# Automatiser l'architecture ? Savoir-faire et calculabilité dans la pratique des courants computationnels en architecture (1965-2020)

Nadja Gaudillière-Jami

UNIVERSITÉ PARIS-EST

**Thèse de doctorat d'Université Paris-Est**

**Architecture**

**NADJA GAUDILLIÈRE-JAMI**

# AUTOMATISER L'ARCHITECTURE?

Savoir-faire et calculabilités dans les pratiques des courants computationnels en architecture, 1965-2020.

***Thèse dirigée par M. Matteo Porrino***
***Co-encadrement par M. Mario Carpo***

Soutenue le 04-02-2022

**Jury :**

M. Pierre LECLERCQ, Professeur des universités, Université de Liège, Rapporteur
Mme Henriette BIER, Associate Professor, Technical University Delft, Rapporteur
M. Olivier COUTARD, Directeur de recherche, CNRS/LATTS
M. Oliver TESSMANN, Professeur des universités, Technische Universität Darmstadt
Mme Antonella TUFANO, Professeure des universités, Paris 1 Panthéon Sorbonne
M. Jean-Aimé SHU, Maître de conférences, École d'architecture de la ville & des territoires Paris-Est

**Résumé.**

Avec le tournant numérique apparaissent à partir des années 60 en architecture de nouvelles méthodes de conception, mais aussi une série de projets expérimentaux, avec pour point commun le recours à des algorithmes et à des outils de programmation – un ensemble de projets que l'on peut regrouper sous l'appellation courant computationnel. Or le recours à un processus de conception procédural, computationnel (informatisé) nécessite de mettre au point une série d'instructions exprimables en langage formalisé, un processus à première vue contradictoire avec les connaissances tacites sur lesquelles s'appuie la pratique architecturale.

Le projet de doctorat propose une analyse à trois niveaux de cette série de projets et du recours aux outils de programmation pour la conception spatiale au cours des 50 dernières années. Dans un premier temps, on cherche à caractériser le courant computationnel d'un point de vue technique : quels processus algorithmiques sont utilisés pour la conception architecturale et quels biais techniques ces processus entraînent-ils ? Dans un second temps, on s'intéresse à la dimension socio-historique de ce courant, en se penchant en particulier sur deux approches de la conception procédurale : l'une favorisant une rationalisation des pratiques architecturales et l'autre choisissant d'explorer le versant heuristique, artisanal, qu'implique la maîtrise des techniques de programmation. Enfin, on s'interroge sur une possible épistémologie de l'architecture computationnelle à partir des notions de savoir-faire et calculabilité.

Le récit qu'offre la thèse du développement historique du champ éclaire le choix épistémologique que représente le recours aux outils algorithmiques. Interroger les pratiques computationnelles, c'est interroger la nature de l'architecture en tant que discipline. La rationalisation des pratiques observée repose sur une ambition de transformation de la pratique architecturale en pratique scientifique. La popularisation de cette approche relève par ailleurs d'une volonté de se conformer à des contraintes socio-économiques pourtant pas toujours en accord avec les ambitions sociales souvent mises au centre de leur pratique par les architectes. La dimension tacite que certaines des expérimentations prospectives du champ computationnel tentent de saisir est au cœur de l'insaisissable définition de l'architecture, dont la réinterprétation par chaque praticien est considérée comme un aspect caractéristique de la discipline. La formalisation que demande la traduction en instructions explicites peut quant à elle être envisagée comme une exploration méthodologique du processus de conception. La cartographie du champ computationnel et de ses enjeux que propose la thèse offre ainsi un point d'entrée privilégié réinterroger la possibilité d'existence d'une épistémologie de l'architecture.

***Abstract.***

*With the digital revolution, new design methods appeared in architecture from the 1960s onwards, leading to a series of experimental projects, with the use of algorithms and programming tools as a common feature - a set of projects that can be grouped together under the common name of computational design. However, the use of a procedural, computational design process requires the development of a series of instructions that can be expressed in a formalised language, a process that at first sight contradicts the tacit knowledge on which architectural practice is based.*

*The doctoral project proposes a three-level analysis of this series of projects and the use of programming tools for spatial design over the last 50 years. First, it seeks to characterise the computational field from a technical point of view: which algorithmic processes are used for architectural design and what technical biases do these processes entail? Secondly, the research looks at the socio-historical dimension of this current, focusing in particular on two approaches to procedural design: one favouring a rationalisation of architectural practices and the other choosing to explore the heuristic, artisanal side of the process, which is involved in mastering programming techniques. Finally, we question a possible epistemology of computational architecture based on the notions of know-how and calculability.*

*The account of the historical development of the field sheds light on the epistemological choice of using algorithmic tools. To question computational practices is to question the nature of architecture as a discipline. The rationalisation of practices observed is based on an ambition to transform architectural practice into scientific practice. The popularisation of this approach is also based on a desire to conform to socio-economic constraints that are not always in line with the social ambitions that architects often place at the centre of their practice. The tacit dimension that some of the prospective experiments in the computational field attempt to capture is at the heart of the elusive definition of architecture, the reinterpretation of which by each practitioner is considered a characteristic aspect of the discipline. The formalisation that translation into explicit instructions requires can be seen as a methodological exploration of the design process. The cartography of the computational field and its stakes that the thesis proposes thus offers a privileged point of view into the possibility of existence for an epistemology of architecture.*

## REMERCIEMENTS

*À ma famille de chercheurs*
*et aux chercheurs de ma famille.*

L'aventure solitaire qu'est la thèse ne l'est pas toujours. L'écriture terminée, il me reste donc à remercier les personnes sans lesquelles ce travail n'aurait pas été ce qu'il est.

À mon directeur de thèse, Matteo Porrino, je voudrais dire merci pour sa patience et son infaillible soutien dans tous les domaines : administratif, professionnel et intellectuel. Il a accepté de s'embarquer avec moi dans l'étude d'un domaine alors peu familier, et il a su me guider tout au long de l'aventure, questionnant avec précision l'ensemble de mes choix pour garantir la qualité du travail produit. Je remercie également mon co-encadrant Mario Carpo pour sa clairvoyance et son humour, ses encouragements à ma prise d'indépendance intellectuelle, les précieuses mises en contact, et nos rendez-vous brefs, intenses et motivants.

Une thèse de doctorat, c'est aussi l'accueil au sein d'un laboratoire : je remercie donc les membres du laboratoire Géométrie Structure Architecture pour nos échanges et nos rares mais joyeuses retrouvailles. Je voudrais également remercier les membres de l'école doctorale Ville, Transports et Territoire de l'Université Paris-Est pour leur soutien logistique tout au long de mes années de doctorat. Je remercie aussi Olivier Coutard et Jean-Aimé Shu pour nos réunions annuelles et pour nos agréables discussions, bien plus larges que le sont souvent celles d'un comité de suivi. J'ai également eu la chance d'occuper en parallèle de mon doctorat un poste de maîtresse de conférence associée à l'École Nationale Supérieure Paris-Malaquais. Chance, car le métier de chercheur auquel prépare la thèse est en fait celui d'enseignant-chercheur, et l'enseignement en est une part importante. Je remercie donc mes collègues de l'École Nationale Supérieure Paris-Malaquais pour les années formatrices passées à leur côté, en particulier Robert Le Roy. Je remercie également l'équipe de Digital Matter, tant pour l'initiation aux pratiques numériques en architecture que pour les années d'enseignement partagé qui ont suivi.

Je voudrais aussi remercier l'ensemble des praticiens qui ont pris le temps de répondre à mes questions sur leurs travaux. Sans eux, je n'aurais pu construire l'image du champ que je dévoile dans ce texte. Universitaires aussi pour beaucoup d'entre eux, ils savaient déjà ce que réserve l'aventure de la thèse à ceux qui la tentent. Je remercie donc en particulier Iain Maxwell et Tim Schork - sans doute

sans même le savoir, ils m'ont donné quelques conseils pratiques auxquels je n'ai eu de cesse de revenir au fil du temps.

Je voudrais remercier Oliver Tessmann, dont les conseils en cours de rédaction et le soutien sans faille, malgré un temps d'écriture qui s'allongeait, ont été précieux. Sa décision de me recruter à l'été 2020 m'a par ailleurs offert des conditions privilégiées pour la rédaction du texte. Ce nouveau poste n'a pas été seulement un confort matériel, il m'a aussi permis d'échanger avec des chercheurs chevronnés, issus du champ computationnel comme de l'histoire de l'architecture. Ces échanges m'ont confortée dans l'aventure entreprise et m'ont été d'un grand soutien au moment de faire les choix qui ont structuré le récit proposé dans ces pages. Je remercie donc également les membres du cluster LOEWE Architectures of Order et les membres du laboratoire Digital Design Unit de la TU Darmstadt : Anton Savov, Max Eschenbach, Andrea Rossi, Samim Mehdizadeh, Shayani Fernando et Mehrzad Esmaeili Charkhab.

À Irène, bien plus chercheuse qu'elle ne le croit elle-même parfois, merci pour sa relecture attentive, durant laquelle elle a su saisir les implications les plus fines du texte et me permettre à travers ses commentaires de les rendre plus claires, et pour son attention à des détails que tout le monde sauf elle oublie malgré leur importance. À Jean-Paul, merci pour son soutien méthodologique, pour les conseils de lecture qui m'ont aidée à positionner ce travail quelque part entre histoire de l'architecture et sociologie des sciences, et aussi pour ses piques affectueuses qui ont tant contribué à normaliser les péripéties émotionnelles qu'on rencontre toujours au cours d'une thèse. À Flora, merci pour les cartes postales et les dinosaures dans la contemplation desquels puiser un peu de motivation lorsque cela devient nécessaire. À Jacques et Léna, merci pour leur soutien affectueux et pour la rituelle question, à laquelle on n'échappe donc pas même dans une famille de chercheurs : "quand est-ce que tu soutiens ?".

Sans les fous rires, les coups de gueule, les recommandations de lecture, les conseils méthodologiques et les astuces administratives partagés avec mes compagnons de route, ce doctorat aurait été bien sombre. À Ian, Justin, Flo, Pierre, Maxime, Matthieu et Alexis, merci pour les encouragements, les relectures de dernière minute et les conversations qui se prolongent. À Claire, merci pour ces précieuses journées d'écriture, pour les riches échanges sur nos travaux respectifs, et pour cette belle initiation à l'histoire de l'art du XIX$^{e}$ au fil des années. À Perceval, merci pour les heures passées de la première à la dernière minute de ce doctorat à m'expliquer des notions de logique variées, pour les parties de Magic et les articles sur les intelligences artificielles qui y jouent, pour la vérification minutieuse des raisonnements, pour l'écoute, et pour tout le reste.

À Adrien, merci pour son endurance incroyable aux anecdotes hasardeuses sur le champ computationnel en architecture, pour ses relectures et ses encouragements à aller toujours plus loin, pour les expérimentations culinaires entre deux longues discussions sur l'architecture, le design et leur enseignement - au fil de ces discussions, beaucoup de choses qui se sont avérées cruciales pour le développement de cette thèse. Merci pour tout et la suite.

***Automatiser l'architecture ? Savoir-faire et calculabilité dans les pratiques des courants computationnels en architecture (1965-2020)***

## LISTE DES FIGURES

### CHAPITRE II. Une filiation avec le monde de l'intelligence artificielle

**CHAPITRE III. Une filiation avec le monde de l'animation**

Figure 1. Réseau du champ computationnel dans sa deuxième période de développement.
Source : Nadja Gaudillière-Jami.

Figure 2. Extraits des règles de composition présentées par George Stiny et James Dips.
Source : Stiny, George et James Dips, "Shape Grammars and the Generative Specification of Painting and Sculpture", *IFIP Congress, Information Processing 71*, 1972, p. 1460-1465.

Figure 3. Exemples de règles et leur résultat dans le projet de grammaire palladienne de George Stiny et William Mitchell.
Source : Stiny, George et William J. Mitchell, "The Palladian Grammar", *Environment and Planning B*, Vol. 5, n°1, pp. 5-18, 1978.

Figure 4. Exemples de règles et leur résultat dans le projet de variations de palais de justices miesiens par Athanassios Economou et James Park.
Source : Park, James et Athanassios Economou, "The Dirksen variations: Towards a generative description of Mies's courthouse language", dans Martens, Bob (éditeur), *eCAADe 2015 : proceedings of the 33rd International Conference on Education and Research in Computer Aided Architectural Design in Europe, 16-18 September 2015, Vienna, Austria. Vol. 1, Real time : extending the reach of computation*, Vienna : eCAADe : Faculty of Architecture and Regional Planning, 2015.

Figure 5. Sommaire annales de 1991 de la conférence Artificial Intelligence in Design 1991.
Source : Gero, John S. (éditeur), *Artificial Intelligence in Design '91*, Elsevier, 1991.

Figure 6. Exemple de production du système expert EAVE, développé par A. Radford et J.R. Mitchell.
Source : Mitchell, J. R. et Antony Radford. "EAVE, a Generative Expert System for Detailing", *Environment and Planning B: Planning and Design*, Vol. 14, p. 281 - 292, 1987.

Figure 7. Logique d'application des règles étape par étape pour la création d'un plan.
Source : Coyne, Richard, D. et John S. Gero, "Design Knowledge and Sequential Plans", *Environment and Planning B: Planning and Design*, Vol. 12, n°4, p. 401-418, 1985.

Figure 8. Bloom House, Greg Lynn FORM.
Source : https://glform.com/

Figure 9. Embryological House, Greg Lynn FORM.
Source : https://glform.com/

Figure 10. Panneau de bois, Objectile.
Source : https://www.frac-centre.fr/collection-2.html

Figure 11.a. Walt Disney Concert Hall, Gehry Partners.
Source : https://www.getyourguide.fr/

Figure 11.b. Fondation Louis Vuitton, Gehry Partners.
Source : Xavier Richer.

Figure 11.c. Fondation Luma, Gehry Partners.
Source : Sabrina Ranvier.

Figure 12.a. Weisman Art Museum, Gehry Partners.
Source : David Joyner.

Figure 12.b. Musée Guggenheim de Bilbao, Gehry Partners.
Source : Erika Ede.

Figure 13.a. Peinture préparatoire pour le projet de la caserne Vitra, Zaha Hadid.
Source : https://www.zaha-hadid.com/

Figure 13.b. Caserne Vitra, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 14.a. Tremplin de Bergisel, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 14.b. Musée d'Ordrupgaard, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 14.c. Centre Heydar Aliev, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 14.d. Abu Dhabi Performing Art Center, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 15.a. Stranded Sear Tower, Greg Lynn FORM.
Source : https://glform.com/

Figure 15.b. Blobwall, Greg Lynn FORM.
Source : https://glform.com/

Figure 16.a. One World Trade Center, Skidmore, Owings & Merrill.
Source : https://www.som.com/

Figure 16.b. 100 Mount Street, Skidmore, Owings & Merrill.
Source : https://www.som.com/

Figure 16.c. Guoco Tower, Skidmore, Owings & Merrill.
Source : https://www.som.com/

Figure 17. AlloExoBio, Marcos Novak.
Source : Roussel, Marion, "A la couture des mondes… Transarchitecture et hypersurfaces : une introduction", consulté le 30 Novembre 2021.
https://dnarchi.fr/analyses/a-la-couture-des-mondes-transarchitecture-et-hypersurfaces-une-introduction/

Figure 18.a. Saltwater Pavilion, ONL.
Source : https://www.frac-centre.fr/collection-2.html

Figure 18.b. FreshWater Pavilion, NOX.
Source : https://www.frac-centre.fr/collection-2.html

**CHAPITRE IV. Un savoir-faire propre au champ**

Figure 1. Réseau de la troisième génération du champ computationnel.
Source : Nadja Gaudillière-Jami.

Figure 2. Productions de l'University of East London.
Source : Coates, Paul, Tom Appels, Corinna Simon et Christian Derix, "Current work at CECA", dans Soddu, Celestino (éditeur), *Proceedings of the 4th Generative Art Conference (GA2001)*, Milan: Generative Design Lab Milan Polytechnic University, Italy, 2001.

Figure 3. Productions du AA Design Research Laboratory.
Source : https://drl.aaschool.ac.uk/

Figure 4. Productions de Sci_Arc.
Source : https://www.sciarc.edu/

Figure 5. Thèmes des numéros de AD sur la période 2000-2020.
Source : Nadja Gaudillière-Jami.

Figure 6. Praticiens cités dans AD.
Source : Nadja Gaudillière-Jami.

Figure 7. Nombre de numéros consacrés à des thèmes du champ computationnel par an.
Source : Nadja Gaudillière-Jami.

Figure 8. 1000 logements avec cours pour Tianherenjia, Hubert & Roy.
Source : http://hubert-roy.com/

Figure 9.a. DFAB House, Gramazio/Kohler.
Source : https://gramaziokohler.arch.ethz.ch/

Figure 9.b. Armadillo Vault, Block Research Group.
Source : https://block.arch.ethz.ch/

Figure 9.c. Noeud rotoformé, Digital Design Unit.
Source : https://www.dg.architektur.tu-darmstadt.de/fachgebiet_ddu/index.en.jsp

Figure 9.d. Chai Gantenbein, Gramazio/Kohler.
Source : https://gramaziokohler.arch.ethz.ch/

Figure 10.a. Babi Yar Memorial, kokkugia.
Source : https://www.kokkugia.com/

Figure 10.b. Pavillon Seroussi, Xefirotarch.
Source : https://www.hda-x.co/

Figure 10.c. theverymany.
Source : https://theverymany.com/

Figure 11.a. Archiglobe & Bollinger + Grohmann.
Source : Tessmann, Oliver, “Collaborative Design Procedures for Architects and Engineers”, Thèse de doctorat, Université de Kassel, 2008.

Figure 11.b. Code dans une présentation de supermanoeuvre.
Source : supermanoeuvre.

Figure 11.c. Code dans une présentation de kokkugia.
Source : https://www.kokkugia.com/

Figure 12. Jardin de la Villa Bloc.
Source : Florent Darrault, Didier Raux, Natalie Seroussi.

Figure 13. Les six réponses au concours du pavillon Seroussi.
Source : Guenoun, Elias (éditeur), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.

**CHAPITRE V. Des outils pour transmettre**

Figure 1. Règles typologiques des systèmes de croissance.
Source : Nadja Gaudillière-Jami.

Figure 2.a. The Tote, Serie Architects.
Source : https://www.serie.co.uk/

Figure 2.b. et 2.c Bibliothèque Nationale Tchèque, OCEAN North.
Source : https://www.ocean-a-e.com/projects

Figure 3.a. Chou romanesco.
Source : https://www.futura-sciences.com/

Figure 3.b. Vol Libre, Loren Carpenter.
Source : https://digitalartarchive.siggraph.org/artwork/loren-carpenter-vol-libre/

Figure 4. Pavillon Bloomberg, Akihisa Hirata.
Source : Takumi Ota (akihisa hirata architecture office)

Figure 5.a. Pavillon Turing, biothing.

Source : https://www.alisaandrasek.com/

Figure 5.b. An Evolutionary Architecture, John Frazer.
Source : Frazer, John, *An evolutionary architecture*, Architectural Association Publications, 1995.

Figure 5.c. Unprintable Forms, Marjan Colletti et al.
Source : Adilenidou, Yota, Zeeshan Yunus Ahmed, Freek Bos et Marjan Colletti, "Unprintable Forms. Complexity as a Robustness Factor for Robotic Fabrication and 3DCP Constraints through Error Elimination and Reinsertion", dans Bieg, Kory, Danelle Briscoe et Clay Odom (éditeurs), *ACADIA 19: Ubiquity and Autonomy. Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture, The University of Texas at Austin School of Architecture, Austin, Texas 21-26 October, 2019*, ACADIA, 2019, p.168-177.

Figure 6. Réseau de développement des systèmes de croissance.
Source : Nadja Gaudillière-Jami.

Figure 7. Chronologie de développement des systèmes de croissance.
Source : Nadja Gaudillière-Jami.

Figure 8. Chronologie de développement des systèmes multi-agents.
Source : Nadja Gaudillière-Jami.

Figure 9. Règles typologiques des systèmes multi-agents.
Source : Nadja Gaudillière-Jami.

Figure 10.a. Mesonic Fabrics, biothing.
Source : https://www.frac-centre.fr/collection-2.html

Figure 10.b. Cliff House, kokkugia.
Source : https://www.kokkugia.com/

Figure 10.c. Trabeculae / Protosynthesis, supermanoeuvre.
Source : https://www.evolo.us/trabeculae-re-imagining-the-office-building/

Figure 11. Réseau de développement des systèmes multi-agents.
Source : Nadja Gaudillière-Jami.

Figure 12.a. Modèle inversé de la Sagrada Familia, Antonio Gaudi.
Source : Geoffrey D'Cruz.

Figure 12.b. Bulles de savon, Frei Otto.
Source : Institut für Leichte Flächentragwerke.

Figure 12.c. Modèles de coques inversées, Heinz Isler.
Source : Asmaljee, Zaahir, "Form-finding of thin shell structures", 2014.

Figure 13.a. Couverture de la cour du British Museum, Foster & Partners.
Source : Diliff (Wikimedia Commons).

Figure 13.b. Stade Olympique de Munich, Frei Otto.
Source : Tiia Monto (Wikimedia Commons).

Figure 13.c. Opéra de Sydney, Jorn Utzon.
Source : Bjarte Sorensen (Wikimedia Commons).

Figure 14.a. Armadillo Vault, Block Research Group.
Source : https://block.arch.ethz.ch/

Figure 14.b. Scénographie de l'exposition *Architectures Non Standard*, EZCT Architecture & Design Research.
Source : Migayrou, Frédéric (éditeur), *Architectures Non-Standard*, Paris Centre Pompidou, 2003.

Figure 14.c. Minima Maxima, The Very Many.
Source : https://theverymany.com/

Figure 14.d. Sensory Playscapes, Sean Ahlquist.
Source : Ahlquist, Sean, "Sensorial Playscape: Advanced Structural, Material and Responsive Capacities of Textile Hybrid Architectures as Therapeutic Environments for Social Play", dans Menges, Achim, Bob Sheil, Ruairi Glynn, et Marilena Skavara, *Fabricate 2017. Rethinking Design and Construction*, UCL Press, 2017.

Figure 15. Règles typologiques d'un système de relaxation dynamique.
Source : Nadja Gaudillière-Jami.

Figure 16. Réseau de développement des systèmes de simulation physique.
Source : Nadja Gaudillière-Jami.

Figure 17. Chronologie de développement des systèmes de simulation physique.
Source : Nadja Gaudillière-Jami.

Figure 18. Règles typologiques d'un algorithme génétique.
Source : Nadja Gaudillière-Jami.

Figure 19. Biomorphs, Richard Dawkins.
Source : Dawkins, Richard, *The Blind Watchmaker*, New York: W. W. Norton & Company, 1986.

Figure 20. Chronologie de développement des algorithmes génétiques.
Source : Nadja Gaudillière-Jami.

Figure 21.a. Chaises, EZCT Architecture & Design Research.
Source : EZCT Architecture & Design Research, "Studies on Optimization: Computational Chair Design", FRAC Centre, 2009.

Figure 21.b. Tour, Skidmore, Owings & Merrill.

Source : Besserud, Keith et Joshua Cotten, “Architectural Genomics”, intervention dans le cadre de la conférence *ACADIA 2008: Silicon + Skin: Biological Processes and Computation*, Université de Minneapolis, 16-19 Octobre 2008.

Figure 22. Réseau de développement des algorithmes génétiques.
Source : Nadja Gaudillière-Jami.

Figure 23.a. Outil GENR8, Emergent Design Group.
Source : Hemberg, Martin, Una May O’Reilly, Achim Menges, Katrin Jonas, Michel da Costa Gonçalves et Steven R. Fuchs, “Genr8: Architects’ Experience with an Emergent Design Tool”, dans Romero Juan et Penousal Machado (éditeurs), *The Art of Artificial Evolution. Natural Computing Series*, Berlin, Heidelberg : Springer,, 2008, p. 167-188.

Figure 23.b. An Evolutionary Architecture, John Frazer.
Source : Frazer, John, *An evolutionary architecture*, Architectural Association Publications, 1995.

Figure 23.c. Projet MARS, kokkugia.
Source : https://www.kokkugia.com/

Figure 24.a. Babi Yar Memorial, kokkugia.
Source : https://www.kokkugia.com/

Figure 24.b. et 24.c. Terminal Aéroport Riga, kokkugia.
Source : https://www.kokkugia.com/

Figure 24.d. National Art Museum of China, kokkugia.
Source : https://www.kokkugia.com/

Figure 25.a. Salle d’escalade, Robert Stuart-Smith.
Source : https://robertstuart-smith.com/

Figure 25.b. Fold Pavilion, Roland Snooks.
Source : http://www.rolandsnooks.com/

Figure 25.c. Kazakhstan Symbol, Roland Snooks.
Source : http://www.rolandsnooks.com/

Figure 25.d. Bibliothèque d’Helsinki, Robert Stuart-Smith.
Source : https://robertstuart-smith.com/

Figure 26. Projets de l’agence biothing utilisant la bibliothèque Genware : Bifid, Orbita, Mesonic fabrics et le pavillon Seroussi.
Source : https://www.frac-centre.fr/collection-2.html

Figure 27. Interface des logiciels Generative Components et Grasshopper.
Source : Paulo Teves Silva et https://www.stress-free.co.nz/internet_enabling_generative_components_for_a_new_breed_of_aec_consultant

Figure 28. Interface du logiciel Processing.
Source : https://doc.ubuntu-fr.org/processing

Figure 29. Interface du logiciel Revit et de son plug-in Dynamo.
Source : Hexabim.

Figure 30. Interface du logiciel Rhinocéros.
Source : Block Research Group.

Figure 31. Interface du plug-in Octopus.
Source : Robert Vier.

Figure 32. Structure d'interface pour deux exemples d'algorithmes - prêt à l'emploi et sur-mesure.
Source : Nadja Gaudillière-Jami.

Figure 33. Exemples de structures d'interface par type d'interface et par typologie.
Source : Nadja Gaudillière-Jami.

Figure 34. Structure d'interfaces types.
Source : Nadja Gaudillière-Jami.

Figure 35. Flux de travail du projet du Pavillon Seroussi.
Source : Guenoun, Elias (éditeur), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.

Figure 36. Flux de travail du projet Associative Component Structure.
Source : Hensel, Michael et Achim Menges, "Material and Digital Design Synthesis: Integrating Material Self-Organisation, Digital Morphogenesis, Associative Parametric Modelling and Computer-Aided Manufacturing", *AD Architectural Design*, Vol. 76, n° 2, p. 88-97, 2006.

Figure 37. Chronologie des apparitions d'interfaces par typologies.
Source : Nadja Gaudillière-Jami.

Figure 38. Chronologie cumulée des apparitions d'interfaces.
Source : Nadja Gaudillière-Jami.

Figure 39. Chronologie des apparitions de plug-ins par typologies.
Source : Nadja Gaudillière-Jami.

**CHAPITRE VI. Une mue du champ**

Figure 1. Profils de connaissances.
Source : Nadja Gaudillière-Jami.

Figure 2. Trajectoires exemples pour le profil architecte-programmeur.
Source : Nadja Gaudillière-Jami.

Figure 3. Trajectoires exemples pour le profil ingénieur-programmeur.
Source : Nadja Gaudillière-Jami.

Figure 4. Trajectoires exemples pour le profil développeur.
Source : Nadja Gaudillière-Jami.

Figure 5. Évolution des profils.
Source : Nadja Gaudillière-Jami.

Figure 6. Terminal d'Anaheim, HOK.
Source : John Linden pour Archdaily.

Figure 7. Utilisateurs potentiels.
Source : Nadja Gaudillière-Jami.

**CHAPITRE VII. Démocratiser, rationaliser**

Figure 1.a. Tour Leeza, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 1.b. Hotel Morpheus, Zaha Hadid Architects.
Source : https://www.zaha-hadid.com/

Figure 2.a. ElbPhilarmonie, Herzog & De Meuron.
Source : https://www.herzogdemeuron.com/

Figure 2.b. Pavillon FFM, Bollinger + Grohmann.
Source : https://www.bollinger-grohmann.com/

Figure 3. "*Get creative with generative design*".
Source : https://www.spacemakerai.com/

## LISTE DES TABLEAUX

Instruction tables will have to be made up by mathematicians with computing experiences and perhaps a certain puzzle-solving ability. There will probably be a great deal of work to be done, for every known process has got to be translated into instruction table form at some stage. The process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself.

*Les tables d'instructions devront être élaborées par des mathématiciens ayant une expérience de l'informatique et peut-être une certaine capacité à résoudre des énigmes. Il y aura probablement beaucoup de travail à faire, car chaque processus connu doit être traduit sous forme de tableau d'instructions à un moment donné. Le processus de construction des tableaux d'instructions devrait être très fascinant. Il n'y a pas de réel danger que cela devienne une corvée, car tous les processus qui sont plutôt mécaniques peuvent être confiés à la machine elle-même.*

Alan Turing, 1946

## NOTE AUX LECTEURS ET LECTRICES

Avant d'entamer le texte qui suit, nous souhaitons fournir des indications sur quelques points pour orienter la lecture et clarifier certains choix de rédaction.

Le premier point concerne les indications techniques et historiques fournies au sujet de certains domaines. Nous avons souhaité rédiger un texte qui permette à tout lecteur ou toute lectrice de saisir les enjeux dont il est question, quel que soit son niveau en informatique et en programmation. Or, d'informatique, de programmation et d'outils numérique il est question tout au long du texte. Il nous a donc fallu expliciter de nombreux concepts issus de ces domaines. Par ailleurs, les recherches présentées comportent une enquête sur les liens entre l'architecture computationnelle et le domaine de l'intelligence artificielle. Pour permettre à tous de saisir les résultats de cette enquête, nous avons également choisi de retracer en détail certains moments de l'histoire du champ de l'intelligence artificielle. Les lecteurs et lectrices déjà familiers de cette histoire et des éléments techniques communiqués dans certaines parties du texte pourront se reporter directement aux paragraphes d'introduction et de fin de partie pour saisir le rôle de ces rappels dans le propos général, s'ils ne souhaitent pas lire ces parties dans leur entièreté.

La bibliographie et le terrain examinés au cours de la recherche sont majoritairement anglophones. Nous avons donc été confrontés à nombre de vocables anglais, qui n'ont pas toujours d'équivalent en français. C'est parfois le cas parce que c'est le mot anglais qui est usuellement employé dans le champ, même dans les discussions francophones : par exemple, on emploie généralement le mot *plugin* plutôt que ses traductions *module d'extension*, *module externe*, *greffon* ou *plugiciel*. C'est aussi parfois le cas parce que le mot anglais ne possède pas d'équivalent approprié en français, comme par exemple le mot *computational*. Pour que le manuscrit puisse être autant que possible intégralement en français, nous nous sommes efforcés de traduire au mieux ces mots. Le texte présenté ici porte les traces de cet effort. Nous espérons que les lecteurs et lectrices ne nous tiendront pas rigueur des éventuelles maladresses en résultant. Nous avons explicité les traductions les plus importantes au Chapitre I. Les vocables originaux sont par ailleurs toujours précisés en note de bas de page lorsqu'une telle traduction a été effectuée.

Enfin, nous avons longuement hésité quant à la féminisation du texte. L'informatique et l'architecture computationnelle sont des domaines où les hommes dominent dans de nombreux contextes. Les femmes n'en sont pourtant pas absentes. Les textes consacrés à ces domaines prennent cependant rarement la peine d'acter leur présence en féminisant l'écriture lorsque cela est nécessaire. Ainsi, on écrit rarement au sujet des programmeuses, et on prend encore moins souvent la peine d'indiquer qu'il

s’agit de “programmeurs et programmeuses”, même quand les deux sexes sont représentés au sein du groupe dont il est question. Écrire pour désigner ces groupes de manière générale “programmeurs”, “utilisateurs” et “développeurs” invisibilise encore un peu plus les femmes qui en font partie. Par crainte de la lourdeur d’écriture résultant de la précision systématique “programmeurs et programmeuses”, “utilisateurs et utilisatrices” et “développeurs et développeuses” ou de l’emploi du point médian, c’est pourtant le choix que nous avons fait. De la même manière, nous avons choisi d’écrire “étudiants”, “enseignants” et “chercheurs”, laissant là aussi le masculin l’emporter par commodité d’écriture. Bien que cela nous paraisse effectivement rendre la lecture plus confortable, nous avons conscience qu’il ne s’agit pas là d’un choix idéal, et nous prions nos lectrices de nous en excuser.

# INTRODUCTION

Pour les générations d'architectes les plus jeunes, l'ordinateur est une évidence. Même pour les praticiens nés avant que nous ayons tous ou presque un smartphone dans la poche, les ordinateurs familiaux et professionnels sont accessibles depuis plusieurs générations. Logiciels et applications les plus divers font partie du quotidien. Pour les plus jeunes, difficile d'imaginer une activité ou un champ professionnel sans les outils informatiques qui lui sont associés. L'architecture ne fait pas exception, elle qui ajoute à la cohorte des outils de communication et de gestion auxquels nous avons tous recours modeleurs 3D et logiciels de dessin. Les outils numériques sont pourtant étrangement peu présents en école d'architecture, nuage d'outils de représentation que les élèves explorent d'eux-mêmes pour produire les dessins attendus. Nombre d'écoles françaises évitent encore le recours au moindre logiciel au cours de la première année. Cette première année est donc surtout peuplée de dessins faits à la main par les étudiants. L'ordinateur semble susciter une certaine méfiance chez beaucoup de professeurs. Quelques départements sont pourtant spécialisés dans ce qu'on appelle parfois dans les écoles françaises "le digital" - anglicisme bâtard visant à démarquer les pratiques du département du simple recours à des outils numériques de dessin comme AutoCAD. L'objet du travail de ces départements est en effet l'application des processus computationnels à la conception architecturale. Autrement dit, le recours à de la programmation informatique au cours de la conception, une toute autre histoire que l'usage des outils de gestion, communication et représentation usuels. À notre époque informatique, il semble en effet difficilement contournable d'apprendre à programmer, ou du moins de s'intéresser à la place de l'informatique en architecture. Étant donné l'omniprésence du numérique, il faut sans doute nécessairement s'y confronter, interroger la place des algorithmes dans la conception architecturale. Pour un novice du domaine se pose alors la question de comment aborder les pratiques qui constituent le champ computationnel en architecture.

Nombre de praticiens sont convaincus de l'importance de l'écrit dans l'appréhension des enjeux de la conception de l'espace. Le champ computationnel est traité par des auteurs décisifs dans le développement des théories de l'architecture contemporaines, Greg Lynn et Bernard Cache en tête. Mais d'autres écrits permettent d'explorer la question de l'architecture computationnelle, et notamment le rapport aux outils techniques qui s'y noue. Des textes d'auteurs aujourd'hui classiques de l'histoire de l'architecture, notamment de l'histoire des mouvements modernes : Siegfried Giedion, Reyner Banham, Alan Colquhoun, Charles Jencks, Robin Evans. Mais aussi des architectes qui ont accordé tout au long de leur carrière autant d'importance à l'écriture qu'à la pratique, et qui ont marqué leur époque par leurs réflexions, Peter Eisenman et Rem Koolhaas en particulier.

L'articulation entre théorie et pratique en architecture est cruciale pour comprendre les pratiques computationnelles. Mais pour découvrir le champ computationnel, il ne s'agit pas seulement de lire, il faut aussi se pencher sur l'écriture du code. L'apprentissage de la programmation elle-même a son importance. Il s'agit d'apprivoiser la grande variété d'environnements et de langages de programmation disponibles. Grasshopper et Processing bien sûr, piliers de l'apprentissage de ces pratiques dans le milieu du design génératif et de l'architecture computationnelle. Mais aussi Mathematica, Python ou C++, piliers non de l'apprentissage de la programmation mais du développement d'outils logiciels. Ces diverses ressources sont mises en application par les praticiens dans des projets multiples, en école comme en agence. À toutes les échelles, de l'assemblage à la planification urbaine en passant par le bâtiment. À toutes les étapes de projet, de l'esquisse à la fabrication de prototypes 1:1. À tout niveau de prospective et un peu partout dans le monde. Ce premier versant de l'apprentissage est essentiel. Difficile de saisir les enjeux dont il est question, difficile de saisir la logique de la programmation, si différente de la logique de conception architecturale usuelle, sans mettre les mains dans le cambouis.

À cet apprentissage de la programmation s'ajoute la découverte du corpus de projets associés, de ce que les architectes font des algorithmes et des machines qui sont les outils essentiels du champ computationnel. Les blobs de Greg Lynn, ces formes courbes présentées comme caractéristiques du recours aux modeleurs 3D. Les panneaux non-standard de Bernard Cache, ondulations de bois parfaitement lissées par les machines à commande numériques utilisées pour les fabriquer. Les drones de Gramazio & Kohler, dont le ballet produit des empilements vertigineux de briques. Les feuillets de bois de l'ICD de Stuttgart, qui se courbent en réaction aux changements hygrométriques et ouvrent ainsi peu à peu des fenêtres au milieu des panneaux ou ils sont installés. Les vaisseaux aliens de Marcos Novak, flottant dans les airs au milieu de paysages virtuels. Les kits d'assemblage de Skylar Tibbits, prenant forme d'eux-mêmes au moment voulu, par la magie d'un changement d'environnement. Les fleurs, les ondulations et les champs magnétiques de biothing, qui se déploient en canopées délicates. Les sculptures colorées de Marc Fornes, tour de force de gestion de l'assemblage de centaines de pièces différentes en des formes inextricables. La prise d'importance de la fabrication numérique, devenue l'un des sujets principaux d'expérimentation dans le domaine, a donné lieu à des dizaines de prototypes et de pavillons, accompagnés d'images et d'animations spectaculaires. Cet aspect spectaculaire, c'est justement ce qui convainc tant d'étudiants d'aller se frotter aux pratiques computationnelles quand ils découvrent les prototypes miraculeux fabriqués grâce aux méthodes de modélisation et de fabrication numériques.

Découvrir les pratiques computationnelles en architecture, c'est découvrir pêle-mêle les systèmes multi-agents, la réalité virtuelle et augmentée, le biodesign, les robots six-axes et les imprimantes 3D, les blobs, les simulations de croissance d'arbres, de coraux ou fleurs, les objets non-standard, la

modélisation paramétrique, les algorithmes génétiques. Au milieu de cette furieuse diversité des pratiques computationnelles, l'un des éléments qui ont guidé ce travail de doctorat est la recherche d'un fil conducteur. La volonté d'acquérir une vue d'ensemble cohérente de ce maelström d'expérimentations, de faire sens d'une telle diversité est un des objectifs principaux de ce travail. Acquérir une vue d'ensemble, mais aussi bien sûr la partager. Le texte qui commence ici s'ancre dans l'ambition d'offrir une initiation aux pratiques computationnelles en architecture, mais aussi une analyse permettant d'aller au-delà du fouillis que semble au premier abord le champ. À ce titre, il s'agit de la formalisation d'une histoire de l'incroyable aventure qu'est la programmation informatique en architecture et de ses enjeux.

Parmi les praticiens du champ computationnel comme parmi leurs élèves perdure une fascination pour les outils algorithmiques. Les dessins comme les détails de la genèse de nombre de projets soulignent en effet les qualités architecturales de certaines productions du champ. Leurs qualités de programmation aussi. La méfiance que semble inspirer la programmation à nombre de praticiens et professeurs en dehors du champ ne tarit pourtant pas. En témoigne la caricaturale injonction d'un enseignant, entendue à l'occasion de l'évaluation d'un étudiant porté sur le numérique: "*mais enfin, sortez de vos jeux vidéos !*". Cette méfiance est parfois atténuée en simple doute par une curiosité de bonne foi. Lorsque les praticiens y recourent, il leur faut pourtant toujours expliquer pourquoi le numérique fait sens comme outil pour la conception architecturale. Pourquoi le numérique serait forcément un atout pour l'architecture. Ce n'est pourtant pas toujours vrai. Le numérique ne fait pas automatiquement sens dans la conception architecturale. La déconstruction de cette idée a fait partie du processus de recherche de ce doctorat. Être formé en tant qu'architecte, c'est apprendre à évaluer la pertinence d'une proposition architecturale. Intégrer les pratiques computationnelles à cette formation mène donc à interroger la pertinence des objets architecturaux produits à l'aide de ces outils. Vouloir saisir ce qui fait les pratiques computationnelles, c'est aussi devoir développer un regard critique. Il faut démêler les différentes pratiques, déchiffrer les objectifs qui les guident, éclaircir les enjeux épistémologiques. Le deuxième objectif de ce travail a donc été de comprendre comment évaluer les productions du champ computationnel. Ce ne sont pas toujours exactement les mêmes que les productions architecturales classiques. Esquisses, plans, coupes, perspectives et modèles 3D font partie de ces productions. Des bâtiments aussi, mais seulement parfois. Rares sont les espaces construits alors que c'est pourtant justement le résultat attendu principal en architecture. Prototypes, diagrammes, objets d'exposition sont bien plus fréquents parmi ces productions dont tant de praticiens clament qu'elles sont une révolution architecturale. Autant que de praticiens qui clament qu'elles ne relèvent pas de l'architecture. Il s'agit donc de comprendre comment décider de leur place au sein de la discipline architecturale.

Pour cela, il faut apprendre à les voir. Ce travail de doctorat prend pour point de départ les algorithmes eux-mêmes pour cerner les productions des courants computationnels en architecture. Le corpus de références traité est somme toute un corpus architectural classique si on le considère sous l'angle de ce qu'il donne à voir : dessins, plans et images de synthèse, maquettes, photographies et vidéos, textes explicatifs. Le code lui-même est peu visible, tout juste mentionné dans les textes comme base du processus de conception mis en œuvre. Au fil des magazines, des catalogues d'exposition, des conférences, le cœur du champ computationnel en architecture, ses algorithmes, échappe encore et toujours au lecteur. Les détails techniques sont rarement explicités. Il est rarement possible de consulter le code écrit dans son entièreté, et les fragments publiés ne suffisent pas à en reconstituer la logique complète. Des diagrammes explicitent la logique de conception globale, mais ils ne suffisent pas à saisir vraiment ce que font les algorithmes, encore moins à savoir s'il existe des spécificités en fonction des environnements de programmation. Les algorithmes en eux-mêmes, pourtant l'outil principal des pratiques computationnelles, sont toujours une sorte de non-dit, de non-détaillé. Au premier abord, se pencher sur les courants computationnels implique donc de rester un peu sur sa faim quant au caractère concret de ces pratiques. A partir des matériaux publiés, il est difficile de parvenir à se faire une idée de l'influence de ces outils sur la production architecturale, ou de leur absence d'influence. Difficile aussi de comprendre comment ils sont écrits, dans quelles conditions et dans quels objectifs, et comment ils sont intégrés au processus de conception. Pour comprendre ce que codent exactement les praticiens du champ computationnel, le premier élément ausculté par ce travail de doctorat, ce sont donc les algorithmes eux-mêmes, dans le détail de leur programmation. La présente recherche prend racine dans la volonté de dévoiler ces algorithmes qui sont la matière des pratiques de programmation pour l'architecture, pour mieux comprendre l'influence des outils numériques et interroger leurs biais.

Le travail de cartographie initialement développé pour pouvoir garder trace des milieux d'usage des différentes familles d'algorithmes identifiées s'est avéré extrêmement important pour la compréhension du développement du champ au fil des décennies. Plus que seulement interroger le comment du recours aux algorithmes, la thèse explore en effet qui est à l'origine de ces différentes pratiques, et s'attaque à la caractérisation de la relation des praticiens à leurs outils. Le second élément essentiel est donc de suivre les personnes qui utilisent ces algorithmes. Ceci met en évidence plusieurs types de praticiens, qui développent différentes pratiques à travers le recours à des types d'outils et des familles d'algorithmes qui varient. Le suivi de ces praticiens s'effectue à travers différents milieux : industries de la construction et du logiciel, agences d'architecture, bureaux d'études, écoles d'architecture et laboratoires de recherche. Ces praticiens ont suivi différentes formations dans des pays variés et exercent différentes professions : architecte, ingénieur, développeur. Difficile en effet de cadrer les pratiques du champ computationnel en examinant les seuls architectes, dont le travail

s’effectue rarement en solitaire. Ces praticiens occupent différents rôles, des rôles qui s’opposent et se complètent en couples emblématiques du champ computationnel.

Le premier couple, c’est celui entre utilisateur et programmeur. Le développement de la programmation depuis les années 1960 a permis non seulement d’augmenter la puissance de calcul, mais aussi d’augmenter les familles d’algorithmes, le nombre des langages de programmation et la variété des outils disponibles. Ce mouvement de développement, c’est aussi celui de l’apparition de boîtes noires, ou plus exactement de mécanismes de facilitation du recours à la programmation. Boite noire ou mécanisme de facilitation, la manière de les considérer dépend notamment de qui les utilise, La programmation orientée objet, par exemple, à l’origine de nombreux mécanismes de facilitation, naît d’une volonté des programmeurs eux-mêmes de se faciliter la tâche pour pouvoir concevoir des programmes plus complexes plus rapidement. Mais d’autres boîtes noires sont pensées pour le bénéfice des utilisateurs, qui n’ont ni ces connaissances de programmation ni le recul qu’elles permettent. Ils entretiennent par le biais de ces boîtes noires une relation fort différente aux outils informatiques. Pour comprendre le champ computationnel, il faut donc examiner comment la répartition des outils et des savoirs s’orchestre entre programmeur et utilisateur, et les conséquences de cette répartition sur la production architecturale.

Dans les pratiques examinées, le couple utilisateur-programmeur se retrouve souvent dans l’association entre deux praticiens ou deux laboratoires. Ce sont en fait l’architecte et le programmeur qui se trouvent associés. Dans cette configuration, l’architecte occupe la place d’utilisateur au sein du premier couple identifié. Or, dans ce couple, l’utilisateur, c’est le non-sachant. Et si l’architecte peut à tout instant être ravalé au rang d’utilisateur non-sachant du programme pour concevoir, il faut donc interroger ce qui le différencie de l’habitant. Le second couple est donc celui de l’architecte et de l’habitant. La participation des habitants au processus de conception de l’espace est un sujet en soi en architecture, auquel nombre d’architectes se sont confrontés. C’est aussi un sujet qui traverse le champ computationnel. Les outils algorithmiques et leurs mécanismes de facilitation de l’usage permettent en effet aux habitants de se saisir d’autant mieux de certains pans du processus de conception. Cet aspect n’a pas échappé à certains des architectes qui s’intéressent au participatif, qui intègrent à leur démarche des outils algorithmiques comme médiateurs entre habitant et architecte au cours de la conception. L’architecte redevient alors le sachant, guidant l’habitant non-sachant. L’architecte est cependant toujours le non-sachant du couple utilisateur-programmeur, même s' il devient parfois aussi le sachant en acquérant des connaissances de programmation. La hiérarchie usuelle entre les acteurs du processus de conception architecturale se trouve ainsi bouleversée par le recours aux outils numériques. La question des savoirs propres à la discipline architecturale se pose avec force. La place de l’architecte comme auteur au sein des pratiques computationnelles est

questionnée également. Là encore, il faut interroger la répartition des outils et des savoirs, et ses conséquences sur la discipline architecturale.

La relation entre programmeur et utilisateur, entre architecte et habitant se construit ici autour de l'ordinateur, pierre angulaire des pratiques computationnelles. Les algorithmes sont cruciaux dans la définition de ces pratiques, mais la machine qui permet d'exécuter ces instructions de programmation l'est aussi. Le troisième couple est donc celui de la machine et de l'humain. Le champ computationnel est en effet aussi le lieu d'interrogations sur la possibilité d'automatiser tout ou partie du processus de conception spatiale. Les liens entretenus par les praticiens avec les champs des sciences de l'informatique et de l'intelligence artificielle donnent lieu à nombre d'expérimentations et de réflexions sur la question de l'automatisation. Les contraintes de l'industrie de la construction et de l'architecture de marché du XXe et XXIe siècle font de plus peser sur les praticiens qui se confrontent à l'automatisation un impératif de productivité qui façonne aussi la manière d'aborder cet enjeu. Automatiser l'architecture navigue ainsi entre contraintes, souhaits des praticiens et facultés réelles de la machine. Le travail de doctorat aborde cette question en examinant ces souhaits et ces facultés, les limites techniques et les biais épistémiques des algorithmes ainsi que la place des opérateurs humains dans le processus de conception computationnel. La machine est vue dans le champ parfois comme un outil, parfois comme un assistant, voire un compagnon, en fonction des tâches qu'elle assure. Elle est parfois omnisciente, toute puissante, parfois réduite à un simple crayon virtuel, plus rapide à effacer que son parent papetier. Or la question de l'automatisation dépend autant des ambitions pour la machine que de ses capacités de calculs. Ses opérateurs humains ne sont jamais loin, mais le rôle qu'ils assurent est à géométrie variable en fonction de leur place dans le processus, de leurs savoirs et de leurs convictions. Entre rôle rêvé et rôle effectif, la place de la machine dans le processus est une question cruciale qui donne son titre à la thèse.

Machine, humain, habitant, architecte, programmeur, utilisateur, le recours aux algorithmes interroge la place de chacun dans le processus de conception. Dès ses débuts, le champ computationnel intègre ces questionnements à ses pratiques. L'identité des praticiens joue, tout comme les domaines dans lesquels le champ prend racine. Ce sont ces origines que le texte commence par interroger. La dimension historique, premier point abordé pour chercher un fil conducteur à ce récit, offre pour point de départ une chronologie des institutions, des ouvrages et des projets. Ce versant historique du questionnement implique une périodisation et met très vite en évidence des générations de praticiens du champ computationnel. Or la notion de génération interroge la question de la transmission entre praticiens. Ceci interpelle la question du savoir-faire, plus exactement des particularités du savoir-faire des praticiens : que changent les algorithmes à la pratique architecturale ? La question du savoir-faire implique par la suite de se pencher sur les outils eux-mêmes, sur les biais qu'ils imposent à la pratique et sur les intentions qui leur donnent forme. Ce sont ces questions que le texte explore,

déployant au fil des chapitres quatre générations de praticiens, mais aussi une analyse des outils qu'ils utilisent et de la relation qu'ils construisent à ces outils.

À partir de ces questionnements, le texte se structure en présentant les méthodes de recherche employées dans un premier chapitre, puis en déroulant le récit de la structuration des pratiques computationnelles en architecture sur six chapitres. Ces chapitres sont construits pour refléter l'évolution chronologique du champ. Le récit propose quatre phases distinctes de cette évolution, avec chacune leurs cercles et leurs pratiques, et chacune héritant certains savoirs et habitudes des précédentes. Les chapitres II, III et IV sont consacrés aux trois premières phases d'existence du champ. Ces périodes l'amènent à une forme de maturité, à des critères de pratique précis et à une reconnaissance en dehors de sa propre sphère. Le chapitre VII décrit la quatrième et dernière phase, la plus contemporaine. Alors que les phases précédentes identifiées amènent le champ à maturité, cette dernière phase est le résultat d'une mue du champ qui le transforme. Les conditions de cette mue sont décrites dans les chapitres V et VI. L'un se penche sur les changements observés dans les algorithmes, l'autre sur les modifications dans les trajectoires des praticiens. L'évolution des deux objets suivis pour comprendre ce qui fait les pratiques computationnelles permet aussi de mieux cerner la transformation qui s'est opérée dans les pratiques du champ, et son visage contemporain. Cette transformation s'explique notamment à travers la place occupée par les enjeux de l'apprentissage des pratiques de programmation en architecture. Ainsi, tout comme cette introduction, le texte est traversé de questions autour du savoir-faire et de la transmission. Il ne sera plus question de mon apprentissage, mais de celui qui a lieu dans le champ étudié : apprivoiser les outils, identifier et transmettre le savoir-faire de la programmation architecturale. Nous allons le voir, les conditions de cet apprentissage sont un choix épistémologique qui permet d'élaborer une réflexion sur les pratiques computationnelles, mais aussi plus largement sur la discipline architecturale.

# CHAPITRE I.

# -

# Méthodes

## 1.1 Le champ computationnel en architecture

### *1.1.1 Programmation et architecture*

Nous commençons ce texte avec pour point de départ une expression : le *champ computationnel en architecture*. Avant de nous plonger dans les conditions d'émergence de ce champ et dans l'analyse des caractéristiques des pratiques concernées, il nous faut définir précisément ce que nous entendons par là. Les pratiques computationnelles en architecture font référence au recours à des algorithmes pour la conception spatiale. Un algorithme, selon le dictionnaire, est un "*ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations*"[1]. Le terme algorithme vient du terme latin *algorithmus*, lui-même latinisation du nom d'un mathématicien arabe du IXe siècle, Muḥammad ibn Mūsā al-Khwārizmī. L'un de ses ouvrages voyage au XIIe siècle jusqu'en Europe, où son titre, "Al-Khwārizmī Concerning the Hindu Art of Reckoning", est traduit par *Algoritmi de numero Indorum*[2]. Son nom ainsi latinisé en vient ensuite à désigner les règles à suivre pour la résolution d'un problème mathématique. Un algorithme peut également être défini comme une série d'instructions à suivre pour parvenir à un objectif. Néanmoins, ainsi défini, de nombreuses choses se qualifient comme des algorithmes. Les étapes de placement des lacets d'une chaussure pour parvenir à faire un nœud, par exemple, où une recette de cuisine. La notion d'algorithme ne suffit donc pas tout à fait à qualifier les pratiques que nous cherchons à définir. En effet, le travail présenté ici n'est pas une étude de la pensée algorithmique en architecture au sens large. Un algorithme est défini comme un ensemble d'instructions, sans qu'il soit précisé par quelle entité ces instructions sont exécutées. Le mot trouve en effet son origine dans la description d'opérations mathématiques entre le IXe et le XIIe siècle, des opérations exécutées par des opérateurs humains, comme c'est usuellement le cas pour des recettes de cuisine ou des noeuds de lacets. Néanmoins, la fin de la définition donnée par le Larousse est la suivante : "*un algorithme peut être traduit, grâce à un langage de programmation, en un programme exécutable par un ordinateur*"[3]. Malgré l'étymologie du terme, sa définition contemporaine est usuellement celle d'un ensemble d'instructions exécutées par une machine. C'est précisément ce que nous entendons ici par les pratiques computationnelles : il s'agit du recours à des algorithmes exécutés par un ordinateur, ou plus exactement du recours à des outils de programmation[4]. La programmation est ici entendue comme

[1] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[2] Encyclopædia Britannica, Encyclopædia Britannica, Inc., 2010.
[3] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[4] Il ne s'agit bien sûr pas ici du programme architectural au sens de la liste des espaces prévus dans un projet.

"*l'ensemble des activités liées à la définition, l'écriture, la mise au point et l'exécution de programmes informatiques ; séquence des ordres auxquels doit obéir un dispositif*"[5].

De nombreux termes sont employés dans le domaine de la programmation informatique pour désigner les outils de programmation. On y parle d'algorithmes, mais aussi de code, de script, de programmes et de logiciels[6]. Chacun de ces termes correspond à un élément différent de l'activité de programmation. Le code désigne ainsi, en informatique, un "*ensemble d'instructions en langage machine ou symbolique constituant un programme*" ou un "*ensemble de règles permettant de représenter des données d'une manière biunivoque sous une forme discrète, en vue de faciliter leur traitement automatique ou leur transmission*"[7]. En bref, le code, c'est le texte saisi pour transmettre les ensembles d'instructions qui composent un algorithme à la machine qui doit l'exécuter. Le script désigne un morceau de code transmis d'un programmeur à l'autre ou d'un programme à l'autre. Ce morceau de code n'est pas nécessairement fonctionnel. Il s'agit d'un anglicisme puisqu'en français, le script désigne une écriture simplifiée conçue pour la prise de notes rapide, ou bien un scénario de film. Les programmes, eux, sont des "*ensemble[s] d'instructions et de données représentant un algorithme et susceptible d'être exécuté par un ordinateur*"[8]. Les programmes contemporains sont en fait souvent composés de plusieurs ensembles d'algorithmes, conçus pour exécuter des sous-tâches, liés entre eux pour traiter une tâche plus large. Enfin, les logiciels sont des ensembles de "*programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de données*"[9]. Les logiciels regroupent donc des programmes permettant de traiter diverses tâches en un seul objet informatique, qui permet usuellement d'accéder à ces différents programmes via une même interface. Des différents objets informatiques que nous venons de définir, ce sont ceux dont les utilisateurs d'un ordinateur sont les plus familiers, puisque ce sont ceux qui sont systématiquement utilisés lorsque face à un ordinateur. Nous avons pris comme point de départ de la définition des pratiques computationnelles les algorithmes, car ils constituent la plus petite unité fonctionnelle de la programmation[10]. Les définitions données ensuite montrent comment chacun des types d'objets est ensuite combiné pour former un objet informatique plus large. Bien que tous ces objets interviennent tous dans les activités de programmations, ces différents niveaux impliquent différentes relations à l'outil informatique en fonction de ce qui est manipulé par l'utilisateur. Dans les pratiques de conception spatiale computationnelles, les praticiens recourent aussi bien à des algorithmes, qu'à des programmes ou des logiciels. Au fil du texte, nous allons examiner les différences qui viennent du recours à l'un plutôt qu'à l'autre. Néanmoins, le

[5] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[6] Environnements de programmation, langages et plug-ins seront abordés au chapitre V..
[7] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[8] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[9] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[10] Le script n'ayant pas vraiment d'existence propre en tant qu'outil puisque ce n'est pas toujours un programme fonctionnel.

dénominateur commun, celui qui nous permet de définir précisément ce que sont les pratiques computationnelles, c'est le recours au code au cours de la conception architecturale.

### *1.1.2 Un vocable aux multiples alternatives*

Ceci posé, nous pouvons maintenant nous pencher sur l'expression elle-même et sur les termes qui la composent, pour expliquer les raisons du choix de ces termes. Computationnel, d'abord, nécessite une explication. Pourquoi choisir ce mot en particulier, parmi les nombreux vocables existant dans la littérature pour qualifier le recours aux outils de programmation ? *Digital*, *numérique*, *informatique*, *paramétrique*, *procédural*, *génératif*… Lorsqu'on se penche sur la littérature du champ computationnel, chacun de ces mots est utilisé comme marqueur de la particularité des pratiques qui nous intéressent. Chaque auteur construit son propos autour de l'un de ces termes en particulier. Rares sont les tentatives de former un glossaire de ces termes qui soit adopté par l'ensemble de la discipline, même si on peut mentionner l'article Design Modelling Terminology de David Stasiuk[11], sur lequel nous reviendrons un peu plus loin. Nous allons voir en les examinant un par un leur définition initiale et leur usage dans le champ, en nous concentrant pour l'instant sur les usages contemporains de ces différents mots[12]. Nous allons aussi voir que la traduction en français de certains de ces termes, issus de la littérature anglophone, peut parfois poser problème. Nous allons néanmoins au fil des termes définir un glossaire des termes que nous retrouverons ensuite au fil du texte.

Commençons par le domaine scientifique à proprement parler dont relèvent les outils qui nous intéressent et leur développement : les sciences de l'informatique. Elles sont définies comme suit :

*informatique*
*nom féminin*
*(de information et automatique)*

*1. Science du traitement automatique et rationnel de l'information considérée comme le support des connaissances et des communications.*
*2. Ensemble des applications de cette science, mettant en œuvre des matériels (ordinateurs) et des logiciels.*

[11] David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.
[12] Dans les chapitres qui suivent, consacrés aux différentes phases de développement du champ, nous verrons à travers les citations et la discussion l'évolution du vocabulaire du champ computationnel au fil du temps.

Les outils auxquels nous nous intéressons sont rarement dits *informatiques* dans la littérature, bien qu'ils relèvent de ce domaine. Ils sont plutôt qualifiés de *numériques*. Concernant la différence entre les termes numérique et informatique, nous proposons au lecteur de se référer à l'article de Christophe Legrenzi *Informatique, numérique et système d'information : définitions, périmètres, enjeux économiques*[13]. La première partie du texte délimite clairement la définition de chacun des deux termes, et souligne également la confusion entre les deux, de plus en plus présente dans les discours contemporains. En effet, le *numérique* tend à se substituer au terme *informatique* de plus en plus souvent et dans des contextes de plus en plus nombreux. Le terme ne désigne pourtant pas la science du traitement de l'information et ses applications, mais le calcul numérique, c'est-à-dire ce qui a trait aux nombres, comme le suggère son étymologie le mot latin latin *numerus*. C'est le sens des deux premières entrées de la définition du mot numérique dans le dictionnaire :

*numérique*
*adjectif et nom masculin*
*(latin numerus, nombre)*

*1. Qui relève des nombres ; qui se fait avec des nombres, est représenté par un nombre.*
*Contraire : littéral*
*2. Qui est évalué ou se traduit en nombre, en quantité : Supériorité numérique.*[14]

Dans le dictionnaire, on trouve cependant également une définition du terme numérique associée au domaine de l'informatique :

*Informatique et télécommunications*

*3. Se dit de la représentation d'informations ou de grandeurs physiques au moyen de caractères, tels que des chiffres, ou au moyen de signaux à valeurs discrètes.*
*Synonyme : digital*
*4. Se dit des systèmes, dispositifs ou procédés employant ce mode de représentation discrète, par opposition à analogique.*[15]
*Synonymes : digital - discret*

Son synonyme *digital* est défini ainsi :

[13] Legrenzi Christophe, « Informatique, numérique et système d'information : définitions, périmètres, enjeux économiques », *Vie & sciences de l'entreprise*, 2015/2 (N° 200), p. 49-76. DOI : 10.3917/vse.200.0049. URL : https://www.cairn.info/revue-vie-et-sciences-de-l-entreprise-2015-2-page-49.htm
[14] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[15] Grande Encyclopédie Larousse, Editions Larousse, 2020.

*digital, digitale, digitaux*
*adjectif*
*(américain digital, de digit, nombre)*
*Vieilli. Anglicisme déconseillé. Synonyme de numérique.*
*Synonyme : numérique*

Parfois utilisé à l'oral en français pour qualifier les pratiques et les outils qui nous occupent, ce synonyme est peu repris à l'écrit. Cependant, son pendant anglophone digital est largement employé dans la littérature consacrée au champ computationnel, et va nous fournir des exemples supplémentaires de la signification donnée au terme. Dans certains cas, le mot est employé au sens initial du terme, en adjectif pour qualifier les outils non-analogiques.

*Une maquette numérique, par sa nature et son mode de construction, semble donc en rupture avec la valeur sémantique de trace qu'elle porte en elle*[16].
E. Cristia et al., 2018.

*De nos jours, la construction d'un parking à étages fait généralement appel à davantage de technologies numériques que celles dont disposait le bureau de Frank Gehry pour la conception du Guggenheim Bilbao au début des années 1990*[17].
M. Carpo, 2012.

On parle de *maquette*, de *fabrication*, de *technologies* ou d'*outils numériques*, en anglais de *digital fabrication*, *tools*, *technologies*. Il s'agit alors d'un simple constat technique sur la nature des outils employés. Les outils concernés par ce constat sont un très vaste champ. Le numérique désigne dans ce cas ce qu'il faudrait en fait appeler des outils informatiques. Ceci a pour effet de tout ramener sur le même plan, en oubliant des distinctions qui sont pourtant importantes lorsque l'on s'attache à caractériser la conception architecturale qui s'appuie sur des outils de programmation. Suivant cette première caractérisation technique, un ordinateur, un smartphone, une tablette graphique, une enceinte connectée sont des objets numériques. Paint, Word ou Photoshop sont des outils numériques au même titre que Processing. Pourtant l'un d'entre eux est un environnement de programmation, alors que les autres sont des logiciels de traitement de texte ou d'image. Certes, tous ces outils peuvent servir au

[16] Emilien Cristia, Pierre-Paul Zalio et François Guéna, "Quand le BIM met la maquette à l'épreuve du numérique", *Actes de la conférence SCAN'18 – 8e Séminaire de Conception Architecturale Numériqu*e, SHS Web Conf., Volume 47, 2018.
[17] "*Building a multistorey car park these days typically involves more digital technologies than were available to Frank Gehry's office for the design of the Guggenheim Bilbao in the early 1990s".* Mario Carpo, "Twenty Years of Digital Design", in Mario Carpo (ed.), *1993-2012, The Digital Turn in Architecture*, Wiley & Sons, 2012.

cours du processus de conception architecturale[18]. Mais les pratiques que nous souhaitons explorer sont moins vastes que cela puisqu'il s'agit seulement des pratiques de programmation. Le mot numérique rassemble tout sous un même vocable : modélisation 3D, B.I.M, réalité virtuelle, code. Les pratiques computationnelles qui nous intéressent en font certes partie, mais l'*informatique* ou le *numérique* font références à des domaines de pratique bien plus vastes. C'est aussi le cas pour le mot anglais *digital*, traduction littérale de *numérique* qui couvre donc un ensemble beaucoup plus large d'outils et de pratiques que celui qui nous occupe.

Le mot *numérique* est cependant employé dans un sens légèrement différent dans les textes qui traitent du recours aux outils informatiques en architecture. Plutôt qu'un constat technique, il s'agit alors de souligner le propre des pratiques de conception qui s'appuient sur ces outils. Le numérique devient alors une pensée spécifique de la conception architecturale. Cet usage du mot vise à saisir les spécificités intrinsèques des pratiques numériques dans la conception architecturale, voire celles du monde dans lequel nous vivons, marqué par la présence de nombre de technologies numériques au quotidien.

> *Le terme numérique peut être trompeur si il est confondu avec l'usage des ordinateurs. Le numérique est en effet omniprésent dans la technologie et la culture, et je ne suis pas sûr qu'on puisse limiter l'étude du numérique en architecture à l'informatique dans la mesure où l'architecture est liée à un grand nombre de dimensions qui relèvent du monde numérique omniprésent dans lequel nous vivons*[19].
>
> Picon 2013

> *Dès lors que l'on définit le numérique comme un état rendu possible par les fondements conceptuels des médias numériques et non nécessairement par ces médias eux-mêmes, les contours de ce moment numérique - la date et les circonstances de son commencement - deviennent moins clairs*[20].
>
> Zardini 2017

> *De nos jours, la construction d'un parking à étages fait généralement appel à davantage de technologies numériques que celles dont disposait le bureau de Frank Gehry pour la conception du Guggenheim Bilbao au début des années 1990. Pourtant, rares sont les parkings d'aujourd'hui qui sont salués comme des exemples de conception numériquement intelligente. En fait, en premier lieu,*

[18] Il est même possible de dessiner ses projets directement sur Excel, comme l'ont fait les architectes Jan De Vylder et Inge Vinck. Voir Jan De Vylder et Inge Vinck, *Gallery Magazine N°1*, Art Paper Editions, 2020.
[19] Antoine Picon, "Histoires du numérique : information, ordinateur et communication", in: Andrew Goodhouse, *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, p. 79-98, 2013.
[20] Mirko Zardini, "Huit millions d'histoires", in: Andrew Goodhouse, *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, p. 9-22, 2013.

*un bâtiment significatif de l'ère numérique n'est pas n'importe quel bâtiment qui a été conçu et construit à l'aide d'outils numériques : c'est un bâtiment qui n'aurait pas pu être conçu ou construit sans ces outils*[21].

Carpo 2012

Comme on peut le voir dans cette dernière citation, Mario Carpo marque dans ses textes la différence entre les deux emplois du terme par le recours au termes *digital* et *digitally intelligent* dès les premières lignes, indiquant ainsi clairement dans quel sens le mot *digital* est employé par la suite. Dans de nombreux textes d'autres auteurs consacrés à la question du numérique en architecture, la distinction n'est cependant pas clairement marquée, ou très vite oubliée. Ceci induit une confusion entre outil et état numérique qui s'ajoute à la confusion entre les termes informatique et numérique, et semble condamner le mot de *numérique* à rester peu clair. Par ailleurs, la définition exacte de cet "état numérique", pour reprendre le terme employé par Mirko Zardini, varie légèrement selon les auteurs, tout comme l'expression employée, même si elle contient toujours le terme *numérique*. Le monde et la pensée numériques soulignés par Antoine Picon ne sont pas exactement la pensée numérique de la conception soulignée par Carpo lorsqu'il recourt au terme *digitally intelligent*, ni celle donnée par Zardini lorsqu'il parle d'état numérique.Ceci ajoute encore un peu de flou à la signification exacte du terme numérique dans les pratiques de conception architecturale. *Numérique* nous semble donc trop peu précis pour pouvoir qualifier correctement les pratiques que nous cherchons à évaluer.

Le besoin de saisir les particularités des pratiques numériques qui transparait à travers l'usage du mot numérique par différents auteurs est cependant une recherche qui traverse toute la littérature du champ computationnel. La nature de ces pratiques y est désignée par plusieurs autres vocables dans le champ, des vocables qui cherchent justement à qualifier plus précisément le processus à l'œuvre : *paramétrique*, *génératif, procédural*. Là encore, les auteurs s'emparent de celui qui leur paraît le plus adéquat pour en faire le vocable principal autour duquel structurer leur pensée, retenant divers aspects de la définition initiale qui leur permettent de souligner ce qui leur paraît important dans ce processus de conception si particulier. Le mot de paramétrique est l'un des plus répandus dans la littérature consacrée à l'usage de l'ordinateur en architecture. Son recours traverse des champs divers : le champ computationnel, mais aussi par exemple des études sur l'adoption des processus informatiques au sein des agences d'architecture[22]. Il est alors question de *modèles paramétriques* ou plus largement de

[21] *"Building a multistorey car park these days typically involves more digital technologies than were available to Frank Gehry's office for the design of the Guggenheim Bilbao in the early 1990s. Yet few of today's car parks are hailed as examples of digitally intelligent design. In fact, in the first instance, a meaningful building of the digital age is not just any building that was designed and built using digital tools: it is one that could not have been either designed or built without them".* Mario Carpo, "Twenty Years of Digital Design", in Mario Carpo (ed.), *1993-2012, The Digital Turn in Architecture*, Wiley & Sons, 2012.

[22] Adeline Stals, Catherine Elsen et Sylvie Jancart, "L'immersion pour l'appréhension des outils de modélisation paramétrique en conception architecturale", *Actes de la conférence SCAN'18 – 8e Séminaire de Conception Architecturale Numériqu*e, SHS Web Conf., Volume 47, 2018.

*conception paramétrique*, comme nous allons le voir dans les exemples qui suivent. *Paramétrique* est défini dans le dictionnaire comme suit :

*paramétrique*
*adjectif*
*Se dit d'une équation algébrique dans laquelle l'un au moins des coefficients dépend d'un paramètre.*[23]

La définition de *paramètre* est la suivante :

*paramètre*
*nom masculin*
*(de grec metron, mesure)*
*1. Grandeur mesurable permettant de présenter de façon plus simple et plus abrégée les caractéristiques principales d'un ensemble statistique.*
*2. Élément en fonction duquel on explicite les caractéristiques essentielles d'un phénomène, d'une question : La pluie, l'obscurité sont des paramètres dont il faut tenir compte.*
*3. Nom donné à certains coefficients, à certaines quantités, autres que la variable ou l'inconnue, en fonction desquels on peut exprimer une proposition ou les solutions d'un problème.*
*4. Pour un arc paramétré (I, f), nombre réel variable qui décrit I ; pour une surface paramétrée (A, g), chacune des deux variables réelles de la fonction g.*
*5. En informatique, variable dont la valeur, l'adresse ou le nom ne sont précisés qu'à l'exécution du programme.*[24]

Daniel Davis est architecte et a d'abord exercé comme enseignant-chercheur à Harvard avant de travailler au sein de l'agence Hassell. Il propose dans son travail de thèse une histoire détaillée de l'usage du terme paramétrique en architecture[25]. Cette histoire, complétée dans l'article *A history of parametric*[26], reprends assez fidèlement la définition du terme paramétrique : "*a set of quantities expressed as an explicit function of a number of parameters*". Son travail inclut par ailleurs un questionnement de l'articulation entre le paramétrique comme concept et comme application informatique. Daniel Davis souligne notamment l'apparition du terme en architecture dès les années 1940 pour désigner des formes modulables selon certains paramètres[27]. Il rappelle aussi l'existence du

[23] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[24] Grande Encyclopédie Larousse, Editions Larousse, 2020.
[25] Daniel Davis, "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture." PhD dissertation, RMIT University, 2013
[26] Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.
[27]Moretti, Luigi. 1971. "Ricerca Matematica in Architettura e Urbanisticâ." Moebius IV no. 1, 30-53. Republished in: Federico Bucci and Marco Mulazzani. 2000. Luigi Moretti: Works and Writings. New York: Princeton Architectural Press et Bucci, Federico, and Marco Mulazzani. 2000. Luigi Moretti: Works and

mot en mathématiques, largement antérieur puisqu'il donne des exemples dès 1837. le sens mathématique largement antérieur du terme en 1837[28]. Dans un article publié un an après la thèse de Daniel Davis, Patrick Janssen et Rudi Stouffs proposent une classification des méthodes de modélisation paramétriques, qui peuvent selon eux être divisés en plusieurs catégories en fonction de la manière dont les itérations sont gérées[29]. Les auteurs y définissent les modèles paramétriques comme ceci :

*En général, un modèle paramétrique consiste en une collection d'opérations de modélisation qui sont liées dans un réseau qui peut être trié topologiquement, c'est-à-dire que l'ordre d'exécution des opérations de modélisation peut être défini avant l'exécution[30].*

La définition du terme paramétrique est donc étendue, en y intégrant la notion qu'il s'agit de définir des relations entre éléments, plutôt que de simplement sous-entendre l'existence de ces relations par la mention des fonctions liant les paramètres entre eux. Cette clarification est faite par plusieurs autres auteurs, avant et après Janssen et Stouffs. C'est le cas par exemple de Robert Woodbury, dont la définition de paramétrique repose sur trois points[31]. Le premier est l'idée qu'un modèle paramétrique relève de règles logiques définies par le concepteur, comme chez Janssen et Stouffs. Mais Woodbury étend ensuite encore la définition en ajoutant deux points relatifs à la temporalité du modèle paramétrique. La modification d'nu modèle paramétrique doit être possible en permanence, et ceci doit permettre le développement de différentes alternatives de conception en parallèle au cours du processus. Cette flexibilité du modèle est également pointée par Davis lui-même dans sa thèse[32]. Autre exemple, la définition donnée par Adeline Stals, Catherine Elsen et Sylvie Jancart dans un article de 2018 :

*Nous pouvons définir la conception paramétrique comme une méthodologie de conception qui permet, entre autres, de générer des formes à partir de l'exploitation et la manipulation d'une grande quantité*

---

Writings. New York: Princeton Architectural Press, cités par Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.

[28] Dana, James. 1837. "On the Drawing of Figures of Crystals." The American Journal of Science and Arts 32: 30-50, cité par Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.

[29] Janssen, P., & Stouffs, R. (2015). Types of parametric modelling. International Conference of Association for Computer-Aided Architectural Design Research CAADRIA, 157–166

[30] "*In general, a parametric model consists of a collection of modelling operations that are linked into a network that can be topologically sorted, that is, the order of execution of the modeling operations can be defined prior to execution".* Janssen, P., & Stouffs, R. (2015). Types of parametric modelling. International Conference of Association for Computer-Aided Architectural Design Research CAADRIA, 157–166

[31] Robert Woodbury, *Elements of Parametric Design,* Routledge, New York, 2010. Robert Woodbury est un professeur d'architecture canadien spécialisé dans les pratiques computationnelles.

[32] Daniel Davis, "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture." PhD dissertation, RMIT University, 2013

*de données de type environnemental, acoustique, structurel, social ou encore urbain repris comme "paramètres"*[33].

L'article reprend la définition donnée par Janssen et Stouffs des modèles paramétriques. Mais les trois chercheuses insistent également dans leur texte sur la diversité des paramètres intégrés aux modèles. Autre point important, leur définition intègre par ailleurs l'idée que les paramètres ne sont pas seulement ajustés en fonction de décisions propres par l'utilisateur, mais en fonction de l'analyse du contexte, ce qui implique la gestion d'une grande quantité de données à travers le recours au modèle paramétrique. Leur texte fait également la transition de la notion de *modèle paramétrique* vers celle de *méthodologie paramétrique*. Avec ce saut, nous retrouvons encore la superposition entre caractérisation technique (ou ici mathématique) et caractérisation épistémologique des pratiques analysées. David Stasiuk, développeur de plusieurs outils computationnels pour la conception architecturale, propose quant à lui une hiérarchisation des familles de modélisation en usage en architecture dans son article *Design Modelling Terminology*[34]. Il y critique la définition de Daniel Davis, qui se limite selon lui aux caractéristiques géométriques d'un objet architectural[35], et choisit de reformuler la définition pour englober toutes les informations numériques relatives à l'architecture. Le paramétrique devient donc "*un ensemble d'équations qui expriment des informations concernant le déploiement d'un système d'information architectural, en tant que fonctions explicites d'un certain nombre de paramètres*"[36]. On retrouve dans ses remarques l'extension de ce que couvrent les paramètres lue chez Stals, Elsen et Jancart, formulée plus ouvertement. Ce travail d'extension du sens du mot paramétrique en architecture culmine chez Patrik Schumacher, qui fait de cette notion la base d'un paradigme de conception généralisé, le *paramétricisme*, sur lequel s'appuie une esthétique architecturale particulière[37].

*L'architecture d'avant-garde récente présente une convergence globale qui justifie sa désignation comme un nouveau style : le paramétrage. Il s'agit d'un style enraciné dans les techniques*

---

[33] Adeline Stals, Catherine Elsen et Sylvie Jancart, "L'immersion pour l'appréhension des outils de modélisation paramétrique en conception architecturale", *Actes de la conférence SCAN'18 – 8e Séminaire de Conception Architecturale Numériqu*e, SHS Web Conf., Volume 47, 2018.
[34] David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.
[35] Ce qui n'est pas explicitement dit dans le texte de Daniel Davis.
[36] "*a set of equations that express information regarding the deployment of an architectural information system, as explicit functions of a number of parameter"*. David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.
[37] Patrik Schumacher, "Parametricism - A New Global Style for Architecture and Urban Design", in: *AD Architectural Design - Digital Cities*, Vol 79, No 4, July/August 2009.

*d'animation numérique, dont les derniers raffinements reposent sur des systèmes de conception paramétrique avancés et des méthodes de script*[38].

Pour un commentaire détaillé du paradigme paramétrique chez Patrik Schumacher, nous renvoyons à l'article de Neil Leach *Parametrics Explained*, ou il souligne les limites de cette idée d'un style numérique global en architecture ainsi que les confusions demeurant entre les notions de *paramétrique* et *algorithmique* dans l'analyse proposée par Patrik Schumacher[39]. Les définitions du terme données par ces différents auteurs intègrent donc toutes des extensions à la définition mathématique du mot, ajoutées pour qualifier plus précisément les modèles architecturaux dit paramétriques. Malgré ces variations, les auteurs partagent un point commun dans leur définition du paramétrique en architecture : le recours à un modèle paramétrique n'implique pas forcément de manipuler du code[40]. Modifier des variables via une interface logicielle classique relève aussi de la modélisation paramétrique, sans que l'utilisateur soit confronté au code de l'algorithme directement. C'est le point que souligne Daniel Davis lorsqu'il indique que s'est produit au cours de ces dernières années un changement dans le recours aux modèles paramétriques : il ne s'agit pour les architectes plus tant d'établir ces modèles que simplement de les utiliser[41]. Prendre Revit comme exemple des pratiques paramétriques montre aussi la possible absence du code. La distinction faite par Stals, Elsen et Jancart ou par Leach entre modélisation paramétrique et algorithmique relève également de cette différence cruciale. Or c'est à la modélisation algorithmique que nous nous intéressons ici, et non à la modélisation paramétrique.

Bien que *paramétrique* soit le plus répandu, dans et en dehors du champ computationnel, plusieurs autres termes sont également en usage. L'ensemble d'entre eux touchent à la caractérisation des pratiques informatiques en conception architecturale, et plusieurs qualifient directement la modélisation algorithmique. Davis Stasiuk articule dans sa proposition de terminologie les plus usités d'entre eux. Il propose la modélisation *paramétrique* comme faisant partie de la catégorie plus large de la modélisation *procédurale*. La modélisation paramétrique englobe selon lui par ailleurs la

---

[38] "*There is a global convergence in recent avant-garde architecture that justifies its designa-tion as a new style: parametricism. It is a style rooted in digital animation techniques, its latest refinements based on advanced parametric design systems and scripting methods".* Patrik Schumacher, "Parametricism - A New Global Style for Architecture and Urban Design", in: *AD Architectural Design - Digital Cities*, Vol 79, No 4, July/August 2009.

[39] Neil Leach, "Parametrics explained", *Next Generation Building*, Vol. 1, p. 1-10, 2014.

[40] Voir par exemple chez Adeline Stals, Catherine Elsen et Sylvie Jancart, "L'immersion pour l'appréhension des outils de modélisation paramétrique en conception architecturale", *Actes de la conférence SCAN'18 – 8e Séminaire de Conception Architecturale Numériqu*e, SHS Web Conf., Volume 47, 2018, chez David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021 ou chez Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.

[41] Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.

modélisation *computationnelle* et la modélisation *générative*, chacune concernant des types de modèles dont la définition comporte plus de contraintes[42]. En hiérarchisant ces différentes notions, Davis Stasiuk résume dans son texte les vocables les plus utilisés du champ. La notion de procédural est définie dans son texte comme suit : "*un processus de modélisation procédurale est un processus qui utilise un ensemble d'instructions explicites pour produire un résultat modèle*"[43]. Dans le dictionnaire français, le mot procédural est simplement défini comme étant "*relatif à la procédure*"[44], un terme lui-même sans définition précise en lien avec l'informatique, simplement listé comme une "*succession d'opérations à exécuter pour accomplir une tâche déterminé*"[45], ce qui se rapproche en effet de la définition donnée par Davis Stasiuk. Le terme *procédural*, utilisé un peu plus rarement dans le champ computationnel que les autres vocables abordés ici, couvre donc un ensemble très vaste d'éléments en comparaison de ces autres termes. Il est moins sujet à des confusions sur sa signification exacte, tant parce qu'il est moins précis dans ce qu'il qualifie que parce qu'il n'est pas utilisé pour qualifier le champ ou les pratiques, mais le processus. Bien qu'il ne soit pas l'objet de confusions entre caractéristiques techniques et manière de penser, nous lui préférons les mots *algorithmique* ou *computationnel* pour qualifier le processus observé, dont les définitions sont plus précises[46].

Concernant le terme génératif, David Stasiuk le définit en indiquant que non seulement un modèle génératif est constitué d'éléments liés entre eux et doit fournir des informations nouvelles à l'issue de son exécution, mais les relations entre les éléments du modèle ne sont par ailleurs pas fixées.

> *[L'idée] de formation progressive et de réutilisation de sous-systèmes intégrés est à la base de la définition présentée ici : un modèle "génératif" sera compris comme un modèle qui non seulement répond à tous les critères d'un modèle de calcul, mais qui, au cours d'une seule instance d'exécution, incorpore en outre une capacité à former des relations topologiques non fixées entre les éléments du modèle, et/ou est actualisé par l'accrétion progressive de nouveaux éléments de modèle qui sont morphogéniquement récursifs, avec des étapes incrémentielles de formation dépendant des étapes précédentes mises en œuvre au cours de l'exécution du modèle*[47].

---

[42] David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.

[43] "*a procedural modeling process is one that uses an explicit instruction set to produce a model outcome".* David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.

[44] Grande Encyclopédie Larousse, Editions Larousse, 2020.

[45] Grande Encyclopédie Larousse, Editions Larousse, 2020.

[46] Pour *algorithme* voir plus haut. Pour *computationnel*, nous y reviendrons dans quelques lignes.

[47] "*[The idea] of progressive formation and the re-use of embedded sub-systems drive the definition presented here : that a "generative" model will be understood as one that not only meets all of the criteria for being a computational model, but which during a single execution instance additionallys incorporates a capacity for unfixed topological relationships between model elements, and/or is actualized through the step-wise accretion*

La notion de *génératif* dans le champ computationnel prend racine dans le domaine dit de l'art génératif, un courant artistique qui se base sur le recours à des algorithmes pour la création des œuvres. Plutôt que de créer l'œuvre directement, les artistes écrivent des programmes, dont les sorties - dessins, vidéos, sons - constituent l'œuvre[48]. Dans le cas de ce courant artistique, le mot génératif est utilisé dans un sens identique à celui donné dans le dictionnaire. En français, le Larousse nous signale que *génératif* est un anglicisme, dérivé du mot *generative*. Les définitions des différents mots correspondants sont les suivantes :

*génératif, générative*
*adjectif*
*(anglais generative)*[49]

*generative*
*: having the power or function of generating, originating, producing, or reproducing*[50]

*generate*
*1: to produce (something) or cause (something) to be produced*[51]

*générer*
*verbe transitif*
*Conjugaison*
*(anglais to generate, du latin generare, engendrer)*
*1. Produire quelque chose, l'avoir pour conséquence inéluctable, l'engendrer, en être la source : L'augmentation des produits pétroliers génère l'inflation.*
*Synonyme : engendrer*
*2. Créer un programme ou un système d'exploitation particulier, à partir de modules de programmes généraux et d'une spécification du programme recherché.*
*3. Engendrer une phrase.*[52]

---

*of new model elements that are morphogenically recursive, with incremental stages of formation dependent upon preceding steps enacted during the model execution".* David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.

[48] Benedikt Gross, Claudius Lazzeroni, Hartmut Bohnacker et Julia Laub, *Generative Design : Visualize, Program, and Create with JavaScript in p5.js*, Princeton Architectural Press, 2018. Au chapitre V, nous reviendrons également sur l'histoire de Processing, logiciel très prisé des artistes génératifs, ce qui fournira quelques précisions sur la relation entre art génératif et champ computationnel.

[49] Grande Encyclopédie Larousse, Editions Larousse, 2020.

[50] *Merriam-Webster Dictionary*, Merriam-Webster, Inc., 2020.

[51] *Merriam-Webster Dictionary*, Merriam-Webster, Inc., 2020.

[52] Grande Encyclopédie Larousse, Editions Larousse, 2020.

La définition initiale de génératif implique donc simplement de voir émerger de nouvelles choses. Or ceci implique un problème dans le champ computationnel, que souligne l'usage du mot génératif. L'affection pour les systèmes algorithmiques qui prends les praticiens par surprise en faisant émerger des formes inattendues tends en effet à brouiller dans leurs discours les frontières entre déterminisme et indéterminisme, entre instructions clairement fournies à la machine et signes d'intelligence artificielle[53]. Le mot génératif est alors érigé en marqueur de cette apparition soudaine de géométries complexes sur l'écran, et marque peu de considérations pour les réalités techniques de l'algorithme à l'œuvre. Si Stasiuk souligne le problème en abordant le terme génératif dans son texte, intégrer à sa définition du mot la notion de morphogenèse le condamne malheureusement à retomber dans les mêmes travers. Il est par ailleurs étonnant de noter qu'après avoir reproché à Daniel Davis sa concentration sur les questions géométriques, sa définition de génératif s'ancre également dans la géométrie. En plus d'être un anglicisme et d'être un terme relativement peu précis quand il s'agit de qualifier les pratiques de programmation en architecture, le mot génératif souffre du même problème que le terme paramétrique. La variété des usages et le flou des diverses définitions employées empêche une démarcation claire entre la qualification d'un mode de pensée et la caractérisation technique.

L'ensemble des auteurs cités jusqu'ici semble tomber d'accord sur le fait que les usages de ces différents termes sont problématiques et créent des confusions[54], sans que leur proposition personnelle n'ait été retenue par la suite dans la communauté. Leur remarque reste donc fondée. Il existe quelques autres vocables utilisés pour qualifier le recours aux outils numériques. Nous nous sommes penchés ici essentiellement sur ceux utilisés de nos jours. La notion de travail de conception assistée par ordinateur, qui a donné l'acronyme CAO et en anglais l'adjectif computer-aided, a été particulièrement utilisée pour parler du recours aux outils de programmation dans les années 1960 et 1970. Si l'acronyme est toujours en usage, il désigne de manière très large les outils numériques de dessin et de conception assistée par ordinateur, et l'emploi de l'adjectif computer-aided pour désigner les pratiques de programmation est tombé en désuétude[55]. Il existe par ailleurs encore un terme que

[53] Nous reviendrons à cet intérêt pour le hasard et l'inattendu plus loin dans le texte.
[54] Janssen, P., & Stouffs, R. (2015). Types of parametric modelling. International Conference of Association for Computer-Aided Architectural Design Research CAADRIA, 157–166. Neil Leach, "Parametrics explained", *Next Generation Building*, Vol. 1, p. 1-10, 2014. David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.
[55] Nous reviendrons sur les débats autour du terme pendant les années 1970 au chapitre II.

nous n'avons pas examiné, celui de *computationnel*. Il s'agit d'un anglicisme[56] dérivé du mot *computation*, qui vient lui-même du latin *computatio* : calcul.

*computation*
*1a: the act or action of computing : CALCULATION*
*b: the use or operation of a computer*[57].

En anglais, l'usage du mot dans le champ qui nous occupe est marqué par une précision faite à de nombreuses reprises. Il s'agit de la différentiation entre *computerisation* et *computation*. Faite par Nicholas Negroponte dès les années 1970, elle est reprise ensuite par Kostas Terzidis au milieu des années 2000, puis par David Stasiuk, Sean Ahlquist, Achim Menges[58]. La différence est définie comme suit :

*Alors que le calcul est la procédure de calcul, c'est-à-dire la détermination de quelque chose par des méthodes mathématiques ou logiques, l'informatisation est l'action de saisir, traiter ou stocker des informations dans un ordinateur ou un système informatique*[59].

Selon ces auteurs, les processus *computationnels* doivent donc être considérés comme des processus qui produisent des informations supplémentaires, ce que les processus de *computerisation* ne font pas. Or David Stasiuk remonte dans son texte *Design Modeling Terminology* à la définition originelle du mot[60] et souligne qu'elle n'implique pas nécessairement d'ordinateur, puisqu'elle désigne juste le fait de calculer. A ce titre la computation est donc simplement un set d'opérations permettant la réflexion. Il note donc que le mot ne devrait pas être utilisé pour désigner un processus dont on veut dire qu'il inclut systématiquement le recours à un ordinateur. Au sens originel donné par Stasiuk, le mot computation correspondait au mot français calcul, et l'opposition entre computerisation et computation, à l'opposition entre numérisation et calcul. Recourir au terme *computationnel* en francais permet néanmoins de contourner le problème - qui se pose comme on l'a vu pour la plupart

[56] Il existe tout de même une définition française, peu usitée et avec une signification légèrement différente de ce que nous allons traiter : *computation - Manière de calculer le temps.* Grande Encyclopédie Larousse, Editions Larousse, 2020.
[57] *Merriam-Webster Dictionary*, Merriam-Webster, Inc., 2020.
[58] Negroponte, Nicholas, "The Architecture Machine: Toward a More Human Environment", *Computer-Aided Design,* 1973. Ahlquist, Sean, and Achim Menges, *Computational Design Thinking*, John Wiley & Sons, Ltd., 2011. David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.
[59] "*While computation is the procedure of calculating, ie, determining something by mathematical or logical methods, computerisation is the act of entering, processing or storing information in a computer or a computer system*". Kostas Terzidis, *Expressive Form : A Conceptual Approach to Computational Design*, Routledge, 2003.
[60] Il s'appuie sur le travail de Heinz Forster - Foerster, Heinz von. 2003. "On Constructing a Reality." In Understanding Understanding, 78:211–27. Springer. doi:10.1006/geno.2001.6652.

des termes en usage. Il nous permet dans le texte de souligner les particularités du processus examiné. Nous utiliserons aussi parfois le mot algorithmique pour désigner outils et processus, mais le champ, lui, sera qualifié de champ computationnel. Cela nous paraît d'autant plus pertinent que les modalités d'usage du terme computationnel nous paraissent un ancrage plus intéressant. En effet, parmi les praticiens qui recourent aux outils de programmation, une partie d'entre eux a déjà adopté le terme d'architecture computationnelle pour définir leur pratique. Or il s'agit de praticiens faisant partie du noyau du réseau. Nous avons donc choisi de reprendre le terme et de l'étendre ici à l'ensemble des praticiens manipulant des algorithmes dans l'objectif de produire des propositions spatiales et architecturales.

### *1.1.3 Courants computationnels et milieux de pratiques*

Intéressons-nous maintenant à la notion de champ intégrée à l'expression qui nous sert de point de départ. Les quelques pages précédentes le montrent, ce que nous désignons sous le nom de pratiques computationnelles en architecture implique en fait plusieurs disciplines : architecture, mais aussi informatique et ingénierie. Les productions sont aussi très diverses : projets non réalisés, outils, prototypes, bâtiments. La construction du bâtiment, qui peut être considérée comme un marqueur essentiel de l'activité de l'architecte, n'est en effet pas prise comme condition ici. Ce qu'on appelle parfois des projets de papier, jamais construits et n'existant donc que sous la forme de dessins et de maquettes, sont pris en compte comme une production architecturale à part entière. Ils montrent en effet au moins autant que leurs cousins construits la place qu'occupe le recours aux algorithmes dans la conception architecturale. C'est à celle-ci qu'on s'intéresse, au sens de la conception de l'espace, quel que soit le support ou l'objet résultant. Le recours aux outils de programmation pour la conception spatiale a lieu dans différents milieux : industrie, enseignement, recherche. Choisir de traiter du recours aux outils de programmation à travers tous ces milieux, tous ces champs et toutes ces productions peut sembler particulièrement large. Cependant, la contrainte de l'écriture du code est très forte : peu s'impliquent directement dans la manipulation de cet outil-là lorsqu'il s'agit de traiter de la conception spatiale.

Les architectes ne sont pas les seuls à explorer la place du code dans la conception spatiale. Nous retrouverons au cours du texte bien sûr des architectes, dont la conception de l'espace est le cœur du métier, mais aussi des ingénieurs en génie civil, des développeurs de logiciels et des chercheurs en informatique. Bien plus nombreux encore, les praticiens avec deux formations différentes, voire plus. Les architectes sont au premier rang des pionniers de la conception architecturale à l'aide d'algorithmes, mais nombre d'autres figures existent. Ils se croisent d'abord pour beaucoup dans le milieu universitaire. En effet, il faut souligner que l'enseignement et la recherche proposent un espace préservé des contraintes industrielles, ou les pratiques expérimentales peuvent prospérer. Les

pratiques computationnelles des débuts du champ, qui relèvent précisément de l'expérimentation, y trouvent donc un espace privilégié pour se développer. Nombre des pratiques que nous allons étudier prennent donc racine dans le milieu de l'enseignement et de la recherche. Ceci peut donner l'impression au premier abord que le champ computationnel en architecture est en fait un microcosme, un refuge universitaire ou seule une poignée de praticiens codant l'espace, dans une bulle protégée. On peut en effet qualifier un partie du champ computationnel de microcosme. D'abord parce que, comme nous l'avons indiqué, les praticiens qui manipulent directement le code plutôt que des logiciels de modélisation ou de dessin ne sont proportionnellement pas si nombreux. Ensuite parce que certains praticiens suivent effectivement une logique de retraite volontaire au sein d'un milieu universitaire qui leur permet d'explorer avec plus de liberté les applications de la programmation informatique à la conception spatiale. Ce microcosme est le noyau, le point de départ du champ. Bien qu'il ne concerne qu'une minorité d'architectes, la relation qu'ils développent aux outils de programmation est particulière, et exacerbe les marqueurs qui permettent d'examiner les particularités de leur pratique. Malgré le fait que ces pratiques ne soient pas toujours celles qui sont identifiées comme les activités clés des architectes - parce que les projets ne sont pas toujours construits, parce que le milieu n'est pas celui de l'agence - elles ont une grande valeur pour l'étude des spécificités du recours aux outils de programmation pour la conception spatiale.

Ce microcosme constitue de plus seulement le noyau du champ, et ne rend donc pas compte de la totalité des pratiques qui y ont lieu. Le choix de certains praticiens d'une carrière dans le milieu universitaire, hors des contraintes industrielles, correspond en fait à des bifurcations à plusieurs moments du développement du champ computationnel. Ces bifurcations entre milieu universitaire et industriel constituent des moments d'extension du microcosme initial. Plus exactement, ces bifurcations correspondent à des moments de sortie des outils de programmation du milieu universitaire vers d'autres milieux. Au fil de son développement, le champ s'agrandit donc au-delà du noyau initial. Les conditions de sortie de ce microcosme sont un des éléments principaux traités dans cette recherche. Le texte des chapitres suivants fait le récit du mouvement de sortie des algorithmes du noyau du champ vers d'autres milieux. La question de la sortie du microcosme, de l'influence des algorithmes au-delà des praticiens qui écrivent et manipulent le code sera donc abordée, en suivant les algorithmes depuis le noyau de ce champ vers des pratiques plus grand public. L'articulation entre les différents milieux de pratiques, industrie, agence, recherche, enseignement, conditionne la sortie des pratiques computationnelles de ce noyau. Le champ computationnel peut donc sembler au premier abord un espace restreint, mais sa croissance au fil des décennies le mène vers d'autres espaces, dessinant des contours moins nets que l'appellation le laisse supposer. Ces contours changeants impliquent, malgré le point commun qu'est le recours aux outils de programmation pour la conception spatiale, une grande diversité des pratiques dans le champ computationnel. Dans le noyau universitaire lui-même, où les sujets de recherches sont multiples, de la robotique à la constitution de

règles informatiques de composition de l'espace En dehors du noyau du champ également, au fil des contraintes qui se diversifient et des ambitions changeantes des praticiens Nous avons pointé en introduction la dimension pléthorique des pratiques dans ce que nous désignons par l'expression champ computationnel. Nous utiliserons en complément l'expression *courants computationnels*, pour pouvoir distinguer les différentes pratiques qui se rejoignent dans le recours aux outils de programmation.

## 1.2 Historiographie des pratiques computationnelles en architecture

### *1.2.1 Des contributions variées*

Le champ computationnel en architecture est un phénomène auquel quelques historiens et théoriciens se sont déjà intéressés, esquissant un récit de l'apparition de ces nouveaux outils dans le paysage architectural du XXe et XXIe siècle et de leurs conséquences. Mario Carpo propose ainsi une étude des fondements théoriques du tournant digital à partir des années 1990, période de généralisation des outils informatiques dans la pratique architecturale qui suit l'apparition du PC et le développement de l'infographie[61]. Sébastien Bourbonnais propose pour la même période une étude des conséquences des outils numériques sur l'imaginaire architectural en s'appuyant sur la pensée de Gilbert Simondon[62]. Antoine Picon est l'auteur d'un petit précis d'architecture numérique généraliste présentant inventions informatiques majeures et illustrant leur usage en architecture par les projets numériques les plus célébrés[63]. L'apparition des algorithmes en architecture précède cependant ce tournant numérique des années 1990. Les travaux de Theodora Vardouli, Molly Wright-Steenson ou Matthew Allen ont retracé le parcours des quelques architectes américains qui, dans les années 1960, se sont parmi les premiers intéressés aux possibilités de la programmation pour la conception architecturale, d'Allen Bernholtz à Nicholas Negroponte[64]. Les acteurs de ce tournant ont eux-mêmes, pour certains, fait le récit de leurs aventures : Pierluigi Serraino, Rick Smith, ou John Walker racontent dans leurs ouvrages leur travail de développement d'outils ayant marqué le champ : Form*Z, AutoCAD, Digital Project[65]. Quelques praticiens emblématiques font leur propre récit de ce tournant à travers des sélections de projets, en particulier Greg Lynn et son Archéologie du numérique, et des ouvrages collectifs

---

[61] Mario Carpo, *The Alphabet and the Algorithm*, The MIT Press, 2011. Mario Carpo, *The Second Digital Turn: Design Beyond Intelligence*, The MIT Press, 2017.

[62] Sébastien Bourbonnais, Sensibilités technologiques : expérimentations et explorations en architecture numérique 1987-2010, Thèse de Doctorat, Université Paris-Est, 2014.

[63] Antoine Picon, C*ulture numérique et architecture : une introduction*, Birkhauser, 2010.

[64] Molly Wright-Steenson, *Architectural Intelligence: How Designers and Architects Created the Digital Landscape* MIT Press, 2017. Matthew Allen, P*rehistory of the Digital: Architecture Becomes Programming, 1935-1990*. Thèse de doctorat, Harvard University, Graduate School of Arts & Sciences, 2019.

[65] Walker, J., *The Autodesk File: Bits of History, Words of Experience*, John Walker, 2017. Smith, R., *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace, 2017. Serraino, P., *History of Form*Z*, Birkhauser, 2002.

racontent à travers des compilations d'essais associant ces mêmes praticiens les plus célébrés à ces quelques chercheurs un peu de ce bouleversement technique[66].

Nous allons revenir plus précisément sur l'apport de ces différents ouvrages dans un instant. Avant, nous souhaitons souligner la variété des contributions existantes, qui se placent dans des registres variés. La littérature qui porte sur le champ computationnel offre à la fois des analyses théoriques diverses, qui proposent d'évaluer ce que le numérique fait à l'architecture, et des analyses historiques, qui proposent de documenter l'arrivée des outils numériques en architecture. Cette diversité dépend notamment des auteurs et de leur appartenance au champ ou non. Sont donc disponibles des analyses théoriques écrites par des personnes du champ[67] - c'est-à-dire que ces auteurs manipulent des outils numériques et participent à la production du champ. Mais aussi des analyses théoriques écrites par des personnes hors du champ[68] - c'est-à-dire que ces auteurs sont historiens ou critiques d'architecture, et ne participent donc pas à la production des objets architecturaux qui relèvent du champ computationnel. Même principe pour l'analyse historique, offerte par des personnes hors du champ[69] ou internes au champ[70]. Ces derniers offrent cependant des récits particulièrement partisans. John Walker et Rick Smith, par exemple, ont chacun livré leur version de l'histoire du développement des outils des entreprises dont ils ont fait partie : Autodesk pour le premier, Gehry Technologies pour le second[71]. Mais malgré la plongée au sein d'une entreprise et d'une période qu'ils offrent, difficile de démêler chez ces auteurs ce qui relève du conflit avec leur ancienne entreprise, notamment au vu des contradictions entre les versions des différents acteurs du conflit. D'autres récits provenant de l'intérieur du champ sont plus apaisés, comme le récit de la conception de Form*Z offert par Pierluigi Serraino ou celui de l'arrivée des outils numériques au sein de l'agence de Frank Gehry par Bruce Lindsey[72]. Plus apaisés, mais peut-être trop : difficile de percer à travers l'admiration éprouvée par les auteurs pour les concepteurs dont ils racontent le travail.

Les recueils offrent une vision moins monolithique, qu'ils proviennent de personnes internes ou

---

[66] Andrew Goodhouse (ed.), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, 2013. Lynn, G. (ed.), *Archaeology of the Digital*, Sternberg Press, 2013. Fankhänel, T. & Lepik A. (eds.), *The Architecture Machine. The Role of Computers in Architecture*, Birkhäuser, 2020.

[67] Voir par exemple les textes de Rivka Oxman, de Kostas Terzidis, de David Stasiuk, d'Achim Menges, de Jane Burry et de nombre des autres praticiens dont nous allons examiner le travail au fil des pages qui vont suivre.

[68] Par exemple les travaux de Sebastien Bourbonnais, Frédéric Migayrou, Marie-Ange Brayer, Christina Cordell ou Antoine Picon.

[69] Voir les travaux de Mario Carpo, Matthew Allen, Alice Upitis.

[70] Walker, J., *The Autodesk File: Bits of History, Words of Experience*, John Walker, 2017. Smith, R., *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace, 2017. Serraino, P., *History of Form*Z*, Birkhauser, 2002. Bruce Lindsey, *Digital Gehry: Material Resistance/Digital Construction*, Birkhauser, 2001.

[71] Walker, J., *The Autodesk File: Bits of History, Words of Experience*, John Walker, 2017. Smith, R., *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace, 2017.

[72] Serraino, P., *History of Form*Z*, Birkhauser, 2002. Bruce Lindsey, *Digital Gehry: Material Resistance/Digital Construction*, Birkhauser, 2001.

externes au champ. Ils prennent des formes diverses : recueils, catalogues d'expositions, numéros spéciaux de journaux ou de magazines. Chacun propose une liste de projets et d'essais en lien avec le numérique, articulés autour d'une question de recherche formulée différemment mais qui se résume toujours à la volonté de percer le secret de l'apport de ces nouvelles technologies à la conception architecturale. La liste est parfois partisane, et critiquée à ce titre : l'*Archéologie du numérique* de Greg Lynn, par exemple, est particulièrement décriée par les connaisseurs du champ. Sans doute parce qu'elle donne la parole à un très petit nombre de praticiens puisqu'elle se concentre sur quatre d'entre eux, sans véritable justification méthodologique de ce choix[73]. Mais surtout, en fonction des récits et des recueils, les textes, projets et praticiens identifiés comme clés n'est jamais la même. Ces listes sans fin offrent donc une vision brouillée, peu délimitée du numérique en architecture. L'ouvrage *Quand le numérique marque-t-il l'architecture?* édité par le Centre Canadien d'Architecture est symptomatique du problème. Les ambitions de l'ouvrage sont décrites comme suit :

> *Tout au long du programme de recherche Archéologie du numérique, le CCA a recueilli les documents d'archives de vingt-cinq projets réalisés entre la fin des années 1980 et le début des années 2000 afin de comprendre cette période en tant qu'origine du numérique. Cependant, dès lors que l'on définit le numérique comme un état rendu possible par les fondements conceptuels des médias numériques et non nécessairement par ces médias eux-mêmes, les contours de ce moment numérique – la date et les circonstances de son commencement – deviennent moins clairs.*
>
> *Il existe huit millions d'histoires des origines du numérique en architecture. Ce livre rassemble quatorze d'entre elles pour former une chronologie des réponses à la question suivante : « Quand le numérique marque-t-il l'architecture? »*[74]

Certes, donner à voir la multiplicité des compréhensions du numérique en architecture est un des objectifs de l'ouvrage. Le livre fournit effectivement des points de vue intéressants mais leur nombre et leur diversité dessine un brouillard de possibilités plutôt que d'aider à apporter une réponse claire à la question du rôle joué par le numérique en architecture. Au problème des définitions variées du numérique vu plus haut s'ajoute donc le problème des dizaines de points de vues différents qui se superposent dans la variété des sources disponibles.

### *1.2.2 Une analyse théorique riche, des informations pratiques minces*

[73] Lynn, G. (ed.), *Archaeology of the Digital*, Sternberg Press, 2013.
[74] Mirko Zardini, "Huit millions d'histoires", in: Andrew Goodhouse (ed.), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, p. 9-22, 2013.

Ces superpositions de sélections font de fait des analyses historiques des propositions un peu bancales ou parcellaires. Le récit fait des travaux numériques en architecture des années 1960s en témoigne : certains des praticiens, par exemple Allen Bernholtz, ne sont mentionnés que dans de très rares textes. L'aspect parcellaire des récits historiques est en effet renforcé par le fait que ce travail d'histoire est fait à partir de praticiens identifiés comme clés sans analyse du champ globale et sélectionnés pour articuler le récit autour d'eux. Ainsi Bernholtz est souvent perçu comme ayant simplement marcher dans les traces de Christopher Alexander, et son apport est minimisé, voir laissé de côté, quand Alexander est omniprésent et occupe une très grande place[75]. L'exemple du Generator le montre aussi : les sources secondaires disponibles décrivent toutes ce projet de Cedric Price, Gordon Pask, John Frazer et Julia Frazer de manière différente[76]. Le fonctionnement algorithmique du projet, son avancement, les acteurs qui y ont pris part, les raisons de son arrêt de développement ne sont jamais expliqués de manière identique. Le flou sur le fonctionnement algorithmique du projet, un problème qu'on retrouve dans la description de nombreux projets du champ computationnel, s'explique notamment par les sources utilisées dans ces travaux. En effet, il s'agit plus souvent de dessins et d'entretiens que de fichiers numériques. Seuls quelques travaux du programme du Centre Canadien d'Architecture, dans le cadre du programme de recherche Archéologie du numérique notamment, se penche sur des fichiers numériques. Or ces travaux sont pour beaucoup axés sur des questions de conservation des archives numériques des projets d'architecture.  En effet, le numérique est plus fragile que l'analogue, au fond. Formats obsolètes, supports physiques volatiles, et autres difficultés techniques rendent la consultation des algorithmes pas toujours aisée. Les descriptions faites des projets utilisés comme cas d'études ou présentés comme clés dans l'histoire du champ relèvent donc plus du commentaire architectural que du commentaire sur les algorithmes, peu consultés.

*Au cours de sept étapes de croissance, deux surfaces corrélées sont générées par calcul dans le cadre d'un processus de conception morphogénétique et évolutif basé sur les contraintes de construction du système. Le processus de croissance computationnelle utilise des règles de réécriture mathématique basées sur les systèmes de Lindenmayer étendus de Hemberg*[77].

Sean Ahlquist & Achim Menges 2011

*Generator est un projet architectural réceptif et reconfigurable pour la White Oak Plantation en Géorgie, à la frontière entre la Géorgie et la Floride, commandité par Howard Gilman. Il s'agit d'une jeu de pièces, composé de 150 cubes de 12 pieds réagencables et de panneaux disponibles dans le*

[75] Plus d'informations à ce sujet sont données au chapitre II.
[76] Plus d'informations à ce sujet sont données au chapitre II.
[77] "*Over seven growth steps, two correlated surfaces are computationally generated as part of a morphogenetic and evolutionary design process based on the constructional constraints of the system. The computational growth process employs mathematical rewriting rules based on Hemberg-Extended Lindenmayer Systems".* Ahlquist, Sean, and Achim Menges, *Computational Design Thinking*, John Wiley & Sons, Ltd., 2011. David Stasiuk, "Design Modeling Terminology", 13 Juin 2018

*commerce. Il y a des sources et des passerelles, et l'idée est que, à partir de cette grille, on puisse se promener et avoir toutes les expériences récréatives ou artistiques imaginables. Au début, un certain nombre de plans étaient fournis, mais ensuite, les usagers pouvaient demander ce qu'ils voulaient de Generator*[78].

Molly Wright-Steenson, 2017

*Les clusters forment un archipel flottant (...). Le schéma de distribution des grappes de cellules suspendues est déterminé par un algorithme qui organise 11 îlots répartis en 4 zones selon un système de circulation pourvu en outre de capteurs de sons*[79].

Catalogue Architectures Non-Standard, 2003

*La seconde particularité de l'HygroSkin est liée à l'emploi des logiciels. (...) L'intérêt réside (...) dans le recours à la logique des automates, selon des modèles d'interaction de plus en plus abstraits offerts par les agencements de l'ordinateur. L'effet de surface n'a plus besoin d'être perceptible en temps réel, dans la matière, mais seulement d'être présent dans sa matrice de conception. Par conséquent, la surface ne doit plus simplement s'animer, mais presqu'au contraire, doit dans sa fixité même faire voir l'activité qui a servi à la générer. C'est comme si la surface « gardait en mémoire » cette énergie de conception*[80].

Sébastien Bourbonnais, 2014

*Ces algorithmes encodent la géométrie et la topologie avec des fonctions complexes et variables et créent ainsi des géométries inédites qui viennent structurer la construction architecturale*[81].

Catalogue Coder le Monde, 2018

L'absence d'analyses détaillées d'algorithmes prive les analyses proposées d'un point de repère technique. Cela se ressent parfois dans les idées sur lesquelles repose l'analyse. Ainsi, dans son ouvrage d'introduction aux pratiques numériques en architecture, Antoine Picon émet l'idée que la pratique du code est pour les architectes source d'émancipation face aux logiciels prédéterminés. Pourtant, l'univers des environnements de programmation et des langages est structuré lui aussi et induit de ce fait des conséquences sur la pratique. Or l'hypothèse ne peut être vérifiée sans un examen

[78] Molly Wright-Steenson, "Archéologies de l'information", in: Andrew Goodhouse (ed.), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, p. 9-22, 2013.
[79] Page consacrée au travail de servo, Frédéric Migayrou; Zeynep Mennan (eds.), *Architectures Non Standard*, Paris : Centre Pompidou, 2003.
[80] Sébastien Bourbonnais, Sensibilités technologiques : expérimentations et explorations en architecture numérique 1987-2010, Thèse de Doctorat, Université Paris-Est, 2014.
[81] Page consacrée au travail de Roland Snooks, in Frédéric Migayrou (ed.), *Coder le monde : mutations, créations*, Éditions HYX et Éditions du Centre Pompidou, 2018.

des algorithmes et des environnements de programmation eux-mêmes. L'absence des algorithmes dans les sources utilisées n'est pas systématiquement un problème. L'enjeu étudié peut dans certains se passer du point de repère technique qu'ils constituent. Du fait de cette absence demeurent néanmoins des questions non résolues. Les extraits ci-dessus nous informent que les fonctions employées sont complexes et variables ou que l'organisation des éléments est déterminée par un algorithme. Mais comment exactement ? S'il s'agit de décortiquer le propre des pratiques avec les outils numériques, encore faut-il observer ces outils numériques d'aussi près que possible.

Comprendre la relation des praticiens aux algorithmes et saisir l'impact de ces outils sur l'espace produit est pourtant l'ambition de nombreux travaux. Parmi ceux produits par des praticiens directement impliqués avec ces outils, les prismes d'analyse proposés sont divers. Mais le manque de mise en relation avec les réalités pratiques du code débouche souvent sur des échafaudages théoriques qui manquent d'une dimension concrète. Ces analyses sont par ailleurs souvent prosélytes : il s'agit avant-tout de convaincre du bien-fondé du recours aux outils numériques et de prouver la révolution qu'ils constituent en architecture. Ces travaux relèvent essentiellement de deux angles d'attaque. Le premier consiste à mettre en relation le recours aux algorithmes avec la conception des formes telle qu'observée dans la nature. Les algorithmes seraient un moyen de se rapprocher de cet idéal formel. Cette vision des choses a été étudiée de près par Christina Cogdell dans son ouvrage *Toward a Living Architecture?: Complexism and Biology in Generative Design*[82]. Bien que le texte superpose les enjeux liées à l'imitation de la nature par le biais des algorithmes et les enjeux lié à l'impact environnemental de la construction, il montre néanmoins particulièrement bien comment un discours sur les productions du champ computationnel s'est construit parmi ses praticiens sur des références aux sciences de la vie. Il souligne aussi combien ce discours est parfois peu cohérent. Le second prisme d'analyse est la mise en relation des travaux de conception numérique avec différents courants philosophiques. La pensée de Gilles Deleuze, en particulier, est identifiée comme fournissant des éléments pertinents pour comprendre les particularités des pratiques computationnelles. La métaphore mathématique employée par celui-ci dans Le Pli, ainsi que la notion de topologie, sont perçues comme des idées décrivant particulièrement bien les productions du champ. Néanmoins, cette pensée est plus tardive que le début des expérimentations avec des outils de programmation puisqu'il s'agit de textes et de projets datant des années 1990 et après. L'histoire de ces idées dans le champ architectural a notamment été traité par Mario Carpo[83]. La notion de topologie a été examinée par Samuel Bernier-Lavigne et Georges Teyssot[84]. L'ouvrage d'Antoine Picon *Culture numérique et architecture: Une introduction* contribue également à ancrer ces réflexions dans les théories

[82] Christina Cogdell, *Toward a Living Architecture?: Complexism and Biology in Generative Design*, University Of Minnesota Press, 2019.
[83] Mario Carpo (ed.), *1993-2012, The Digital Turn in Architecture*, Wiley & Sons, 2012.
[84]Georges Teyssot, avec Samuel Bernier-Lavigne, « Forme et information. Chronique de l'architecture numérique, » in : Action Architecture, (dir.) Alain Guiheux, Paris, Éditions de la Villette, 2011, pp. 49-87.

contemporaines de l'architecture[85]. Ces travaux brossent un tableau clair des réflexions théoriques produites dans le champ computationnel. Néanmoins, l'absence des algorithmes comme objets d'étude dans ces travaux contribue à creuser un fossé entre discours et pratiques du champ computationnel Nous ne souhaitons pas ajouter à l'analyse fournie par ces auteurs sur les théories du champ computationnel, mais plutôt proposer une étude socio-historique de ce même champ.

### *1.2.3 Tournant numérique, tournant computationnel*

Ces travaux d'analyse théorique s'entrecoupent de récits historiques, mais des récits historiques parfois trop détachés de la pratique algorithmique elle-même. En effet, malgré le riche corpus de travaux internes au champ disponible, il en existe beaucoup moins produit par des chercheurs externes, et peu s'attaquent à un récit historique global. Ces recherches amorcent la caractérisation de ce qu'on nomme désormais le tournant numérique. Or cette qualification est la source d'une confusion entre le recours aux algorithmes et le recours à des outils numériques au sens large. Chez Antoine Picon notamment, les deux sont superposés, et les algorithmes sont à peine mentionnés, l'ouvrage se bornant à souligner que l'algorithmique n'a guère changé la pratique architecturale pour l'instant[86]. L'ouvrage s'intéresse aux avant-gardes digitales dans un sens beaucoup plus large. Or la chronologie proposée par Antoine Picon du recours aux outils de programmation plutôt qu'à des fonctions logicielles prédéfinies se trouve ainsi décalée. C'est aussi le cas chez Georges Teyssot et Samuel Bernier, dans l'Archéologie du numérique proposée par le Centre Canadien d'Architecture, ou encore chez Mario Carpo[87]. En effet, les premières utilisations d'ordinateurs en architecture remontent aux années 1960, comme en atteste en particulier le travail de Molly Wright Steenson sur les quelques pionniers de la période[88]. Or les travaux que nous venons de mentionner se concentrent sur la popularisation des outils numériques dans les années 1980 à 2000, c'est-à-dire sur la popularisation des logiciels CAD au sens large. Le postulat d'un tournant numérique est fait au sujet d'une période ou l'ordinateur devient la norme comme outil de dessin. Cette période est marquée d'abord par le développement de l'infographie, puis par l'avènement de l'ordinateur personnel et la généralisation dans les agences de ces machines et des logiciels qui les accompagnent. Le tournant numérique, traite donc du recours à l'ordinateur pour la conception architecturale, mais de manière très générale, puisqu'il s'agit du recours à des outils numériques quels qu'ils soient[89].

---

[85] Antoine Picon, C*ulture numérique et architecture : une introduction*, Birkhauser, 2010.
[86] Antoine Picon, C*ulture numérique et architecture : une introduction*, Birkhauser, 2010, p. 94.
[87] Georges Teyssot, avec Samuel Bernier-Lavigne, « Forme et information. Chronique de l'architecture numérique, » in : Action Architecture, (dir.) Alain Guiheux, Paris, Éditions de la Villette, 2011, pp. 49-87. Mario Carpo (ed.), *1993-2012, The Digital Turn in Architecture*, Wiley & Sons, 2012.
[88] Molly Wright-Steenson, *Architectural Intelligence: How Designers and Architects Created the Digital Landscape* MIT Press, 2017. Matthew Allen, P*rehistory of the Digital: Architecture Becomes Programming, 1935-1990*. Thèse de doctorat, Harvard University, Graduate School of Arts & Sciences, 2019.
[89] Usuellement des outils de dessin assisté par ordinateur tout de même.

Du recours des logiciels, plus exactement, et non du recours à des outils de programmation. L'histoire du champ computationnel est donc pour l'instant faite incluse dans l'histoire du tournant numérique, une superposition à notre sens problématique. Il s'agit en effet non seulement d'une confusion entre enjeux techniques et épistémologiques, comme nous l'avons vu plus haut, mais aussi d'une confusion entre pratiques numériques et pratiques computationnelles. Le récit fait est celui d'une irruption technologique certes passionnante mais qui évalue sans nuance le changement que constituent les outils numériques. Pourtant ces récits interrogent les changements qu'amènent ces nouveaux outils, mais en les plaçant tous sur le même plan. Seul à marquer la distinction, Mario Carpo a consacré deux livres au récit de ces deux tournants : *The Alphabet and the Algorithm* et *The Second Digital Turn*[90]. La chronologie proposée y est cependant très différente de celle que nous envisageons. En effet, elle se repose d'une part sur une histoire des théories des pratiques numériques en architecture à partir des années 1990, notamment la place de la notion de non-standard, et d'autre part sur une analyse du rôle de l'architecte comme auteur depuis la Renaissance. Son travail s'ancre donc notamment dans la question des outils de représentation employés dans la conception architecturale, bien plus que dans l'examen des algorithmes eux-mêmes. Beaucoup a été écrit sur le tournant numérique, quelques éléments aussi sur le tournant computationnel, mais toujours en les mélangeant, ce qui biaise la chronologie du tournant computationnel en le ramenant en arrière. En effet, le champ computationnel, c'est-à-dire le groupe des praticiens qui recourent à des outils de programmation, prédate le tournant numérique de deux décennies. Mais le tournant computationnel, c'est-à-dire la popularisation de ces outils, est plus tardif que la généralisation de l'ordinateur en agence d'architecture. C'est à cette chronologie que nous nous intéressons dans le présent texte.

Enfin, numérique ou computationnel, une question demeure. S'agit-il véritablement d'un tournant ? Les textes consacrés au numérique en architecture commencent fréquemment par des assertions sur le changement total que représente le numérique pour les pratiques architecturales. Hernan Diaz Alonso, Neil Leach, Daniel Davis, Sanford Kwinter et bien d'autres appuient volontiers cette idée[91]. Mario Carpo présente le numérique comme un bouleversement équivalent à l'invention de l'impression[92]. C'est précisément pour cette raison qu'il s'agirait d'un tournant. La notion de tournant questionne cependant nombre d'usagers de ces outils dans le monde architectural. A ce titre, le numérique est bien un tournant, puisque presque tous les architectes ont aujourd'hui recours à des outils de dessin

[90] Mario Carpo, *The Alphabet and the Algorithm*, The MIT Press, 2011. Mario Carpo, *The Second Digital Turn: Design Beyond Intelligence*, The MIT Press, 2017.
[91] Hernan Diaz Alonso, "seingemer", in Elias Guenoun (éditeur), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007. Sanford Kwinter, *Far from Equilibrium: Essays on Technology and Design Culture*, Actar Press, 2008. Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021. Neil Leach, "Parametrics explained", *Next Generation Building*, Vol. 1, p. 1-10, 2014.
[92] Mario Carpo, *The Alphabet and the Algorithm*, The MIT Press, 2011.

assisté par ordinateur et à des modeleurs 3D[93]. Les outils computationnels sont également présentés comme de plus en plus présents en agence d'architecture[94]. Marquer une différence nette entre outils numériques au sens large et outils computationnels nécessite cependant de réexaminer cette affirmation, pour vérifier l'usage fait des outils de programmation directement. Quelle place occupent-ils dans la conception architecturale ? Au-delà de la prédominance de ces outils, la notion de tournant demande aussi de comprendre si le changement de paradigme si souvent mentionné au sujet de ces outils est bien là. Le recours aux algorithmes modifie-t-il les pratiques de conception architecturale ? Ce sont ces questions que nous nous proposons d'examiner à travers l'examen des algorithmes eux-mêmes et le récit du développement du champ computationnel en architecture.

## 1.3 Saisir les changements de la pratique architecturale

### *1.3.1 Caractériser la pratique architecturale : le tacite comme marqueur*

Pour tracer le tableau des pratiques computationnelles en architecture, il faut les observer et les cartographier, selon une méthodologie que nous détaillerons un peu plus loin. Cependant, parvenir à évaluer le changement que ces pratiques constituent par rapport à la pratique architecturale conventionnelle implique une analyse plus poussée. Or, pour cadrer cette possible différence, encore faut-il pouvoir cerner la pratique architecturale elle-même. Pourtant, cette pratique architecturale est si différente d'un praticien à l'autre qu'en parler au singulier semble dérisoire. Il nous faut pourtant établir des points de repère communs à tous les praticiens impliqués dans des travaux de conception architecturale, afin de déterminer si les pratiques computationnelles s'en éloignent. Avant d'établir ces points communs, il nous faut définir ce que nous entendons exactement par *architecture*. Après tout, nous nous sommes jusqu'ici beaucoup intéressés à la première partie de notre expression, *champ computationnel*. Mais c'est du champ computationnel en *architecture* que nous souhaitons traiter. Par architecture nous entendons ici la production de l'espace et l'ensemble des étapes de ce processus de production. Nous englobons donc la conception et la fabrication (ou la construction) dans le terme production, d'où l'usage au fil du texte de la notion de conception spatiale. Les objets architecturaux résultent usuellement de ce processus de production exécuté dans son ensemble. L'espace est conçu, représenté, construit, et le résultat est un bâtiment, Mais certains travaux d'architecture n'accomplissent qu'une partie du processus : ce sont des projets "de papier"[95]. Nous les considérons ici comme relevant autant de l'architecture que les bâtiments réalisés, d'autant que les algorithmes

[93] Voir à ce sujet le travail d'Adeline Stals, notamment Adeline Stals, P*ratiques numériques émergentes en conception architecturale dans les bureaux de petite taille. Perceptions et usages de la modélisation paramétrique*, Thèse de doctorat, Université de Liège, 2019.
[94] Neil Leach, "Parametrics explained", *Next Generation Building*, Vol. 1, p. 1-10, 2014. Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.
[95] De l'anglais *paper project* - projet non réalisé.

que nous cherchons à observer interviennent au stade de la conception. De nombreux projets spéculatifs font également partie du corpus intégré, sans que cela ne nous paraisse sortir du spectre de la production architecturale. De la même manière, l'espace n'est pas toujours produit par des architectes, et nous prenons en considération aussi des espaces qui seraient conçus par leurs habitants, par des artistes, des ingénieurs ou par des programmeurs.

L'élément clé de notre définition des pratiques architecturales, celui qui va nous permettre d'interroger la différenciation des pratiques computationnelles en regard des pratiques conventionnelles, c'est l'idée que les pratiques architecturales les plus diverses ont un point commun : elles reposent sur des savoirs tacites. Tacite et explicite sont des qualificatifs associés avec la connaissance : on oppose les connaissances explicites, formalisées et de ce fait facilement transmissibles, aux connaissances tacites, non formalisables et donc impossibles à transmettre directement d'un individu à un autre. Le champ des connaissances explicites est donc clairement délimité, puisqu'il s'agit de ce qui peut être réduit à une structure formelle, ce qui peut être exprimé sur un support matériel physique. L'explicite est ainsi difficile à dissocier des supports sur lesquels il s'inscrit : procédures, équations, recettes et autres documents écrits sont autant d'exemples de connaissances explicites. A l'inverse, les connaissances tacites sont celles qui ne peuvent être codifiées, et ne peuvent donc être transmises ni sur des supports matériels, ni par la parole. La notion de connaissances tacites est d'abord développée à partir des années 1950 par le philosophe Michael Polanyi[96], avant de devenir dans les années 1980 un outil classique de travail dans les théories de la connaissance ainsi que dans le domaine du management, où des méthodes d'évaluation des connaissances tacites personnelles et de leurs conditions de transmission émergent[97]. Les connaissances tacites ont également acquis une place importante en sociologie des sciences ou la dimension objective érigée en totem de la production des connaissances scientifiques est remise en perspective grâce à un travail d'identification des connaissances tacites présentes au sein de tout travail[98], quelle que soit sa nature et son objectif.

Polanyi, dont le travail sur la nature de la connaissance découle de l'intuition que "nous croyons plus que nous ne pouvons prouver et savons plus que nous ne pouvons dire"[99], définit les connaissances tacites comme n'étant pas exprimables par des règles, et n'obéissant donc pas aux cadres logiques qui nous servent habituellement de références communes. Toutes les connaissances tacites que nous

[96] Polanyi, M., *The Tacit Dimension*, Routledge, 1966 et Polanyi, M., *Personal Knowledge : Towards a post-critical philosophy*, University of Chicago Press, 1958.
[97] On reviendra en particulier dans le chapitre 6 sur le travail d'Ikujiro Nonaka.
[98] On reviendra en particulier dans le chapitre 6 sur le travail de Harry Collins.
[99] "*We believe more than we can prove, and know more than we can say*" - Polanyi, M., *The Tacit Dimension*, Routledge, 1966.

possédons se basent sur un ensemble d'informations conceptuelles et sensorielles subjectives[100], et sont donc personnelles à chaque individu. Elles sont ce qui permettent de développer compétences, créativité et capacité de jugement - des composantes essentielles du savoir-faire[101]. Polanyi donne de nombreux exemples des connaissances tacites : la capacité à exécuter une action physique, composée d'un ensemble de mouvements, sans pour autant être capable de décrire l'ensemble de ces mouvements que le corps exécute pour pouvoir accomplir cette action - par exemple pour faire du vélo; la capacité à identifier le visage d'une personne connue, sans pour autant être capable de décrire ou dessiner intégralement ces traits; ou encore la capacité à apprendre et parler un langage sans avoir besoin d'en connaître les règles grammaticales exactes, par l'exposition répétée à l'écoute et par la pratique verbale - comme le font les enfants au moment d'apprendre une langue maternelle. Selon Polanyi, connaissances explicites et tacites sont par ailleurs indissociables : chaque connaissance explicite a en quelque sorte son miroir tacite, qui nous permet de savoir comment utiliser les connaissances explicites que nous possédons dans une situation donnée, de savoir comment adapter aux particularités de cette situation unique des connaissances antérieures[102]. Les connaissances tacites que nous possédons sont donc essentielles à notre appréhension du monde. Cependant, au moment de la transmission des connaissances, seule la partie explicite peut être communiquée. La personne émettrice ne peut donc que laisser la personne réceptrice reconstituer la partie tacite qui n'a pu être transmise[103]. Les modèles d'analyse de la transmission des connaissances tacites développés par la suite cherchent donc à identifier comment, en fonction des modes d'apprentissage, la facilitation de cette reconstitution peut être permise. Ainsi, si pour Polanyi le prélude à l'existence des connaissances tacites est que "tout savoir est personnel"[104], les domaines du management ou de la philosophie des sciences s'intéressent à la dimension collective de ce savoir, cherchant à identifier les points de passage du tacite vers l'explicite et de l'individuel vers le collectif[105].

La conception architecturale a pour caractéristique la mobilisation de nombreuses et diverses connaissances au cours des opérations successives qui mènent de l'esquisse d'un espace à sa construction. Nombre de praticiens font le parallèle entre la pratique de l'architecte et celle d'un chef

---

[100] "*Tacit knowledge comprises a range of conceptual and sensory information that is featured with strong personal subjectivity*" - Polanyi, M., *The Tacit Dimension*, Routledge, 1966.

[101] skills, experience, insight, creativity, judgment (decision) (+ intuition) - summed as know-how
(Ryle 1945 paper difference between knowing how and knowing what)

[102] "*It supplies us with the context within which our articulations have meaning*" - Polanyi, M., *The Tacit Dimension*, Routledge, 1966.

[103] "*Our message had left something behind that we could not tell, and its reception must rely on it that the person addressed will discover what we have not been able to communicate*" - Polanyi, M., *The Tacit Dimension*, Routledge, 1966. Selon certains travaux, un minimum de confiance entre les deux parties prenantes de la transmission est même nécessaire pour garantir la passation de connaissances - article Creating a knowledge management culture for Ganga River

[104] "*All knowing is personal*" - Polanyi, M., *The Tacit Dimension*, Routledge, 1966.

[105] Collins, H. *Tacit and Explicit Knowledge*, University of Chicago Press, 2010. Nonaka, I., Toyama, R. & Konno,N., "SECI, Ba and Leadership: a Unified Model of Dynamic Knowledge Creation", *Long Range Planning* Vol. 33, n.1, pp. 5-34, 2000.

d'orchestre, voyant en l'architecte un acteur du projet dont le travail est de coordonner l'ensemble des spécialistes des différents savoirs mobilisés dans l'objectif de parvenir à donner forme et corps au bâtiment. Parmi les connaissances explicites mobilisées, les exemples les plus évidents sont celles relevant du domaine de la thermique ou de celui de la structure et les modèles de calcul qui leur sont associés, ou encore la géométrie, partagée par tous, ou les modèles de calcul des coûts financiers d'un projet. Cependant, si l'on suit la définition des connaissances tacites données par Polanyi, chacun des ces ensembles de connaissances explicites possède un corpus de connaissances tacites associé, mobilisé au même moment par les différents spécialistes. Corpus de connaissances spécialisées de plus en plus souvent utilisées dans le monde de l'architecture, l'analyse de cycle de vie donne un bon exemple de cette association entre tacite et explicite, car elle donne à voir particulièrement bien la nécessité de choix guidant la construction d'un modèle sur-mesure à chaque nouvelle évaluation - des modèles qui reposent sur des connaissances explicites de la mesure de l'impact environnemental, alors que ces prises de décisions permettant de créer un modèle adéquat pour chaque projet reposent sur le savoir tacite des experts de ces outils. Néanmoins, en parallèle de ces ensembles de connaissances explicites, usuellement reconnues comme objectives car les formes sous lesquelles elles sont explicitées sont des modèles applicables à la conception de n'importe quel espace, la conception spatiale mobilise aussi des connaissances qui n'ont jamais été formalisées dans des modèles généralisables. Ainsi, recherches formelles et hiérarchisation spatiale font rarement l'objet de règles de conception complètes qui soient reconnues comme valides par une majorité de la profession, qui cultive plutôt une image de l'architecte proche de celle de l'artiste, avec un travail et un savoir unique à chaque praticien - une perception contemporaine du métier d'architecte pas toujours fidèle à la réalité mais néanmoins tenace. Il existe quelques exceptions : les schémas de logement proposés par l'architecture moderne suivent des règles d'orientation et de connexion des différents espaces du logement qui sont encore aujourd'hui suivies dans de nombreux bâtiments. Ce sont cependant malgré tout des règles de conception informelles : bien qu'elles soient explicitées verbalement, elles ne sont pas codifiées dans des supports considérés comme incontournables à l'apprentissage et à la pratique de l'architecture, auxquelles on ne pourrait pas déroger sans conséquences majeures. Si les choix qui sont fait dans ces domaines le sont en accord avec la mise en application des connaissances explicites des autres domaines déjà pointés - les décisions qui ont trait à la forme construite d'un bâtiment se font rarement sans prendre en compte les performances structurelles de cette forme en complément de la hiérarchisation des espaces abrités -, ils sont généralement perçus comme l'apanage de l'architecte, dont c'est donc l'expertise. S'il est parfois difficile de pointer les connaissances explicites qui seraient propres à l'architecte en raison de la structuration contemporaine des métiers du domaine et de la séparation des connaissances entre architectes, ingénieurs et constructeurs qu'elle implique, il existe donc néanmoins un ensemble de connaissances tacites propres à la conception spatiale - celles dont Christopher Alexander ou Nigel Cross parlent dans leurs écrits.

En 1982, dans son article *Designerly Ways of Knowing*, Nigel Cross pose l'existence d'un savoir-faire propre aux activités de conception[106]. L'article et l'idée sont ensuite repris ensuite largement par ses pairs de la Design Research Society puis au-delà, dans tout le champ de la théorie du design qu'ils ont contribué à créer. Cette idée est alors le support d'un plaidoyer pour l'intégration aux études secondaires d'une formation à la conception. Le savoir-faire propre au design est en effet, selon Cross et ses collègues, un outil utile au quotidien quelle que soit la profession exercée. La théorie exposée dans *Designerly Ways of knowing* s'appuie sur une distinction faite dès 1945 par Gilbert Ryle, philosophe anglais contemporain de Nigel Cross, entre savoir et savoir-faire[107]. Le savoir, aussi appelé savoir descriptif ou déclaratif, concerne les connaissances qui peuvent être verbalisées. Le savoir-faire, aussi appelé savoir performatif ou pratique, concerne les connaissances relatives à l'exécution d'une tâche. L'expertise du concepteur relève selon Cross de la seconde, et implique donc des connaissances qui ne peuvent être verbalisées. Nigel Cross ne va néanmoins pas beaucoup plus loin que la simple reconnaissance de cette dimension indicible. Néanmoins, son travail de promotion de la théorie du design par le biais de la Design Research Society et de ses activités contribue beaucoup à populariser l'idée de connaissances propres au design, et par extension à l'architecture. Cross et ses collègues s'interrogent essentiellement sur le programme pédagogique qui permet d'acquérir ce savoir-faire. En effet, ils considèrent que s'il s'agit bien d'une expertise propre à l'architecte et au designer, tout le monde devrait cependant l'acquérir et donc suivre la formation nécessaire. Christopher Alexander soutient à l'inverse qu'il ne s'agit pas d'une expertise à acquérir par la pratique, mais de connaissances que tout le monde possède. Ces connaissances sont ce qu'il faut parvenir à mobiliser pour parvenir à une architecture à un environnement de qualité. Alexander émet de plus l'idée que la qualité à recherche est une *qualité sans nom*, qu'on reconnaît en la voyant[108]. Bien qu'il ne nomme pas cette qualité, et bien qu'il argumente que tout le monde possède les connaissances nécessaires pour l'atteindre sans formation, Alexander décrit au fil de ces travaux nombre des composantes de ce savoir-faire de la conception. Le contexte tel qu'il est discuté dans *Notes on the Synthesis of Form* et les motifs développés dans *A Pattern Language* et dans *The Timeless Way of Building* en font partie[109]. Les capacités mentales de synthèse et d'association sur lesquels le processus de travail qu'il décrit s'appuient sont également des éléments clés du savoir-faire qu'il discute dans ses textes. Bien que ni Cross ni Alexander n'utilisent la notion de tacite dans leurs travaux, leurs réflexions s'ancrent dans la reconnaissance de cette propriété des connaissances mobilisées au cours du processus de conception.

---

[106] Cross, N., "Designerly Ways of Knowing", in *Design Studies* Vol. 3 n. 4, pp. 221-227, 1982.
[107] En anglais, *knowing that* et *knowing how*. Ryle, G The concept of mind Hutchinson, London 1949
[108] En anglais, *quality without a name*. Alexander C., *Notes on the Synthesis of Form*, Harvard University Press, 1964.
[109] Alexander C., *Notes on the Synthesis of Form*, Harvard University Press, 1964. Alexander, C., Silverstein M., et Ishikawa, S., *A Pattern Language : Towns, Buildings, Construction*, Oxford University Press, 1977. Alexander, C., *The Timeless Way of Building*, Oxford University Press, 1979.

Seules quelques études existantes identifient directement la connaissance architecturale comme une connaissance tacite[110]. Parmi les travaux accomplis ou en cours, il faut mentionner le projet de recherche Communities of Tacit Knowledge: Architecture and its Ways of Knowing[111], qui regroupe dix travaux de doctorat consacrés à cet enjeu. Ces travaux sont consacrés à des réflexions théoriques sur la notion de tacite, à des recherches historiques sur la place de cette notion en architecture, à des études de cas, au développement de nouvelles méthodes de travail reposant sur cette notion, et à un examen de l'enseignement de l'architecture à l'aune de cette notion. Bien que le groupe de travail n'aie pas encore publié de résultats de recherche, le projet en cours questionne notamment le besoin d'explicitation des savoirs tacites sur lesquels repose la pratique architecturale. La difficulté de formalisation d'instructions à suivre pour apprendre à concevoir ou pour décrire les activités de conception est au cœur de beaucoup de discussions sur la nature de la pratique architecturale. En effet, si cette question est rarement identifiée directement comme relevant d'une formalisation du tacite vers l'explicite, la difficulté à parler d'architecture précisément est bien identifiée. Cet indicible est désigné par une multitude de notions mobilisées ici ou là par des auteurs de la recherche en design et de la théorie de l'architecture pour désigner cet indicible. Un grand nombre d'études consacrées à la question de l'apprentissage en architecture identifient la particularité des connaissances mobilisées dans la conception architecturale, bien qu'elles ne la désignent pas par ce terme. À titre d'exemple, Uluoglu traite par exemple de connaissances personnelles en matière de conception, une notion qui peut être assimilée aux connaissances tacites[112].

Créativité, intuition, subjectivité, implicite, empirique - pour décrire l'expertise de l'architecte, on retrouve chacun de ces mots utilisés[113]. Chacun permet de cadrer un aspect particulier de l'expérience du praticien et de sa méthode de travail. Tous impliquent cependant une tension entre connaissances tacites et méthodes de formalisation. Dans les textes du champ computationnel, nous observons la même diversité de vocabulaire. La question est cependant très souvent discutée en lien avec la notion de décision, et la méthodologie mise en œuvre par les praticiens pour prendre des décisions dans le cadre du travail de conception en cours. La méthode identifiée est caractérisée d'empirique, c'est-à-dire qu'elle s'appuie sur l'expérience et non sur le raisonnement. La notion d'explicite est par ailleurs très bien identifiée, puisqu'il s'agit de rendre explicite cette méthodologie de travail pour pouvoir la transformer en programme informatique. Explicite est cependant usuellement opposé à implicite, et non à tacite[114]. Or implicite ne veut pas nécessairement dire qu'on ne peut pas formaliser,

[110] Kruchten, Philippe, Patricia Lago, Hans van Vliet, et Timo Wolf. "Building up and Exploiting Architectural Knowledge." Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, 2005.
[111] https://tacit-knowledge-architecture.com/
[112] Uluoglu, Belkıs. 2000. "Design knowledge communicated in studio critiques," https://doi.org/10.1016/S0142-694X(99)00002-2.
[113] En anglais, on retrouve les mots *skills, experience, insight, creativity, judgment, intuition, know-how*
[114] Par exemple dans les textes de Nicholas Negroponte ou Robert Aish. Voir Negroponte, N., *The Architecture Machine*, The MIT Press, 1970 ou Robert Aish and Emmanuel Mendoza. 2016. DesignScript: a domain specific

cela atteste simplement du fait que ce n'est pas formalisé. Ainsi, et nous discuterons plus longuement de cette question dans la suite du texte, le savoir-faire de l'architecture est considéré dans le champ computationnel comme à formaliser plutôt que non formalisable. Bien que ce choix de vocabulaire dénote des ambitions différentes, nous retrouvons néanmoins la caractéristique tacite du savoir-faire de la conception spatiale. La mobilisation des notions de tacite et d'explicite en architecture ne se fait donc pas en ces termes, mais l'idée traverse de nombreux corpus de textes. Or cette diversité de vocabulaire nuit à une compréhension fine des implications de l'idée que le savoir-faire architectural est tacite. L'ensemble des notions utilisées incluent pourtant tous cette idée comme point de départ. Ceci qui nous permet de poser comme point commun des pratiques architecturales ce savoir-faire tacite de la conception spatiale, idée que nous allons utiliser comme point de départ pour évaluer en quoi le recours aux algorithmes change le travail de conception.

### *1.3.2 Traduction tacite-explicite*

Comme ailleurs, entre les savoirs tacites et explicites existe un équilibre essentiel en architecture. Pour mieux caractériser la pratique architecturale et les particularités du recours aux outils computationnels, il nous faut donc déterminer ses formes explicites. Bien qu'il soit difficile de formuler une méthode de conception architecturale générale, la discipline offre tout de même des formes explicites communes et reconnues. Construction et représentation sont en effet usuellement identifiées comme la formalisation en vigueur en architecture. Le dessin et le bâtiment, formes physiques de l'architecture, sont aussi ses formes explicites. C'est ce que Chris Yessios, architecte et développeur du logiciel de modélisation Form*Z dans les années 1990, entend par la séparation qu'il marque entre conceptualisation et exécution, et par la définition de cette dernière comme la matérialisation physique[115]. C'est aussi ce que l'on peut déduire du modèle SECI de transmission des connaissances, développé par Ikujiro Nonaka, professeur d'économie, qui identifie l'usage de croquis comme une forme d'externalisation, c'est-à-dire d'explicitation[116]. À partir de l'idée que représentations et bâtiments sont les formes explicites propres à l'architecture, nous pouvons postuler que la conception architecturale est toujours un acte de traduction d'un ensemble de connaissances tacites elles aussi propres à l'architecture en cette forme explicite. Il n'y a donc pas de méthode commune et reconnue de conception, mais une traduction qui a systématiquement lieu. Donner leur forme explicite aux connaissances tacites dans le champ computationnel, c'est formuler des instructions transmises à l'ordinateur plutôt que représenter et construire. C'est ensuite l'ordinateur qui en suivant les instructions va produire une représentation, ou la machine à commande numérique

---

language for architectural computing. In Proceedings of the International Workshop on Domain-Specific Modeling (DSM 2016). Association for Computing Machinery, New York, NY, USA, 15–21.

[115] Serraino, P. History of Form*Z, Birkhäuser, 2002.

[116] Ikujiro Nonaka et Noboru Konno, "The Concept of 'Ba': Building a Foundation for Knowledge Creation", California Management Review, 40:3, p. 116-132

qui en suivant ces instructions va produire une construction. Il existe donc là une autre forme explicite de la conception architecturale, qui donne corps à une étape intermédiaire usuellement non formalisée.

Des mentions de l'explicitation des instructions nécessaires aux pratiques computationnelles sont faites ici et là à travers les textes, sans que l'enjeu que cela représente ne devienne un point clé des discours. David Stasiuk ou Daniel Davis, notamment, mentionnent directement la notion d'instructions explicites[117]. C'est la même idée qu'on retrouve chez John Frazer ou Paul Coates, deux praticiens du champ computationnel des années 1970 et 1980, à travers la notion de lisible par la machine.

*La façon dont Frazer explique la relation entre l'architecture et le script informatique dans son texte "A Natural Model for Architecture" publié en 1995, explique comment, entre ses mains, l'architecture devient lisible par la machine. Il pense en termes de langage, avec un vocabulaire et une syntaxe*[118].

*Vous avez, j'en suis sûr, une base de données visuelle complète (bien que sommaire) des formes architecturales passées. Mais tout cela se passe dans notre tête. Ce que nous devons faire, c'est sortir les idées de nos têtes et les mettre sous une forme lisible par une machine*[119].

Il s'agit ici d'un point clé du discours des pionniers du champ sur la place de la machine dans le processus de conception architecturale. L'enjeu est ici la traduction de l'humain vers la machine. Il s'agit de rendre assimilable par la machine une série de perceptions et réflexions humaines. Il n'est donc pas question uniquement de l'explicitation du savoir-faire du praticien, mais aussi de l'automatisation du processus de conception. Dans son introduction au catalogue de l'exposition Pavillon Seroussi, un texte introduisant le travail de six agences alors parmi les principales dans le champ computationnel, Elias Guenoun développe par exemple la notion de traductibilité générale pour souligner l'instantanéité du processus de conception algorithmique[120]. Gordon Pask, cybernéticien proche des pionniers du champ computationnel, pour qui les systèmes et leurs relations

---

[117] David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021. Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.

[118] "*The way Frazer explains the relation of architecture to computer script in his text "A Natural Model for Architecture" published in 1995, explains how, in his hands, architecture becomes machine-readable. He thinks in terms of language with vocabulary and syntax*" Eliza Pertigkiozoglou, "1976. Cedric Price & John Frazer", 8 Avril 2017, https://eliza-pert.medium.com/1976-22121bb498c4, consulté le 15 Novembre 2021.

[119] "*You have I trust a comprehensive (though sketchy) visual database of past architectural forms. But all this goes on inside our heads. What we have to do is to get the ideas out from our heads and into a machine readable form*". Paul Coates, Tom Appels, Corinna Simon et Christian Derix,'Current work at CECA', *Proceedings of the 4th Generative Art Conference (GA2001)*, Milan: Generative Design Lab Milan Polytechnic University, Italy, 2001.

[120] Elias Guenoun (ed.), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.

ne sont pas fixes mais évolutifs, est de ce fait convaincu que la cybernétique permet la traduction nécessaire des contraintes architecturales en instructions informatiques[121].

Bien qu'elle ne soit pas identifiée comme la clé de voûte permettant de définir les pratiques architecturales computationnelles, la notion de traduction entre tacite et explicite en travers néanmoins les textes. Nous faisons ici le choix de faire de cette traduction un élément clé pour mieux comprendre le propre de ces pratiques. Puisque leur marqueur principal est le recours aux algorithmes, alors implémenter des connaissances tacites dans une structure intégralement explicite, celle de la programmation informatiques, est aussi une composante essentielle. Nous pourrons ainsi examiner de plus près en quoi le recours à la programmation relève ou non d'une automatisation de la prise de décision par le recours à des outils algorithmiques au cours du processus de conception spatiale. En effet les étapes de traduction peuvent se découper de plusieurs manières, la traduction peut se faire de plusieurs manières. Le premier objectif est de formuler des instructions, mais le processus procédural final pour la machine a aussi des impératifs pratiques. Il faut pouvoir mettre au point des programmes plus ou moins complexes, avec plus ou moins d'étapes. Il faut pouvoir assez vite tester les résultats, les visualiser et ajuster le script. Il faut aussi prendre en compte les contraintes du projet. La gestion de ces impératifs se fait différemment en fonction des praticiens et des outils. Il existe de plus des dispositifs de traduction intermédiaires, sortes de passerelles entre l'intuition et la formalisation, entre l'intention et la contrainte. Ce sont ces dispositifs que nous allons examiner à travers la cartographie des outils du champ.

## 1.4 Cartographier les pratiques computationnelles en architecture

### *1.4.1 Objets suivis*

L'historiographie existante du champ computationnel démontre un besoin intact de décrire concrètement ses pratiques, en particulier sur le plan technique. Malgré les apports des travaux existants, nous manquons encore d'une vision globale tant sur le plan historique que sur le plan du rapport à l'outil entretenu par les praticiens. Plus que définir, il nous faut donc commencer par cartographier le champ à l'étude. Pour mener à bien cette cartographie, trois types d'objets doivent être suivis : les algorithmes, les projets et les praticiens. Chacun doit nous permettre de répondre à certaines des questions posées, par l'obtention d'un certain ensemble de données à leur sujet.

L'aspect technique dont nous avons souligné qu'il nous paraît si lacunaire, nous souhaitons le comprendre en consultant les algorithmes utilisés pour la conception architecturale au sein du champ.

[121] Pickering, A., "Gordon Pask: From Chemical Computers to Adaptive Architecture", in Pickering, A., *The Cybernetic Brain. Sketches of Another Future*, University of Chicago Press, 2009.

Plus exactement, nous voulons consulter le code écrit pour produire les projets, le texte saisi dans un environnement de programmation dont l'exécution produit les éléments du projet. La formation reçue nous permettant de lire et comprendre les codes des différents projets, l'objectif est de saisir comment les praticiens articulent leurs intentions architecturales et l'écriture de ce code. Les informations à obtenir pour pouvoir mener à bien le travail d'analyse des algorithmes sont relatives au code, à ses entrées et à ses sorties, ainsi qu'au set-up technique utilisé. Ce dernier couvre l'ensemble des logiciels et machines utilisés au cours du processus. Langages de programmation, environnements de programmation, logiciels, mais aussi outils d'interaction avec l'ordinateur peuvent varier. Nous avons par exemple aujourd'hui l'habitude d'utiliser une souris et un clavier, mais il existait auparavant d'autres dispositifs d'interfaces, comme les stylets, que nous découvrirons dans le chapitre II. Nous verrons dans ce même chapitre et dans le suivant l'existence de dispositifs physiques de dessin ou de construction de maquettes eux aussi complémentaire au set-up de programmation en lui-même. Les entrées sélectionnées pour l'algorithme, c'est-à-dire le type des paramètres et les valeurs explorées pour chacun, sont importantes pour comprendre l'articulation du savoir-faire architectural et de la formalisation du code. Nous reviendrons sur cette articulation en fin de chapitre, mais les entrées jouant un rôle clé dans sa compréhension, il est important de les documenter. En fonction des valeurs de paramètres, l'algorithme résulte en différentes sorties. Nous avons vu plus haut avec Robert Woodbury qu'une des caractéristiques de la conception paramétrique et algorithmique est la production d'un grand nombre de variantes[122]. Il nous faut donc comprendre la place de ces variantes dans le processus. Par ailleurs, la nature exacte des sorties est à étudier, puisque nous n'observons pas simplement des algorithmes, mais des algorithmes pour la conception architecturale. David Stasiuk pointait justement l'existence de sorties géométriques, mais aussi de sorties numériques ou textuelles[123]. L'une des caractéristiques des pratiques computationnelles est justement la transformation de certains types de données en d'autres types de données au fil du processus. Cette transformation s'exécute parfois intégralement au cours du processus algorithmique, mais parfois aussi en dehors. Documenter les sorties permet donc de comprendre quand et comment les différents types de données s'articulent. Les sorties permettent également de comprendre comment l'espace produit au cours du processus algorithmique est lu par les architectes, en fonction de la nature de ces sorties.

En regard de ces variations dans la nature des sorties, il nous faut également interroger leur statut. Les sorties de l'algorithme constituent-elles le projet en lui-même ? Pour répondre à cette question, il nous faut observer non seulement les algorithmes, mais aussi le projet dont ils font partie, ou plutôt qu'ils permettent de produire. Les algorithmes consultés sont en effet écrits pour des projets donnés.

[122] Robert Woodbury, *Elements of Parametric Design,* Routledge, New York, 2010.
[123] David Stasiuk, "Design Modeling Terminology", 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.

Certains sont parfois réutilisés ensuite pour d'autres projets. Nous avons vu que les algorithmes sont usuellement combinés entre eux pour former des programmes. Ils peuvent donc être recombinés pour former d'autres programmes. Il faut donc documenter ces recombinaisons, observer l'usage des différents algorithmes au sein des projets pour lesquels ils sont mobilisés. En complément des algorithmes utilisés, il nous faut également documenter le programme du projet, son contexte géographique et ses conditions de réalisation, ainsi que les conditions d'accès à la commande pour les praticiens. En effet, nous avons souligné que les conditions de réalisation des projets peuvent varier beaucoup : commandes privées, concours, commandes pour expositions. L'objet des projets varie également : pavillons, musées, prototypes pour démontrer la faisabilité d'un processus de fabrication. Nous avons choisi de tout prendre en compte malgré cette diversité, mais il faut la documenter pour en garder trace autant que pour comprendre ce qu'elle peut nous apprendre des pratiques de conception dans le champ computationnel. Là où pour les algorithmes, nous souhaitons consulter le fichier de code lui-même, l'exécuter, le manipuler, pour les projets, il s'agit à la fois de lister les informations pertinentes et de consulter les documents classiques de la réalisation d'un projet d'architecture. Parmi les objets physiques se trouvent donc des maquettes, des prototypes à diverses échelles et des pavillons ou des bâtiments à l'échelle 1:1. Ces objets peuvent être observés directement ou documentés par le biais de photographies. Ces photographies font partie de l'iconographie usuelle d'un projet architectural, tout comme les dessins : esquisses, plans, coupes, perspectives, axonométries. Il s'agit donc de recenser pour chaque projet examiné l'iconographie disponible, tant les représentations classiques que des images particulières, propres au champ computationnel. Ceci nous aidera à comprendre comment la conception computationnelle est observée et communiquée par les praticiens impliqués. Nous souhaitons également nous appuyer sur cette documentation des projets d'architecture conçus à l'aide d'outils de programmation pour essayer de saisir ce qui fait le propre des pratiques computationnelles. Nous explorerons donc si certains marqueurs esthétiques ou spatiaux se retrouvent dans les différents projets à l'aide de cette production graphique.

Pour comprendre comment se structurent les pratiques computationnelles, il faut également documenter les différents rôles tenus par les différents praticiens, déterminer la liste de ceux participant à chaque projet et leur qualité : utilisateur, concepteur, programmeur. Pour chacun de ces participants, il faut également déterminer leur statut professionnel et leurs lieux d'exercice. Nous avons souligné la diversité des praticiens à l'œuvre, entre architectes, ingénieurs et informaticiens. Nous avons également vu que les pratiques computationnelles naissent dans plusieurs milieux de pratique : industrie du logiciel, campus universitaires, agences d'architecture, bureaux d'études. Pour documenter les usages de la programmation informatique dans ces différents milieux, nous souhaitons identifier les lieux d'exercice de ces pratiques et le parcours des praticiens impliqués dans les projets recensés. Il nous font donc recenser, en fonction des différents projets, quels praticiens ont participé.

Puis, pour chacun de ces praticiens, leurs trajectoires, de leur formation à leurs occupations professionnelles successives et à leur investissement dans des associations, des comités scientifiques d'expositions ou de journaux, ou encore dans la pédagogie des établissements d'enseignement qui les accueillent.

### *1.4.2 Constitution du corpus*

Puisqu'il s'agit de documenter projets, algorithmes et parcours des praticiens, il faut donc déterminer à la fois comment établir la liste des projets et praticiens examinés et comment obtenir les informations nécessaires. Pour établir une liste préliminaire de projets et de praticiens, nous nous sommes appuyés sur le magazine AD. La revue AD, compte tenu de son histoire et de sa ligne éditoriale, nous semble en effet être une source particulièrement adaptée à la constitution du corpus initial. Le magazine Architectural Design (AD) a été fondé en 1930 et a depuis été continuellement édité, d'abord par Academy et ensuite par Wiley. 12 numéros par an ont été publiés jusqu'à ce que le magazine passe à 6 numéros par an en 1987, chaque numéro étant l'occasion pour AD d'inviter un rédacteur invité à se concentrer sur un sujet spécifique de la conception architecturale de l'époque. Le magazine AD représente depuis des décennies une référence clé pour de nombreux architectes, en particulier pour les développements technologiques de la discipline. Depuis les débuts du virage informatique en architecture, il est de plus reconnu dans l'ensemble comme une référence pour ce domaine spécifique. Le corpus étudié pour constituer la liste de projets est constitué de 452 numéros d'AD, du n°1 de 1965 au n°2 de 2020 (mars/avril), accessibles dans les bibliothèques de KADK (Copenhague), de l'ENSAPB (Paris) et de l'ENSAMV (Paris). Sur ces 452 numéros, 104 contenaient des articles présentant des projets, recherches, outils, réflexions en rapport avec les outils computationnels. 1398 projets cités dans les articles ont été identifiés et intégrés à la base de données pour une analyse plus approfondie.

L'établissement de cette première liste de travaux nous donne un premier ensemble de travaux sur lesquels nous pencher. Ceci nous permet d'obtenir une vue d'ensemble de l'évolution du champ computationnel tel qu'il est présenté par le magazine AD. Cette vision implique néanmoins un biais de sélection dû aux choix éditoriaux du magazine qui doit être pris en compte. À titre d'exemple, certains rédacteurs invités ont été invités pour plusieurs numéros, tandis que d'autres praticiens ayant contribué à la constitution du domaine n'ont jamais été invités. En outre, certains rédacteurs invités ont tendance à présenter une grande partie de leur propre travail ou des travaux de proches collaborateurs, ce qui fausse également la vue d'ensemble donnée par AD[124]. Par ailleurs, AD n'a commencé à s'intéresser de très près aux pratiques computationnelles et à les publier régulièrement qu'au début des années 2000. Ceci biaise la chronologie en fournissant beaucoup moins d'exemples des pratiques antérieures.

[124] Les biais éditoriaux d'AD sont abordés plus en détail au chapitre IV.

Afin de contourner ce biais éditorial, le corpus a été complété par une liste de projets présentés dans diverses expositions dédiées au sujet, essentiellement au cours des années 2000, dont nous avons pu consulter les catalogues[125]. Afin de compléter le corpus, notamment sur les périodes pour lesquelles ces sources ne permettent pas d'obtenir une vision claire des pratiques, nous nous sommes appuyés sur des regroupements d'articles scientifiques publiés par les praticiens du champ : revues, annales de conférences et bases de données. Concernant les annales de conférences, nous nous sommes appuyés particulièrement sur celles du Design Modelling Symposium[126] et de la conférence Smart Geometry[127], ainsi que sur la base de données Cumulative Index about publications in Computer Aided Architectural Design (CUMINCAD), qui regroupe les annales des conférences ACADIA, CAADRIA, eCAADe, SIGraDi, ASCAAD and CAAD futures[128]. Ces conférences étant les principales du champ, ces annales permettent d'en observer les pratiques de manière globale. CUMINCAD liste des publications dans le champ computationnel à partir des années 1960, et nous a donc notamment permis de compléter le corpus pour les périodes antérieures à celles documentées en détail par AD. Les annales de l'Association for Computer Machinery et de la revue Design Studies, ou nombre des praticiens des premières années du champ computationnel ont publié avant que des revues et des conférences dédiées à leur domaine ne se créent, ont également été utilisées.

Un exemple type des informations recherchées pour les différents projets est donné dans le tableau 1. Il s'agit du tableau des informations exploitées pour le cas d'étude du concours du Pavillon Seroussi, discuté en détail au chapitre IV. Le recours aux articles scientifiques a permis de compléter en partie les informations recherchées au sujet de certains projets, mais aussi d'approfondir la recherche concernant d'autres éléments ponctuels. La période la plus tardive du champ n'a pu être examinée tout à fait de la même manière que les autres en raison du très grand nombre de praticiens qui en font partie. Établir le réseau des praticiens en activité et examiner les projets produits par ce groupe de praticiens fonctionne jusqu'à la fin des années 2000. La sortie du noyau observée ensuite, que nous

[125] D'autres sources secondaires diverses ont également servi ponctuellement à compléter cette liste - en particulier des recueils de projets et d'articles comme ceux mentionnés en partie 1.2.1.

[126] Christoph Gengnagel, Olivier Baverel, Jane Burry, Mette Ramsgaard Thomsen et Stefan Weinzierl (eds.), *Impact: Design With All Senses: Proceedings of the Design Modelling Symposium, Berlin 2019,* Springer, 2020. Klaas De Rycke, Christoph Gengnagel, Olivier Baverel, Jane Burry, Caitlin Mueller, Minh Man Nguyen, Philippe Rahm et Mette Ramsgaard Thomsen (eds), *Humanizing Digital Reality: Design Modelling Symposium Paris 2017* Springer, 2018. Mette Ramsgaard Thomsen, Martin Tamke, Christoph Gengnagel, Billie Faircloth et Fabian Scheurer (eds), *Modelling Behaviour. Design Modelling Symposium 2015*, Springer, 2015. Christoph Gengnagel, Axel Kilian, Julien Nembrini et Fabian Scheurer (eds), *Rethinking Protoyping. Proceedings of the Design Modelling Symposium Berlin 2013,* epubli, 2013. Christoph Gengnagel, Axel Kilian, Norbert Palz et Fabian Scheurer (eds), *Computational Design Modeling: Proceedings of the Design Modeling Symposium Berlin 2011,* Springer, 2011.

[127] Neuf éditions depuis 2008. https://www.smartgeometry.org/, consulté le 30 Novembre 2021.

[128] http://papers.cumincad.org/, consulté le 30 Novembre 2021. Association for Computer Aided Design In Architecture (ACADIA), conférences annuelles depuis 1981. Computer Aided Architectural Design Research In Asia (CAADRIA), conférences annuelles depuis 1996. Education and Research in Computer Aided Architectural Design in Europe (eCAADe), conférences annuelles depuis 1983. Sociedad Iberoamericana de Gráfica Digital (SIGraDi), conférences annuelles depuis 1997. Arab Society of Computer Aided Architectural Design (ASCAAD), neuf éditions depuis 2005. CAAD futures, conférences bi-annuelles depuis 1985.

racontons en détail dans le chapitre VI, implique une grande augmentation du nombre de praticiens, qui rend difficile l'application de la même méthode. Les bases d'articles ont donc alors servi pour des recherches statistiques permettant d'obtenir une vue d'ensemble de certains points, notamment l'évolution chronologique de l'exploration de certains sujets et de certaines familles d'algorithmes. AD ayant aussi servi pour repérer des essais écrits à intégrer à l'état de l'art et à utiliser pour observer l'usage de certains mots-clés et références, les bases d'articles ont également servi pour compléter cette observation. Ce travail sur le discours et les références vise à comprendre, en complément des pratiques du champ computationnel, comment les praticiens les discutent.

| | | Practices | | | | | |
|---|---|---|---|---|---|---|---|
| | | biothing | labDORA | EZCT Architecture & Design Research | IJP Corporation | Gramazio & Kohler | Xefirotarch |
| Algorithm | Family | Formal Research | Performance-driven design | Performance-driven design | Formal Research | Fabrication-driven design | Formal Research |
| | Type | Attractors (agent-based; magnetic fields) | Finite Element Analysis, Particle System Optimization | Genetic Algorithm | Periodic Equation | Robotic Brick Placing | Growth |
| | Whole or Split | Whole | Split / Not related | Split / Related | Whole | Split / Related ?? | Whole ?? |
| | Main Technical Objective | Global Shape Generation | Structural Optimisation | Light Optimisation | Global Shape Generation | Detailed Shape Generation | Global Shape Generation |
| | Secondary Technical Objectives | / | Global Shape Generation, Detailed Shape Generation | Structural Optimisation, Global Shape Generation, Detailed Shape Generation | Mathematical Sobriety | Fabrication files generation | / |
| | Output | "Flower" Shells (Roofing) | Shell Shape and Structural Elements | Main Shell Shape, Structural Pattern for Surface Subdivision | Modules | Wall Shape and Bricks Placement | ? |
| | Significant Variables | Position of Attractors | Double Shell Structure | Shape of retrieved blocks | Periodic Equation / Cylinder | Robotic Placing Constraint | ? |
| Technical Set Up | Software | Rhinoceros + FlowerPower | Rhinoceros, Finite Element Analysis Software | Rhinoceros, Mathematica, Radiance, Maya, Blender, VTK, Polytrans, Getfem++, 3Dmax, Vray, EO, Catia | MathCAD | ? | ? |
| | Programmation Language | C#* | C#* | Python, C++, Wolfram | Mathematics | ? | ? |
| | Interface type | Plug-in with slider parameters | / | Scripting | Scripting | ? | ? |
| | Graphic Representation Tool for Base Geometry Visualisation | Rhinoceros | Rhinoceros | Rhinoceros | AutoCAD | ? | ? |
| | Final Output Type | 3D + 2D Drawings | 2D Drawings | 3D + 2D Drawings | 2D Drawings | 3D + 2D Drawings + Fabrication File | 3D + 2D Drawings |
| Chart | Date of Office Creation | 2001 | 2001 | 2000 | 2000 | 2003 | 2001 |
| | Number of people (Team + Consultants) | 7 + 1 | 17 + 7 | 11 | 10 | 12 | 10 |
| | Background of principal | Architecture | Architecture | Architecture | Architecture | Architecture | Architecture |
| | Background of team | Architecture, Computer Science | Architecture | Architecture, Computer Science, Mathematics, Daylight Modelling, Structural Engineering, Visual Rendering | Architecture | Architecture | Architecture |
| | Background of Consultants | Mathematics, Visual Rendering | Architecture, Computational Design, Structural and Environmental Design | / | Landscape Design | Structural Engineering | / |
| Architectural Design | Drawn Architectural Design independant of the Algorithmic Output | Yes (plan)* | Yes (structural principle) | Yes (plan)* | Yes (plan) | Yes (plan) | |
| | Site-Related items (verbal mention, written mention, site plan, sculptures present on drawings) | 2 | 2 | 4 | 3 | 3 | 4 |
| | Brief compliance (program) | Partial | No | Partial | Yes | Partial | Partial |
| Available documents | Site Plan | Y | Y | X | Y | Y | Y |
| | Ground Plan | Y | X | Y | Y | Y | Y |
| | Drawings (Sections, Renders) | Y | Y | Y | Y | Y | Y |
| | Code | X | X | X | Y | X | X |
| | Diagram/Illustration of algorithmic method | Y | Y | Y | Y | X | Y |
| | Credits | Y | Y | Y | Y | Y | Y |
| | Interviews | Y | Y | Y (informal) | Y | Scheduled 15/01/19 | X |

*Tableau 1. Éléments d'analyse pour les projets du concours Seroussi*[129]

Concernant les praticiens mentionnés au fil des numéros d'AD et des catalogues, en raison de leur grand nombre, nous avons opéré une sélection en retenant ceux cités plus de trois fois. Pour chacun

[129] Voir le récit fait de ce concours au chapitre IV.

d'entre eux, nous avons examiné leur portfolio[130] pour repérer d'autres projets à intégrer au corpus. Cet examen a parfois servi à disqualifier des praticiens dont les travaux sont prisés du champ computationnel et donc souvent cités, mais dont le travail n'en relève pas directement. Pour retracer la carrière de ces praticiens, nous avons consulté des sources diverses : biographies dans des monographies et catalogues d'expositions, CV académiques, notices nécrologiques parfois. Le travail d'obtention des données sur le parcours des praticiens a été marqué par peu de problèmes. Hormis les praticiens des années 1960, la majeure partie des praticiens dont la trajectoire a été examinée sont vivants et ont donc pu être contactés en cas de besoin. Leurs carrières récentes ont de plus laissé beaucoup de traces de leurs trajectoires qui ont pu être consultées. Si les sources secondaires permettent dans l'ensemble de bien reconstituer la carrière des praticiens, elles ne disent néanmoins rien du détail des interactions et de l'investissement dans les projets.

Concernant les projets également, la base de données fournie par la consultation de cet ensemble de sources secondaires peut être très lacunaire. L'établissement de la liste des projets et praticiens a été satisfaisant par le biais de cette méthode de constitution du corpus. Mais obtenir l'ensemble des données souhaitées pour chacun des projets examinés est difficile sur la base de sources souvent parcellaires à ce sujet. En complément d'une description de quelques lignes mentionnant tout juste la famille d'algorithmes employée et l'objectif du projet, le projet est montré en quelques images (parfois juste une seule). Ces images sont les images usuelles du projet : perspectives, coupes, axonométries. Elles sont souvent assorties d'un schéma concernant le processus algorithmique mis en œuvre. Ce schéma est extrêmement variable. Dans le cadre des projets de la série Genware de biothing[131] par exemple, tous les projets sont assortis du même schéma sur la bibliothèque algorithmique employée. Cela donne des informations sur la structure de la bibliothèque et les différents algorithmes qui la composent, mais rien sur l'assemblage d'algorithmes précis employés pour un projet donné. Dans certains cas, un schéma expliquant les relations géométriques ou le processus de fabrication numérique est disponible, ce qui ne donne pas non plus d'informations précises sur les algorithmes employés. Autres possibilités, des extraits de code ou des schémas du flux de travail, qui donnent des informations sur les différents outils logiciels employés mais par forcément sur la logique algorithmique mise en place.

Le problème rencontré dans ces sources secondaires est celui que nous avons déjà souligné plus haut du manque d'informations techniques. Malheureusement c'est un problème qu'on retrouve fréquemment non seulement dans les magazines et les catalogues d'expositions, mais aussi dans les articles scientifiques publiés dans les bases de données consultées. Plus exactement, c'est le cas pour les articles scientifiques récents. Jusque dans les années 1990, les articles incluent beaucoup de détails

[130] En consultant ouvrages publiés et sites web.
[131] Plus de détails aux chapitres IV et V.

sur les algorithmes employés. Ceux-ci sont transcrits au moins en partie et au moins en pseudocode[132]. C'est une chance notamment parce que cela concerne souvent les articles des praticiens les plus vieux, avec qui nous n'avons pas pu parler. Cela nous fournit donc des détails sur leur travail. Les détails sur les algorithmes employés disparaissent cependant ensuite. Même le set-up technique, c'est-à-dire les machines, les logiciels, les langages et les environnements de programmation employés, ne sont mentionnés que dans certains cas. Il y a plusieurs raisons à cela. Les articles de conférence sont des formats trop courts[133] pour pouvoir vraiment détailler ces aspects techniques tout en commentant sur les applications de l'outil développé et en replaçant le travail fourni dans son contexte. Les algorithmes deviennent par ailleurs trop longs pour être retranscrits dans des articles quel que soit leur format : des centaines ou des milliers de lignes de code. Par ailleurs, les praticiens ne semblent plus considérer qu'intégrer des extraits de code ou de pseudo-code fasse sens pour expliquer l'algorithme utilisé. Il s'agit d'un changement interprétable comme un signe de la mue du réseau, ce que nous allons voir en détail au chapitre VI, il existe aussi des raisons techniques à cette volte-face. En raison de la longueur des algorithmes, en montrer seulement des extraits peut ne pas être très adéquat. Néanmoins certaines fonctions précises pourraient pourtant s'avérer pertinentes. Mais les set-ups techniques sont néanmoins de plus en plus souvent simplement des outils avec interface ou des assemblages de fonctions classiques, déjà décrits ailleurs dans la littérature. Pour peu qu'on précise y avoir recours, pas besoin de les retranscrire - même si cette précision n'est pas toujours là justement. Toujours est-il que ces sources fournissent souvent trop peu d'informations sur le cadre de réalisation, sur les algorithmes et les outils utilisés, sur la logique de projet et sur le processus de conception par rapport à nos attentes. Pour pallier ce manque, nous avons choisi de mener des entretiens avec les praticiens auteurs de ces projets.

Ces entretiens ont eu pour objectif autant de vérifier certaines informations des sources secondaires et d'obtenir des informations complémentaires sur les trajectoires professionnelles et les projets que d'interroger les praticiens sur leur approche de la conception computationnelle de manière générale. Nous avons donc déterminé d'une part des questions précises sur les projets et la carrière, visant à compléter la base données, et d'autre part des questions ouvertes[134] très larges, ayant vocation à faire parler les praticiens librement de leur recours aux algorithmes. Une interview type comporte donc des questions comme :

[132] Le pseudocode est la description des étapes de programmation en langage naturel (c'est-à-dire en langage parlé ou écrit usuel).
[133] Généralement de 8 à 10 pages, bibliographie et résumé inclus.
[134] Pour la méthodologie des entretiens, nous nous sommes appuyés sur les conseils fournis dans le livre de Florence Weber et Stéphane Beau *Guide de l'enquête de terrain: produire et analyser des données ethnographiques*, La Découverte, 1997.

- *Comment en êtes-vous venu à vous investir dans les pratiques computationnelles en architecture ?*
  *Pouvez-vous me raconter votre trajectoire professionnelle en détail ?*
- *Concernant le projet X, pouvez-vous en raconter les conditions d'accès à la commande, le déroulé du travail, la constitution de l'équipe et les rôles de chacun ? Pouvez-vous détailler le fonctionnement de l'algorithme utilisé, me montrer le code ?*
- *Pouvez-vous décrire le déroulé type du développement d'un projet et la place occupée de l'écriture d'algorithmes sur-mesure pour le projet ?*
- *Comment intégrez-vous les outils algorithmiques utilisés dans votre pratique quotidienne à votre enseignement ?*
- *Comment construisez-vous la relation entre intention architecturale initiale et développement de la logique algorithmique ?*
- *Quelles particularités voyez-vous dans les pratiques computationnelles en architecture ?*

Les interrogations sur des projets en particulier, repérés en amont et souvent la raison du choix de ces praticiens pour conduire les entretiens, ont permis d'obtenir les informations souhaitées. Elles ont souvent également mené à la description d'autres projets qui paraissaient pertinents aux praticiens, et ont permis de réévaluer l'intérêt de différents projets comme cas d'études sur certains enjeux. Les questions plus larges ont visé à questionner la nature tacite des pratiques de programmation en architecture, les habitudes de travail, mais aussi le niveau de maîtrise de code des différents praticiens et leur relation aux outils en fonction de cela. Les hypothèses étudiées au moment de la réalisation des entretiens, qui ont eu lieu tôt au cours du travail de doctorat, avaient essentiellement pour objectif de saisir la connexion entre usage des algorithmes et projet architectural chez les praticiens. L'un des objectifs de la partie plus ouverte des entretiens était de faire parler les praticiens de ce qu'ils trouvaient de particulier à la conception architecturale appuyée sur le recours à la programmation[135], et ainsi de chercher à comprendre ce qui se cache derrière les affirmations souvent rencontrées que ces outils impliquent une révolution de la conception architecturale.D'autres hypothèses ont émergé plus tardivement, en particulier sur l'articulation de la pratiques des individus et des enjeux de l'industrie de la construction et du logiciel. Les questions étant formulées de manière particulièrement large, les entretiens ont également fourni matière à analyse sur ces enjeux. Enfin, il faut préciser qu'aux entretiens classiques s'ajoutent d'autres échanges avec des praticiens du champ. Les conversations exploitées dans le texte sont des échanges au sujet desquels les participants n'auraient vraisemblablement pas été aussi transparents si il s'était agit d'entretiens à proprement parler[136]. Ces conversations sont référencées comme des conversations personnelles, et la liste complète des praticiens interrogés est disponible en annexe.

[135] Ou à l'inverse, ce qu'ils ne trouvaient pas particulier.
[136] Notamment la franchise de certains développeurs et ingénieurs sur l'obsolescence des architectes

Enfin, les entretiens ont également été l'occasion de demander à accéder aux archives privées des praticiens concernant les projets discutés. Ceci a permis de consulter des images non publiés des projets, mais aussi le code de certains algorithmes, ces documents ayant toujours été sélectionnés par les praticiens puis mis à disposition. Néanmoins, les algorithmes se sont avérés un matériau difficile à consulter et à travailler. Consulter le code lui-même n'a pas toujours été possible, pour plusieurs raisons. Les praticiens, malgré les demandes, n'ont pas toujours fait l'effort de mettre ce matériau à disposition, peut-être en raison du format unique des entretiens, parfois réalisés à distance. Ensuite, même les praticiens de bonne volonté n'ont pas tous conservé le code original de projets anciens. L'ensemble des praticiens interrogés a montré beaucoup d'étonnement face à la requête de consultation du code original. Peu prisé comme matériau d'analyse ou d'exposition usuellement, le code n'est pas vraiment considéré comme digne de conservation. C'est d'autant plus le cas que les progrès dans la maîtrise de la programmation accomplis par les praticiens au fil des années rendent de vieux algorithmes obsolètes. Ils sont moins efficaces, moins élégants que ceux que les praticiens sont à même de coder dix ou quinze en plus tard. Les codes peuvent aussi devenir obsolètes techniquement très vite, en raison de la volatilité des objets numériques. Le recours à d'anciens langages de programmation, d'anciens logiciels ou d'anciens ordinateurs peut rendre compliqué la consultation et l'exécution d'un vieux programme. Les praticiens conservent rarement les anciennes machines. Ils conservent usuellement les données relatives au projet, y compris les fichiers de code, mais n'ont plus rien pour les lire.

En regard des informations initialement espérées, dont nous avons souligné qu'elles correspondent à la documentation des entrées, du code et des sorties, tout n'a pas toujours été obtenu. D'abord, il faut signaler que la lecture du processus de programmation comme un simple enchaînement d'entrées, d'exécution du code et de sorties est un idéal rarement atteint. C'est également un idéal de programmeur formé à l'usage d'environnements de programmation tardifs, qui ont l'avantage de rassembler de vastes bibliothèques de fonctions toutes disponibles au même endroit. Il y a dix ans ou plus, ces environnements n'existaient pas ou leurs bibliothèques étaient beaucoup plus réduites. Les programmes développés auparavant l'étaient à travers un bien plus grand nombre d'environnements et de logiciels différents. Ils constituent donc des patchworks d'entrées, de code exécuté, de sorties devenant des entrées pour un autre pan de code dans un autre environnement, et ainsi de suite. En complément du code lui-même, saisir la logique de ces flux de travail entre environnement de programmation s'est avéré extrêmement important. Ensuite, les entrées[137] sont souvent mises en valeur dans la documentation annexe au code, notamment quand elles varient, puisque cela permet de souligner les résultats de ces variations sur les sorties. Il est assez facile d'identifier les paramètres du

[137] En tout cas certaines d'entre elles, considérées comme importantes par le praticien parmi les différentes entrées existantes, potentiellement très nombreuses.

programme considéré comme clés par les praticiens puisque ces entrées sont visibles dans la documentation conservée et exposée du projet. Les sorties sont mises en valeur suivant les mêmes principes. La sortie finalisée est également aisée à consulter, puisqu'il s 'agit des images du projet lui-même. Pourtant, cette vision finalisée implique usuellement des transformations entre la sortie brute de l'algorithme et les images du projet. Opérations informatiques complémentaires, extraction des images, collages de données 2D ou 3D, dessin du projet à partir de sorties géométriques : le processus peut inclure de nombreuses opérations extérieures à l'algorithme principale, numériques ou analogiques. Les sorties à proprement parler de l'algorithme demandent donc de reconstituer la encore le flux de travail et d'enquêter parmi les documents disponibles pour identifier les composants du processus.

Le code en lui-même étant parfois difficile à consulter, il faut développer des stratégies alternatives pour contourner le problème. Pour comprendre un projet donné, parcourir les autres matériaux disponibles s'est avéré dans plusieurs cas permettre d'obtenir une vision claire du fonctionnement de l'algorithme utilisé. Extraits de code, pseudo-code, environnements de programmation et logiciels utilisés, diagrammes explicatifs, descriptifs textuels et audios, flux de travail permettent en les combinant de reconstituer les détails du processus algorithmique d'un projet en particulier. Au fil des recherches, un document en particulier nous a paru intéressant. De nombreux praticiens ont montré des esquisses de projet, non des esquisses spatiales comme nous avons l'habitude de voir chez les architectes, mais des esquisses algorithmiques. Mélanges de logigrammes et de croquis éclairant les sorties que les praticiens cherchent à obtenir, ces esquisses ont été précieuses pour saisir la logique d'aller retour entre intentions architecturales et écriture de l'algorithme. Pour comprendre les habitudes de programmation, parcourir d'autres codes en complément de ceux en lien avec des projets que nous avons pu consulter a permis de se faire une idée des manières de programmer des praticiens. Le livre Coders at Work[138], ainsi que plusieurs textes du mouvement des Artisans du logiciel[139] nous ont permis d'établir des points de repères sur les pratiques de programmation. De très nombreux exemples de code sont par ailleurs disponibles en ligne, dans la plupart des langages et des environnements de programmation dans le champ computationnel. Ces exemples, écrits par des amateurs ou des professionnels, forment une bibliothèque d'exemples riches qui permettent de se faire une idée des pratiques de programmation en vigueur dans le champ. Ce sont ces exemples qui nous ont permis de rendre compte du passage des algorithmes sur-mesure pour des projets particuliers vers le développement d'outils algorithmiques pensés pour être réutilisés dans des contextes différents, sur des projets variés, par un plus grand nombre de praticiens que juste leur développeur[140]. En effet, une partie du travail a été d'observer le lien entre algorithmes, programmes et logiciels dans le champ. Or

[138] Peter Seibel, *Coders at Work: Reflections on the Craft of Programming*, Apress, 2009.
[139] Pete McBreen, *Software Craftsmanship : The New Imperative*, Boston, Addison Wesley, 2001. Plus d'informations au Chapitre IV.
[140] Nous rendrons compte de cette transformation au chapitre VI.

consulter le code qui permet à un outil de fonctionner est bien plus facile que de consulter le code écrit pour un projet particulier par un praticien en particulier. Ces codes, mis à disposition dans le cadre de pratiques open-source, ont permis d'étoffer les exemples consultés.

Concernant la constitution du corpus, enfin, deux points doivent être mentionnés. Le premier est que la conception urbaine qui s'appuie sur le recours à des outils de programmation a été intégrée au corpus au même titre que les autres projets en ce qui concerne l'évaluation de l'évolution du recours à ces outils. Les projets d'urbanisme présents dans le corpus sont cependant des projets conçus par des praticiens déjà dans le corpus, qui sont usuellement formés comme architectes et évoluent en agence d'architecture ou dans des facultés d'architecture à l'université. Nous avons choisi d'utiliser ponctuellement quelques uns de ces projets comme exemples au cours du texte, parce que certains aspects les rendait particulièrement emblématiques du point discuté. Cependant les enjeux auxquels nous nous sommes intéressés au cours de la recherche n'ont pas mis en évidence de différence majeure entre les projets d'urbanisme relevés et les autres qui indiquent la nécessité d'un traitement différent. Nous n'avons pas non plus réservé de traitement particulier aux projets de fabrication robotisée. Comme les autres projets documentés, ils font appel à des outils de programmation pour générer les formes fabriquées. Puis une seconde partie du processus implique de recourir à des outils de programmation pour guider les machines. Or c'est surtout la première partie qui nous intéresse, et nous l'avons donc documentée comme pour les autres projets. Nous avons cherché à intégrer la question de la fabrication à notre réflexion d'ensemble sur le champ, et à interroger la place qu'elle occupe au fil des décennies en regard des dynamiques de développement générales observées. La fabrication numérique occupe une place prépondérante à deux moments de développement du champ, une prépondérance qui s'explique justement par des dynamiques d'ensemble. La place occupée par la question dans le texte reflète ces observations. Fabrication et urbanisme, que nous aurions pu choisir de traiter différemment, sont donc des parties du corpus comme les autres.

Les objets suivis l'ont été en Europe et dans le monde anglophone. Certaines périodes de développement du champ que nous avons traitées sont de plus presque uniquement focalisées sur le monde anglophone. Cette concentration s'explique par plusieurs éléments. Nombre des sources secondaires consultées sont elles-mêmes centrées sur la sphère anglophone, dont les recherches ont beaucoup de portée dans le monde contemporain. Mais la focalisation des sources s'explique par ailleurs aussi par le fait que les recherches dans les différents domaines abordés par la thèse - architecture computationnelle et informatique - sont particulièrement développées dans les pays anglophones. Les sources existantes sur l'histoire du domaine de l'intelligence artificielle s'accordent pour pointer la densité des recherches dans le domaine aux Etats-Unis, en comparaison d'autres

pays[141]. Ces sources donnent des raisons économiques, politiques et institutionnelles au fait que le développement ait été si soutenu la-bas. Ces mêmes raisons, et le fort développement de l'informatique et de l'intelligence artificielle qui précède celui du champ computationnel en architecture, font du monde anglophone un de ses épicentres. Au fil du travail, cette prédominance anglophone est devenue de plus en plus évidente, et nous nous sommes donc interrogés sur l'existence de pôles du champ computationnel alternatifs qui n'auraient pas été mis en évidence au cours de la constitution initiale du corpus. Nous avons découvert au fil des recherches quelques pistes qui pourraient indiquer l'existence de pôles précurseurs alternatifs, investiguant la place des outils computationnels dans la conception architecturale aussi tôt que leurs homologues américains décrits dans le texte. Par exemple, en Allemagne, l'HfG Ulm pourrait avoir accueilli des recherches beaucoup plus poussées que celles découvertes dans le présent travail, qui touchent rapidement à des questions de théorie du design plutôt que de conception algorithmique. Autre exemple, quelques sources font état de recherches sur la conception participative en Europe au cours des années 1960 et 1970, les citant en lien avec le travail de Yona Friedman. Or Yona Friedman s'est à cette période beaucoup intéressé à la conception algorithmique. Certaines de ces recherches européennes, très brièvement mentionnées par nos sources, pourraient avoir partagé ces intérêts. Néanmoins, le temps de cette recherche ne nous a pas permis d'aller plus loin dans l'examen de ces hypothèses. Elles sont néanmoins signalées en notes au fil du texte pour permettre au lecteur de les identifier.

### *1.4.3 Cartographies constituées*

A partir des données rassemblées au fil de la consultation de ces différentes sources, plusieurs outils d'analyse ont été mis en œuvre. Certains d'entre eux relèvent d'une analyse des données classiques, il ne nous semble donc pas nécessaire de détailler ici comment les données ont été travaillées dans ce cadre-là. Les chronologies constituées pour suivre l'évolution des différents objets étudiés sont disponibles sous forme de figures au fil des chapitres. Diverses analyses statistiques ont également été faites : la méthode de travail exacte a été précisée aux différents endroits du texte qui font appel aux résultats de ces analyses[142]. En complément de ces éléments, deux analyses parallèles ont été mises en œuvre pour observer les pratiques computationnelles en architecture et la constitution du champ qui les rassemble. La première est l'étude des réseaux de praticiens, et la seconde est un ensemble d'analyses portant sur les algorithmes et les outils algorithmiques employés par ces praticiens. Chacune de ces analyses se penche donc sur l'un des deux objets principaux suivis au cours de la

[141] Notamment les ouvrages de référence de Daniel Crevier et Pamela McCorduck. Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993. McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

[142] C'est le cas par exemple pour l'étude détaillée du magazine AD au chapitre IV, pour l'évolution chronologique des outils présentée au chapitre V ou encore pour l'étude des acquisitions d'autodesk au chapitre VI.

recherche. Le travail sur le réseau des praticiens se base d'abord sur une prosopographie, définie comme suit :

> *Une prosopographie pourrait être définie, a minima, comme une étude collective qui cherche à dégager les caractères communs d'un groupe d'acteurs historiques en se fondant sur l'observation systématique de leurs vies et de leurs parcours. Son ambition première est donc descriptive : il s'agit de rechercher la structure sociale d'un collectif par l'accumulation de données structurées sous la forme de fiches individuelles relatives à chacun de ses membres, avec l'objectif final d'en saisir la structure de groupe par-delà les discours qu'il produit.*[143]

Le réseau des praticiens est cartographié en documentant les liens entre les praticiens et les entreprises, les praticiens et les institutions - où chacun a étudié et enseigné - ainsi qu'en documentant quel projet a été créé par qui en utilisant quelle typologie, ce qui permet de cartographier les groupes de développement à travers le temps et les groupes de développement associés à l'usage d'algorithmes spécifiques. Les réseaux de praticiens représentés au cours de ce travail de cartographie le sont donc pour représenter le système socio-technique que constitue le champ computationnel en architecture. La sociologie des réseaux fait de la société un ensemble de relations d'interdépendance, des réseaux d'influence permettant d'expliquer les comportements individuels[144]. Dans notre cas, il s'agit plus simplement d'observer les relations entre praticiens, des relations qui se constituent notamment par les lieux géographiques, les institutions et les agences qui les accueillent ensemble. Nous supposons que l'observation de ces relations au fil du temps nous permettra de comprendre comment les pratiques computationnelles se sont développées. La cartographie de ces liens n'a pas tant vocation à expliquer le comportement des praticiens par le réseau de relations dans lequel ils s'insèrent qu'à retracer le développement chronologique du champ. Il ne s'agit pas non plus d'une analyse ancrée dans la théorie des acteurs réseaux développée par Madeleine Akrich, Bruno Latour ou Michel Callon[145]. Bien que l'idée de suivre des outils techniques au fil de leurs usages décrive très bien notre entreprise, nous n'avons pas voulu faire des algorithmes un élément à part entière de nos réseaux. Au contraire, nous verrons au fil du texte que les algorithmes et les ordinateurs sont souvent personnifiés. L'ombre de l'intelligence artificielle renforce cette idée d'une intentionnalité de la machine, une idée pourtant mise à mal par les réalités techniques. Nous souhaitions éviter cette manière de voir les choses, et avons donc adopté une méthode d'analyse qui nous en éloigne. Les réseaux qui nous intéressent sont

---

[143] DELPU Pierre-Marie, « La prosopographie, une ressource pour l'histoire sociale », Hypothèses, 2015, volume 18, n°1, p. 263-274. Nous remercions Claire Dupin de Beyssat pour sa suggestion et ses conseils concernant ce point méthodologique.

[144] Jacques Coenen-Huther, "Analyse de réseaux et sociologie générale", FLUX Cahiers scientifiques internationaux Réseaux et Territoires, Vol. 13-14, p. 33-40, 1993.

[145] Madeleine Akrich, Michel Callon et Bruno Latour (éd.), Sociologie de la traduction : textes fondateurs, Paris, Mines ParisTech, les Presses, « Sciences sociales », 2006. Textes rassemblés par le Centre de sociologie de l'innovation, laboratoire de sociologie de Mines ParisTech.

donc somme toute des réseaux très simples de relations entre praticiens, dont la vocation est de représenter le système socio-technique que constitue le champ computationnel en architecture. Ce système socio-technique, au sens défini par Madeleine Akrich ou Gilbert Simondon, est le réseau des acteurs tissé autour des algorithmes utilisés pour la conception architecturale et autour des outils logiciels qui permettent de recourir à ces algorithmes[146]. Par leur représentation, nous souhaitons obtenir une description du champ qui nous serve de base pour l'analyse des différentes stratégies de développement et pour l'analyse de la trajectoire des différents outils en jeu.

Le tracé des cartes a été effectué grâce au logiciel YED, en recourant aux méthodes dites de configuration organique et orthogonale[147], puis le graphisme retravaillé manuellement pour rendre les cartes plus lisibles. Ces méthodes de représentation des réseaux ont permis d'observer différents aspects. L'évolution chronologique du réseau s'est faite en établissant quatre phases distinctes et quatre cartes correspondantes. Les méthodes de configuration employées ont notamment permis de rendre visibles les différents pôles institutionnels présents à ces différentes phases. Ces pôles apparaissent à partir d'une étude des institutions et du voyage des praticiens de l'une à l'autre au fil du temps. Nous nous sommes de plus penchés sur la relation entre certains de ces pôles et les domaines voisins de celui de l'architecture computationnelle : théorie du design, intelligence artificielle, informatique, Ceci nous a permis de comprendre l'influence et les apports théoriques et pratiques de ces différents domaine, et d'identifier différentes écoles de pensée dans le champ qui nous occupe. De plus, comprendre les dynamiques du système socio-technique étudié nécessite de comprendre la place des différents milieux de pratiques à chaque période étudiée. Dans un second temps, les cartes ont donc été modifiées pour montrer, à partir du même réseau de relations, les professions des différents praticiens et la manière dont l'industrie du logiciel et de la construction, le monde universitaire, les agences d'architecture et les bureaux d'études techniques s'y rencontrent. Le travail sur ces cartes nous a également permis d'étudier la dimension de microcosme ou non du champ computationnel, et de saisir le développement du réseau au fil des années et comment l'origine des praticiens et leur milieu de pratique affecte l'évolution du champ.

Une fois le réseau des relations du champ identifié, cette cartographie a ensuite été exploitée pour représenter les sous-réseaux constitués autour d'outils algorithmiques particuliers. Nous avons jusqu'ici interrogé le recours aux algorithmes dans la conception architecturale de manière générale. Pourtant, il ne s'agit pas tout à fait de la même pratique selon qu'on observe les algorithmes sur-mesure écrits pour un projet en particulier, ou les éléments algorithmiques qui peuvent être

[146] Madeleine Akrich, « La construction d'un système socio-technique. Esquisse pour une anthropologie des techniques », Anthropologie et Sociétés, vol. 13, no 2, 1989. Gilbert Simondon, Du mode d'existence des objets techniques, Paris, Aubier, 1958; dernière réédition corrigée et augmentée, Paris, Flammarion, 2012.
[147] *organic layout* ou *orthogonal layout*
https://yed.yworks.com/support/manual/layout_smartorganic.html, consulté le 30 Novembre 2021.
https://yed.yworks.com/support/manual/layout_orthogonal.html, consulté le 30 Novembre 2021.

réutilisés d'un projet sur l'autre. Parmi les algorithmes sur mesure employés se trouve en effet des fonctions et d'assemblages de fonctions, des structures de code répétées d'un algorithme sur-mesure à l'autre. Lorsque ces structures sont formalisées, elles constituent des familles algorithmiques. Lorsqu'elles sont formalisées et combinées à une interface logicielle qui permet à l'utilisateur d'en modifier les paramètres sans toucher au code de la structure algorithmique, elles deviennent des outils algorithmiques[148]. Le travail d'identification des familles algorithmiques en usage dans le champ computationnel a été fait à partir des algorithmes sur-mesure recensés dans les différents projets. Cette catégorisation des algorithmes par familles a ensuite permis d'effectuer des études comparatives, notamment la trajectoire des différentes familles au sein du réseau, ou plus exactement les sous-réseaux d'usage de ces familles. Ce travail de cartographie en fonction des familles a mis en évidence des réseaux bien particuliers pour chacune, et a aussi permis de souligner la trajectoire chronologique de ces familles algorithmiques d'un pôle du réseau à d'autres, ou parfois l'arrêt de cette trajectoire. Ceci a fourni des informations précieuses sur le développement historique de ces familles d'algorithmes. La superposition de la présence des familles algorithmiques au réseau des praticiens montre de plus que le recours à certaines familles dans les pratiques individuelles des praticiens s'explique notamment par l'insertion dans certains pôles plutôt que d'autres. Enfin, ce travail de cartographie du réseau par familles d'algorithmes a permis de mettre en évidence des détails complémentaires sur la maîtrise du code et ses enjeux au sein des pratiques computationnelles.

Les algorithmes et les outils associés ne peuvent cependant pas uniquement être étudiés par le prisme de leur insertion dans un réseau de praticiens. Passée leur relation aux réseaux de praticiens se pose la question de l'analyse technique de ces outils. Or cette analyse se heurte à un premier problème : les algorithmes sont implémentés dans différents langages et environnements, qui peuvent varier drastiquement. Le même problème se pose pour les outils algorithmiques. Pour étudier certains points précis de leur usage, nous avons donc fait le choix de produire des diagrammes permettant de mettre ces différentes implémentations sur le même plan. Algorithmes et outils algorithmiques ont donc été examinés à travers une série d'analyses diagrammatiques. Ces analyses diagrammatiques ont vocation à mettre les algorithmes sur-mesure sur le même plan à travers une série de représentations communes. Une deuxième série de représentations est consacrée aux outils algorithmiques. Chacun de ces diagrammes a pour objectif de mettre les objets étudiés sur le même plan pour pouvoir les comparer concernant un point précis. En effet, les points analysés ne sont pas les mêmes en fonction de si on se penche sur les algorithmes ou sur les outils algorithmiques, néanmoins le problème rencontré est commun. Le recours aux diagrammes permet dans les deux cas de le résoudre, même s'il ne s'agit pas de diagrammes identiques.

[148] Nous décrivons en détail la forme de ces divers outils algorithmiques au chapitre V.

Concernant les algorithmes, qu'il faut donc documenter projet par projet, l'objectif est d'observer l'aller-retour entre le travail de conception architectural et la structuration du code pour mieux le comprendre. Nous supposons que les éléments clés à observer pour comprendre ces allers-retours sont les données manipulées et leurs transformations successives. Nous avons noté que les algorithmes sont compréhensibles comme un ensemble d'entrées, d'opérations et de sorties. Ces opérations sont le lieu de transformation des données. Informatiquement, ce sont des données numériques transformées en d'autres données numériques. Mais parmi les types de données manipulées par les praticiens au cours du processus complet se trouvent aussi des données textuelles et géométriques - visuelles. Le processus algorithmique est un travail de structuration de ces transformations successives, avec des points d'observation de ces données. L'enjeu est donc de comprendre comment ces données sont transformées, mais aussi sous quelle forme elles sont observées par les praticiens à différents stades. Pour ce faire, les logigrammes nous ont paru être une forme de diagramme particulièrement adéquate. Les logigrammes sont des organigrammes de programmation suivant une charte graphique prédéterminée, qui ont pour objectif de formaliser la représentation de l'enchaînement des opérations au sein d'un programme[149]. Ils sont particulièrement utilisés en ingénierie des procédés et en ingénierie mécanique, ou ils permettent de formaliser le fonctionnement d'une machine avant sa fabrication. Nous avons donc produit des logigrammes pour les différents projets étudiés, qui permettent de mieux en comprendre et d'en comparer la structure algorithmique[150]. Nous avons également souligné l'importance des flux de travail et des différents environnements, langages, outils de programmation constituant le set-up technique. Nous nous sommes également appuyés sur des diagrammes permettant de reconstituer ces flux de travail et de les comparer, en nous basant soit sur les souvenirs des praticiens partagés au cours des entretiens, soit sur des documents existants représentant déjà ces flux de travail.

D'autres diagrammes d'analyse des outils algorithmiques ont été réalisés pour mieux comprendre leur utilisation par les différents usagers du champ. Observer les outils plutôt que les algorithmes sur-mesure décale l'observation des projets individuels au réseau des praticiens et des institutions. Puisqu'il s'agit de comparer les différentes familles d'algorithmes, les objectifs pour lesquelles elles sont mobilisées, et les praticiens qui les utilisent, des cartes du réseau pour les différents types d'outils ont donc été réalisées. Elles montrent les communautés de praticiens derrière les outils. L'interaction avec les outils passe par l'interface, que nous avons donc voulu examiner aussi. Or, que ce soit les

[149] Ravi K. Walia et Y. S. Parmar, *Algorithm & Flowchart Manual*, Computer and Instrumentation Center, University of Horticulture & Forestry, Nauni Solan, Inde.

[150] Par ailleurs, nous avons également tenté de documenter une autre sorte de transformation, pour saisir l'interaction entre les différentes formes de savoir qui nous intéressent. Nous avons exploré différentes manières de créer des sortes de logigrammes alternatifs, montrant la transformation du savoir architectural vers la formalisation du code. Ces expérimentations n'ont pas rencontré un grand succès, et nous ne présentons donc pas les cartes correspondantes dans le travail, mais elles ont néanmoins participé à la réflexion autour des savoirs en jeu.

interfaces elles-mêmes ou les flux de travail dans leur ensemble, l'interaction avec les outils algorithmiques du champ computationnel est caractérisée par la forêt de couches techniques qui se superposent. En complément des réseaux, nous avons donc mis au point une représentation en couches pour montrer les différents niveaux d'interaction d'un utilisateur avec son outil. Cette logique d'évaluation de l'interaction est tirée d'études des interfaces dans le domaine de l'expérience utilisateur, et plus particulièrement de la notion d'épaisseur des interfaces développée par Anthony Masure, qui vise à qualifier l'importance prise par l'interface dans le dispositif technique et son influence sur le recours aux outils algorithmiques[151].

Le point essentiel que nous cherchons à documenter à travers ces analyses diagrammatiques est l'interaction entre le praticien et son outil. Ces diagrammes ont donc pour ambition d'interroger la place de l'utilisateur au sein du set-up technique. Il s'agit d'observer la relation entretenue par le praticien à l'outil en fonction des caractéristiques de cet outil. Cette relation varie en fonction des connaissances et des ambitions des praticiens. Nous cherchons à observer comment l'utilisateur interagit avec les outils algorithmiques en fonction de tout ceci. En croisant les observations sur les praticiens, leurs parcours et leurs connaissances et les observations sur les outils algorithmiques, nous souhaitons interroger la mobilisation des outils par leurs utilisateurs et leurs développeurs. En fonction de leurs connaissances et de leurs rôles, la maîtrise des outils, la capacité à les exploiter dans un objectif donné varie. Puisqu'il s'agit d'examiner comment l'automatisation se produit dans le champ computationnel, il faut donc examiner les outils au regard de leurs utilisateurs. Comprendre quel contrôle de leurs outils ont les praticiens respectifs impliqués dans un processus de conception algorithmique permet d'interroger les biais techniques qui se glissent dans le développement et l'usage de ces outils. De nombreux travaux ont souligné le besoin d'études en détail des algorithmes en raison de leur impact croissant sur nos vies, et en raison des biais qu'ils induisent dans les tâches qu'ils effectuent ou aident à effectuer[152]. Saisir les dessous des algorithmes est aujourd'hui un enjeu crucial pour cerner leurs effets. L'usage d'outils algorithmiques en architecture n'étant pas encore totalement généralisé, ce constat y est rarement fait. Il n'en est pas moins nécessaire. La somme des diagrammes et des cartes réalisées a de ce fait pour objectif d'interroger les biais sociologiques, techniques, épistémologiques du champ computationnel en architecture au fil de son développement.

*L'expérience du praticien comme outil méthodologique*

Enfin, nous souhaitons aborder un dernier point méthodologique, celui du rôle joué par l'expérience du praticien dans l'analyse. Nous avons indiqué les éléments sur lesquels se basent l'analyse proposé

[151] Masure, Anthony. 2014. Le design des programmes : des façons de faire du numérique, Ph.D. Thesis, Université Paris-1 Panthéon Sorbonne.
[152] À ce sujet, on peut consulter le travail de l'IA Now Institute de New York, qui couvre dans ses rapports de nombreux exemples de ces biais. https://ainowinstitute.org/, consulté le 30 Novembre 2021.

du champ computationnel, en particulier l'ambition de consulter les algorithmes eux-mêmes, qui constituent l'objet de l'explicitation dans le cadre d'une pratique de conception architecturale qui repose sur des savoirs tacites. Nous avons indiqué quelles données ont été collectées et quelles cartographies ont été tracées pour visualiser l'évolution des outils algorithmiques et les différents aspects de cette explicitation. Mais en complément de ces objets d'études, un dernier élément a été particulièrement utile à l'analyse : l'expérience du praticien. S'appuyer sur une formation d'architecte permet en effet un regard informé sur la conception architecturale, sur les intentions des concepteurs et sur les espaces produits. La pratique de l'enseignement de l'architecture comme de la programmation donne une meilleure idée des enjeux de la transmission des savoirs dans ces disciplines. S'appuyer sur une formation à la programmation et sur la pratique de la programmation permet de pouvoir lire le code lui-même, d'évaluer les alternatives possibles, d'émettre des hypothèses pour combler les lacunes en cas de besoin. Enfin, conjuguer les deux permet de mieux saisir la traduction à l'œuvre dans les pratiques architecturales computationnelles. Les historiens et philosophes des sciences ont établi de longue date l'importance que peut revêtir la connaissance en tant que praticien de la discipline étudiée.  H. Otto Sibum en particulier a souligné l'importance de l'expérience dans la compréhension du geste scientifique, notamment dans le texte *From the Library to the Laboratory and Back Again: Experiment as a Tool for Historians of Science*, écrit avec Hjalmar Fors et Laurence M. Principe[153]. Nous rejoignons ici ce postulat méthodologique : avoir étudié l'architecture, appris à coder, développé des projets d'architecture computationnelle et enseigné l'apprentissage de ces pratiques nous a permis d'acquérir un regard précieux pour l'exploitation des matériaux rassemblés.

[153] Hjalmar Fors, Lawrence M. Principe & H. Otto Sibum, "From the Library to the Laboratory and Back Again: Experiment as a Tool for Historians of Science", *Ambix*, 63:2, 85-97, 2016.

# CHAPITRE II.

-

# Une filiation avec le monde de l'intelligence artificielle

## 2.1 Les prémisses d'un écosystème de recherche

### *2.1.1 Cambridge, Massachusetts*

Les premières réflexions sur le rôle de la programmation informatique dans la conception architecturale se développent avec l'apparition dans les années 1960 des recherches dans le domaine de l'intelligence artificielle[1] (IA). L'Architecture Machine Group (ArchMach[2]), fondé en 1967 par Nicholas Negroponte et Leon Groisser au Massachusetts Institute of Technology (MIT), compte parmi les premiers groupes de recherche dédiés à la question de l'architecture computationnelle. Il se différencie peu des autres groupes de recherche du MIT qui se penchent sur la question informatique. Multidisciplinaire, le groupe accueille cependant plus d'informaticiens que d'architectes, que ce soit parmi ses membres permanents ou ses étudiants. L'ArchMach est essentiellement financé par la DARPA[3] au cours de ses premières années d'existence. L'ambition initiale du groupe, formulée par Negroponte, est de mettre les outils informatiques au service des architectes en vue de simplifier leur travail de conception. Les outils informatiques ne peuvent cependant selon Negroponte fournir cette aide aux architectes qu'à travers une forte interactivité entre les deux entités[4]. A cette période pionnière, étant donné l'état de l'art embryonnaire, les recherches dans le champ de l'IA tiennent plus de recherches sur les outils informatiques eux-mêmes. Les projets de l'Architecture Machine Group ne font pas exception et portent plus souvent sur des questions d'interface entre humain et ordinateur que sur des enjeux architecturaux. Se pencher sur l'interface qui permet aux utilisateurs d'interagir avec les machines est nécessaire pour donner naissance aux conditions posées par Negroponte, puisqu'elles n'existent pas encore. Le projet URBAN 5 notamment, développé en 1973, explore la possibilité de converser avec un système informatique au sujet de la conception d'un projet en cours. Le programme est assimilé à un assistant qui documente les étapes du travail. Le système propose deux types d'interactions : le dessin d'espaces à travers l'assemblage d'unités cubiques et des séries de questions qui permettent d'attribuer des qualités aux espaces créés, de documenter la prise de

[1] J'utilise dans ce chapitre l'expression "intelligence artificielle" pour désigner à la fois le champ de recherches qui s'y consacre, et ce que ce champ s'efforce d'obtenir : des machines intelligentes.
[2] Diminutif du nom du groupe utilisé par ses membres et leurs collègues du domaine.
[3] Defense Advanced Research Project Agency.
[4] Negroponte, N., "Towards a Theory of Architecture Machines", in *Journal of Architectural Education*, Vol. 23 n. 2, p. 9-12, 1969.

décision des utilisateurs et de suggérer des modifications[5]. S'il s'agit bien d'observer le processus de conception architecturale, la majeure partie du travail accompli pour URBAN 5 ne porte pas sur la mise en place d'une démarche procédurale, mais sur le développement d'interfaces qui offrent la possibilité de numériser le processus de conception classique. Ces recherches sont aujourd'hui considérées comme les prémices de ce qu'on appelle maintenant la conception assistée par ordinateur, ou CAO. URBAN 5 fait également partie des programmes pionniers de l'interaction verbale avec un ordinateur. Le projet possède ainsi plus de similarités avec ELIZA, un célèbre programme conversationnel[6], qu'avec un quelconque autre projet d'architecture de la même période.

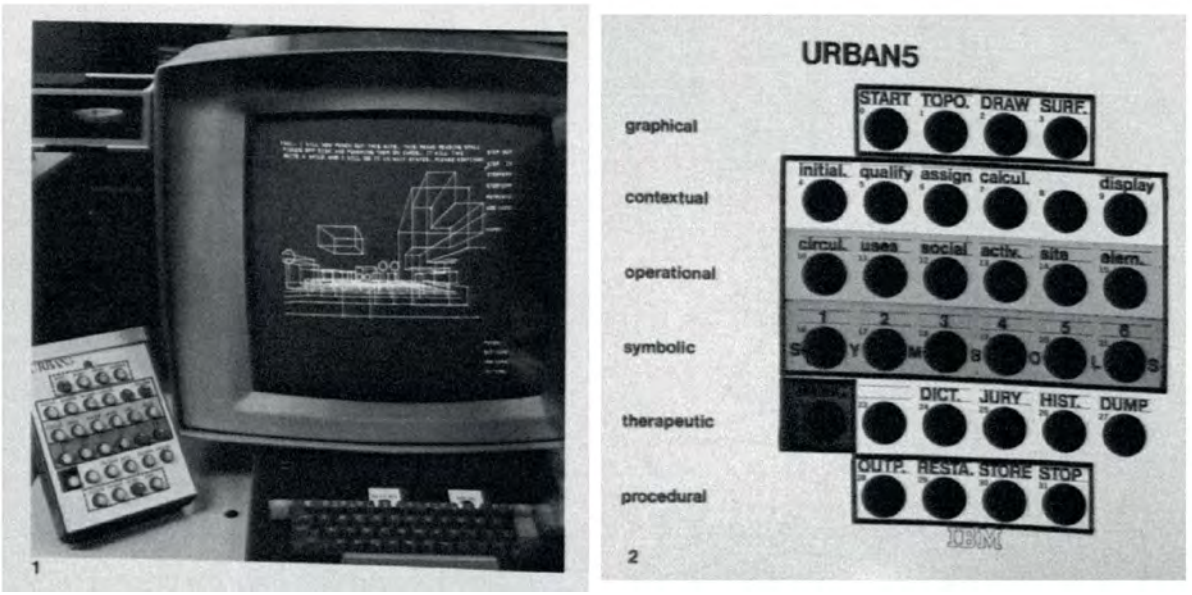



*Figure 1. Le programme URBAN 5 et son clavier d'attribution de qualités aux espaces conçus.*

En 1985, l'Architecture Machine Group est intégré dans ce qui devient le Medialab. La nouvelle structure rassemble en un seul l'ensemble des groupes de recherche sur l'IA préexistants au MIT. Il faut ensuite attendre une dizaine d'années avant que de nouveaux groupes dédiés spécifiquement à la question de l'architecture computationnelle soient créés au sein du Medialab. Au moment de cette fusion, les travaux de l'Architecture Machine Group n'ont plus grand chose à voir avec l'architecture[7] . L'ArchMach a acquit au cours de ses vingt ans d'existence une influence significative dans le monde de l'informatique et de l'intelligence artificielle, notamment grâce à ses apports dans le domaine des

[5] Negroponte, N., *The Architecture Machine*, The MIT Press, 1970. Steenson, M., "Architectures of Information: Christopher Alexander, Cedric Price & Nicholas Negroponte", thèse de doctorat, Princeton University, 2013,
[6] ELIZA est un programme de psychothérapie par la conversation développé par Joseph Weizenbaum au MIT entre 1964 et 1966, ayant pour objectif d'aider les personnes dépressives à discuter de leurs problèmes, et un des tout premiers projets de communication avec un ordinateur en langage naturel. Voir Weizenbaum, J., "ELIZA—a computer program for the study of natural language communication between man and machine" in *Communications of the ACM*, Vol. 9 n. 1, pp. 36-35, 1966.
[7] Brand, S., *The Medialab : Inventing the Future at MIT*, Penguin Books, 1987.

échanges verbaux avec un ordinateur et dans le domaine du data management[8]. Au moment de son intégration dans le Medialab, l'essentiel de l'activité du groupe se concentre sur le développement de systèmes média divers comme la Media Room ou l'Aspen Movie Map[9]. Entre 1967 et 1985 émerge cependant au sein et autour de l'Architecture Machine Group un noyau dur de praticiens qui partagent l'intérêt exprimé dans ses textes par Nicholas Negroponte pour le développement de ce que ce dernier nomme des *machines à architecture*. Ces machines à architecture sont des ordinateurs dont il a l'ambition qu'ils deviennent des assistants à la conception architecturale. Ces intelligences artificielles doivent être capables d'assister des architectes comme des habitants non-sachants dans leurs travaux. Les machines à architecture doivent pouvoir ajuster leurs suggestions de conception à tous les détails du contexte du projet, qu'il s'agisse des caractéristiques géographiques ou économiques ou les préférences formelles du concepteur. Negroponte imagine aussi ces machines comme capables de gérer le fonctionnement quotidien des bâtiments de manière automatisée et d'en réguler tous les aspects pour y accueillir au mieux les usagers. Ces machines à architecture sont donc la matérialisation d'une automatisation de tous les aspects de l'architecture, permise par les progrès informatiques que Nicholas Negroponte voyait déjà se déployer au moment de la fondation de l'Architecture Machine Group. Des collaborations de long terme se nouent entre les praticiens membres du groupe et donnent lieu à des projets devenus par la suite emblématiques. Ainsi le projet Seek[10], développé entre 1969 et 1970 par Negroponte, Groisser et un groupe d'étudiants[11], a pour objectif d'adapter le plan d'une ville en fonction du comportement de ses usagers, déplaçant les lieux aux endroits les plus pertinents en regard des allées et venues[12]. Un prototype du système est construit, où des gerbilles hébergées dans une grande cage en verre font office d'usagers. Elles évoluent au milieu de piles de blocs métalliques légers qui font office d'environnement urbain. En fonction des déplacements qu'elles causent dans les blocs qui forment la "ville" autour d'elles en les cognant lors de leurs mouvements, le système robotique qui les surplombe remet les blocs en place ou les déplace, adaptant le plan aux activités des gerbilles. La démonstration du projet faite au Jewish Museum à l'automne 1970 se solde par la rupture du portique. Les gerbilles qui errent au milieu des vestiges du système font ensuite l'objet de descriptions apocalyptiques dans la presse de l'époque[13]. Mais le

[8] L'architecture des systèmes informatiques doit en effet plus que son nom au champ de l'architecture.
[9] La Media Room est un système de navigation d'informations et d'interfaces informatiques multiples développée par le groupe entre 1976 et 1978. L'Aspen Movie Map est une visite touristique virtuelle de la ville d'Aspen, développée en 1978 par Andrew Lippmann, Peter Clay, Bob Mohl et Michael Naimark.
[10] Le projet est aussi connu sous le nom d'URBAN 2. Les appellations du projet varient selon les sources. Il semble que *URBAN 2* ait été le nom du projet dans son ensemble, et que Seek désigne plus spécifiquement l'installation mettant en scène des gerbilles réalisée pour le Jewish Museum en 1970.
[11] Randy Rettberg et Mike Titelbaum (Electrical Engineering) pour la robotique, Steven Gregory (Architecture and planning) pour la programmation, Steven Peters et Ernest Vincent pour la fabrication.
[12] En imaginant que les bâtiments soient conçus pour être facilement déplaçables ou reconfigurables.
[13] Dans un éditorial de *Art News*, le critique Thomas Hess fait une description dantesque des gerbilles livrées à une mort certaine au milieu des décombres du système, concluant par "*Artists who become seriously engaged in technological processes might remember what happened to four charming gerbils*." ("*Les artistes qui souhaitent*

groupe n'abandonne pas pour autant ses ambitions. De nombreux projets font suite à Seek et URBAN 5 : le FLATWRITER, mais aussi les programmes YONA, SQUINT, IMAGE, HUNT ou ARCHIT sont développés au cours des années 1970 et 1980. Tous explorent les mêmes enjeux du potentiel des machines à architecture imaginées par Negroponte.

*Pour autant que je sache, la seule chose ressemblant de près ou de loin à une machine à architecture qui ait été publiquement exposée par l'Architecture Machine Group de Negroponte donnait à des gerbilles le rôle des architectes. C'était à l'automne 1970 au Jewish Museum de New York. Une grande cage en verre contenant 500 blocs métalliques de deux pouces de côté et une colonie de gerbilles était surplombée par une machine à redresser commandée par ordinateur. Dans leur incessante activité, les gerbilles heurtaient légèrement les blocs au cours de leurs déplacements*[14].

S. Brand, 1987.

*Figure 2. La cage en verre, les blocs métalliques et les gerbilles du projet Seek.*

*se consacrer sérieusement à des procédés technologiques pourraient se rappeler ce qui est arrivé à quatre charmantes gerbilles*:") Hess,T., "Gerbils ex Machina," *Art News* (December, 1970), p. 2.

[14] "*So far as I know, the only thing resembling an architecture machine ever publicly demoed by Negroponte's Architecture Machine Group involved gerbils as the architects. It was in fall 1970 at the Jewish Museum in New York. A large glass case containing 500 two-inch metallized blocks and a colony of gerbils was watched over by a computer-driven straightening machine. As the restless gerbils went on about their business, they knocked the light blocks around a bit by their traffic.*
" Brand, S., *The Medialab : Inventing the Future at MIT*, Penguin Books, 1987

*2.1.2 Londres*

L'Architecture Machine Group est présenté dans de nombreux compte-rendus historiques des débuts de l'architecture computationnelle comme le premier groupe de recherche sur le sujet[15]. Il compte en effet parmi les plus renommés des groupes des années 60-70 qui s'y consacrent. Mais les travaux qui émergent au même moment en Grande-Bretagne au sein et autour de l'Architecture Association (AA) de Londres jouent un rôle tout aussi important. Au tout début des années 60, Cedric Price, architecte renommé pour ses idées provocantes et novatrices[16], se prend d'intérêt pour les questions computationnelles, et plus particulièrement pour la cybernétique[17]. Gordon Pask[18], cybernéticien lui aussi déjà renommé pour ses apports à ce champ, fait quant à lui preuve depuis longtemps d'intérêt pour l'art et l'architecture. Ceci les mènent à une collaboration célèbre : le projet du Fun Palace[19]. Si le Fun Palace, imaginé en 1961, n'inclut aucun programme informatique ou mise en pratique des principes développés, Pask propose tout de même un modèle de fonctionnement complet pour un théâtre cybernétique inclu dans le projet. Ce théâtre cybernétique prévoit la participation du public, dont l'interaction avec le théâtre influe sur les actions des acteurs et sur le scénario de la pièce en cours de représentation[20]. Price et Pask cultivent une amitié de longue date, depuis leurs études ensemble à Cambridge[21], et le Fun Palace est une première opportunité de mettre en œuvre dans un projet leurs intérêts communs. Malgré les échecs successifs, ils poursuivront d'ailleurs jusqu'en 1976 l'ambition de faire construire le projet[22]. Leur collaboration continue par la suite, d'abord en tant qu'enseignants. Price, alors en poste à la AA, y invite régulièrement Pask pour des jurys, ateliers ou conférences, jusque dans les années 80, moment où Pask occupe finalement lui-même un poste d'assistant d'enseignement à l'école. Et entre 1962 et 1969, deux de leurs étudiants se piquent

[15] Par exemple chez Theodora Vardouli ou Alice Upitis. Le premier dont on ait trouvé trace dans le présent travail de recherche est le Laboratory for Computer Graphics and Spatial Analysis de Harvard, créé deux ans plus tôt en 1965.

[16] Des idées qui donnent rarement lieu à des projets construits, et surtout à des dessins et des textes. Hardingham, S., *Cedric Price Works 1952–2003: A Forward-minded Retrospective*, Architectural Association Publications, 2016.

[17] La cybernétique désigne l'étude des mécanismes d'échange d'information entre entités. Le terme émerge dans les années 1940 alors qu'un groupe de scientifiques renommés issus de diverses disciplines cherche à donner naissance à une compréhension unifiée de la communication.

[18] Pickering, A., "Gordon Pask: From Chemical Computers to Adaptive Architecture", in Pickering, A., *The Cybernetic Brain. Sketches of Another Future*, University of Chicago Press, 2009.

[19] Hardingham, S., *Cedric Price Works 1952–2003: A Forward-minded Retrospective*, Architectural Association Publications, 2016.

[20] Hernandez, J., "From the Fun Palace to the Generator. Cedric Price and the conception of the first intelligent building", in *ARQ (Santiago),* n.90, 2015.

[21] Entre 1949 et 1953.

[22] En particulier Price, très attaché à ce projet. S'il ne parviendra jamais à faire construire le Fun Palace, ce projet a eu une influence durable sur l'architecture des années qui suivent. En particulier, l'influence des propositions faites pour le Fun Palace sur la conception du Centre Georges Pompidou, dont la construction se termine en 1977, a été établie par de nombreux observateurs.

d'intérêt pour les thèmes promus par Pask et Price, jusqu'à en faire leurs sujets de diplôme, puis l'objet de leurs carrières : Paul Coates, Julia May Connor et John Frazer.

L'un de ces projets en particulier réunit Cedric Price, Gordon Pask, John Frazer et Julia May Connor, devenue en 1977 Julia Frazer : le Generator. En 1976, Howard Gilman commande à Cedric Price un bâtiment reconfigurable, basé sur les mêmes principes que le Fun Palace[23]. Ce bâtiment doit servir de centre de loisirs aux employés de son usine, la Gilman Paper Company - une grosse entreprise américaine qui fabrique du papier. L'entreprise met donc à disposition de Price un terrain au sein de la White Oak Plantation, sur la côte de Floride. Price fait de nouveau appel à Gordon Pask, mais aussi aux Frazer. En effet, si le Fun Palace n'est pas allé au-delà de l'esquisse, le projet pour la Gilman doit être construit. Price a donc besoin, en complément de Pasl et de ses schémas cybernétiques[24], de spécialistes capables d'écrire les programmes informatiques sur lesquels le système va reposer. Le groupe travaille sur le projet jusqu'en 1979, date à laquelle le conseil d'administration de la Gilman refuse de poursuivre le financement du projet malgré les ambitions de son président[25]. Alors que Seek et URBAN 5 s'intéressent à l'échelle urbaine, le Generator Project, lui, transpose les mêmes principes d'adaptabilité à l'échelle du bâtiment. Le système se compose de fondations qui suivent un plan en forme de grille. A ces fondations s'ajoutent une grue et des éléments constructifs : cloisons, portes, fenêtres. Ces éléments contiennent une puce qui permet de les identifier. Ils sont réarrangés en permanence à l'aide de la grue sur la grille des fondations dans des positions différentes et permettent d'accommoder différentes activités. Les adaptations du plan dépendent de quatre programmes informatiques. Le premier est un programme de dessin qui trace le plan en suivant les règles constructives du système d'éléments. Le deuxième est un programme d'interaction avec les usagers qui vise à définir leurs besoins. Le troisième programme observe la réaction des usagers à une configuration donnée pour évaluer cette configuration. Enfin, le quatrième programme est conçu pour permettre au Generator lui-même de proposer spontanément des configurations en fonction du comportement des usagers : c'est le programme *boredom*.

[23] Wright Steenson, M., "Cedric Price's Generator" http://www.girlwonder.com/blog/wp-content/uploads/2010/04/crit-piece.pdf, consulté le 12 Novembre 2021.
[24] Qui font office de ce qu'on appellerait aujourd'hui un logigramme algorithmique.
[25] Il faut noter que Howard Gilman a été toute sa vie un mécène, et qu'il a peut-être approché ce projet comme du mécénat sans trop d'attentes quant au résultat, contrairement au conseil qui, voyant s'amenuiser la probabilité d'un véritable bâtiment émerger, a choisi d'y mettre fin.

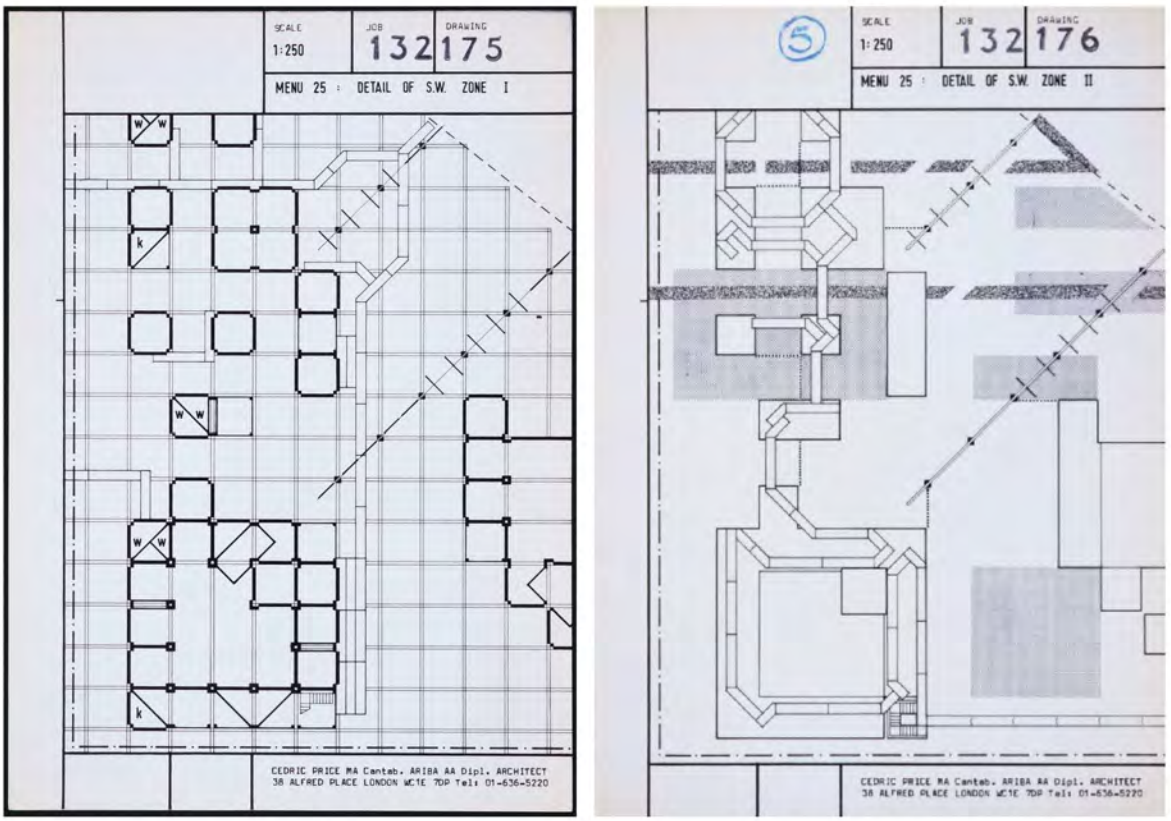



*Figure 3. Deux plans résultant de l'exécution des programmes du projet Generator.*

En parallèle, John Frazer, Julia Frazer occupent, comme Paul Coates, leurs premiers postes en tant qu'enseignants dans plusieurs écoles d'architecture à Londres. Ces postes leur permettent de développer plus avant les thèmes de recherche qui leur sont chers. Ainsi, John Frazer démarre en 1969 à la AA, d'abord en tant qu'étudiant dans le cadre de son projet de diplôme[26] puis en tant qu'enseignant, ses travaux sur ce qu'il baptise l'*architecture évolutionnaire*. L'expression désigne la conception d'objets architecturaux conçus à l'aide d'algorithmes eux aussi dit évolutionnaires, qui simulent la croissance d'êtres vivants[27]. John Frazer fait par ailleurs avant sa collaboration sur le Generator un passage à l'université de Cambridge où il fonde avec Alex Pike la *Technical Research Division* du département d'architecture. Il retourne ensuite avec Julia Frazer à la AA. Tous deux sont à l'origine dans cette école du second des plus importants pôles universitaires du champ computationnel. Les activités de ce pôle contribuent largement à populariser les algorithmes évolutionnaires. John Frazer choisit ensuite dans les années 1990 d'aller enseigner à l'université de Hong Kong. Julia Frazer, quant à elle, enseigne encore aujourd'hui à la AA, où elle est à la tête du département de computation. Après un passage à l'Ecole Polytechnique de Liverpool, Paul Coates est

[26] Frazer, J., "Autotectonics : The choice of choice", Diplôme de fin d'études, Architectural Association, 1969.
[27] Frazer, J., *An evolutionary architecture*, Architectural Association Publications, 1995.

recruté par l'Université de East London (UEL), où il reste toute sa carrière. Il y crée d'abord un programme de master, au sein duquel les étudiants développent de nombreuses recherches sur les questions procédurales. Ils étudient notamment les applications à l'architecture des théories de linguistique générative de Noam Chomsky, dont les travaux ont beaucoup d'influence sur Coates. Ce programme devient ensuite petit à petit un centre de recherche à part entière, structuré en tant que tel à partir des années 1990. L'existence de ce groupe est ensuite reconnue officiellement en 2001 avec la fondation du Centre for Evolutionary Computing in Architecture (CECA) par Paul Coates, en association avec Christian Derix qui vient de terminer ses études au sein du master de l'UEL[28]. John Frazer, Julia Frazer et Paul Coates entretiennent des échanges et collaborations régulières depuis leurs établissements respectifs pendant plusieurs décennies après leur rencontre. Parmi ces collaborations, la création d'Autographics, un des tout premiers logiciels de dessin architectural sur ordinateur, précurseur d'Autocad, qui les occupe de 1978 à 1996. Pendant très longtemps, ces expérimentations londoniennes collectives ne se font cependant pas autour d'un groupe de recherche à proprement parler. Elles se font donc sans structure de fonctionnement et de financement officielle et avec des liens avec le monde de l'intelligence artificielle moins forts. Mais elles rassemblent néanmoins un groupe de praticiens récurrents autour d'un ensemble de projets aussi fondateurs pour l'architecture computationnelle que ceux de l'Architecture Machine Group.

### *2.1.3 Monde anglophone*

Si la AA et le MIT deviennent très vite les pôles les plus marquants de la période, ce ne sont pas les seuls. Un petit nombre d'autres chercheurs se consacre également à la question de la programmation informatique mise au service de l'architecture. Ils réunissent autour d'eux, dans les structures où ils enseignent, des groupes de quelques collègues et étudiants. Toujours à Londres, Ben Hillier passe la majeure partie des années 70 et 80 à développer avec son équipe de la Bartlett[29] la *space syntax*. L'appellation désigne un système d'analyse qui repose sur une décomposition de l'espace en un système d'objets mathématiques. La démarche de conception architecturale est envisagée comme une série de combinaisons entre ces objets, et peut ainsi être représentée par un graphe. Ceci permet la transformation de la démarche en un système procédural facilement exploitable via des outils informatiques. Dès la fin des années 70, le système de la *space syntax* fait de nombreux émules dans le petit monde de l'architecture computationnelle. Michael Benedikt en particulier développe à l'université du Texas les isovists, une technique d'analyse de la perception de l'espace qui repose sur

[28] West, M., "Paul Coates 1945-2013 research emerging", Essay for the MSc IDBE, University of Cambridge, 2014.
[29] En particulier Adrian Leaman, Alan Beattie, Julienne Hanson.

les mêmes principes computationnels[30]. A l'université de Californie, à Los Angeles, William Mitchell, nommé directeur de la faculté d'architecture, contribue par son enseignement et ses publications[31] à promouvoir la conception assistée par ordinateur au sens large, des simples logiciels de dessin aux processus procéduraux. Ceci favorise dans l'écosystème architectural californien une habitude précoce de recours aux outils informatiques. William Mitchell devient par la suite, en 1985, le directeur de l'école d'architecture du MIT. Sa présence au sein de l'institution et son travail de promotion contribuent à asseoir le MIT comme une autorité majeure au sein des courants computationnels en architecture. À Harvard puis à Berkeley, Christopher Alexander, à cheval dès le début de ses études entre les domaines de l'architecture, des mathématiques et de l'informatique, participe aussi largement à légitimer le champ computationnel, notamment par ses écrits. Ses recherches portent sur ce qu'il nomme des *patterns* (motifs), qu'on peut comprendre comme des séries d'instructions paramétrables potentiellement adressées à un non-sachant ou à un ordinateur - de manière indifférenciée pour Alexander. Il expérimente ensuite avec des programmes informatiques, cherchant à mettre en place des systèmes qui puissent garantir la production d'une architecture de qualité. Un projet tardif, le Gatemaker (1996), qui permet de dessiner une grille de jardin, rassemble nombre des sujets auxquels s'intéresse Alexander. L'interface vise à offrir un confort de conception à l'usager et à empêcher que la nature informatique de l'outil ne le distraie de son travail de conception. Les instructions données à l'utilisateur sont calibrées pour fluidifier les étapes de dessin pour un néophyte, le tracé de la porte sur des photos de sites existants doit permettre de prendre en compte au mieux le contexte du projet - élément essentiel de la conception architecturale selon Alexander. Allen Bernholtz fonde pendant son séjour à Harvard entre 1965 et 1972 le Laboratory for Computer Graphics and Spatial Analysis. Il y teste plusieurs des programmes développés par Alexander dans des cas d'études sur le logement, et y poursuit les recherches de ce dernier une fois celui-ci parti pour Berkeley. A l'université technologique de Sydney (UTS) enfin, John Gero fonde en 1968 la Computer Applications Research Unit[32], qu'il dirige jusqu'en 2007. Il se penche au cours des premières années notamment sur la question de l'optimisation architecturale. John Gero contribue également largement à la naissance d'une filière universitaire en Australie en créant un master en *computation architecturale* et en organisant des cycles de conférences dédiées à ce sujet.

Un peu partout dans le monde anglophone émergent donc des lieux d'expérimentation, majoritairement centrés autour d'une figure charismatique en particulier promouvant ces thèmes de

[30] Benedikt, M., "To Take Hold of Space: Isovists and Isovist Fields", in *Environment and Planning B Planning and Design,* Vol.6 n.1, pp. 47-65, 1979.
[31] Notamment Mitchell, W.J., *Computer-Aided Architectural Design*, John Wiley & Sons, 1977.
[32] Qui deviendra l'Architectural Computing Unit en 1973, la Design Computing Unit en 1982, le Key Centre of Design Computing en 1990, le Key Centre of Design Computing and Cognition en 1998 et finalement le Design Lab en 2007.

recherche. Chacun représente le point de départ d'un pôle significatif et durable des courants computationnels en architecture. Ces praticiens sont issus tant du domaine de l'architecture que de ceux de l'ingénierie et de l'informatique. Ce croisement de disciplines caractérise le champ par la suite, avec un équilibre durable entre ces trois axes. Si ces projets et groupes pionniers n'entretiennent pas toujours des liens aussi proches que le réseau londonien, les prémisses d'un écosystème universitaire et d'un domaine de recherche à part entière apparaissent. Des articles sont publiés dans des revues d'autres champs dans un premier temps (*Architectural Science Review*, *Engineering Optimization*, *Environment and Planning*). Puis des ouvrages consacrés au sujet sont édités, dont nous avons cité plusieurs exemples. Des conférences dédiées sont organisées, comme les workshops de *Design Automation* de l'ACM à partir de 1963 ou la *International Conference on Computers in Architectur*e à Londres en 1972). Enfin, des revues spécialisées voient également le jour (*Computer Aided-Design*, *CAD Systems*), et en 1981 naît l'Association for Computer Aided Design in Architecture (ACADIA), qui rassemble depuis la majeure partie des praticiens du champ.

Ce qui apparente ces groupes et permet la mise en réseau des praticiens qui en font partie, et est à l'origine du champ computationnel, ce sont leurs thèmes de recherche, tous orientés vers les implications de l'utilisation de la programmation en architecture. Mais un autre élément les rassemble également: leur proximité avec le monde de l'IA. Celle-ci est parfois évidente au vu de l'origine des financements et de l'institution qui héberge le groupe, comme on l'a vu pour l'ArchMach. Cette proximité est parfois aussi plus discrète mais bien présente, à travers un réseau d'influences et un intérêt partagé pour certains des enjeux de l'imitation et de l'automatisation de la pensée humaine. Elle est également perceptible dans la pratique. En effet, les architectes impliqués dans des programmes de recherche sur l'architecture computationnelle sont quasi systématiquement accompagnés d'informaticiens. Ces derniers prennent en charge tout ou partie des opérations de programmation à accomplir pour développer les projets. Par exemple, les interventions de Gordon Pask dans les projets de Price le sont à titre d'expert cybernéticien. Par extension il prend donc en charge la mise en place de la structure informatique des programmes, par exemple celui du théâtre cybernétique pour le Fun Palace. La cybernétique repose en effet après tout sur un principe majeur, celui de la rétroaction, qu'il faut donc intégrer au fonctionnement du bâtiment. Pour le Generator, Price est l'architecte et instigateur du projet et Pask pilote les décisions concernant la structure informatique d'ensemble sur laquelle repose le bâtiment. Mais ce sont les Frazer qui vont mener à bien le travail de programmation à proprement parler et ainsi servir pour l'occasion d'experts en informatique. Lionel March, collègue de Frazer lors de son bref passage à Cambridge, est à la même période lui aussi appelé à servir de programmeur et expert informatique. Il intervient notamment sur le projet de reconstruction de Whitehall mené par Leslie Martin. Le projet porte sur une série de bureaux pour des fonctionnaires du gouvernement britannique, et March doit mettre au point un système de

calcul de répartition des employés dans les bâtiments en fonction de leur hauteur et de leur emprise au sol. À l'aide de cet outil, Martin souhaite explorer diverses configurations architecturales et urbaines tout en respectant les besoins formulés par le commanditaire[33]. Christopher Alexander quant à lui collabore pendant une longue période - notamment pour le projet du Gatemaker - avec le même programmeur, Greg Bryant. Le positionnement très progressiste de ce dernier, un peu décalé de celui de la majeure partie des experts informatiques à l'époque[34] suscite des discussions poussées avec Alexander sur l'impact du recours à des outils numériques sur le processus de design. Bien que plus marquées par les hiérarchies en place, des discussions régulières sur les possibilités et enjeux techniques avec les programmeurs chargés de donner corps aux instructions imaginés par les architectes membres de l'institution ont dû avoir lieu fréquemment au MIT. L'Architecture Machine Group y dispose non seulement de l'aide de collègues chercheurs en informatique mais aussi d'un personnel spécialisé conséquent[35]. Durant cette période pionnière du champ computationnel en architecture, l'interaction avec des spécialistes issus du domaine de l'informatique et de l'intelligence artificielle est donc constante, et cette interaction implique une influence tant pratique que théorique sur les projets conçus.

L'influence de Gordon Pask, mais aussi de plusieurs autres cybernéticiens de l'époque, fers de lance de la seconde cybernétique[36], se fait sentir partout dans le champ naissant de l'architecture computationnelle. La vision systémique offerte par la discipline présente de nombreux avantages pour l'établissement de processus procéduraux de conception architecturale. La dimension sociale que la cybernétique a pour objectif de prendre en compte dans les modèles qu'elle établit suscite notamment l'intérêt des architectes. L'influence de Pask se fait donc sentir à la AA et en particulier dans la conception du Fun Palace puis du Generator - dont le principe de reconfiguration spontanée en réaction à l'ennui s'inspire directement d'un projet précédent de Pask, Musicolour. Mais les cybernéticiens sont aussi cités ailleurs dans les pôles naissants du champ computationnel. Paul Coates s'appuie largement sur la vision de l'anthropologue et psychologue américain Gregory Bateson, spécialisé dans les questions de communication. Ben Hillier s'appuie sur les travaux fondateurs de

[33] March, L., "Mathematics and Architecture since 1960", in Williams, K. and Rodrigues, J.F. (eds.), *Nexus IV : Architecture and Mathematics*, pp. 7-33, Kim Williams Books, 2002.
[34] Dans le sens où il s'intéresse à des explorations prospectives de l'usage de la programmation plus qu'à des mises en applications les plus efficaces possibles. Voir à ce sujet les comptes-rendus de Greg Bryant de sa collaboration avec Christopher Alexander, en particulier le texte de 2014 "Gatemaker, or the Aspen Summit 1997". https://www.gregbryant.com/grogbrat/aspen97/index.html
[35] Allen.M., "Architecture becomes Programming. Invisible technicians and local theories in the late 1960s", in Loosen, S., Heynickx, R. et Heynen, H., *The Figure of Knowledge: Conditioning Architectural Theory*, 1960s - 1990 Leuven university Press, 2020.
[36] Si la cybernétique première est ce qui donnera naissance au champ de l'intelligence artificielle, la cybernétique seconde se détache ensuite des deux pour constituer un champ à part entière, étendant l'idée de la cybernétique première que tout peut être décrit comme un système d'éléments et de relations qui les lient, en y ajoutant la notion d'évolution permanente de ces systèmes.

Norbert Wiener pour son ouvrage *The Social Logic of Space*, qui investigue les applications de la dimension sociale de la cybernétique à la transformation procédurale de l'architecture. Christopher Alexander est guidé dans le développement de ses motifs par les travaux de William R. Ashby, psychiatre et ingénieur anglais pionnier de l'automatisation. Nicholas Negroponte quant à lui développe URBAN 5 en s'appuyant sur le paradigme des *microworlds* (micro-mondes) développé par Marvin Minsky et Seymour Papert dans le cadre de leurs travaux sur l'intelligence artificielle[37]. La plupart des pionniers du champ computationnel en architecture s'inscrivent cependant dans le paradigme théorisé par les cybernéticiens qui leur sont contemporains. Il s'agit d'un paradigme particulièrement adéquat pour le développement d'une architecture automatisée puisqu'il est supposé permettre de tout formaliser sous la forme d'échanges d'informations.

La proximité intellectuelle est telle qu'on verra par la suite, dans les années 2000, une résurgence dans le milieu architectural computationnel. Celle-ci se cristallise autour des notions d'émergence et de morphogénétique développées par la seconde cybernétique, en complément de la notion de rétroaction chère aux chercheurs des années 1960-1970. Cette résurgence a lieu alors même que la cybernétique n'est plus un paradigme très en vogue par ailleurs[38]. Ce regain d'intérêt pour la cybernétique se fait donc de manière un peu déconnectée des recherches en cours dans le monde de l'informatique et de l'intelligence artificielle. Mais dans les années 1960-1970, les premiers praticiens du champ computationnel en architecture rencontrent un écho très fort à leurs recherches chez leurs collègues informaticiens. Une influence à double sens s'établit pendant presque deux décennies, en particulier autour de la figure de Christopher Alexander. Ainsi, ses *Notes sur la Synthèse de la Forme*[39] deviennent une lecture obligatoire pendant de nombreuses années peu après leur publication… dans les cursus de sciences informatiques américains. Son ouvrage suivant, *A Timeless Way of Building*[40], paru en 1979, est également très lu par la communauté informatique américaine. Les cercles universitaires se piquent d'intérêt pour l'ouvrage, en particulier en Californie où Alexander est en poste. Mais les réseaux entrepreneuriaux émergents de la Silicon Valley aussi font la promotion de l'ouvrage, notamment parce qu'ils y perçoivent un reflet de leur propre influence[41]. Alexander a

[37] La majeure partie des recherches du MIT sur le sujet à l'époque s'appuient également sur ce paradigme. Minsky, M. and Papert, S. Proposal to ARPA for Research on Artificial Intelligence at MIT, 1970–1971. MIT Artificial Intelligence Lab, Cambridge, MA, 1970, 34. Steenson, M., "Microworld and Mesoscale", in *Interactions 22*, pp. 58-60, 2015.

[38] Pour une analyse détaillée des relations entre théoriciens de la cybernétique et architectes du champ computationnel, voir le travail de Paul Pangaro et Hugh Dubberly présenté dans Dubberly, H. et Pangaro, P., "How cybernetics connects computing, counterculture, and design", in Blauvelt, A. (ed.), *Hippie Modernism: The Struggle for Utopia*, Walker Art Center, 2015.

[39] Alexander C., *Notes on the Synthesis of Form*, Harvard University Press, 1964.

[40] Alexander, C., *The Timeless Way of Building*, Oxford University Press, 1979.

[41] "*Among the readers chewing through its 1,171 pages were countless programmers, product designers and computer entrepreneurs. The book was infectious, and for Bay Area technologists it was, in part, considered a local brainchild. Among the many who promoted it was Stewart Brand, a quintessential local counterculture*

pourtant lui aussi pesé sur le développement de l'informatique. Les motifs qu'il développe pendant toute sa carrière, sujets notamment de *A Timeless Way of Building*, ont beaucoup compté dans l'émergence de la programmation orientée objet. Celle-ci a en retour influencé durablement les structures de programmation adoptées dans le champ computationnel[42] en architecture, contribuant à un aller-retour régulier entre les deux disciplines au cours de leur développement.

Il est souvent difficile d'identifier la toute première occurrence de l'utilisation d'un ordinateur pour accomplir telle ou telle tâche, ou du moins d'être certain que celles qui ont été identifiées comme telles sont bien les premières. Les praticiens et les travaux décrits jusqu'ici sont cependant unanimement acceptés comme les premiers à faire de l'architecture avec des outils informatiques, et à se pencher sur les conséquences théoriques de cette transformation. Mais ce qui fait l'intérêt de cette première génération du champ computationnel, plutôt que l'éventuelle primauté des recherches, c'est la mise en réseau des praticiens qui la composent. Les rares travaux d'histoire menés sur ces débuts se concentrent sur quelques figures, les présentant comme des praticiens autonomes et extraordinaires à l'influence durable[43]. Pourtant, c'est l'apparition de pôles de recherche où se regroupent non seulement les figures les plus en vue, mais aussi des collaborateurs et des étudiants moins réputés, qui permet de donner corps à un champ à part entière apparu à cette période. Deux pôles en particulier émergent à ce moment, nous l'avons vu, l'un autour du MIT et l'autre autour de la AA. Ces pôles comportent chacun de nombreux satellites et créent ainsi deux réseaux reliés seulement par quelques rares praticiens. Si on est encore loin des milliers de praticiens et des dizaines de groupes qu'on trouve aujourd'hui dans le champ computationnel, cette génération est la première à faire date précisément grâce à la possibilité de faire réseau, une possibilité dont ses membres ont su se saisir. Plus que ce qui est usuellement raconté de la genèse des courants computationnels en architecture, récit des coups de génie de quelques rares grands hommes[44], c'est la constitution d'un réseau d'influence qui en marque les débuts. Ces praticiens identifiés comme majeurs ont néanmoins un point commun. Au-delà de leur

---

*figure, founder of the Whole Earth Catalog, and chronicler of important early computer experiments.*" Bryant, G., "Gatemaker: Christopher Alexander's dialogue with the computer industry", in Neis, H.J., Brown, G.A. et Bryant, G., *Battle for the Life and Beauty of the Earth, PUARL Conference Proceedings*, pp.77-91, 2013.

[42] "*Despite the shortcomings of the older approach to patterns, eight years after The Timeless way of Building, increasing interest in patterns among computer people led to a formalization, for an increasingly fashionable software engineering approach known as Object-Oriented Programming.*" Bryant, G., "Gatemaker: Christopher Alexander's dialogue with the computer industry", in Neis, H.J., Brown, G.A. et Bryant, G., *Battle for the Life and Beauty of the Earth, PUARL Conference Proceedings*, pp.77-91, 2013.. On reviendra plus en détail sur le rôle joué par la programmation orientée objet dans les pratiques du champ computationnel dans les chapitres IV et V.

[43] Voir les travaux cités d'Alise Upitis, Matthew Allen ou Molly Wright Steenson, mais aussi les très nombreux compte-rendus des travaux d'un praticien ou l'autre en particulier ceux cités produits par Martin West, Greg Bryant, ou Andrew Pickering.

[44] Jamais de femmes, malgré leur présence aussi dans le réseau dès cette époque - Voir Recommandations de lecture.

vision de l'architecture computationnelle et des expériences qu'ils mènent dans ce champ au cours de leur carrière, ils possèdent une capacité remarquable à communiquer cette vision, à multiplier les collaborations et donc à faire réseau. Ainsi Nicholas Negroponte fait office de meneur dans le pôle du MIT, Christopher Alexander dans celui de Californie, John Frazer, bien que plus jeune et donc à l'impact plus tardif, porte le développement du pôle londonien à la suite de Cedric Price. Enfin, Gordon Pask, par son investissement à la AA et par ses fréquentes visites à l'Architecture Machine Group puis au Medialab, contribue fortement à faire circuler les idées entre les deux pôles du réseau.

Les sous-pôles que représentent les différentes universités où naissent des activités de recherche dans le domaine sont eux aussi portés par des praticiens qui développent des réseaux locaux. Tous n'ont cependant pas la même postérité qu'Alexander, Negroponte ou Frazer. Paul Coates, John Gero ou Charles Eastman sont par exemple un peu moins renommés. Chacun d'entre eux forge sa propre trajectoire au sein du réseau, et on peut observer des similitudes entre leurs carrières. Leurs parcours relèvent de trajectoires typiques qui reflètent les préoccupations de cette génération du champ computationnel. Sur le plan institutionnel, certains choisissent de faire carrière dans un seul et même établissement. C'est le cas de Paul Coates à l'UEL, de John Gero à l'UTS ou de Ben Hillier à la Bartlett. D'autres contribuent à structurer le réseau en circulant beaucoup d'un établissement à un autre, comme Michael Benedikt, William Mitchell ou Charles Eastman. Peu d'entre eux se préoccupent de très près des possibles applications pratiques de leurs travaux. Mais on observe tout de même quelques incursions dans le monde professionnel. C'est le cas d'Eastman avec Formtek, des Frazer et Coates avec Autographics, de Hillier avec Space Syntax Limited. Ces incursions se font souvent à travers l'édition ou l'utilisation de programmes informatiques plutôt que dans des pratiques architecturales classiques. Sur le plan de la recherche, certains se consacrent au champ computationnel pendant l'intégralité de leur carrière, comme Paul Coates ou John Gero. D'autres en revanche alternent entre plusieurs champs de recherche en architecture, comme Yona Friedman ou Christopher Alexander. Certains se détournent même complètement de l'architecture après un temps, comme Nicholas Negroponte qui se reconvertit en investisseur dans le domaine des nouvelles technologies après avoir dirigé le Medialab jusqu'en 2000. Il fonde aussi plus tard l'ONG One Laptop Per Child, qui promeut la démocratisation de l'accès aux technologies numériques[45]. Certains des praticiens du champ explorent un unique thème et projet de recherche au cours de leur vie. C'est le cas de Michael Benedikt avec les isovists, de Ben Hillier avec la Space Syntax ou de John Frazer avec l'architecture évolutionnaire. D'autres en revanche passent d'un projet et d'un enjeu de recherche à l'autre, comme Negroponte, Alexander ou Coates.

[45] https://www.onelaptopperchild.org/

Tous s'investissent dans la constitution d'un réseau, mais certains se consacrent plus que d'autres à la dissémination à travers l'enseignement et la création de revues. Ceci devient même l'héritage principal de quelques-uns des praticiens, qui ne laissent presque aucune trace des projets qu'ils ont menés mais beaucoup plus de leurs activités institutionnelles. Ces trajectoires diverses sont la marque d'un réseau qui se constitue non seulement par les collaborations et l'institutionnalisation, mais aussi par la promotion au sein des institutions du champ qui s'esquisse. La promotion se fait par des stratégies variées en fonction des praticiens. William Mitchell ou Michael Benedikt optent ainsi pour s'intégrer dans les structures institutionnelles en place pour promouvoir le champ de l'intérieur, quitte à s'affirmer moins fortement. Ils ont donc tous deux une forte influence sur l'adoption des outils numériques au sens large au cours des décennies, mais moins sur les enjeux conceptuels du procédural. Mitchell arrive, comme on l'a vu, à des postes à haute responsabilité dans le monde américain de la formation architecturale. Benedikt dirige le Center for American Architecture and Design pendant la majeure partie de sa carrière, en complément de ses postes d'enseignant et de direction de l'enseignement. D'autres praticiens choisissent la création d'un espace personnalisé et fondent des programmes d'enseignement ou des centres de recherches spécialisés autour d'eux au sein de l'établissement où ils exercent. Ces espaces privilégiés sont souvent rendus possibles par une longue fidélité au même établissement, avec des moyens plus restreints mais des programmes de recherche plus libres. Ceci permet aux praticiens d'explorer des questions très prospectives, éloignées des préoccupations de la plupart des architectes en exercice ou de l'industrie de la construction. C'est le cas pour Paul Coates à l'UEL ou John Gero à l'UTS.

La promotion de ce nouveau champ de recherche passe également par les ouvrages publiés. Christopher Alexander, Nicholas Negroponte ou William Mitchell jouent un rôle central sur ce point. Ils publient en effet très tôt des livres clés pour faire appréhender à un plus vaste public le tournant amorcé. D'autres publient seulement beaucoup plus tard, privilégiant au cours de leur carrière l'enseignement à l'écriture. Paul Coates par exemple ne fait paraître son livre principal *Programming Architecture*[46] qu'en 2010. L'influence de ces publications et de ces activités est néanmoins limitée par la culture d'un certain entre-soi promue au sein des différents pôles, en particulier au sein de la AA et du MIT. Ces environnements sont déjà notoirement élitistes indépendamment des réseaux du champ computationnel qui s'y créent. Ces réseaux ne font donc dans une certaine mesure que perdurer une culture déjà pré-existante. Le réseau de relations qui se tisse entre tous les praticiens cités et l'enchevêtrement d'influences des uns sur les autres existent par le jeu des interactions que crée l'environnement universitaire - proximité géographique et/ou institutionnelle, conférences communes, invitations aux jurys, etc. Cet entre-soi qui se développe résulte pourtant de facteurs sociologiques

[46] Coates, P., *Programming Architecture*, Routledge, 2011.

autant que pour le confort de recherche qu'il permet. En effet, l'écosystème fermé qui se structure alors, embryon du microcosme qu'est encore aujourd'hui l'architecture computationnelle, offre à ses membres un espace privilégié d'expérimentation. Ils peuvent y tester les hypothèses les plus surprenantes sur le futur de l'architecture, car il s'agit un espace protégé des contraintes matérielles grandissantes auxquelles est soumise la pratique professionnelle de l'architecture. Cette culture d'un entre-soi et ce système de cooptation qui lie les membres de la première génération du champ computationnel est une caractéristique qui perdure par la suite. Elle est encouragée à la fois par la structuration de la recherche tous domaines confondus et par le confort offert par une situation universitaire pour les architectes attirés par les questions prospectives.

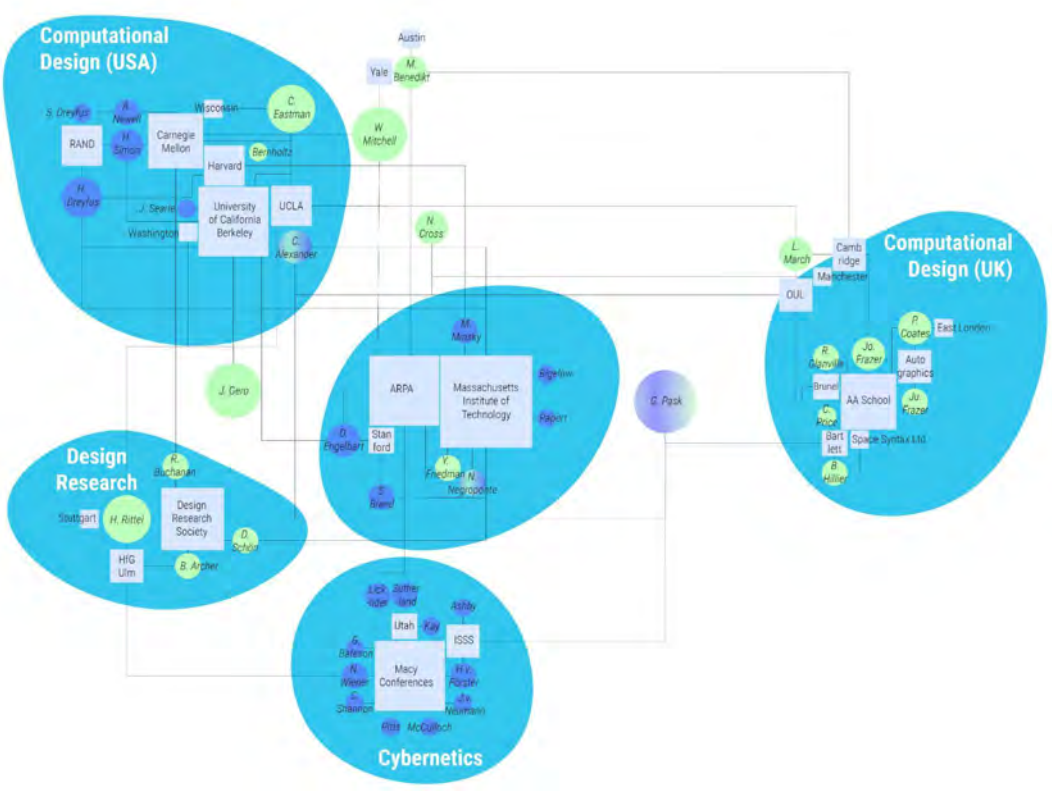



*Figure 4. Réseau de la première génération du champ computationnel, avec les affiliations respectives des praticiens entre 1960 et 1985.*

Les recherches en architecture aux États-Unis cultivent également une proximité intellectuelle et institutionnelle très forte avec le milieu de l'intelligence artificielle. La proximité géographique avec de grands noms de la recherche en IA favorise les échanges. Allen Newell et Herbert Simon se partagent entre Carnegie Mellon et la RAND Corporation[47] à Boston. Cette dernière dispose en complément de bureaux en Californie et accueille également Hubert et Stuart Dreyfus. Marvin Minsky, Seymour Papert et John McCarthy sont en poste au MIT. Boston et le MIT sont alors un lieu

[47] Un think tank militaire créé en 1948 pour soutenir les efforts de recherche de l'armée américaine, financé par le gouvernement.

clé du champ de l'intelligence artificielle, ce qui contribue à donner forme à la pensée de l'ArchMach. Cette pensée circule  par la suite dans le réseau et influence ainsi l'ensemble des recherches en architecture computationnelle de cette première génération. Nombre des publications écrites par les chercheurs de la période sont publiées dans des revues et conférences consacrées à la recherche en informatique, en particulier les conférences de l'Association for Computer Machinery (ACM). John Gero, Michael Benedikt ou Charles Eastman et leurs étudiants y publient régulièrement tout au long de leur carrière. Cette proximité institutionnelle est aussi encouragée par les structures et sources de financement, identiques à celles des recherches sur l'intelligence artificielle : DARPA, RAND, et financements privés sur le modèle du *corporate funding* adopté par le MIT. Si la DARPA finance les débuts de l'ArchMach, elle soutient aussi nombre de projets portés par d'autres praticiens du champ computationnel, et ce jusqu'à la fin des années 1990. William Mitchell, Michael Benedikt ou Ben Hillier aux Etats-Unis mais aussi John Gero à l'Université de Sydney bénéficient par exemple de ces financements. Les premiers essais de l'ArchMach, qui portent tant sur le développement des Technologies de l'Information et de la Communication (TIC) que sur la mise au point de programmes qui facilitent la conception architecturale, comptent parmi les projets financés. Les essais plus tardifs de John Gero, qui portent sur des tentatives plus avancées d'automatisation et d'optimisation des processus de conception[48], sont également financés. C'est aussi le cas des les systèmes de description systématique de l'espace développés par Michael Benedikt ou Ben Hillier. Par ailleurs, la capacité de Nicholas Negroponte à offrir une vision technologique qui va bien au delà de la question architecturale ou de Christopher Alexander à convaincre de l'intérêt du sujet de l'architecture pour l'avancée de l'IA aident à inscrire l'architecture computationnelle dans le paysage de l'intelligence artificielle et à obtenir des financements pour les projets.

Il faut également rappeler que la Defense Advanced Research Project Agency (DARPA) est à sa création la Advanced Research Project Agency (ARPA). Créée par Eisenhower en 1958, après le lancement du satellite Sputnik par les Soviétiques, avec l'objectif de rattraper le retard américain dans la conquête spatiale, l'ARPA est ensuite supplantée dans cette mission par la NASA, créée quelques mois plus tard et qui monopolise projets et financements sur le sujet. L'ARPA est réorientée vers des projets beaucoup plus prospectifs, avec un fort soutien de chercheurs, et la mission de l'agence devient d'aller au-delà des besoins immédiats de l'armée et des technologies immédiatement disponibles. En 1973, le Mansfield Amendment limite de nouveau les recherches de l'ARPA à des projets à vocation militaires, dans les domaines de la balistique, de la surveillance ou de l'armement. L'agence devient alors la DARPA. Cependant, entre 1958 et 1973, l'ARPA joue un rôle prédominant

[48] Notamment les projets *Design Creativity* et *Situated Design Computing*.

dans le développement des TIC[49]. L'agence travaille sur ARPANET - l'ancêtre d'internet -, sur le langage Processing - aujourd'hui encore très utilisé -, et finance de très nombreux projets prospectifs. L'ARPA porte ainsi les débuts de l'intelligence artificielle avec un spectre de recherche très large. Dans ce spectre sont incluses les prémisses de l'architecture computationnelle. En effet, l'architecture est un sujet qui rejoint exceptionnellement bien les grands enjeux de l'IA, en raison des formes de connaissances que les processus de conception nécessitent de mobiliser. La diversité de ces connaissances et la difficulté pour les concepteurs d'expliciter les savoirs et les méthodologies auxquels ils ont recours, mais aussi la variabilité du processus, qui change et s'adapte à chaque nouveau projet, font de l'architecture un cas exemplaire de la complexité du fonctionnement de la pensée. Il s'agit donc d'un support privilégié d'expérimentation pour l'IA. Les liens entre champ de l'intelligence artificielle et architecture computationnelle qui naissent dans les années 60 se construisent donc autour d'une fascination commune pour l'esprit humain et autour d'un effort collectif pour tenter de le comprendre et de le modéliser.

### 2.2 Les prémisses d'axes de recherche durables

Les systèmes informatiques de l'époque sont cependant pour le moins primitifs : la souris d'ordinateur est tout juste en développement[50], les écrans cathodiques sont encore petits - surtout quand il s'agit d'afficher plans et perspectives -, et les processeurs lents et encombrants. Certains des projets de l'époque ne se confrontent ainsi même pas à l'enjeu de la réalisation technique et se cantonnent à une exploration théorique de la mobilisation de la computation pour la conception architecturale. Cette limitation a souvent des raisons pratiques d'accès restreint aux ordinateurs, alors des machines coûteuses réservées à certains départements des universités. En 1961, Cedric Price n'a accès à aucune de ces machines à la AA et le Fun Palace n'est donc pas expérimenté en pratique. En 1969, Paul Coates aussi préfère élaborer pour son diplôme un système algorithmique manuel de détermination de la géométrie des bâtiments à base d'une roulette de poker miniature, de gommettes et d'un livret d'instructions rédigés par ses soins[51]. Les équipes du MIT, de la Bartlett ou d'UTS recourent elles à des stylets, des boutons et des claviers sur-mesure qu'il faut prendre le temps de développer et de fabriquer à chaque nouveau projet. Il faut également écrire chaque fois des programmes à l'aide des premiers langages de programmation disponibles et des quelques progiciels dédiés à l'affichage graphique existants. URBAN 2 et URBAN 5 utilisent par exemple un ordinateur IBM 360, un

[49] Chiou, S., Music, C., Sprague, K. et Wahba, R., "A Marriage of Convenience : The Founding of the MIT Artificial Intelligence Laboratory" - http://web.mit.edu/6.933/www/Fall2001/AILab.pdf, consulté le 12 Novembre 2021.
[50] Si la souris est développée à partir de 1963 à l'université de Stanford, le dispositif n'est révélé au public qu'en 1968, et commercialisé seulement à partir de 1973.
[51] West, M., "Paul Coates 1945-2013 research emerging", Essay for the MSc IDBE, University of Cambridge, 2014.

moniteur IBM 2250 - un écran cathodique sur lequel sont manipulés les cubes grâce à un crayon optique - et un ensemble de 32 boutons qui permettent d'assigner des attributs à chacun des cubes. Le programme est écrit en FORTRAN IV et fait appel au progiciel GPAK, qui permet de générer, d'afficher et de manipuler des images sur l'écran[52]. Pour les travaux qui le mènent à la publication de *Notes sur la Synthèse de la forme*, Alexander utilise lui aussi des ordinateurs IBM, de la série 700 cette fois, et le langage FORTRAN[53]. Il est alors chercheur associé au Computer Engineering Systems Laboratory (CESL) du MIT, et est impliqué dans un programme de recherche qui réunit plusieurs laboratoires de recherche de l'institut, le Computer-Aided Design Project. Les discussions que tiennent ses participants influencent beaucoup les choix techniques de l'époque, et notamment la promotion de l'usage de la programmation orientée objet pour la conception architecturale assistée par ordinateur. En dépit de la complexité et l'encombrement du matériel informatique, les programmes en eux-mêmes sont pourtant relativement simples et incluent peu d'opérations en regard des algorithmes contemporains. Les quatre programmes qui permettent au Generator project de fonctionner tiennent ainsi sur un simple circuit intégré[54]. La complexité de la conception d'un bâtiment est alors fréquemment décomposée en un ensemble de sous-programmes simples en interaction entre eux, comme ceux du Generator ou ceux du projet YONA. Nombre des programmes développés jusque dans les années 80 font l'objet de publications, où leur fonctionnement détaillé est décrit à travers la retranscription de seulement quelques lignes d'opérations logiques ou de code[55]. L'encombrement des systèmes et la sobriété des programmes à leur disposition n'empêche cependant pas les praticiens d'explorer les mécanismes de conception architecturale procéduraux. Ils progressent d'autant plus que les outils disponibles font rapidement de grands progrès, du premier microprocesseur en 1971 au premier PC en 1981 et aux avancées de la programmation orientée objet jusqu'en 1985. Deux visions possibles pour l'architecture computationnelle se dégagent ainsi peu à peu au cours de ces années. Elles sont définies notamment en regard de la proximité plus ou moins accentuée des équipes de recherche avec le champ de l'intelligence artificielle et de l'influence de ce champ sur les praticiens des équipes, laquelle donne lieu à deux axes de recherche distincts.

[52] Steenson, M., "Architectures of Information: Christopher Alexander, Cedric Price & Nicholas Negroponte", Thèse de doctorat, Princeton University, 2013.
[53] Upitis, A., "Two or More Architectures. Computers and Design at MIT until 1963" in Dutta, A. (ed.), *A Second Modernism. MIT, Architecture and the "Techno-Social" Momen*t, The MIT Press, 2013.
[54] Steenson, M., "Architectures of Information: Christopher Alexander, Cedric Price & Nicholas Negroponte", Thèse de doctorat, Princeton University, 2013.
[55] Voir notamment les publications montrées dans le chapitre IV.

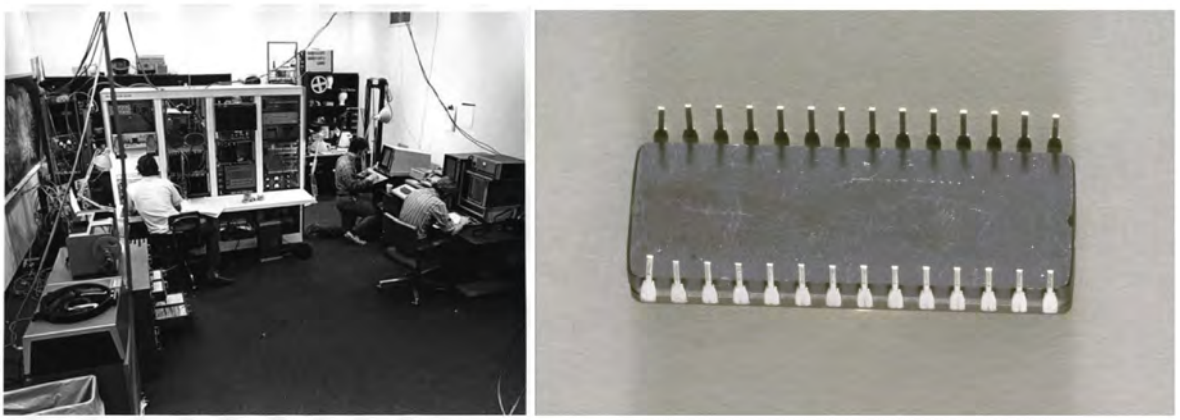


*Figure 5. a. Installation de l'Architecture Machine Group au début des années 1970 ; b. Puce du projet Generator.*

### *2.2.1 Axe 1 : l'architecte et la machine*

D'une part, la notion de machines à architecture, dispositifs computationnels d'aide à la conception de l'espace, telle que développée par Nicholas Negroponte et son équipe, interroge la place de l'architecte dans ce processus de conception. Le processus computationnel brouille le statut d'auteur de l'architecte[56] en l'associant à un programmeur. En fonction de ses capacités de programmation, l'architecte peut être lui-même programmeur, mais peut aussi être ravalé au rang de simple utilisateur d'un programme. En tant qu'utilisateur de ce programme, seule son expertise différencie l'utilisateur-architecte d'un autre utilisateur qui serait simple usager des espaces qu'il s'agit de concevoir. Le processus computationnel bouscule les statuts des acteurs d'un projet. Les hiérarchies des couples programmeur-utilisateur et architecte-habitant sont remises en question par le recours à des outils informatiques. Si l'architecte est programmeur, il est en position de définir l'ensemble des éléments qui interviennent dans le processus, et reste l'auteur privilégié du projet conçu[57]. Mais s'il est simple utilisateur du programme, son expertise architecturale lui permet-elle de produire un résultat différent d'un habitant qui serait tout aussi novice en programmation ? Est-il encore l'auteur du projet, ou est-ce que le programmeur partage ce statut ou prend le relais ? Lorsque Stewart Brand, à propos de Seek, désigne les gerbilles comme les architectes du projet, il s'aligne avec les propos tenus par Negroponte au sujet des ambitions générales de l'ArchMach. Il s'éloigne cependant en partie de la description du projet faite par ce même Negroponte, où les gerbilles sont décrites comme des usagers. Les concepteurs et programmeurs du système sont les personnes à l'origine des règles de création de l'espace et donc ceux tenant le rôle de l'architecte[58]. Il faut cependant noter l'influence

[56] Un statut en vigueur depuis la Renaissance. Voir à ce sujet Carpo, M., *The Alphabet and the Algorithm*, The MIT Press, 2011.
[57] Nous reviendrons sur la question de l'autorat en architecture et dans le champ computationnel au chapitre VII.
[58] Negroponte, N., *The Architecture Machine*, The MIT Press, 1970.

primordiale de l'ouvrage *Architecture sans architectes*, publié par Bernard Rudovsky en 1964[59], sur Nicholas Negroponte comme sur les Frazer, Cedric Price, Paul Coates ou Yona Friedman. L'ouvrage questionne la nécessité de la présence d'un architecte au sens moderne de la profession et passe en revue de nombreux espaces vernaculaires conçus et construits sans intervention d'un architecte. Le texte qui accompagne la présentation de ces projets est cité dans nombre des écrits de cette génération du champ computationnel[60]. L'influence des idées qui sont présentées dans l'ouvrage qui se fait sentir dans les interrogations sur la place de l'architecte qui sous-tendent les projets produits.

Dans la continuité de Seek, URBAN 5 et le Generator continuent d'interroger les places respectives de l'usager et de l'architecte dans le processus de conception, et ajoutent une entité supplémentaire à la réflexion : l'ordinateur. En créant pour URBAN 5 un programme qui a pour faculté de converser avec les usagers et d'émettre des propositions sur la configuration spatiale, l'ArchMach transforme ce programme en un interlocuteur à part entière dans le processus de conception. Par extension, l'ordinateur devient un acteur de ce processus. Au croisement des couples utilisateur-programmeur et architecte-habitant s'ajoute donc celui entre humain et machine. Les différents projets se penchent généralement de manière privilégiée sur l'une ou l'autre de ces relations. Mais l'ensemble d'entre elles jouent un rôle dans la définition du projet. Le Generator va plus loin dans l'examen du croisement humain-machine. Le quatrième programme sur lequel repose le système, *boredom*[61], a pour objectif, lorsque peu de changements sont observés dans les activités des usagers au sein du bâtiment, de déclencher des modifications spontanées du plan et de la structure par le système. En bref, lorsque le système s'ennuie, il tente d'y remédier. Les Frazer et Price dotent donc en quelque sorte leur système[62] d'une personnalité, et personnifient encore un peu plus l'ordinateur dans un rôle de concepteur. Au sein de l'unité de master dirigée par John Frazer à la AA, la Morphogenesis Unit, sont entreprises avec les étudiants successifs de multiples expériences d'architecture computationnelle entre 1989 et 1996. Les Frazer et Pask y poursuivent leurs questionnements, offrant à l'ordinateur le statut d'entité à part entière prenant part au travail de conception[63]. Ils se penchent notamment sur le rôle de l'interface dans cette question. La question de la communication avec un ordinateur est étudiée, dans le cadre du projet Universal Constructor, à travers le développement d'un système de cubes à manipuler à la place des traditionnels clavier et souris. Le motif créé par les cubes est prévu

---

[59] Rudovsky, B., *Architecture without Architects: A Short Introduction to Non-Pedigreed Architecture*, Doubleday & Company, 1964.

[60] West, M., "Paul Coates 1945-2013 research emerging", Essay for the MSc IDBE, University of Cambridge, 2014.

[61] "ennui"

[62] Qui est en même temps le bâtiment - encore une superposition des intentionnalités.

[63] Frazer, J.. "The Cybernetics of Architecture: A Tribute to the Contribution of Gordon Pask", in *Kybernetes. The International Journal of Systems & Cybernetics* Vol. 30 n.5-6, pp. 641-651, 2001. Frazer J., *An Evolutionary Architecture*, Architectural Association Publications, 1995.

pour être lu comme un motif structurant un système, selon plusieurs possibilités d'interprétation : structure du programme, structure du bâtiment ou encore structure des interactions.

Les travaux de la Morphogenesis Unit sont largement influencés par la perception de Pask du bâtiment comme une entité à part entière impliquée dans le processus de gestion de l'espace à travers les programmes informatiques qui le font fonctionner. Ceci permet à la machine de construire une relation directe avec ses usagers lui permettant de se passer de l'intermédiaire qu'était jusqu'ici l'architecte. Mais l'idée de Frazer que les bâtiments peuvent croître comme des structures biologiques joue également un rôle primordial. Cette idée est mise en œuvre dans les programmes successivement développés pour le Morphogenesis Project. Ceux-ci s'appuient sur d'autres travaux menés plus tôt par Pask, qui portent sur les processus de croissance modélisés via des algorithmes génétiques et des automates cellulaires. Ces programmes contribuent donc à explorer la notion d'architecture évolutionnaire forgée par John Frazer. Pask collabore donc avec Price et les Frazer autour de la relation bâtiment-usager telle qu'envisagée par la cybernétique architecturale. Mais il collabore aussi avec les Frazer autour de l'autonomisation du bâtiment. Enfin, Pask travaille également avec Nicholas Negroponte et l'ArchMach pour poser les bases de la notion de machine à architecture. Ces travaux sont le lieu d'explorations de la relation entre l'architecte et son outil, et examinent une symétrie de contribution à la conception entre les deux entités. Le cybernéticien fait donc le lien entre les deux pôles, mais aussi entre les différentes implications du triple croisement architecte-habitant, programmeur-utilisateur et humain-machine. Gordon Pask rassemble ainsi presque seul l'existence liée de ces trois dimensions plutôt que l'une ou l'autre d'entre elles de manière isolée. Dans son texte consacré au cybernéticien[64], Andrew Pickering traite largement de l'intérêt de Gordon Pask pour l'art et l'architecture. Il y développe pour résumer cette approche la notion d'*architecture paskienne* au sujet des projets développés par Pask avec ses collègues de la AA, puis par ses étudiants de Brunel University, dont un grand nombre ont d'abord été formés comme architectes dans d'autres établissements. Cette architecture paskienne se base sur l'idée d'une relation entre humain et non-humain en évolution permanente. Elle pose non seulement la nécessité de l'étude des interfaces entre humain et ordinateur, mais aussi l'existence de cette double relation à explorer qu'implique la cybernétique architecturale, entre acteurs humains du processus de conception comme entre humain et machine. Le Morphogenesis Project joue un rôle important dans la définition de cette architecture paskienne et du devenir réactif des bâtiments projetés. Mais il joue aussi un rôle prépondérant dans le développement des thématiques portées par Frazer sur l'architecture évolutionnaire et la morphogénétique des bâtiments, alors une idée cantonnée aux travaux de Frazer, mais qui deviendra

[64] Pickering, A., "Gordon Pask: From Chemical Computers to Adaptive Archictecture", in Pickering, A., *The Cybernetic Brain. Sketches of Another Future*, University of Chicago Press, 2009.

plus tard un courant majeur du champ computationnel[65]. Les projets de l'unité de John Frazer à la AA ont donc une double importance dans la genèse du champ.

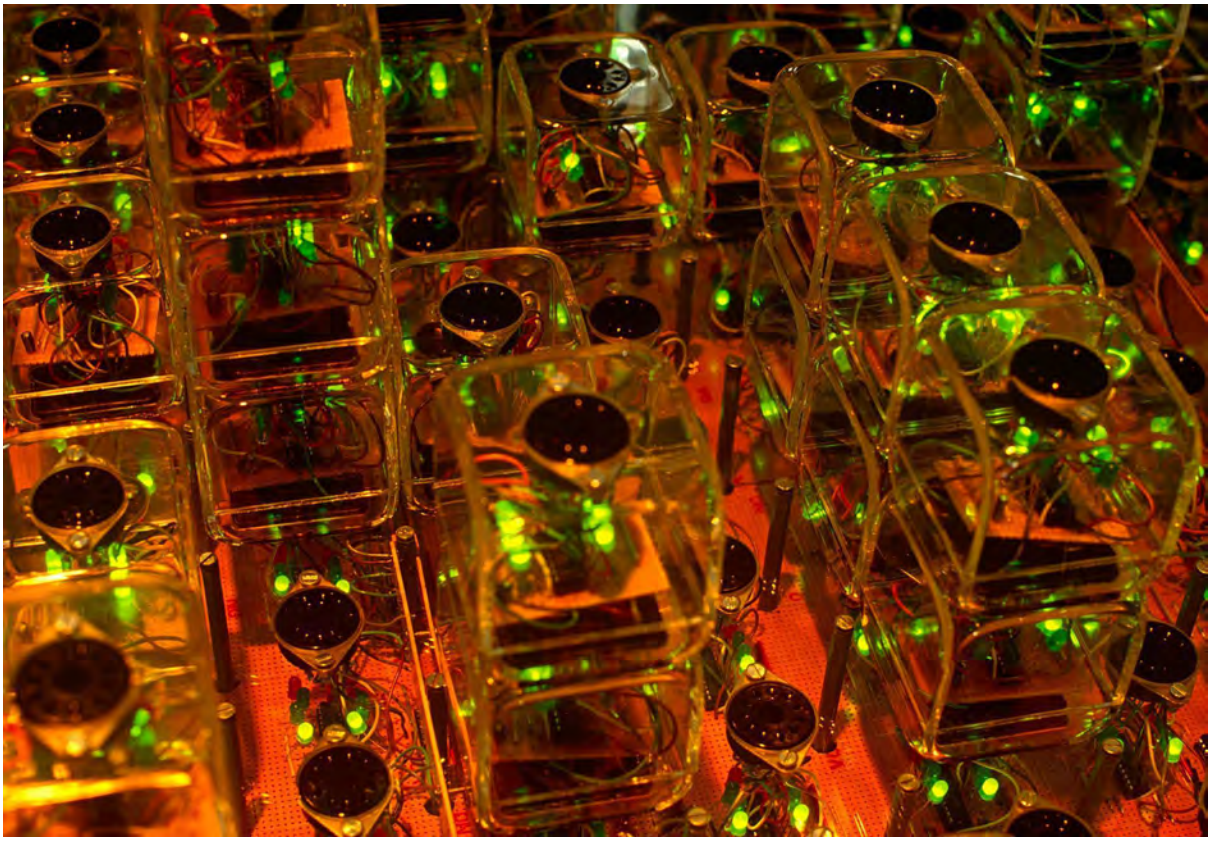


*Figure 6. Cubes de l'Universal Constructor développé par la Morphogenesis Unit*

Alors que les travaux de Pask avec Price et les Frazer interrogent le rôle joué par un bâtiment devenu presque conscient, le projet Flatwriter de Yona Friedman, développé lors d'un séjour au MIT en 1967, s'intéresse à la place à donner à l'habitant. Le projet cristallise les préoccupations du moment sur l'enchevêtrement des rôles et des expertises - des préoccupations qu'on retrouve aussi dans nombre d'autres projets d'architecture hors du champ computationnel, un peu partout en Europe notamment[66]. Le Flatwriter est un programme informatique conçu sur le principe d'une machine à écrire, remplaçant sur les touches du clavier les lettres par des espaces fermés, dont on sélectionne la forme, le placement et l'orientation en tapant sur les touches, "écrivant" au fur et à mesure le plan désiré. Yona Friedman, dont toute la carrière avant et après le Flatwriter raconte l'ambition de rendre accessible à tous tant la discussion que la conception architecturale, transpose les principes qui le guident dans une version logicielle dont l'interface est pensée précisément pour être utilisable par

[65] Voir Chapitre IV.
[66] Vardouli, T., "Architecture-by-yourself. Early studies in computer aided participatory design", Mémoire de Master, M.Arch Design and Computation, MIT

n'importe qui - architecte, usager, programmeur, utilisateur, ou tout ça à la fois. Negroponte a raconté plus tard plus tard la forte influence que les travaux de Friedman ont sur lui à l'époque. La collaboration entre l'Architecture Machine Groupe et ce dernier se poursuit après le Flatwriter, entre 1975 et 1977, dans le cadre du projet "Architecture by Yourself: An Experiment with Computer Graphics for House Design". Ce projet est lui-même un morceau de l'axe de recherche de l'ArchMach "Machine Recognition and Inference Making in Computer Aids to Design". Cette collaboration prolongée résulte dans le développement du programme YONA, qui doit guider ses utilisateurs dans la planification de leur propre appartement. Un couple est invité à participer et à utiliser le programme au cours d'un test qui dure huit semaines. Au cours de cet essai, le découpage en étapes du processus de design et la formulation des instructions données par le programme à ses utilisateurs sont modifiées et ajustées jusqu'à ce que l'ensemble soit suffisamment simple à comprendre et facile d'utilisation pour permettre à n'importe qui, sans formation d'architecte ou d'informaticien, de s'en servir. On retrouve là les préoccupations exprimées par Christopher Alexander dans le cadre du projet Gatemaker, et plus largement dans ses recherches, autour de la nécessité d'un processus simple. Ce processus doit être explicité étape par étape pour garantir une clarté et un confort dans le processus de conception et permettre à n'importe quel utilisateur d'explorer ses intuitions. Alexander baptise cette démarche *unconscious design*[67], et développe pour celle-ci la notion de *graphic conversation theory* ("théorie de la conversation graphique"), en écho à la théorie de la conversation sur laquelle Pask construit ses systèmes cybernétiques[68].

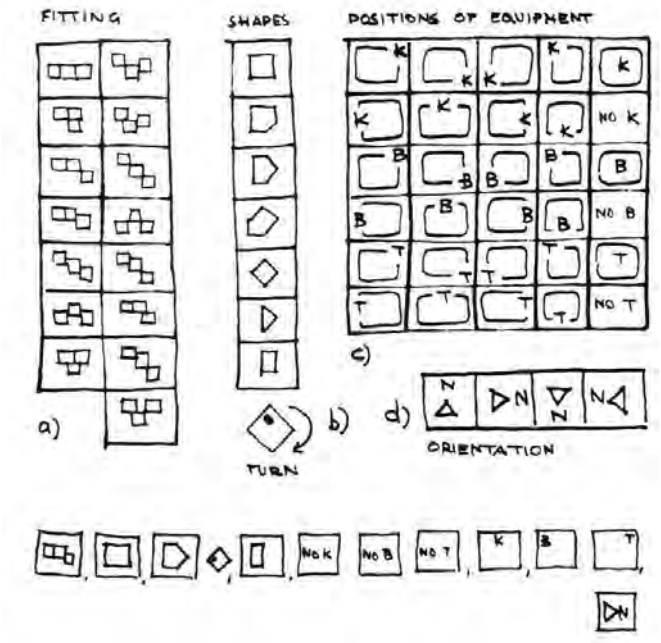



*Figure 7. Opérations proposées par le Flatwriter.*

[67] Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, 1964.
[68] Pask, G., *Conversation, cognition and learning*, New York: Elsevier, 1975.

*2.2.2 Axe 2 : décomposer l'espace*

La rencontre au sein des programmes informatiques de la logique et de l'architecture amène une nécessaire traduction en instructions de programmation des intentions architecturales. Le second axe de recherche qui se développe est l'exploration d'une décomposition des éléments architectoniques en des corpus d'unités combinables entre elles à l'infini. Ces recompositions peuvent s'exprimer facilement en instructions de programmation et offrent donc un espace privilégié pour l'étude des processus de conception computationnels. L'identification du rôle que peut jouer la logique dans la formalisation du processus de conception entraîne le développement de systèmes architecturaux divers. Ces systèmes ont pour ambition commune de permettre l'automatisation de la génération d'espaces, et, pour un même type d'espace, la production de variations infinies. Les propositions développées par les différentes équipes de recherche sont de multiples exemples de ces *grammaires architecturales* et des possibilités formelles qu'elles génèrent. Les systèmes mis au point par William Mitchell et compilés dans son ouvrage *The Logic of Architecture*[69] ou par Bill Hillier et Michael Benedikt dans le cadre de leurs travaux sur la *space syntax*[70] vont plus loin. Au-delà d'éléments architectoniques aux géométries variées, dont la forme entraîne l'existence de règles d'assemblage elles aussi géométriques, il s'agit de tentatives de création de règles s'affranchissant des questions formelles pour s'intéresser aux qualités spatiales. Les matrices de proximité, par exemple, synthétisent pour une liste de pièces données celles qui doivent être reliées entre elles. A partir de ces informations, diverses méthodes de placement dans l'espace permettent de tracer différents plans - chacun d'entre eux ayant une forme différente mais la matrice de proximité étant toujours respectée. Le programme YONA, par exemple, est décomposé en cinq étapes :

- identification des espaces[71] à inclure dans le plan,
- indication des connections existantes entre les différents espaces,
- production d'un graphe de proximité[72],
- ajustement par l'utilisateur du placement des différents espaces,
- dessin d'un diagramme montrant non seulement le placement des espaces, mais aussi leur tailles respectives.

Enfin, le plan définitif est dessiné manuellement par l'utilisateur. Les deux premières étapes sont résumées en une matrice de proximité listant les différents espaces et leurs relations et constituent

[69] Mitchell, W.J., *The Logic of Architecture. Design, Computation, and Cognition*, The MIT Press, 1990.
[70] Hillier, B. et Hanson, J., *The Social Logic of Space*, Cambridge University Press, 1984.
[71] Ou pièces.
[72] Sur ce graphe sont montrés les différents espaces placés les uns par rapport aux autres.

ainsi un ensemble d'informations abstraites au sujet du projet en cours de conception. Dans cette phase ces informations sont déconnectées du placement dans l'espace et de la géométrie spécifique des éléments du plan. C'est précisément ce genre d'information, exprimable autrement que par des représentations visuelles, par des relations logiques ou des équations mathématiques, que ce deuxième axe de recherche a pour objectif de formaliser. L'objectif de formalisation s'applique à l'ensemble des dimensions qu'implique la conception architecturale, de la géométrie aux usages en passant par les qualités lumineuses ou les performances structurelles et thermiques. Étant donné les capacités informatiques disponibles à l'époque, seuls des modèles encore relativement simples peuvent cependant être développés. Les recherches se concentrent donc plus particulièrement sur la hiérarchisation des composants architecturaux entre eux et sur le placement des éléments d'un plan dans l'espace. C'est dans le cas de l'utilisation par Bernholtz du programme HIDEC d'Alexander pour concevoir des logements, à travers un processus assez similaire à celui du programme YONA. Bill Hillier rassemble nombre des techniques d'analyse de l'espace développées à cette période sous le nom de *space syntax*, proposant une approche unifiée basée sur la prise en compte de la dimension sociale de l'architecture que son équipe applique en particulier à des projets d'urbanisme. L'objectif, tel qu'exprimé par nombre des praticiens de l'époque, est de fournir une description mathématique de l'architecture. Quelques rares praticiens, par exemple Paul Coates à l'UEL, parlent déjà de processus procéduraux, actant la contrainte technique représentée par l'usage d'un ordinateur.

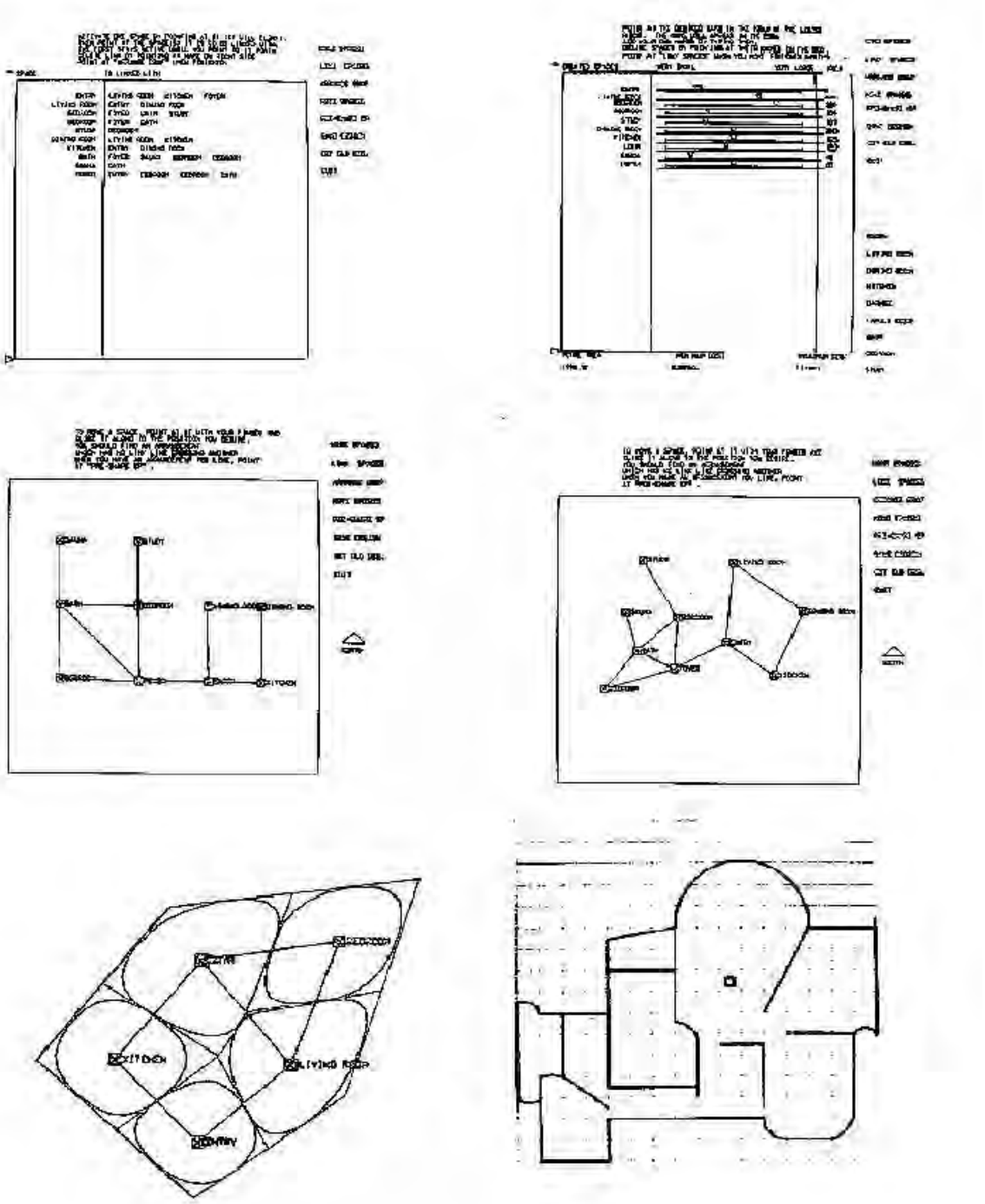


*Figure 8. Visualisation lors des différentes étapes du programme YONA.*

### *2.2.3 Interroger la nature du processus de conception*

Le découpage de l'espace en sets de données finis et règles d'inférence compréhensibles par un ordinateur est impératif pour l'automatisation du processus de conception architecturale, celle-ci étant elle-même une condition nécessaire aux ambitions de l'ArchMach et des autres. Le second axe de

recherche se penche donc sur l'une des questions fondamentales soulevées par la possibilité d'une architecture computationnelle. En effet, pour parvenir à entretenir une conversation avec un programme au sujet d'un projet d'architecture en cours, encore faut-il apprendre à une machine par le biais de ce programme ce qu'est l'architecture. Pour ce faire, il faut parvenir non seulement à définir ce qu'est l'architecture, mais aussi à le formuler dans un langage assimilable par un ordinateur. Au-delà des expériences sur la décomposition formelle ou abstraite d'un objet architectonique, ces explorations ouvrent des questions épistémologiques beaucoup plus profondes sur la nature du processus de conception et par extension sur la nature de la pensée humaine. Les influences mutuelles entre le champ de l'intelligence artificielle et l'architecture comme le design computationnel ne sont pas un hasard : l'architecture est perçue comme un exercice particulièrement représentatif de la diversité des activités de l'esprit humain. La décomposition en sous-programmes évoquée pour des raisons techniques plus haut met en évidence la nécessité d'une réflexion sur la nature de ce processus. Elle facilite de plus cette réflexion par le découpage en plus petits éléments et la définition de ces derniers. Les instructions développées par Christopher Alexander pour le Gatemaker, ou par l'ArchMach pour le programme YONA, défrichent des premières possibilités de description du processus. Les travaux de Bernholtz avec le programme HIDEC constituent autant de tentatives de représentation de ce même processus de conception, de cartographie des connaissances inhérentes à la prise de décision architecturale pour ensuite les rendre lisibles par un ordinateur. Le rôle que joue le contexte dans la variabilité des pratiques et des projets et l'influence de cette variabilité sur une modélisation éventuelle du processus de conception sont vite identifiés. La question est perçue comme un élément décisif par Negroponte, qui s'interroge tout particulièrement sur la manière de prendre en compte ce contexte toujours changeant dans les machines à architecture de l'ArchMach. Le groupe ne parvient cependant pas pour autant à faire autrement que de laisser l'utilisateur des machines prendre les décisions sur ce point[73].

L'enjeu occupe aussi un rôle central dans la pensée de Christopher Alexander, qui développe d'abord dans le cadre de sa thèse une analyse de l'impossibilité à caractériser suffisamment et explicitement un contexte donné, malgré la nécessité pour le processus de conception d'une description aussi complète que possible de ce contexte[74]. Puis, dans le projet de l'Oregon Experiment, un exercice de planification urbaine participative pour le campus de l'université d'Oregon, il pose l'idée de diagnostic urbain par le ressenti, qu'il tente de définir comme une mesure objective[75]. Alexander défriche ainsi la notion de ressenti à laquelle il va par la suite donner une place majeure dans sa

[73] Nicholas Negroponte interrogé par Nick Axel, *Digital x Nicholas Negroponte : Terms and Conditions,* 2019 : https://www.e-flux.com/architecture/digital-x/260419/terms-and-conditions/, consulté le 12 Novembre 2021.
[74] Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, 1964.
[75] Alexander, C., Silverstein, M., Angel, S., Ishikawa, S., et Abrams, D., *The Oregon Experiment*, Oxford University Press, 1975.

compréhension du processus de conception et dans ses travaux ultérieurs. L'interface pensée pour le Gatemaker en 1996 a pour objectif de rendre intelligible à n'importe quel utilisateur le processus de conception. Mais cette interface se veut aussi suffisamment confortable pour se faire oublier, et donc permettre à l'utilisateur de concevoir en restant en lien avec son propre ressenti. Alexander considère en effet que cette connexion disparaît lorsqu'il faut formaliser des instructions lors du recours à un programme informatique usuel. Or si l'interface est suffisamment bien pensée pour permettre à l'outil numérique de se faire oublier, alors l'utilisateur peut se concentrer sur son ressenti. C'est justement ce ressenti qui est source selon Alexander d'une architecture de qualité. Autre prérequis à la production d'une architecture de qualité selon lui, la prise en compte du contexte est favorisée au sein du projet du Gatemaker. Le traçage du portail se fait en effet sur des photos de lieux plutôt que sur un fond vierge comme c'est le cas dans tout autre outil de dessin numérique. Le dessin est accompagné de l'invitation dans les instructions à imaginer les ornements de ce portail comme une passerelle vers l'environnement permettant une intégration harmonieuse.


*Figure 9. Interface du Gatemaker, avec, en blanc, le dessin proposé par l'utilisateur.*

Si Negroponte tend à s'appuyer sur des postulats sur la nature du processus de conception sans jamais les poser explicitement[76] dans ses écrits, la plupart des autres praticiens de la période documentent leur travail sur ce plan dans de nombreux textes. Alexander publie d'abord *Notes on the Synthesis of Form* pour sa thèse en 1964, *the Oregon Experiment* en 1975 pour le récit de cette expérience de planification, mais aussi de nombreux autres ouvrages tout au long de sa carrière. Ce travail d'écriture culmine en 2003 avec *The Nature of Order*, supposé condenser l'ensemble de ses réflections en une théorie de l'architecture unifiée[77]. Bernholtz s'intéresse à une mise en application des programmes HIDEC d'Alexander avant tout pour en faire un support d'expérimentation sur la nature de la conception, plus que pour explorer plus avant la décomposition architecturale systématique proposée par ce dernier. Il retranscrit cette ambition et ses résultats dans son article de 1968 *Some thoughts on computer role playing and design*[78].Charles Eastman, considéré aujourd'hui comme un des pères du B.I.M, démarre sa carrière en 1969 avec l'écriture d'un article qui fera date dans le champ des design studies, *Cognitive processes and ill-defined problems: a case study from design*[79]. Cet article est fondateur de la manière d'envisager la question de la conception pour le champ des design studies : il s'agit, comme dans le domaine de l'ingénierie, de résoudre des problèmes. Mais là ou l'ingénierie pose clairement le problème et identifie explicitement des critères d'évaluation permettant de déterminer si la solution proposée fonctionne, le design ne parvient à formuler correctement ni problème ni critères d'évaluation précis, rendant extrêmement difficile la formalisation des méthodes de travail et la modélisation du processus de conception[80].

Une réflexion intense sur la nature du processus de conception architectural s'engage à cette période en parallèle des recherches pratiques. Les programmes conversationnels ayant vocation à accompagner ce processus sont autant de prétextes à la documentation des étapes de réflexion engagées par un architecte ou un designer. Cette documentation a vocation à formaliser pour automatiser. Le développement des théories du design en tant que champ à part entière, notamment la fondation de la Design Research Society par Nigel Cross en 1966, contribue beaucoup à ces tentatives de formalisation, en s'intéressant aux mêmes enjeux. Nigel Cross lui-même, dans le cadre de sa thèse, développe des expériences qu'il appelle des "tests de Turing inversés", et qui prendront plus tard le

[76] Nicholas Negroponte interrogé par Nick Axel, *Digital x Nicholas Negroponte : Terms and Conditions,* 2019 : https://www.e-flux.com/architecture/digital-x/260419/terms-and-conditions/, consulté le 12 Novembre 2021.
[77] Alexander, C., *The Nature of Order: An Essay on the Art of Building and the Nature of the Universe*, Center for Environmental Structures, Berkeley, Californie, 2003.
[78] Bernholtz, A., "Some thoughts on computer role playing and design", in *Ekistics*, Vol.26, n. 157, pp. 522-524, 1968.
[79] Eastman, C., "Cognitive processes and ill-defined problems: a case study from design", in *IJCAI'69: Proceedings of the 1st international joint conference on Artificial intelligence*, pp. 669–690, Morgan Kaufmann Publishers, 1969.
[80] Nous reviendrons au chapitre VII sur la question des problèmes de conception.

nom d'*expériences du Magicien d'Oz*[81]. Ces expériences consistent à fournir un programme pour un projet à un groupe d'architectes, à disposition desquels on met également un ordinateur capable de répondre à toutes leurs questions au sujet du projet. Il s'agit d'observer quelles questions sont posées par les architectes, et comment elles sont formulées ou reformulées en fonction des réponses successives données par l'ordinateur. Ce que le groupe d'architectes ignore, c'est que ce n'est pas à un programme informatique qu'ils posent leurs questions, mais à un deuxième groupe d'architectes. Ces deux groupes d'architectes ne peuvent communiquer que par écrit via "l'ordinateur". Ils se trouvent ainsi forcés à formaliser leur pensée sous une forme pouvant potentiellement être intégrée à un programme informatique existant du type URBAN 5[82]. Si Cross s'intéresse au départ comme les autres à la question informatique[83], au fil de ses expériences il en vient à développer un intérêt beaucoup plus aigu pour le processus de conception en lui-même. Au fur et à mesure de son parcours, il s'éloigne des applications informatiques pour contribuer davantage au développement du champ des design studies, en particulier par le biais de ses activités de promotion au sein de la Design Research Society. Cross se sert d'ailleurs presque dès le départ de l'ordinateur comme d'un prétexte pour poser d'autres questions, ne le considérant par exemple jamais vraiment comme un acteur du processus à part entière, contrairement à Pask ou Negroponte. Comme quelques autres, il se plonge dans l'interrogation du processus de design jusqu'à délaisser la question computationnelle. Il envisage donc les programmes comme des supports de réflexion forçant une formalisation, simples jouets d'expérimentation sur la nature de l'esprit plutôt que véritables objets de recherches en informatique. Comme le dira plus tard Paul Pangaro, professeur de design à Carnegie Mellon dont le travail sur les processus de conception et sur la cybernétique s'inscrit dans la lignée de ces praticiens pionniers, "*les ordinateurs semblaient à l'époque le terreau le plus fertile pour une réflexion sur la nature de la réflexion*"[84].

Après Eastman et Cross, d'autres praticiens, ceux-ci rarement formés comme architectes, contribuent à l'éclosion du champ des design studies. Ces praticiens construisent eux aussi dans un premier temps leurs travaux à partir d'une influence de la cybernétique, proposant des analyses systémiques du processus de design. Ils promeuvent une formation en design dès le cycle secondaire plutôt que comme une spécialisation professionnelle. Les méthodologies de résolution des problèmes de design qu'ils proposent peuvent selon eux être appliquées dans de très nombreuses situations au quotidien et

[81] En référence à l'histoire du magicien d'Oz, ou un homme dissimulé derrière un rideau se fait passer pour un magicien.
[82] Cross, N., "Human and Machine Roles in Computer Aided Design" Thèse de doctorat, Design Research Laboratory, Department of Building, University of Manchester, Institute of Science and Technology, 1974.
[83] Cross N., "Can a machine design?", in *Design Issues*, Vol. 17, n. 4, pp. 44-50, 2001.
[84] "*Computers seemed a more likely substrate for thinking about thinking*", Pangaro, P., "An Idiosyncratic History of Conversation Theory in Software, and its Progenitor, Gordon Pask", in *Kybernetes*, Vol. 30 n.5-6, 2001.

dans de nombreux univers professionnels[85]. Horst Rittel puis Richard Buchanan développent la notion de complexité de la définition des problèmes de design, passant de l'appellation proposée par Charles Eastman *ill-defined problems* ("problèmes mal définis"`) à l'expression *wicked problems* ("problèmes vicieux"), Rittel caractérisant ces problèmes vicieux toujours essentiellement par leur difficulté à être formulés clairement[86]. Si Rittel est professeur de design, Buchanan se situe quant à lui à la frontière entre management, design et gestion de l'information, un mélange des genres et des domaines typique de la période. Nombre des praticiens gravitant autour des champs naissants des design studies et de l'architecture computationnelle présentent en effet ces casquettes multiples. Cette expertise multiple représente un apport extérieur qui leur permet de développer une vision autre du design et de l'architecture, de leur formalisation et de leur traduction en instructions de programmation. Enfin, contributeur majeur au champ des design studies, Donald Schön, philosophe et professeur d'urbanisme au MIT, s'intéresse à partir des années 1970 à la question fondamentale de l'apprentissage du design ainsi qu'aux rôles respectifs du penser et du faire dans cette discipline et ses réflexions culminent avec la publication en 1983 de *The Reflective Practitioner*[87]. Il se penche sur l'enseignement de cette pratique dont il écrit qu'elle ne peut vraiment complètement se formaliser. Schön propose ainsi un autre angle d'analyse qui contribue à définir les contours du champ par une approche légèrement différente puisque qu'il s'oppose à la possibilité d'existence une méthode générale de design - et donc par extension à la possibilité d'automatiser l'architecture.

Malgré l'apparition de cette approche alternative au sein des design studies, l'objectif reste pour la plupart des praticiens du champ computationnel de cette génération d'explorer la question de l'automatisation, prélude à un enjeu de rationalisation des pratiques. Cette rationalisation des pratiques est à l'œuvre *via* l'explicitation nécessaire des processus pour des raisons techniques de communication avec les ordinateurs. Les contraintes techniques de programmation mènent à l'externalisation de tout un ensemble de décisions hors des programmes eux-mêmes. C'est le cas par exemple du dessin final du plan pour YONA et HIDEC ou du choix des usages de l'espace dans le projet Generator. Cette externalisation pousse à laisser de côté ce qui se joue dans les phases empiriques du processus de conception pour se concentrer sur une explicitation appauvrie, cantonnée à l'implémentation informatique d'éléments mathématiques simples, comme dans le cas des matrices de proximité. Deux processus coexistent donc, dont le déroulement est renforcé par le travail parallèle des architectes et des programmeurs dans de nombreux projets. Ceux-ci sont développés par des équipes interdisciplinaires, mais au sein desquelles chaque spécialité s'occupe d'une étape spécifique

[85] Voir par exemple Archer, B., "Design as a Discipline", *Design Studies*, Vol. 1 n. 1, pp.17-20, 1979 ou Cross, N., "Designerly Ways of Knowing", in *Design Studies* Vol. 3 n. 4, pp. 221-227, 1982.
[86] Rittel, H. et Webber, M., "Dilemmas in a General Theory of Planning" in *Policy Sciences*, Vol. 4, pp. 155-169, 1973. Buchanan, R. "Wicked Problems in Design thinking", in *Design Issues*, Vol. 8, n. 2, pp 5-21, 1992.
[87] Schön, D., *The Reflective Practitioner. How professionals think in actio*n, Temple Smith, 1983.

du processus d'implémentation. Toute une partie des écrits autour des design studies de cette période amorce également une formulation de la question dans des termes de rationalisation, par la description de la conception comme acte de résolution d'un problème[88]. L'influence du champ de l'intelligence artificielle favorise cette approche en promouvant l'idée qu'à terme, toute forme de pensée humaine pourra être analysée et rationalisée pour être répliquée par une machine. Il est également question chez beaucoup de praticiens d'appliquer des méthodes scientifiques à la conception architecturale pour l'optimiser ensuite. Le titre de l'ouvrage de Friedman *Pour l'architecture scientifique*[89] ou la conviction d'Alexander qu'il est possible de développer des mesures quantitatives objectives du ressenti permettant de systématiser la production d'une architecture de qualité trahissent cette intention. Car s'il s'agit bien d'automatisation, il s'agit d'automatisation de l'architecture, et d'architecture de qualité. Le processus est donc en tension entre deux pôles, à l'équilibre entre empirisme architectural et formalisation informatique. Les praticiens du champ computationnel élaborent donc des méthodes de travail dont les étapes sont le reflet de cette tension. La prise de décision initiale relève de l'empirique, et sa traduction en instructions informatiques et le calcul par la machine de la formalisation. S'ajoute ensuite éventuellement finalement une dernière étape d'interprétation des résultats fournis par la machine, lieu d'un retour à des choix empiriques.

Bien que la formalisation de processus procéduraux conduise donc à une ébauche de rationalisation de la conception architecturale, l'ampleur des interrogations sur la nature de cette dernière que dévoilent les textes de la période montre une conscience aiguë de l'équilibre existant entre cet impératif d'explicitation et la dimension empirique de l'architecture, vue comme clé de sa qualité. Les moments du processus qui demeurent extérieurs à la machine permettent aux praticiens conscients de l'enjeu que représentent ces étapes de formalisation de préserver un apport empirique, implicite. La structuration en un système cybernétique du Generator, la décomposition en étapes de conceptions de YONA ou du Gatemaker, ou encore l'interprétation en un projet final des résultats produits par la machine à l'issue du processus de calcul, comme en redessinant un plan à partir du graphe de proximité, en sont autant d'exemples. Autre trace de la conscience de cette tension, lorsque le Generator s'ennuie, c'est-à-dire quand son programme *boredom* est activé, c'est parce qu'une configuration donnée est maintenue dans le bâtiment. Ceci peut être interprété comme le fait que les usages ne varient plus, qu'une configuration optimale fixe a été atteinte pour ces usages, mais est pourtant remise en question par les modifications spontanées. On peut interpréter ce dispositif de programmation comme une manière de rechercher d'autres configurations optimales n'ayant pas été mises en évidence par les premières évolutions, de la même manière que les algorithmes génétiques

[88] Nous reviendrons sur cette idée en détail au chapitre VII.
[89] Friedman, Y., *Pour l'architecture scientifique*, Pierre Belfond, 1971.

développés plus tard intègrent des mutations leur permettant de s'extraire des optimums locaux. Mais cela peut également être interprété comme une volonté de conserver une prise de décision subjective dans le processus de conception - qu'elle soit humaine ou non - ce que les écrits de Cedric Price sur le Generator laissent à penser. La décision d'intégrer le programme *boredom* au Generator vient de l'inquiétude de Price que ses utilisateurs ne perçoivent pas toujours l'intérêt d'adapter en continu le bâtiment, ayant l'habitude d'évoluer dans des lieux statiques, et donc n'exploitent pas vraiment toutes ses possibilités de reconfiguration. Le programme *boredom* intègre de plus un principe d'apprentissage basé sur l'idée qu'au bout d'un certain temps, le Generator serait plus à même que ses utilisateurs de savoir ce qui est bon pour eux. Le Generator est donc un vrai doublé entre une approche d'optimisation, et une mise en forme assez facétieuse de l'idée qu'un bâtiment puisse avoir sa propre subjectivité. Le projet est à ce titre très représentatif de l'équilibre recherché par les praticiens de cette première génération. Christopher Alexander, quant à lui, revendique le côté enfantin de l'interface du Gatemaker et des esquisses qui résultent de l'utilisation des programmes. Il explore cette même dimension ludique, mobilisée comme un mécanisme permettant l'équilibre entre subjectivité de l'utilisateur et rationalité du programme. Le Gatemaker témoigne aussi de l'intérêt de cette génération pour la pratique de la programmation comme un acte dans lequel les programmeurs et utilisateurs mettent beaucoup d'eux-mêmes[90]. Un équilibre se crée donc ainsi entre la dynamique de rationalisation amenée par l'exploration de l'automatisation et le savoir-faire des architectes impliqués dans le développement du champ computationnel. Les productions de la période peuvent ainsi être interprétées comme des tentatives de préservation de cet équilibre tout en explorant les possibilités de la computation.

### 2.3 Philosophies de l'automatisation

L'équilibre du champ est aussi nourri par les relations entretenues par le champ computationnel naissant avec le monde de l'intelligence artificielle. La proximité pratique qui vient des travaux de programmation accomplis ensemble et la proximité institutionnelle et géographique encouragent en effet une proximité philosophique entre les deux milieux. Les débuts du champ computationnel en architecture se faisant comme cas d'étude de ce travail de développement de l'IA, le rôle qu'y jouent les modèles théoriques y est prépondérant. Les machines à architecture de Nicholas Negroponte, les motifs de conception de Christopher Alexander ou le savoir du designer de Nigel Cross en sont autant de témoignages. Ces idées sont autant de reflets des préoccupations du champ de l'IA dans lequel s'insère la pratique de l'architecture computationnelle. La partie qui suit offre un détour par le monde de l'intelligence artificielle, pour mieux en comprendre la genèse théorique. Ceci nous permettra

[90] Nous verrons dans le chapitre VI en quoi cela peut s'apparenter à une pratique artisanale.

ensuite de mieux comprendre les pratiques du champ computationnel en architecture dans les chapitres suivants. Le détour que nous proposons ici se fait par ailleurs à travers la retranscription de la pensée d'une petite sélection de philosophes, dont nous avons trouvé les idées et la biographie particulièrement utiles pour mettre en lumière les différents enjeux de la genèse du domaine de l'intelligence artificielle. Un grand nombre d'autres auteurs ont néanmoins traité de ces sujets, et le récit qui suit n'est donc pas exhaustif[91], il est seulement destiné à fournir un point de repère au lecteur pour la suite de l'analyse du développement du champ computationnel.

### *2.3.1 De l'oracle de Turing au symbolisme*

Les théories en discussion à partir des années 1960 dans le monde de l'intelligence artificielle le sont pour parvenir à cadrer comment donner naissance à l'intelligence artificielle à partir des technologies de l'informatique alors disponibles, ou pour trouver lesquelles développer en complément. Pour saisir la nature de ces discussions, il faut d'abord définir l'intelligence artificielle que cherchent à atteindre les praticiens du domaine. Les explications qui suivent sont tirées de l'article consacré à l'intelligence artificielle de la *Stanford Encyclopedia of Philosophy*, et résument les différents enjeux de la quête de l'intelligence artificielle des praticiens des années 1960 à aujourd'hui[92]. La définition qui fait aujourd'hui consensus a été donnée par Peter Norvig et Stuart Russell dans leur ouvrage commun *Artificial Intelligence : A Modern Approach*, qui compte parmi les classiques du domaine depuis sa sortie en 1995 et ses rééditions successives[93]. Les deux informaticiens proposent de cadrer ce qu'est l'intelligence artificielle à partir des objectifs que l'on souhaite atteindre. Selon eux, la majeure partie des définitions proposées par les chercheurs du champ peuvent se résumer sous la forme d'une phrase commençant par "*Le champ de l'intelligence artificielle vise à construire…*"[94]. Ils classent ensuite les définitions de leurs pairs en quatre catégories. Ces quatres catégories correspondent aux phrases suivantes :

- *Le champ de l'IA vise à construire des systèmes qui pensent comme les humains.*
- *Le champ de l'IA vise à construire des systèmes qui agissent comme les humains.*
- *Le champ de l'IA vise à construire des systèmes qui pensent rationnellement.*

---

[91] En particulier, nous avons choisi de laisser de côté les cybernéticiens pour se concentrer sur leurs successeurs directement impliqués dans le domaine de l'intelligence artificielle, mais leur travail a eu un impact sur l'analogie homme-machine, comme nous l'avons vu plus haut.
[92] Bringsjord, Selmer and Naveen Sundar Govindarajulu, "Artificial Intelligence", The Stanford Encyclopedia of Philosophy (Summer 2020 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/sum2020/entries/artificial-intelligence, consulté le 14 Octobre 2021.
[93] Russell, S. & Norvig, P., 1995, *Artificial Intelligence: A Modern Approach*, Saddle River, NJ: Prentice Hall.
[94] Bringsjord, Selmer and Naveen Sundar Govindarajulu, "Artificial Intelligence", The Stanford Encyclopedia of Philosophy (Summer 2020 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/sum2020/entries/artificial-intelligence, consulté le 14 Octobre 2021.

- *Le champ de l'IA vise à construire des systèmes qui agissent rationnellement.*

Les deux premières propositions sont les plus populaires de ces définitions, celles qu'on retrouve le plus dans le débat public et la littérature, des androïdes de *Blade Runner* à la Samantha du film *Her*[95]. Les deux autres sont cependant aussi présentes: c'est ce qu'incarnent les inquiétants ordinateurs super-rationnels de *WarGames* ou d'*Alphaville*, supposés servir de démonstration des dérives potentielles de l'intelligence artificielle[96]. Il est intéressant d'observer les deux critères selon lesquels ces définitions sont formulées. Le premier consiste à déterminer si l'on vise une performance humaine ou si l'on vise une rationalité idéale, l'autre à déterminer si l'on cherche à construire un système qui pense ou qui agit[97]. Il est également intéressant de noter dans ces définitions la place de la rationalité, qui occupe une place prépondérante dans une proposition sur deux. Cette division est le reflet direct de l'établissement de différents modèles de l'esprit humain par les chercheurs du domaine de l'intelligence artificielle, et des expérimentations qui en découlent. Au fil des avancées des recherches, la division entre des modèles de pensée rationnels et des modèles de pensée humains se cristallise. Mais les discussions des années 1960 sont encore un moment de réflexion sur la nature de l'esprit humain, et d'établissement de modèles théoriques qui doivent permettre de l'imiter au mieux. Vouloir reproduire l'intelligence humaine implique en effet de savoir comment un ordinateur fonctionne, mais aussi d'adopter une position philosophique sur la nature de l'esprit humain. L'ordinateur traite des ensembles de symboles et d'opérations logiques. Cela suffit-il à émuler l'esprit humain ? La réponse à cette question relève pour les différents participants aux débats de conviction sur la nature de l'esprit humain et de sa connaissance du monde. Par ailleurs, l'intelligence artificielle est un champ qui s'est formé à partir du champ de la logique, lui-même une branche de la philosophie. Les interrogations philosophiques jouent donc un rôle particulièrement fort dans la naissance de l'intelligence artificielle dans les années 1960[98]. Les pages suivantes retracent les principales controverses du champ au fil des décennies qui suivent, entre recherches sur les capacités d'un ordinateur et débats sur la nature de l'esprit. Ceci nous permettra de saisir les différentes approches en

---

[95] Ridley Scott, *Blade Runner*, 1982. Spike Jonze, *Her*, 2013.

[96] John Badham, *WarGames*, 1983. Jean-Luc Godard, *Alphaville, une étrange aventure de Lemmy Caution*, 1965.

[97] *The answers all fall under a quartet of types placed along two dimensions. One dimension is whether the goal is to match human performance, or, instead, ideal rationality. The other dimension is whether the goal is to build systems that reason/think, or rather systems that act.* Bringsjord, Selmer and Naveen Sundar Govindarajulu, "Artificial Intelligence", The Stanford Encyclopedia of Philosophy (Summer 2020 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/sum2020/entries/artificial-intelligence, consulté le 14 Octobre 2021.

[98] *"AI runs deep into the past, and has always had philosophy in its veins. This is true for the simple reason that computer science grew out of logic and probability theory,[13] which in turn grew out of (and is still intertwined with) philosophy".* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

vigueur dans le champ, et la façon dont elles résultent en la formulation de différents objectifs pour l'intelligence artificielle, entre rationalité et reflet fidèle des humains. Nous examinerons enfin comment l'existence de ces différents objectifs et théories se répercute dans le champ computationnel en architecture.

La controverse principale du champ de l'intelligence artificielle est parfaitement résumée par une métaphore proposée par Alan Turing, celle de l'Oracle. Non content d'être considéré comme le père de l'informatique moderne en raison de la mise au point des machines qui portent son nom[99], Alan Turing est aussi considéré comme un philosophe de l'esprit majeur, ayant écrit entre 1936 et 1950 une dizaine d'articles fondamentaux pour la discipline. Au tout début de sa carrière, Turing s'intéresse au *Entscheidungsproblem*, ou problème de la décision. Puisque ce problème pose la question de savoir si la validité d'un énoncé peut être déterminé mécaniquement, c'est-à-dire décidable, Turing cherche à définir en premier lieu ce qu'est une procédure mécanique. Dans ses propres termes, il cherche à définir une méthode de calcul universelle. Ceci le mène à postuler l'existence de ce qu'il nomme une machine universelle, dont il se sert pour apporter une réponse au problème de la décision[100]. Cette machine universelle, qu'on connait depuis sous le nom de machine de Turing, lui permet de définir ce qui est calculable. C'est ce qui peut être effectué par une machine de Turing opérant seule - une machine *automatique*, selon les mots de Turing[101]. Cette première période du travail de Turing débouche sur l'article *On computable numbers* en 1937, sur ce qu'on nomme aujourd'hui la thèse de

---

[99] La machine de Turing est un modèle de fonctionnement des machines de calcul,, qui permet de définir ce qu'est un processus mécanique.

[100] La machine universelle de Turing équivaut au lambda-calcul défini à la même période par Church, et également aux fonctions récursives de Gödel. Turing explique les propriétés des trois systèmes comme suit : *A function is said to be 'effectively calculable' if its values can be found by some purely mechanical process. Although it is fairly easy to get an intuitive grasp of this idea, it is nevertheless desirable to have some more definite, mathematically expressible definition. Such a definition was first given by Gödel at Princeton in 1934… These functions were described as 'general recursive' by Gödel… Another definition of effective calculability has been given by Church… who identifies it with lambda-definability. The author [i.e. Turing] has recently suggested a definition corresponding more closely to the intuitive idea… It was stated above that 'a function is effectively calculable if its values can be found by a purely mechanical process.' We may take this statement literally, understanding by a purely mechanical process one which could be carried out by a machine. It is possible to give a mathematical description, in a certain normal form, of the structures of these machines. The development of these ideas leads to the author's definition of a computable function, and to an identification of computability with effective calculability. It is not difficult, though somewhat laborious, to prove that these three definitions are equivalent.* Turing, A., 'Systems of logic defined by ordinals', *Proc. Lond. Math. Soc*., Ser. 2, 45: 161–228, 1939. Cité dans Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[101] *'Calculable by finite means' was Turing's characterisation of computability (...) Turing's machine formulation allowed the precise definition of the computable: namely, as what can be done by a Turing machine acting alone. More exactly, computable operations are those which can be effected by what Turing called automatic machines. The crucial point here is that the action of an automatic Turing machine is totally determined by its 'table of behaviour'.* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

Church-Turing[102] et sur les machines universelles, tout ceci ayant des répercussions cruciales en mathématiques et surtout dans le domaine de l'informatique. En effet, son article *On computable Numbers* donne une définition de la calculabilité et une limite à cette notion, sur laquelle repose l'informatique contemporaine[103].

Dès le départ, lorsqu'il se confronte à la question de ce qui est décidable, Turing, bien qu'il se serve de la notion de machine, a pourtant en tête les capacités d'un esprit humain[104]. Certes, la calculabilité est définie par Turing comme ce qui est calculable par des moyens finis, mais il justifie le choix de cette formulation et de la prise en compte des moyens finis par l'idée que "la mémoire humaine est nécessairement limitée"[105]. D'autant que Turing ne s'intéresse initialement à la machine universelle que comme une construction logique théorique. La possibilité d'une machine physique équivalente qui pourrait véritablement effectuer les calculs définis par Turing est formulée initialement par Church, son directeur de thèse et proche collègue. Turing ne se penchera sur cette hypothèse que des années plus tard. Il s'intéresse initialement au problème de la décision, puisque ce dernier, à travers la notion d'indécidabilité, souligne l'existence dans l'esprit humain d'opérations calculables, lorsque l'esprit suit des règles et exécute des instructions, mais aussi d'opérations non calculables lors de la prise de certaines décisions. La trajectoire de Turing se poursuit après l'écriture de *On computable numbers* sur le même sujet : le processus de pensée humain. Mais il bascule de l'examen du processus de l'esprit humain suivant des règles à celui de l'esprit humain plus généralement. Après les bases mathématiques et logiques jetées sur la notion de calculabilité, Turing se penche plus en détail sur ses implications pour le problème corps-esprit[106].

Au cours de son travail sur la machine universelle, Turing démontre qu'il existe des énoncés non décidables par le calcul, c'est-à-dire non calculables. L'arrêt de la machine universelle fait partie de

---

[102] Tout ce qui est calculable est ce qui est calculable par une machine.

[103] *The paper "On Computable Numbers…" (Turing 1936–7) was his first and perhaps greatest triumph. It gave a definition of computation and an absolute limitation on what computation could achieve, which makes it the founding work of modern computer science.* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[104] "*Nevertheless Turing's purpose was to embody the most general mechanical process as carried out by a human being. His analysis began not with any existing computing machines, but with the picture of a child's exercise book marked off in squares. From the beginning, the Turing machine concept aimed to capture what the human mind can do when carrying out a procedure*." Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[105] *'Calculable by finite means' was Turing's characterisation of computability, which he justified with the argument that 'the human memory is necessarily limited.'* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[106] Le problème corps-esprit est une question fondamentale de la philosophie, qui interroge la nature des liens entre le corps et l'esprit, plus particulièrement entre le cerveau et l'esprit..

ces énoncés. Turing conçoit donc plusieurs dispositifs pour permettre à sa machine de continuer à fonctionner même si elle rencontre un énoncé indécidable. Le premier de ces dispositifs est ce que Turing nomme la machine à choix, pour lesquelles l'état (0 ou 1) n'est pas déterminé par un calcul, mais par l'action (la décision) d'un opérateur externe[107]. Une description plus tardive de la machine à choix indique que cette décision peut en fait être effectuée par le choix au hasard d'une décision, plutôt que par l'action d'un opérateur externe[108]. Mais dans sa thèse, publiée en 1938, Turing fait évoluer la machine à choix dans une autre direction, vers un dispositif qu'il nomme machine-oracle. La machine-oracle est une machine capable d'effectuer des opérations non calculables, qui consiste en une machine universelle à laquelle est associé un oracle, qui peut prendre en charge les opérations non calculables. La définition donnée par Turing de cette entité a donné lieu à plusieurs interprétations et suscité nombre de discussions quant à sa nature, tant chez ses contemporains que chez ses successeurs philosophes. L'oracle originel est clairement décrite par Turing comme n'étant pas une machine : *Nous ne nous étendrons pas davantage sur la nature de cet oracle, si ce n'est qu'il ne peut pas être une machine*[109]. Son objectif à travers le développement de ce dispositif est d'étudier le non calculable [110], que Turing associe alors à la notion d'intuition, qu'il définit comme la capacité d'établir des vérités sans suivre de processus mécanique[111]. Certains des collègues de l'époque de Turing vont

---

[107] *The machine is an automatic machine (a-machine) which means that at any given moment, the behavior of the machine is completely determined by the current state and symbol (called the configuration) being scanned. This is the so-called determinacy condition (Section 3). These a-machines are contrasted with the so-called choice machines for which the next state depends on the decision of an external device or operator.*
De Mol, Liesbeth, "Turing Machines", The Stanford Encyclopedia of Philosophy (Fall 2021 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/fall2021/entries/turing-machine/, consulté le 14 Octobre 2021.

[108] *Turing also mentions the idea of choice machines for which the next state is not completely determined by the state and symbol pair. Instead, some external device makes a random choice of what to do next.*
De Mol, Liesbeth, "Turing Machines", The Stanford Encyclopedia of Philosophy (Fall 2021 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/fall2021/entries/turing-machine/, consulté le 14 Octobre 2021.

[109] "*We shall not go any further into the nature of this oracle apart from saying that it cannot be a machine"*
Turing, A., 'Systems of logic defined by ordinals', *Proc. Lond. Math. Soc*., Ser. 2, 45: 161–228, 1939

[110] *In his investigation, Turing introduced the idea of an 'oracle' capable of performing, as if by magic, an uncomputable operation. Turing's oracle cannot be considered as some 'black box' component of a new class of machines, to be put on a par with the primitive operations of reading single symbols, as has been suggested by (Copeland 1998). An oracle is infinitely more powerful than anything a modern computer can do, and nothing like an elementary component of a computer. Turing defined 'oracle-machines' as Turing machines with an additional configuration in which they 'call the oracle' so as to take an uncomputable step. But these oracle-machines are not purely mechanical. They are only partially mechanical, like Turing's choice-machines. Indeed the whole point of the oracle-machine is to explore the realm of what cannot be done by purely mechanical processes.* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[111] *One can only safely say that Turing's interest at this time in uncomputable operations appears in the general setting of studying the mental 'intuition' of truths which are not established by following mechanical processes (Turing 1939, p. 214ff.).* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.
*Turing makes it clear that the 'intuition' being discussed is related to the human act of seeing the truth of a formally unprovable Gödel statement.* Andrew Hodges, "Uncomputability in the work of Alan Turing and Roger

au-delà de cette définition et font directement le lien entre cette notion d'intuition et les facultés mentales des humains (notamment des mathématiciens). En complément des textes de Turing, d'autres réflexions autour de lui contribuent à lier l'oracle et l'esprit humain, à les faire équivaloir bien que ce ne soit pas explicite chez lui initialement, puisque sa définition de l'intuition est circonscrite à ce que nous avons indiqué plus haut[112].

Néanmoins, pendant la guerre, Turing est envoyé à Bletchley Park, où il est chargé parmi d'autres mathématiciens et cryptographes de décrypter les messages saisis par les Allemands sur la machine Enigma. Cette période représente selon Andrew Hodge, qui a consacré de nombreux textes au parcours intellectuel et biographique d'Alan Turing, un tournant pour le chercheur. Les équipes de Bletchley Park, dirigées par Turing, fabriquent des séries de machines universelles capables de trouver les clés de décryptage d'Enigma rapidement en testant de nombreuses solutions. Avec l'EDVAC de Newman, il s'agit là des premiers ordinateurs modernes - qui contrairement au travail de Newman resteront très longtemps un secret militaire. Turing se confronte alors à l'enjeu d'une transposition physique de sa machine. Ce n'est pas la première fois car il a fait quelques essais mineurs avant, mais le travail accompli à Bletchley Park est de bien plus grande envergure, avec des enjeux nationaux bien plus pressants. Alan Turing est par ailleurs entouré des meilleurs cryptographes et mathématiciens que la Grande-Bretagne a pu mobiliser. Il ressort de cette expérience avec une conception très différente de la computabilité de l'esprit[113]. Hodge résume ce changement par l'idée qu'Alan Turing passe de considérations sur la limite des capacités des machines, qu'il exprimait dans ses travaux sur les limites de la calculabilité, à des considérations sur l'étendue des capacités des machines[114]. Hodge attribue ce

---

Penrose", présentation donnée à Hambourg le 06 Octobre 2000 dans le cadre de la conférence Interface 5. https://www.calculemus.org/MathUniversalis/NS/10/hodges1.html, consulté le 14 Octobre 2021.

[112] *However, there is also a possible interpretation in terms of human cognitive capacity. On this interpretation, the oracle is related to the 'intuition' involved in seeing the truth of a Gödel statement. M. H. A. Newman, who introduced Turing to mathematical logic and continued to collaborate with him, wrote in (Newman 1955) that the oracle resembles a mathematician 'having an idea', as opposed to using a mechanical method. However, Turing's oracle cannot actually be identified with a human mental faculty. It is too powerful: it immediately supplies the answer as to whether any given Turing machine is 'satisfactory,' something no human being could do. On the other hand, anyone hoping to see mental 'intuition' captured completely by an oracle, must face the difficulty that Turing showed how his argument for the incompleteness of Turing machines could be applied with equal force to oracle-machines (Turing 1939, p. 173).* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[113] *But more profoundly, it appears that Turing emerged in 1945 with a conviction that computable operations were sufficient to embrace all mental functions performed by the brain. As will become clear from the ensuing discussion, the uncomputable 'intuition' of 1938 disappeared from Turing's thought, and was replaced by new ideas all lying within the realm of the computable.* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

[114] *until that point, the young Turing had been most absorbed by the two questions which are essentially in Penrose's direction, i.e. casting doubt on the mechanizability of mind. (...) It is therefore a striking fact, which I pointed out, that in Turing's later work his central thrust is the exploration of how much machines can do, not of what they cannot, culminating in the 1950 position discussed above.* Andrew Hodges, "Uncomputability in the

changement à ce que Turing a vu du potentiel de ses machines une fois construites à Bletchley Park. Celles-ci sont en effet parvenues à décrypter les messages d'Enigma, alors que tous les efforts humains en la matière ont été vains. La seule manière pour un opérateur humain de parvenir à résoudre des problèmes de cryptographie de cette nature est par ailleurs de procéder comme une machine. Turing semble très marqué par l'étendue des possibilités offertes par les opérations mécaniques. Il formule alors l'idée que *mécanique* ne veut pas seulement dire que l'esprit exécute des séries d'instructions logiques[115]. Turing passe ensuite de l'usage de la notion d'intuition à celle de la notion d'initiative pour qualifier l'activité de l'esprit lorsqu'il ne suit aucune règle préétablie, et considère que l'initiative peut émerger d'un système mécanisable. Il formule alors des propositions qui évoquent les réseaux neuronaux d'aujourd'hui[116].

Les travaux de Turing après la guerre jettent une lumière différente sur le rôle de l'oracle, qui devient alors pour les interprètes des textes de Turing beaucoup moins certain dans sa nature de non-machine. De la même manière que des collègues d'avant-guerre de Turing faisaient le lien entre oracle et

---

work of Alan Turing and Roger Penrose", présentation donnée à Hambourg le 06 Octobre 2000 dans le cadre de la conférence Interface 5. https://www.calculemus.org/MathUniversalis/NS/10/hodges1.html, consulté le 14 Octobre 2021.

[115] *I now go on to the Turing of 1941, who by that time was masterminding the decipherment of naval Enigma messages. Not only had he devised physical machines of enormous logical ingenuity, but there were people carrying out logical and statistical operations 'like machines'. It must have been very striking that human guessing and judgement had been overtaken by mechanical methods to great effect, and Turing must have been perfectly aware that machines and mechanical methods working on the logical data were embodiments of Turing machines. The astonishing power of the 'merely' mechanical did, I believe, influence him towards the shift of view which I previously attributed to the 1936 period. From now on, he was committed to exploring the range of the 'purely mechanical', and it appears that it was now that Turing concluded that the scope of computability was not limited to processes where the mind follows an explicitly given rule.* Andrew Hodges, "Uncomputability in the work of Alan Turing and Roger Penrose", présentation donnée à Hambourg le 06 Octobre 2000 dans le cadre de la conférence Interface 5. https://www.calculemus.org/MathUniversalis/NS/10/hodges1.html, consulté le 14 Octobre 2021.

[116] *He announced ideas for how to achieve this: he thought 'initiative' could arise from systems where the algorithm applied is not consciously designed, but is arrived at by some other means. Thus, he now seemed to think that the mind when not actually following any conscious rule or plan, was nevertheless carrying out some computable process.* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

*These remarks are to my mind evidence of how the post-war Turing, even in a technical report, felt obliged to respond to the implications of Gödel's work and his own 1938/9 discussion of 'intuition.' It was not long before Turing (1948) described the problem of 'intelligent machinery' as that of how to create machines with 'initiative.' This is not the same word as the 'intuition' of 1938/9 but has the same role as describing that what the mind does when not following any apparent rule. In Turing's postwar thought, initiative does not need uncomputable steps; it is as computable as the 'mechanical processes' even though this goes against one's expectations of 'machines.' It is merely necessary to widen the scope away from computations that follow a programmer's explicit plan. To this purpose Turing sketched nets of logical elements which he proposed could have the capacity for training and adaptation by indirect means, rather than by explicit functional design.* Andrew Hodges, "Uncomputability in the work of Alan Turing and Roger Penrose", présentation donnée à Hambourg le 06 Octobre 2000 dans le cadre de la conférence Interface 5. https://www.calculemus.org/MathUniversalis/NS/10/hodges1.html, consulté le 14 Octobre 2021.

caractéristiques opérations non calculables de l'esprit humain[117], d'autres font ensuite le lien entre oracle et machine à partir de la vision d'après-guerre de Turing du potentiel de ses machines universelles. Ceci a été largement défendu par le philosophe B.J. Copeland, qui voit l'oracle comme une super machine, bien plus puissante que nos ordinateurs modernes, mais une machine tout de même. Hodge avance cependant que ceci relève d'une mésinterprétation des textes de Turing et de l'appellation donnée par ce dernier au dispositif machine-oracle. En effet, pour Turing, la machine-oracle est l'assemblage d'une machine et d'un oracle, et non une super-machine avec les capacités d'un oracle comme l'avance Copeland. Hodge note de plus que l'oracle de Turing peut être vu comme un simple outil mathématique, plutôt que soit comme une machine, soit comme un humain. Mais bien que les remarques de Hodge fassent parfaitement sens à la lecture des textes d'avant-guerre de Turing, sa pensée d'après-guerre, ses travaux sur l'initiative et sur des processus proches de ceux des réseaux neuronaux jettent un doute sur sa pensée plus tardive de la mécanisation de l'esprit, qui va au-delà de la question de la calculabilité. Par ailleurs, quelles que soient les interprétations de ses écrits, le flou autour de cette question de la mécanisation de l'esprit au travers du travail de Turing reflète la contradiction à laquelle il se confronte tout au long de ses recherches. Comme le formule Andrew Hodge : *"Comment l'intelligence pouvait-elle naître d'opérations qui étaient elles-mêmes totalement routinières et sans intérêt - "entièrement sans intelligence" ? C'est le cœur du problème auquel Turing a été confronté, et le même problème se pose aujourd'hui à la recherche sur l'intelligence artificielle"*[118]. Parce que le travail d'Alan Turing débouche sur une définition nette de la calculabilité, qui cerne ce que les machines universelles, c'est-à-dire nos ordinateurs, sont capables de faire, sa contribution est cruciale à la compréhension de l'informatique et aux théories de l'esprit qui y sont attachées. La controverse sur le rôle de l'oracle a par la suite des répercussions importantes dans plusieurs domaines : informatique, mathématiques, logique, philosophie. Non seulement le travail d'Alan Turing a des conséquences sur l'informatique pratique, appliquée, mais elle plante aussi les graines d'un débat durable, que la question de savoir si l'oracle est humain ou machine résume parfaitement.

Le développement du champ de l'intelligence artificielle après-guerre, entre autres à partir des travaux d'Alan Turing, ne s'embarrasse pourtant pas de doutes. Les deux courants qui s'affrontent entre 1945

---

[117] Newman mais aussi Roger Penrose plus tard, et peut-être Michael Polanyi, dont il est intéressant de noté qu'il était proche d'Alan Turing.

[118] *How could the intelligent arise from operations which were themselves totally routine and mindless—'entirely without intelligence'? This is the core of the problem Turing faced, and the same problem faces Artificial Intelligence research today. Turing's underlying argument was that the human brain must somehow be organised for intelligence, and that the organisation of the brain must be realisable as a finite discrete-state machine. The implications of this view were exposed to a wider circle in his famous paper, "Computing Machinery and Intelligence," which appeared in Mind in October 1950.* Hodges, Andrew, "Alan Turing", The Stanford Encyclopedia of Philosophy (Winter 2019 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/win2019/entries/turing/, consulté le 14 Octobre 2021.

et 1969 ont beau présenter des différences majeures dans leurs théories et leurs stratégies, ils sont d'accord sur une chose : faire penser une machine, c'est possible. Les premiers contributeurs à la question ne sont pourtant pas tout à fait préoccupés par cette question, puisque ce qui les intéresse, c'est plutôt de modéliser le cerveau pour en comprendre le fonctionnement. C'est ce qui motive l'article fondateur de Warren McCulloch, neuroscientifique, et Walter Pitts, logicien, en 1943 : *A Logical Calculus of Ideas Immanent in Nervous Activity*[119]. Celui-ci présente un modèle de neurone artificiel et un modèle mathématique de réseaux de neurones. Bien que l'objectif initial de McCulloch et Pitts soit de proposer un outil de modélisation en neurologie, leur modèle de réseau de neurones est repris par plusieurs chercheurs dont l'objectif est de produire des machines qui pensent. Oliver Selfridge, mathématicien formé au MIT, y consacre ensuite toute sa carrière. Il y développe le Pandemonium, un système de reconnaissance de motifs. Mais au-delà des applications pratiques du Pandemonium, que Selfridge utilise notamment pour reconnaître des lettres, c'est aussi une architecture logique dont son développeur espère qu'elle est la clé de l'automatisation de mécanismes d'apprentissage[120]. Le Pandemonium est un système composé de plusieurs catégories de "démons". Les premiers sont chargés de la perception de l'image. Les seconds sont associés chacun une caractéristique différente. Pour les lettres, certains sont associés à la caractéristique "petit trait droit", "long trait droit" ou encore "trait courbe". Ces démons sont supposés "hurler" lorsque l'image présente leur caractéristique. Les démons du groupe suivant ont chacun un motif associé. En fonction des différents hurlements du groupe précédents, ils s'activent si cela correspond à leur motif. Dans le cas de la reconnaissance de lettres, "petit trait droit", "long trait droit" et "trait courbe" ensemble correspondent au R, c'est donc le démon chargé de ce motif qui s'y reconnaîtra. Enfin, le quatrième ensemble de démons est chargé de contrebalancer cette déduction avec d'autres facteurs. Dans le cas des lettres, par exemple, si l'image est floue, il faudra écouter d'autres signaux venant du troisième groupe de démons, puisque le R est peut-être une erreur. Le nom et la description du Pandemonium viennent de l'enfer de John Milton, que Selfridge utilise comme métaphore pour décrire son système, qui relève du même modèle mathématique que le réseau de neurones de McCulloch et Pitts. Le Perceptron du psychologue et informaticien Frank Rosenblatt, développé en 1958 à Cornell, est lui aussi utilisé pour de la reconnaissance d'images. Le principe du Perceptron est calqué sur celui des réseaux de McCulloch et Pitts, mais Rosenblatt y ajoute un mécanisme d'apprentissage, en renforçant les connexions qui donnent les résultats les plus pertinents. Pour ce faire, il substitue à l'organisation logique prédéfinie du réseau de McCulloch et Pitts une méthode d'évaluation statistique[121]. Initialement implémenté sur un ordinateur IBM, il a pourtant vocation à être une machine à part

[119] McCulloch, W. S., Pitts, W., A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.
[120] Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993.
[121] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173-220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.

entière. Malgré les difficultés techniques, le Mark I Perceptron est construit en 1967. Il est composé de 400 cellules photoélectriques, et ce sont des moteurs électriques qui permettent d'activer les différents neurones. Il s'agit là d'un dispositif assez proche de celui du réseau neuronal construit en 1951 par Marvin Minsky, mathématicien passionné de sciences cognitives, dans le cadre de sa thèse à Princeton. Minsky, comme beaucoup d'autres, est pourtant rebuté par les limites techniques auxquelles se heurtent les tentatives de fabrication de machines qui pensent grâce à des réseaux de neurones. Son réseau neuronal ne comporte que quarante neurones, et bien qu'il soit convaincu de l'intérêt de ce modèle, il se détourne du sujet durant plusieurs années. Mais la découverte des travaux du mathématicien et informaticien Ray Solomonoff sur la formalisation logique des inférences le fait ensuite basculer vers une autre approche, que d'autres ont déjà commencé à défricher.

En effet, à la même période, un autre groupe de chercheurs s'intéresse aussi à la fabrication de machines qui pensent. Ou plutôt à l'écriture de programmes qui pensent, puisqu'ils se sont déjà tournés vers l'ordinateur, dont ils considèrent qu'il constitue un excellent support de recherche pour parvenir à obtenir des machines qui pensent. John McCarthy, condisciple de Marvin Minsky à Princeton, se prend lui aussi d'intérêt pour la question, et décide d'organiser avec plusieurs collègues deux mois de travail sur la question à l'Université de Dartmouth, où il est alors assistant professeur. C'est, à l'été 1956, la célèbre Conférence de Dartmouth, que l'on considère aujourd'hui comme le moment de la naissance de l'intelligence artificielle. Plus exactement, c'est le moment où John McCarthy forge le terme, mais les participants sortent peu impressionnés de cet atelier de deux mois. À aucun moment les dix invités n'ont été présents en même temps, et ils ne sont pas vraiment d'accord sur leurs ambitions ni leurs stratégies[122]. Pamela McCorduck, auteure d'un livre de référence sur l'histoire de l'intelligence artificielle, argue cependant que l'importance de la conférence réside dans la constitution du groupe. Malgré les divergences d'intérêt, chacun se rend en effet compte à Dartmouth qu'il fait partie d'une communauté de chercheurs qui ont pour objectif commun la mise au point d'une machine intelligente[123]. C'est le moment de l'émergence d'un réseau, qui par la suite donne naissance au champ de l'intelligence artificielle, qui se constitue autour de dix chercheurs présents ce jour-là : John Mccarthy, Marvin Minsky, Claude Shannon, Nathaniel Rochester, Herbert Simon, Allen Newell, Trenchard More, Arthur Samuel, Oliver Selfridge et Ray Solomonoff. Chacun d'entre eux évolue par ailleurs dans un des centres de développement du champ : I.B.M, la RAND, Bell Telephone Laboratories, Harvard, le MIT ou Stanford. Autre élément important de la conférence Dartmouth, la présentation par Allen Newell et Herbert Simon du programme sur lequel ils travaillent alors, qu'ils viennent de parvenir à faire fonctionner : le Logic Theorist. Si leur présentation fait peu

[122] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[123] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

d'effet aux autres sur le moment, le Logic Theorist est pourtant le premier représentant fonctionnel du paradigme qui deviendra dominant dans le champ par la suite : l'approche symboliste. Les dix pionniers, en structurant autour d'eux de manière presque clanique le domaine de l'intelligence artificielle[124], verrouillent en effet au fil de la décennie suivante la recherche autour du paradigme introduit par Newell et Simon comme le seul valide pour de nombreuses années.

Allen Newell et Herbert Simon se rencontrent à la RAND, d'abord pour quelques discussions brèves puis rapidement pour plusieurs années de collaboration intensive. Allen Newell est un mathématicien formé à Princeton, où son travail entre 1949 et 1950 le convainc très vite qu'il n'est pas fait pour les mathématiques fondamentales. Attiré aussi bien par la renommée scientifique croissante de l'organisation que par la possibilité qui lui y est offerte de résoudre des problèmes concrets, il rejoint la RAND. Celle-ci est alors encore complètement gérée par l'armée de l'air américaine, qui lui fournit un budget de recherche très conséquent et une liberté complète. Newell est chargé de travailler sur divers problèmes logistiques, en particulier la modélisation de la prise de décision chez les opérateurs surveillant les survols d'appareils inconnus. Il faut en effet déterminer s'il est nécessaire d'envoyer un avion de l'armée de l'air identifier l'appareil, ou si le risque est faible et que la dépense n'en vaut pas la peine[125]. Newell doit donc rationaliser les logiques de décision à l'œuvre. Herbert Simon est un spécialiste du management et de la modélisation des organisations dont le travail dans le domaine est déjà très reconnu et auquel Newell fait appel au bout de deux ans à la RAND. Ensemble, ils travaillent sur le problème du centre de contrôle aérien pendant un temps, avant de se tourner vers l'élaboration d'un modèle de prise de décision plus général. Pour Simon, la découverte du travail de Newell est très importante. Observer la manière dont Newell parvient à faire afficher la positions des avions détectés par les radars sur papier lui fait prendre conscience que les ordinateurs sont capables d'accomplir un travail non numérique. Ceci souligne pour lui que les ordinateurs ne sont pas simplement de grosses calculettes, mais des processeurs de symboles[126]. C'est ce qu'il nomme les capacités de fonctionnement symbolique[127] de l'ordinateur, les symboles pouvant représenter n'importe quoi : chiffre, mots, objets ou notions du quotidien. Leur rencontre renforce par ailleurs la conviction de Simon que l'esprit est une machine logique, et que les organisations, les interactions et les décisions humaines qu'il étudie peuvent être modélisées sur un ordinateur. C'est à partir de cette idée qu'ils développent ensemble ce qu'ils nomment le *information-processing model* (modèle de traitement de l'information), et c'est en raison de cette intuition et du succès qui couronne les essais qu'ils effectuent à partir de cette intuition que le travail de Newell et Simon est clé pour le nouveau

[124] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004. Même si l'auteur excuse un peu ce verrouillage, elle en fait tout de même le constat.
[125] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[126] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004, p. 152.
[127] "symbolic-functioning capacities"

paradigme.

Le Logic Theorist est le premier des programmes qu'ils développent. Il s'agit d'un programme permettant de trouver des preuves pour les théorèmes des *Principia Mathematica* d'Alfred North Whitehead et Bertrand Russell, un ouvrage de 1910 aux fondements de la logique moderne. Le Logic Theorist est programmé sur le Johnniac, un ordinateur primitif - nommé en hommage à John von Neumann dont il utilise l'architecture informatique - construit par la RAND et mis à disposition de ses chercheurs[128]. Newell et Simon reçoivent pour cela l'aide de Cliff Shaw, qui les avait déjà aidés à programmer une simulation des prises de décisions du centre de contrôle aérien. Newell fait une partie du travail de structuration du programme et Simon sait lui aussi à peu près programmer[129], mais c'est Shaw qui se charge de la majeure partie de la mise en œuvre du Logic Theorist. À la conférence de Dartmouth, Newell et Simon refusent l'appellation intelligence artificielle pour leur programme, lui préférant leur nom de *information-processing model*. Daniel Crevier, auteur d'une histoire du champ, considère cependant le Logic Theorist comme le premier programme d'intelligence artificielle, un avis partagé depuis lors par la majeure partie des contributeurs au champ[130]. Non seulement le Logic Theorist fonctionne très bien, retrouvant les preuves données par Whitehead et Russell pour leurs différents théorèmes, mais il trouve même une preuve qu'ils n'avaient pas donnée, plus courte que celle qu'ils indiquent dans les *Principia Mathematica*. Ceci ajoute d'autant plus à l'idée d'intelligence du programme. L'incertitude du résultat en sortie contribue en effet à donner l'image de l'intelligence. L'ordinateur exécute certes les instructions données par le programme, des instructions qui sont donc connues, mais il arrive que les programmeurs soient ensuite pris de court par les résultats[131].

Le Logic Theorist, qui vient tout juste de produire ses premières preuves, n'impressionne pourtant pas grand monde à Dartmouth. Les réseaux de neurones et le connexionnisme sont alors considérés comme les pistes les plus prometteuses, ce qui relativise peut-être l'importance de l'accomplissement de Newell et Simon. Pamela McCorduck avance aussi l'hypothèse d'une certaine mauvaise grâce des autres participants, légèrement vexés de voir deux chercheurs qu'ils avaient pourtant un peu hésité à inviter les prendre de court et présenter une version fonctionnelle de ce qu'ils ont tant de mal à

[128] Le trio de chercheurs crée un langage spécifique pour l'occasion, nommé ILP-I, puis ILP-II dans sa seconde version.
[129] Dans un entretien avec Patricia McCorduck, il indique avoir appris en un été essentiellement parce que c'était la chose à faire, presque la chose à la mode, plus que par intérêt ou nécessité. McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[130] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993, p.27. Newell indique néanmoins a posteriori que d'autres programmes écrits avant comme celui d'Oliver Selfridge sur la reconnaissance de motifs relèvent aussi de l'IA.
[131] Une caractéristique très appréciée des architectes du champ computationnel comme de beaucoup d'autres. Nous y reviendrons dans les chapitres ultérieurs.

réaliser, de surcroît avec une grande fierté[132]. Pamela McCorduck retranscrit également le témoignage de Marvin Minsky, qui émet l'hypothèse que Newell et Simon n'aient pas fait grande impression à Dartmouth parce qu'ils "parlaient comme des psychologues"[133]. Newell et Simon présentent en effet le Logic Theorist et sa suite, le General Problem Solver, qu'ils ont déjà en tête, comme de bonnes imitations du comportement des humains. Or Minsky explique qu'il n'envisageait pas alors l'intelligence artificielle comme devant imiter l'humain, mais simplement comme une machine intelligente, et que l'intervention de Newell et Simon n'a alors suscité que peu d'intérêt chez lui[134]. Cela montre que la période qui suit voit aussi un changement dans la question de l'imitation ou non de l'humain pour les programmes d'intelligence artificielle, et que la tension entre rationalité et imitation de l'être humain, décrite dans la définition de l'IA donnée par Norvig et Russell évoquée plus haut, est là depuis les tout débuts du champ. La réception de leur travail ne décourage néanmoins pas Newell et Simon, qui s'attaquent à la programmation d'un nouveau système : le General Problem Solver. L'expérience de Dartmouth les convainc en effet d'autant plus que ce qu'ils cherchent à faire, ce n'est pas de tenter d'exploiter les capacités de l'ordinateur comme leurs collègues, mais de s'en servir pour modéliser l'esprit humain. Leur ambition prend racine dans des recherches en psychologie récentes, qui montrent que le processus de pensée humain n'a rien à voir avec ce qu'ils ont implémenté dans le Logic Theorist. Ils décident alors de reproduire les expériences en psychologie sur lesquelles sont basées ces conclusions, qui consistent à poser à des personnes des problèmes de logique similaires à ceux posés au Logic Theorist. Alors que ces personnes s'attaquent à la résolution des problèmes, il leur est demandé d'exposer le cheminement de leur pensée tout haut pour pouvoir la documenter. C'est à partir de ces données que Newell et Simon élaborent l'architecture du General Problem Solver, toujours avec l'aide de Cliff Shaw. Sa caractéristique principale est de séparer la base de données de sa stratégie de résolution des problèmes. Ainsi, quel que soit le problème posé, le General Problem Solver peut appliquer sa stratégie, pour peu que les données du problèmes aient été formalisées et saisies sous une forme lisible par le programme. Pour peu qu'elles aient été traduites, donc.

Pendant que Newell et Simon se remettent au travail sur leur General Problem Solver, les années qui suivent la conférence de Dartmouth sont un moment de changement dans le champ, qui débouche sur la reconnaissance de leur approche par les huit autres participants comme par l'ensemble des chercheurs en intelligence artificielle. Plusieurs facteurs contribuent à cette reconnaissance. D'abord, les machines de l'approche connexionniste s'avèrent difficiles à construire en raison des limites

[132] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[133] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[134] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004. Minsky ajoute également que les travaux de Newell et Simon étaient de toute façon basés sur des principes qu'il avait déjà théorisés avant.

techniques de l'époque. Ensuite, l'établissement par un nombre croissant de chercheurs d'un lien entre ordinateur et intelligence artificielle, et le recours croissant à l'ordinateur pour leurs travaux, renforce l'idée d'une intelligence artificielle symboliste et précipite la fin du connexionnisme. Dans les grands centres de l'intelligence artificielle, les tenants du connexionnisme sont peu à peu supplantés par des symbolistes. Les connexionnistes qui avaient assisté à la conférence de Dartmouth se tournent peu à peu eux aussi vers l'approche symboliste, dont Newell et Simon ont apporté la preuve qu'elle permet d'obtenir des résultats bien plus convaincants. Et puisque les membres de la conférence de Dartmouth sont parmi les plus influents du champ, ils entraînent la majeure partie de ses praticiens avec eux dans cette volte-face. Parmi eux, Marvin Minsky joue un rôle crucial. En 1963 d'abord, il publie *Steps Towards AI*, un rapport qui fait partie d'une série sur l'état du champ, *Working Models of AI*[135]. L'ensemble de ces rapports mentionnent le General Problem Solver de Newell et Simon comme témoin majeur de l'approche à adopter, mais c'est celui de Minsky qui aura le plus d'écho. Six ans plus tard, il publie le livre *Perceptrons* avec Seymour Papert, mathématicien et informaticien devenu à partir de 1963 son collègue et complice au MIT[136]. Ils y démontrent que l'architecture des Perceptrons de Frank Rosenblatt ne peut calculer la fonction OU exclusive, un opérateur logique pourtant essentiel. Papert et Minsky y soulignent plusieurs autres limites des Perceptrons, et bien qu'ils proposent des architectures de réseaux neuronaux alternatives qui pourraient permettre d'atteindre l'intelligence artificielle, ils soulignent aussi qu'une partie des problèmes ont déjà été résolus par d'autres méthodes. Des méthodes qui relèvent de l'approche symboliste. Le livre a un impact désastreux pour l'approche connexionniste, qui n'est déjà plus très en vogue, et l'enterre pour 30 ans [137]. L'approche symboliste devient alors le paradigme dominant jusque dans les années 1990 pour la quête de l'intelligence artificielle.

Dans leur article *La revanche des neurones*, Dominique Cardon, Jean-Philippe Cointet et Antoine Mazières résument les deux approches[138]. Pour les symbolistes, "penser, c'est calculer des symboles qui ont à la fois une réalité matérielle et une valeur sémantique de représentation". Pour les connexionnistes, "penser s'apparente à un calcul massivement parallèle de fonctions élémentaires". Autre différence entre les deux, le symbolisme travaille à partir de la séparation entre software et hardware, se concentrant exclusivement sur les modalités logiques du traitement de l'information,

[135]Minsky, M. "Steps toward Artificial Intelligence," in Proceedings of the IRE, vol. 49, no. 1, pp. 8-30, Jan. 1961, doi: 10.1109/JRPROC.1961.287775.
[136] Papert, S. & Minsky, M., *Perceptrons*, MIT Press, 1969 (Enlarged edition, 1988), ISBN 0-262-63111-3
[137] Les historiens de l'IA assignent en tout cas au livre de Minsky et Papert un rôle très important dans l'effondrement de l'approche connexionniste. Voir chez Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993 et chez McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[138] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173-220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.

alors que les connexionnistes sont persuadés de la nécessité du couplage entre traitement de l'information et processus physiques, dont les processus biologiques[139]. Mais dans les deux cas, penser, c'est calculer. La pensée est donc mécanisable, et l'Oracle de Turing une machine. Les idées des uns nourrissent par ailleurs les idées des autres pendant tout l'âge d'or de l'intelligence artificielle, entre 1956 et 1974[140]. Par exemple, la présentation en 1954 à la RAND par Oliver Selfridge de ses travaux sur ce qui devient plus tard le Pandemonium fait très forte impression sur Allen Newell. Pour ce dernier, l'idée d'une addition de processus simples faisant émerger des résultats complexes est une grande source d'inspiration, qu'il décrit comme un tournant dans son approche du modèle de traitement de l'information[141]. La conférence de Dartmouth a beau être le point de départ d'un rejet des théories connexionnistes, notamment au vu du peu de résultats qu'elles donnent, ces travaux pionniers ancrent déjà l'idée que le cerveau est une "meat machine"[142]. Seule différence entre connexionnisme et symbolisme, le modèle de machine. L'analogie entre esprit et machine est une conviction partagée par tous dans le domaine et soutenue avec fougue. Indice de la conviction des chercheurs en intelligence artificielle de la validité de leur modèle, les nombreuses et très ambitieuses annonces de l'arrivée imminente de programmes qui pensent au fil des années 1960 et 1970. Dans son article *Computing Machinery and Intelligence*, écrit en 1950, Alan Turing estime ainsi que 70% des machines feront preuve de comportements intelligents très rapidement[143]. En 1958, Allen Newell et Herbert Simon affirment que d'ici dix ans, les ordinateurs battront les humains aux échecs et pourront découvrir de nouveaux théorèmes mathématiques[144]. En 1965, Simon annonce que dans les vingt années à venir, les machines pourront accomplir n'importe quelle tâche humaine[145]. En 1967, Marvin Minsky clame que d'ici un génération, la création de l'intelligence artificielle sera un problème résolu [146], et en 1970, qu'il ne faudra que trois à huit ans pour voir apparaître une machine aussi intelligente qu'un être humain[147].

---

[139] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.
[140] L'âge d'or est le nom donné par les historiens à cette première période de développement intensif de recherches très libres pour l'intelligence artificielle. Voir Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993 et McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[141] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004, p. 157.
[142] "machine de viande" McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004, p. 85.
[143] Turing, A., 'Computing machinery and intelligence', *Mind*, 50: p. 433–460; 1950.
[144] Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993
[145] Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993
[146] Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993
[147] Crevier, D., *AI: The Tumultuous Search for Artificial Intelligenc*e. New York, NY: BasicBooks, 1993

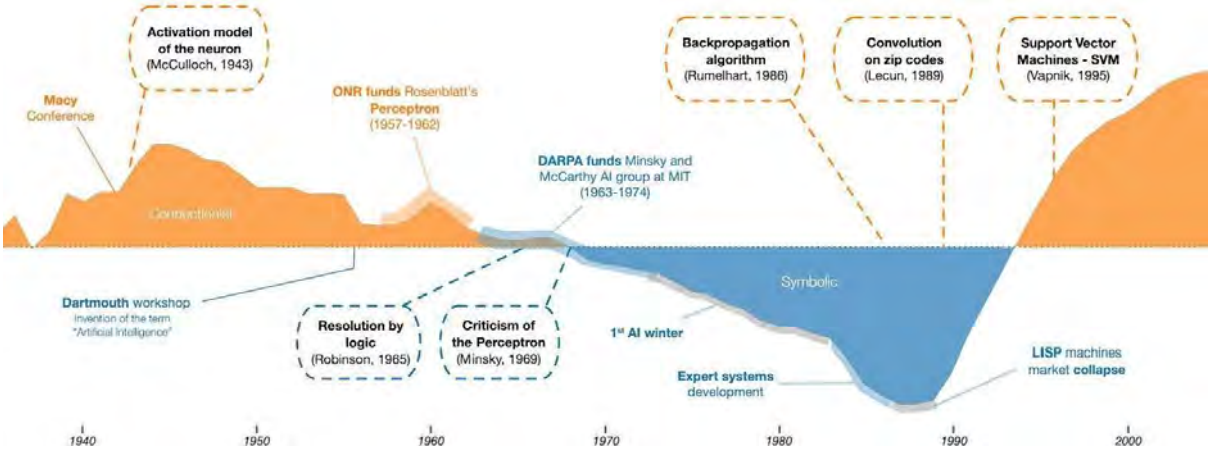



*Figure 10 - Chronologie des paradigmes dominant la recherche en intelligence artificielle. Source : Dominique Cardon et al., La Revanche des Neurones*[148].

### *2.3.2 Des savoirs logiques au monde des perceptions*

Avec son recours à la logique et sa séparation entre software et hardware, la victoire du symbolisme sur le connexionnisme met plus que jamais les séquences d'opérations logiques au cœur des essais de simulation de la pensée par des machines. Mais en dépit de l'enthousiasme des tenants de cette théorie et des chercheurs en intelligence artificielle, ce modèle ne convainc pas tout le monde. L'approche symboliste est de plus en plus vivement critiquée par les philosophes, dont plusieurs établissent des réfutations majeures de ses présupposés[149]. John Searle, spécialiste des questions philosophiques que soulèvent les tentatives de mécanisation de la pensée humaine, basé à l'Université de Berkeley de 1959 à sa retraite en 2019, développe à partir de 1980 une expérience de pensée visant à interroger la capacité d'une machine à comprendre les symboles qu'elle manipule[150]. Cette expérience, qui porte le nom de chambre chinoise, est conçue par Searle comme une réponse au test de Turing. Celui-ci a été proposé en 1950 par Alan Turing pour permettre d'évaluer la capacité d'une machine à penser, et est encore aujourd'hui perçu comme un des moyens principaux de déterminer si une machine a franchi le cap de l'intelligence artificielle. Le test de Turing consiste à placer un examinateur face à deux entités, avec lesquelles il peut communiquer par écrit et leur poser des questions. L'une de ces entités est une machine, l'autre un humain, mais l'examinateur ne sait pas laquelle des entités est la machine et laquelle est humaine - c'est ce qu'il doit déterminer par le biais des échanges écrits avec les deux entités. Si la machine parvient au cours des échanges à passer pour humaine aux yeux de

[148] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.
[149] Ls difficultés techniques rencontrées vont aussi jouer. Nous les aborderons au chapitre suivant.
[150] Searle, J., 1980, 'Minds, Brains and Programs', *Behavioral and Brain Sciences*, 3: 417–57

l'examinateur, alors elle réussit le test de Turing, que Turing décrivait comme un *jeu de l'imitation*[151]. Celui-ci fait figure de référence dans le monde de l'intelligence artificielle et a suscité jusqu'à aujourd'hui de nombreuses discussions : Turing avait lui-même identifié neuf contre-arguments qu'il est possible d'opposer au test, des contre-arguments qui font l'objet d'une réponse dans ses écrits. D'autres chercheurs ont proposé des variantes au test, le considérant comme dépassé, trop facile ou trop difficile[152], ou voulant simplement évaluer un autre aspect, comme on l'a vu avec le test du magicien d'Oz, ou test de Turing inversé, de Nigel Cross.

Bien que Alan Turing ait donné dès 1950 une réponse à nombre des objections faites à la pertinence du jeu de l'imitation, un grand nombre d'autres réfutations ont été formulées au fil des années, la *chambre chinoise* décrite par John Searle étant l'une des plus citées. Celle-ci cherche à montrer qu'imiter n'est pas comprendre, et propose une métaphore visant à mettre en évidence le fait qu'un ordinateur ne fait que traiter des symboles abstraits au cours des calculs qu'il effectue, sans saisir ce que ces symboles représentent. Searle imagine une personne dont la langue maternelle est l'anglais, assise dans une pièce avec des boîtes remplies de feuilles de caractères chinois, et un livre d'instructions pour la manipulation de ces caractères[153]. Si une autre personne, hors de la pièce, fait parvenir à la personne qui est à l'intérieur une question écrite en chinois, alors la personne à l'intérieur de la pièce peut utiliser le livre d'instructions et les feuilles de caractères pour écrire une réponse à la question. La personne à l'extérieur, ayant fourni une question en chinois et reçu une réponse dans la même langue, peut donc en conclure qu'à l'intérieur se trouve quelqu'un qui parle couramment le chinois. Searle assimile cette conclusion à la réussite du test de Turing par cette chambre chinoise, alors même que la personne qui se trouve à l'intérieur ne maîtrise pas du tout la langue et ne fait que suivre des instructions donnant l'illusion de sa compréhension. Les différents composants de l'expérience représentent l'ensemble des éléments auxquels fait appel une machine traitant de l'information : la personne à l'intérieur correspond au processeur, les boîtes remplies de feuilles de caractères chinois correspondent à une base de données, le livre d'instructions pour la manipulation de ces caractères à un programme, la question écrite en chinois à des données d'entrée, et la réponse à la question à des données de sortie. John Searle, qui a aussi beaucoup contribué à la linguistique et à la philosophie du langage au cours de sa vie, pointe à travers la chambre chinoise la différence entre syntaxe et sémantique, et l'incapacité de la syntaxe seule à produire de la compréhension[154]. Et si

[151] Turing, A., 1950, "Computing Machinery and Intelligence," Mind, 59 (236): 433–60.
[152] Oppy, Graham and David Dowe, "The Turing Test", The Stanford Encyclopedia of Philosophy (Winter 2020 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/win2020/entries/turing-test/>.
[153] Il ne s'agit pas d'une grammaire ou d'un dictionnaire, simplement d'instructions pour associer des caractères entre eux.
[154] Cole, David, "The Chinese Room Argument", The Stanford Encyclopedia of Philosophy (Winter 2020 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/win2020/entries/chinese-room/>

l'ordinateur (ou la chambre chinoise) manipule bien une syntaxe pour communiquer avec son utilisateur, il n'associe pas de signification aux mots ou symboles employés, et ne mobilise donc aucune sémantique. Or c'est en associant une signification aux symboles que nous comprenons ce dont il est question et, pour Searle, que nous faisons preuve d'intelligence. Ainsi le test de Turing ne peut vraiment permettre de mesurer l'intelligence de la machine qui y est soumise, puisqu'il ne fait qu'évaluer sa capacité à manipuler une syntaxe donnée. L'expérience proposée par Searle vise donc également à montrer que la maîtrise du langage ne peut suffire comme mesure de l'intelligence, puisqu'il peut facilement s'imiter une fois les règles formelles saisies. Searle décrit donc l'ordinateur comme faisant l'exact opposé de ce que Polanyi donne comme exemple de l'existence de connaissances tacites : contrairement à un enfant apprenant à exprimer des idées dans une langue donnée sans en apprendre les règles grammaticales, l'ordinateur n'exprime jamais d'idées, il ne fait que manipuler des règles grammaticales.

Searle offre avec la chambre chinoise une réponse au test de Turing, mais aussi une réfutation de l'approche symboliste, remettant en cause le rôle qu'y jouent les symboles et l'idée de l'esprit comme un système de traitement de l'information. Le débat faisant rage dans le milieu de l'intelligence artificielle et chez les philosophes de l'esprit, des réponses sont également formulées à l'expérience de Searle. L'une des principales est celle que Searle nomme la *Other Minds Reply* : si on ne peut accepter que la chambre chinoise a une compréhension du chinois sur la base des réponses qu'elle fournit, alors sur la base de preuves similaires on n'accepte pas non plus que les humains aient de l'esprit, même s'ils sont capables de parler[155]. Mais l'argument offert par la chambre chinoise de Searle, qui réfute l'association de la pensée à un système de procession d'information, porte un premier coup à la définition de l'intelligence offerte par ces théories. Les tenants de l'approche symboliste ont en effet une vision bien particulière de l'intelligence. Les cas d'études les plus fréquemment choisis pour leurs programmes en sont un marqueur : la géométrie, les échecs et la logique dominent. Il s'agit de tâches intellectuelles perçues par les chercheurs en intelligence artificielle des années 1950 à 1970 comme un signe d'intelligence. Mais il s'agit en fait d'une forme bien particulière d'intelligence parmi d'autres. Cette idée devient par la suite bien plus communément admise, notamment grâce au développement par le philosophe Hubert Dreyfus d'autres arguments qui traitent de tout un pan de l'expérience du monde des humains, la perception, et de son rôle dans l'intelligence humaine. Celui-ci travaille au MIT en tant qu'assistant professeur de philosophie au moment où Marvin Minsky et Seymour Papert y mènent leurs premières recherches. Dreyfus prise particulièrement la philosophie continentale - Merleau-Ponty et Heidegger comptent parmi ses penseurs favoris. Le contraste entre les idées que

[155] Cole, David, "The Chinese Room Argument", The Stanford Encyclopedia of Philosophy (Winter 2020 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/win2020/entries/chinese-room/>

Dreyfus enseigne à ses étudiants et l'approche symboliste développée sur le même campus le frappe fortement[156]. Il commence alors à s'intéresser au champ de l'intelligence artificielle. Il échange d'abord à ce sujet avec son frère, Stuart Dreyfus, ingénieur en génie industriel alors en poste à la RAND, et donc très familier du domaine. En 1964, Stuart Dreyfus fait recruter son frère Hubert par la RAND pour un été, dans le but de produire une évaluation philosophique des recherches en cours sur l'intelligence artificielle. Celle-ci, intitulée *Alchemy and AI*, est particulièrement agressive, et suscite des réactions très vives dans le domaine auquel elle s'attaque[157]. Mais Hubert Dreyfus est convaincu de la différence fondamentale entre intelligence humaine et intelligence artificielle, et persuadé que le projet d'intelligence artificielle est voué à l'échec. Il approfondit donc sa vision de la question dans le livre *What Computers Can't Do*, où il reprend et développe les points principaux de son rapport initial pour la RAND[158]. Le livre est réédité en 1979, puis en 1992 sous le titre *What Computer Still Can't Do*, prenant en compte dans ces versions successives les réponses apportées aux arguments de Dreyfus par ses opposants, mais aussi les progrès techniques du champ[159]. En 1986, Hubert Dreyfus publie également *Mind over Machine*, qui a pour sujet non une critique de l'intelligence artificielle, mais plutôt une proposition du fonctionnement de l'esprit humain[160].

Ces publications successives sont l'occasion pour Dreyfus d'exposer ses convictions sur la nature de l'intelligence et les raisons pour lesquelles le modèle symboliste utilisé par les chercheurs en intelligence artificielle est obsolète. Selon lui, ce modèle repose sur quatre suppositions invalides : biologique, psychologique, épistémologique et ontologique. Il les formule comme suit :

- *Le cerveau traite l'information en opérations discrètes par le biais d'un équivalent biologique des interrupteurs marche/arrêt.*
- *L'esprit peut être considéré comme un dispositif fonctionnant sur des bits d'information selon des règles formelles.*
- *Toute connaissance peut être formalisée.*
- *Le monde est constitué de faits indépendants qui peuvent être représentés par des symboles indépendants.*[161]

---

[156] Dreyfus, H. & Dreyfus, S., Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer, New York, NY: Free Press, 1987. Préface.
[157] Dreyfus, H. L., *Alchemy and Artificial Intelligence*. Santa Monica, CA: RAND Corporation, 1965.
[158] Dreyfus, H., *What Computers Can't do. The Limits of Artificial Intelligenc*e, The MIT Press, 1972.
[159] Dreyfus, H., *What Computers Can't do. A Critique of Artificial Reason*, The MIT Press, 1992.
[160] Dreyfus, H. & Dreyfus, S., Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer, New York, NY: Free Press, 1987
[161] "*The brain processes information in discrete operations by way of some biological equivalent of on/off switches. The mind can be viewed as a device operating on bits of information according to formal rules. All knowledge can be formalized. The world consists of independant facts that can be represented by independent*

La première est la supposition que l'activation des neurones est comparable aux 0 et aux 1 de l'ordinateur. La seconde est la supposition que le modèle de traitement de l'information est une description fidèle de l'activité cognitive humaine. La troisième est la supposition qu'il n'y a pas de différence entre la règle que quelqu'un suit pour faire quelque chose et la règle qui permet de décrire ce que quelqu'un fait. Pour Dreyfus, formé à la physique avant de devenir philosophe, confondre les deux fonctionne en physique, mais pas pour modéliser la pensée humaine. La supposition qu'il décrit implique que même si le modèle cognitif humain n'est pas celui du traitement de l'information, ce dernier permettrait quand même à l'ordinateur de faire preuve d'intelligence. Ceci implique la quatrième supposition : tout ce qui existe est assimilable à des objets, à leurs propriétés et à leurs relations. Selon Dreyfus, la phénoménologie du XX$^{e}$ siècle montre que l'idée que les comportements intelligents peuvent être décrits par des règles est obsolète. Les modèles réductionnistes comme le symbolisme ne fonctionnent pas pour modéliser la pensée humaine. Il existe une différence fondamentale entre l'esprit humain et la machine, que Dreyfus résume par l'idée d'*horizon intérieur*, un espace de semi-conscience toujours présent, qui nous permet de hiérarchiser ce vers quoi notre attention se porte. Cet espace d'ambiguïté serait impossible à trouver dans une machine. La différence entre humain et machine telle que analysée par Dreyfus repose par ailleurs sur l'idée que l'expérience du monde à travers le corps humain est fondamentale, et que la séparation entre corps et esprit - entre hardware et software - prônée par les symbolistes est insensée.

Dreyfus s'impose très vite comme un des critiques principaux du champ de l'intelligence, avec des arguments qui font mouche malgré la mauvaise volonté des chercheurs du camp symboliste, qui rechignent à engager le débat[162]. La plupart d'entre eux n'opposent que du silence à Dreyfus, arguant qu'il est trop partisan d'une impossibilité d'existence de l'intelligence artificielle pour qu'on puisse argumenter avec lui. Marvin Minsky publie tout de même en 1982 l'article *Why people think computers can't*, où il cherche à répondre à une partie des critiques de Dreyfus[163]. Il explique d'abord que ses travaux ont montré que les ordinateurs sont capables de raisonnement logique ou quantitatif, mais aussi de raisonnement par analogie. Il mentionne à ce titre le travail de son étudiant Tom Evans sur le programme ANALOGY, qui compare des formes géométriques pour reconnaître celles qui sont

---

*symbols*."
Dreyfus, H., *What Computers Can't do. The Limits of Artificial Intelligenc*e, The MIT Press, 1972.

[162] Dreyfus note même dans la préface de *Mind over Machine* que plus aucun d'entre eux ou presque ne voulait lui adresser la parole, qu'ils refusaient d'échanger dans des débats publics avec lui. McCorduck cite plusieurs chercheurs symbolistes qui confirment cette crispation, l'un d'entre eux résumant leur point de vue par "he was simply too silly to be taken seriously". Elle intitule même son chapitre à ce sujet "l'Affaire Dreyfus".
McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

[163] Minsky, M., "Why People Think Computers Can't",. *AI Magazine*, 3(4), 3, 1982. https://doi.org/10.1609/aimag.v3i4.376

proches ou identiques. Minsky soutient également dans l'article que ce n'est pas parce que les chercheurs se servent de la logique pour comprendre les mécanismes de fonctionnement d'un ordinateur que ce dernier n'est capable de fonctionner que selon des raisonnements logiques. Il émet également l'hypothèse que le scepticisme des opposants à l'analogie esprit-machine vient d'une méconnaissance du fonctionnement de l'esprit humain : "*Les théories [sur les différences fondamentales entre les machines et les esprits] ne sont pas prouvées aujourd'hui, non pas à cause des machines, mais simplement parce que nous en savons trop peu sur le fonctionnement de l'esprit humain*"[164]. Dans un argumentaire rappelant la pensée de son mentor B.F. Skinner, psychologue à Harvard qui estimait que l'"esprit" n'existait pas et que nos comportements n'étaient que des réponses à des stimulus électro-chimiques, il réduit les objections à l'analogie homme-machine à des *a priori* sans fondement sur la pensée humaine. Les titres des parties de l'article sont autant de questions sur les capacités de l'ordinateur, comme *Les ordinateurs peuvent-ils être créatifs ?*, *Un ordinateur peut-il vraiment comprendre quelque chose ?* ou *Un ordinateur peut-il avoir un soi ?*. Ses réponses à ces questions ne portent pourtant pas sur l'ordinateur, mais sur pourquoi nous considérons généralement que ces caractéristiques sont nécessaires à l'intelligence. Il cherche ainsi à réfuter l'idée que créativité, compréhension ou conscience de soi sont des marqueurs valables de l'intelligence humaine. La créativité ne serait rien de plus qu'une invention que nous plaquons sur un processus d'apprentissage particulier. Il faudrait donc simplement doter les ordinateurs de capacités d'apprentissage pour voir apparaître ce que les naïfs assimilent à de la créativité. Au sujet de la notion de compréhension, il indique par exemple : *Le mot "comprendre" est-il même une idée que nous pouvons demander à la science de traiter ? (...) Je ne me sens pas obligé de définir des mots tels que "sens" et "comprendre"*." [165]. Puisqu'une véritable définition de la compréhension n'existe pas, mais seulement une version idéalisée de ce que nous pensons que cela devrait être, il n'est pas nécessaire de pouvoir le programmer. Concernant la question de la conscience de soi, il écrit :

"*Habituellement, nous disons des choses comme ceci :*

*Un ordinateur ne peut pas faire (xxx), car il n'a pas de soi.*

---

[164] *"Theories [of how machines must differ fundamentally from minds] are unproved today - not because of anything about machines, but just because we know too little about how human minds really wor"*
Minsky, M., "Why People Think Computers Can't",. *AI Magazine*, 3(4), 3, 1982.
https://doi.org/10.1609/aimag.v3i4.376

[165] *Is "understand" even an idea we can ask science to deal with? (...) I feel no obligation to define such words as "mean" and "understand"*
Minsky, M., "Why People Think Computers Can't",. *AI Magazine*, 3(4), 3, 1982.
https://doi.org/10.1609/aimag.v3i4.376

*Et ces affirmations semblent souvent parfaitement sensées jusqu'à ce que nous abandonnions cette vision d'agent unique. Aussitôt, ces affirmations se transforment en bêtises, comme celle-ci :*

*Un ordinateur ne peut pas faire (xxx), parce que tout ce qu'un ordinateur peut faire est d'exécuter des processus incroyablement complexes, peut-être des millions à la fois, tout en construisant des structures interactives élaborées sur la base de réseaux presque inimaginablement ramifiés de fragments de connaissances interreliés.*

*Cela n'a plus tellement de sens, n'est-ce pas ?*"[166].

En somme, un ordinateur peut déjà faire beaucoup de choses complexes, alors pourquoi ne pas accepter l'idée qu'un ordinateur pense, puisque de toute façon nous ne sommes pas capables de définir ce qu'est penser ?

Les arguments de Minsky dans l'article ne paraissent guère plus que des artifices rhétoriques, et la faiblesse générale de la réponse des symbolistes à Dreyfus achève d'enterrer la prééminence de leurs idées, déjà mises à mal par un sévère ralentissement de leurs succès pratiques[167]. Si l'approche symboliste n'est plus en vogue dans le domaine de l'intelligence artificielle, cela n'enterre cependant pas les convictions des praticiens qui ont contribué à son succès, ni l'analogie esprit-machine, qui connaîtra plusieurs regains d'intérêt par la suite[168]. En témoigne la position de Searle, à qui la chambre chinoise sert pour une critique des symbolistes, mais pas l'analogie en elle-même, puisqu'il reste de l'avis que les cerveaux sont des machines[169]. Mais le craquèlement des modèles connexionniste et symboliste, qui culmine dans les années 1980, s'il ne fait pas disparaître l'analogie esprit-machine pour autant, fait émerger une autre théorie de la pensée. Cette théorie concurrente repose sur trois

---

[166] *"Usually, we say things like this:*
*A computer can't do (xxx), because it has no self.*
*And such assertions often seem to make perfect sense until we shed that Single Agent view. At once those sayings turn to foolishness, like this:*
*A computer can't do (xxx), because all a computer can do is execute incredibly intricate processes, perhaps millions at a time, while constructing elaborately interactive structures on the basis of almost unimagineably ramified networks of interrelated fragments of knowledge.*
*It doesn't make so much sense any more, does it?"*
Minsky, M., "Why People Think Computers Can't",. *AI Magazine*, 3(4), 3, 1982.
https://doi.org/10.1609/aimag.v3i4.376

[167] Voir chapitre III.

[168] Rescorla, Michael, "The Computational Theory of Mind", The Stanford Encyclopedia of Philosophy (Fall 2020 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/fall2020/entries/computational-mind/>.

[169] "The Chinese Room argument is not directed at weak AI, nor does it purport to show that no machine can think – Searle says that brains are machines, and brains think. The argument is directed at the view that formal computations on symbols can produce thought." Simon, H. and Eisenstadt, S., 2002, 'A Chinese Room that Understands', in Preston and Bishop (eds.) 2002, 95–108.

points essentiels. Le premier est l'idée que coexistent plusieurs manières de penser dans l'esprit humain. Ceci est résumé par Daniel Kahneman, qui propose un modèle baptisé "système 1, système 2" qui repose sur deux systèmes de pensée distincts. Il compare la première, intuitive et heuristique à un lièvre qui se déplace rapidement, et la seconde plus lente, rationnelle et logique, à une tortue prenant son temps[170]. Si la manière de penser que représente la tortue peut être mécanisée car représentée par des opérations logiques, l'autre ne peut pas être réduite à du traitement des symboles. Second point essentiel, cette théorie acte ainsi les limites de l'analogie machine-esprit, qui permet d'imiter seulement une partie du processus de pensée humain. L'Oracle de Turing serait donc un humain. Les différentes manières de penser impliquent par ailleurs plusieurs formes de connaissances, certaines ne pouvant s'acquérir que par l'expérience physique du monde. Le débat porte en effet également sur ce que c'est qu'apprendre. L'apprentissage est identifié comme facteur clé chez les tenants des deux parties, mais dans le cas de l'analogie machine-esprit, le processus d'apprentissage est mécanisable, contrairement aux tenants de la seconde théorie de l'esprit, qui proposent d'autres modèles. Celle-ci offre donc comme troisième point une réflexion sur la place de l'expertise[171]. L'expertise implique en effet de mobiliser ces différentes manières de penser, ces différentes formes de connaissances, toutes ensemble, et c'est cette capacité, donc l'expertise, qui serait le marqueur de l'intelligence. Dreyfus propose par exemple dans *Mind over Machine* un modèle d'acquisition des connaissances en paliers successifs[172], qui n'est pas sans rappeler le modèle de Nonaka que nous avons vu au chapitre précédent. Si le modèle de Nonaka se construit à partir d'une théorie sur le transfert des connaissances tacites et explicites, c'est parce que devenir un expert, c'est accumuler des connaissances des deux types. La notion de connaissances tacites est en effet clé dans cette théorie alternative de la pensée : c'est justement parce qu'elles existent et ne peuvent être explicitées que les opérations logiques ne suffisent plus à modéliser le processus de pensée. Le débat peut donc se lire comme une controverse sur la place respective des connaissances tacites et explicites dans nos processus de pensée, qui a influencé les progrès de l'intelligence artificielle et les différentes approches adoptées dans le champ.

**Deux appareils théoriques possibles pour les courants computationnels**

De Turing à Kahneman, les débats mettent en évidence une histoire de l'intelligence artificielle qui partage beaucoup avec la philosophie de l'esprit, avec des débats vifs sur la nature de l'intelligence et la capacité d'une machine à devenir intelligente. Les débuts de l'intelligence artificielle voient

[170] Kahneman, D., *Thinking, Fast and Slow*. New York: Farrar, Straus & Giroux, 2011.
[171] Au sens général. Il ne s'agit donc pas seulement d'expertise spécialisée, mais du savoir sur le monde en général. Reconnaître les objets du quotidien - porte, tasse ou stylo - serait dans ce sens une forme d'expertise, les connaissances emmagasinées pour pouvoir bouger dans l'espace également, etc.
[172] Dreyfus, H. & Dreyfus, S., Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer, New York, NY: Free Press, 1987.

l'émergence de deux appareils théoriques distincts, entre mécanisation totale de la pensée et impossibilité d'y parvenir, mais aussi entre quête de rationalité et appréciation de l'intuition humaine. Ces deux appareils théoriques tentent de répondre en parallèle à deux questions. Qu'est-ce que penser ? Qu'est-ce qu'un ordinateur peut reproduire de ce processus de pensée humain ? L'une des deux approches affirme que penser peut être intégralement explicité, et donc modélisé avec une machine - quitte à ce que ce ne soit pas un ordinateur moderne. L'autre affirme que les machines ne sont pas capables de modéliser complètement l'esprit humain, car elles ne manipulent que des connaissances explicites quand l'esprit se repose autant sur des savoirs tacites - quitte à ce que d'autres machines que les ordinateurs modernes parviennent à modéliser une plus grande part des processus de pensée. Cette controverse n'est pas résolue. Il s'agit d'un débat qui prend avant tout racine dans des convictions profondes de part et d'autre, plus que dans des preuves scientifiques quelconques. Comme l'écrit Pamela McCorduck au sujet des premiers tenants de ces théories :

> *[Il s'agit d']un engagement passionné envers une idée : de même que Simon, Minsky et d'autres s'accrochent avec ténacité à l'idée que, quelles que soient les preuves qui s'y opposent, les preuves que les machines - les ordinateurs - seront un jour très intelligentes sont écrasantes, de même Hubert Dreyfus soutient que, quelles que soient les preuves que les machines peuvent effectuer des tâches intelligentes, les preuves qu'elles ne seront jamais capables d'être réellement, humainement intelligentes, sont écrasantes*[173].

Deux approches de l'automatisation différentes découlent de ces points de vue. La première nécessite seulement de trouver les bonnes instructions, qui permettront à l'ordinateur de travailler ensuite en toute autonomie. L'intégralité des processus mentaux nécessaires lui ayant déjà été transmis, il pourrait ainsi construire sa propre représentation du monde, en saisir les enjeux et produire ses propres objets en réponse à ces enjeux. La seconde implique de revenir sans arrêt sur la traduction en instructions des savoirs humains, puisque les limites de l'ordinateur ne lui permettent pas de saisir les enjeux toujours renouvelés du monde tel que le perçoivent les humains[174].

---

[173] *"a passionate commitment to an idea: as tenaciously as Simon and Minsky and others hold on to the idea that whatever the evidence against it, the evidence for machines - computers - someday being highly intelligent is overwhelming; so Hubert Dreyfus holds that whatever the evidence that machines can perform intelligent tasks, the evidence against their ever being able to be really, humanly intelligent, is overwhelming"*
McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004, p. 226. Mais Patricia McCorduck, qui a évolué pendant des années dans le milieu de l'intelligence artificielle américain, ne fait pas exception, comme le trahit son scepticisme à l'égard de Dreyfus, qui transparaît dans le récit qu'elle fait de son rôle.

[174] Selon l'auteure de ces lignes, c'est le sens de la citation d'Alan Turing qui se trouve en exergue de ce texte.

Nous avons pointé la naissance commune du champ computationnel et du champ de l'intelligence artificielle. Ceci implique que les deux appareils théoriques dont nous avons mis en évidence la genèse au cours de ces années sont à la portée des architectes du champ computationnel dès ses débuts. Les interrogations de Nigel Cross, de Charles Eastman ou d'Horst Rittel sur la nature de processus de conception sont justement en lien avec les réflexions du champ de l'intelligence artificielle sur la nature de l'esprit. L'importance qu'elles ont à la naissance du champ computationnel témoigne de l'influence de ces réflexions. La vivacité de ces interrogations démontre par ailleurs l'intérêt du complexe processus de conception spatiale comme cas d'étude. Les recherches de Bill Hillier ou de William Mitchell sur les règles de composition de l'architecture relèvent de la même ambition que celles d'Allen Newell, Herbert Simon ou Marvin Minsky pour trouver des règles logiques de description de la pensée humaine. Les prises de décisions qui sont au centre de la pratique architecturale relèvent de l'incalculable au sens défini par Alan Turing. C'est pour cette raison que c'est un si beau cas d'exemple et que le champ computationnel en architecture est né dès la naissance de l'informatique. Au cœur de ce débat se trouve donc la possibilité d'automatiser tout ou partie du processus de conception architectural. Pour les praticiens du champ, cerner le rôle de l'ordinateur dans leur travail de conception, l'ampleur de ce qu'il peut prendre en charge et le poids relatif de leurs décisions est directement impacté par celles de ces théories sur la nature de l'esprit qu'ils choisissent pour point d'ancrage. Il s'agit donc d'un débat qui a des conséquences majeures sur les choix de développement au sein du champ computationnel. Le fait que l'appartenance à l'une des deux approches soit de l'ordre de la conviction a aussi une conséquence sur les praticiens du champ computationnel et leur travail de traduction des savoirs tacites de la conception architecturale en instructions de programmation explicite. La manière dont on envisage la traductibilité est tributaire de contraintes idéologiques, et le champ computationnel dispose de deux appareils théoriques développés dès ses débuts pour penser le rôle de l'ordinateur dans la conception spatiale. Les praticiens n'y choisissent pas les mêmes modèles, les mêmes algorithmes, ni les mêmes modalités de pratique en fonction de la manière dont ils envisagent la validité de l'analogie esprit-machine. Et malgré leur nom, les courants computationnels en architecture ne relèvent pas seulement d'une approche computationnaliste, ou symboliste : des modalités de pratique relevant d'une autre vision du rôle de l'ordinateur y existent aussi. Les différentes convictions à l'origine des deux approches de l'automatisation ne sont pas sans conséquences sur la forme prise par les pratiques du champ, comme nous allons le voir. Les deux appareils théoriques du monde de l'intelligence artificielle vont nous fournir pour la suite de ce récit un prisme d'analyse pour le développement du champ, qui se fait en tension entre ces deux positions.

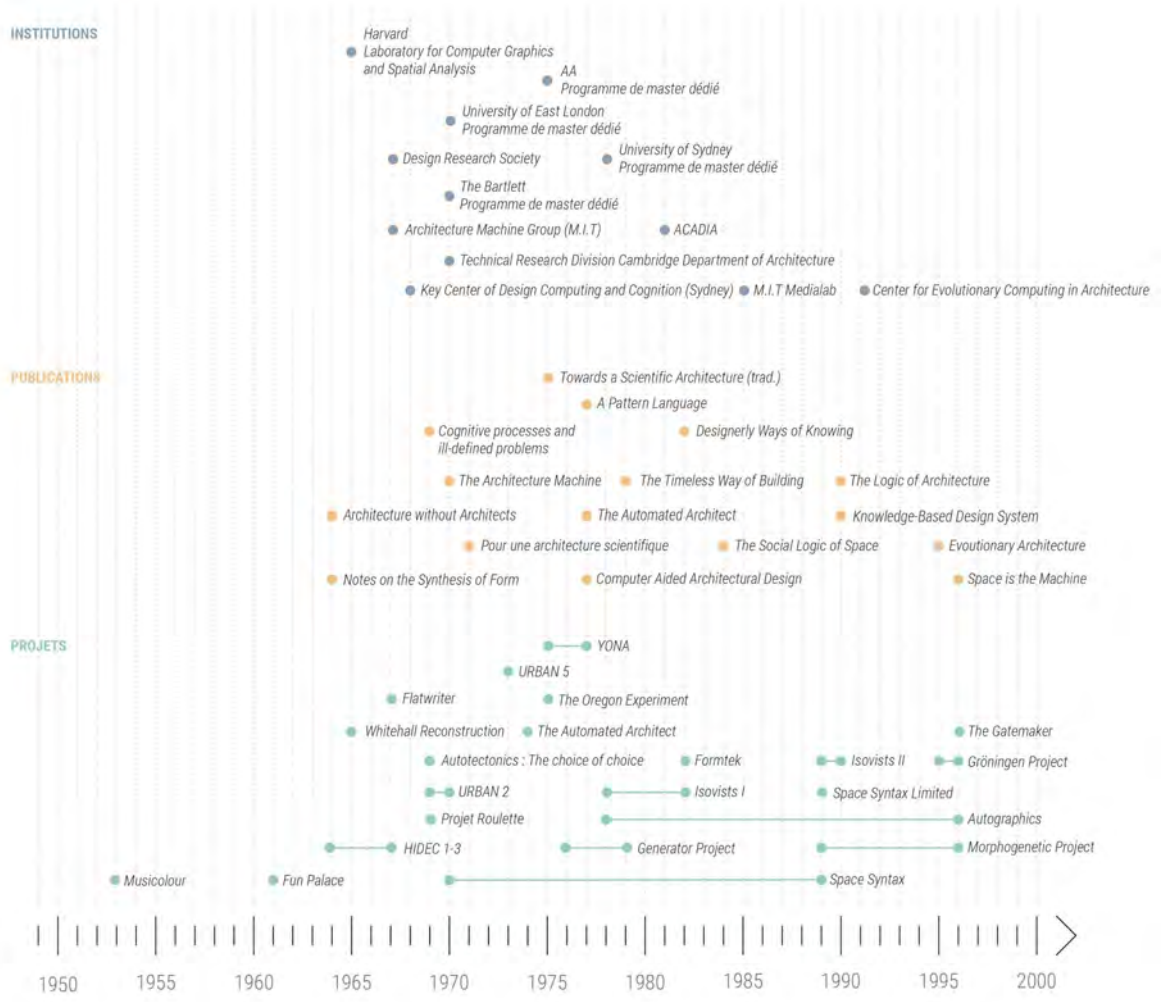



*Figure 11. Chronologie des productions de la première génération du champ computationnel en architecture.*

## CHAPITRE III.

## -

## Une filiation dans le monde de l'animation

### 3.1 L'hiver procédural

*3.1.1 Une quête pour l'intelligence artificielle freinée*

*Je crois que dans une cinquantaine d'années, il sera possible de programmer des ordinateurs (...) pour qu'ils jouent si bien le jeu de l'imitation qu'un interrogateur moyen n'aura pas plus de 70 % de chances de faire la bonne identification après cinq minutes d'interrogatoire. Je crois qu'à la fin du siècle, l'usage des mots et l'opinion générale instruite auront tellement changé que l'on pourra parler de machines pensantes sans s'attendre à être contredit*[1].

Alan Turing, 1950

*Mon but n'est pas de vous surprendre ou de vous choquer, mais la façon la plus simple de résumer est de dire qu'il y a maintenant dans le monde des machines qui peuvent penser, qui peuvent apprendre et qui peuvent créer. De plus, leur capacité à faire ces choses va augmenter rapidement jusqu'à ce que - dans un avenir visible - l'éventail des problèmes qu'elles peuvent traiter soit aussi large que celui auquel l'esprit humain a été appliqué*[2].

Herbert Simon, 1957

*Dans 10 ans, un ordinateur numérique sera le champion du monde d'échecs*[3].

Herbert Simon & Allen Newell, 1958

*Les machines seront capables, d'ici 20 ans, de faire n'importe quel travail qu'un homme peut faire*[4].

Herbert Simon, 1965

---

[1] *"I believe that in about fifty years' time it will be possible to programme computers, with a storage capacity of about 109, to make them play the imitation game so well that an average interrogator will not have more than 70 percent chance of making the right identification after five minutes of questioning. ... I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted".*

[2] *"It is not my aim to surprise or shock you—but the simplest way I can summarize is to say that there are now in the world machines that can think, that can learn and that can create. Moreover, their ability to do these things is going to increase rapidly until – in a visible future - the range of problems they can handle will be coextensive with the range to which the human mind has been applied".*

[3] "*Within 10 years a digital computer will be the world's chess champion".*

[4] *"Machines will be capable, within 20 years, of doing any work a man can do".*

*D'ici une génération, le problème de la création d'une "intelligence artificielle" sera en grande partie résolu*[5].

Marvin Minsky, 1967

*Dans 3 à 8 ans, nous aurons une machine dotée de l'intelligence générale d'un être humain moyen*[6].

Marvin Minsky, 1970

Ces quelques affirmations sont rassemblées par Daniel Crevier dans son livre de 1993 *AI: The Tumultuous History of the Search for Artificial Intelligence*[7]. Elles brossent le tableau des attentes dans le domaine de l'intelligence artificielle des années 1950 à 1970, des attentes faramineuses. On peut ajouter que Herbert Simon avait prédit non seulement qu'un ordinateur serait champion d'échecs sous peu, mais aussi qu'une de ces machines découvrirait et prouverait un nouveau théorème mathématique d'importance, et que la plupart des travaux de psychologie prendraient la forme de programmes informatiques. Ou encore que John McCarthy avait obtenu le financement par l'université de Stanford en 1963 d'un projet se donnant pour objectif de construire une machine "pleinement intelligente"[8] en une décennie. Les dates des déclarations comme les laps de temps et les objectifs précis prédits varient dans ces déclarations, mais même en prenant les prédictions les plus prudentes, l'intelligence artificielle aurait dû nous arriver au plus tard un peu avant les années 2000. Inutile sans doute de rappeler que vingt ans plus tard, au moment où ces lignes sont écrites, c'est toujours loin d'être le cas. La première génération de spécialistes de l'intelligence artificielle fait donc preuve d'un peu trop d'optimisme au moment de ses premières prédictions. Si le temps a donné raison à Herbert Simon et que des programmes informatiques sont aujourd'hui parfaitement capables de battre des maîtres aux échecs, il a fallu attendre non pas 1968 mais 1981 pour que cela se produise une première fois, et 1989 pour que le programme batte un grand maître. Des joueurs humains continuent par ailleurs régulièrement à battre des programmes informatiques jusqu'en 2005 dans les conditions de jeu classiques des tournois d'échecs[9]. En 2016, le programme AlphaGo parvient à battre au jeu de go Lee Sedol, l'un des meilleurs joueurs du monde. Le jeu de go est considéré comme bien plus complexe que les échecs, et cette victoire est donc une étape très importante pour la communauté de

---

[5] *"Within a generation, the problem of creating 'artificial intelligence' will substantially be solved".*
[6] *"In 3 to 8 years, we will have a machine with the general intelligence of an average human being".*
[7] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.
[8] *fully intelligent*
[9] Ces oppositions ont lieu dans des tournois réguliers avec des règles diverses depuis 2005, et des humains gagnent encore dans des cadres de jeu particuliers - mais de simples programmes d'échecs sur téléphone battent cependant maintenant sans problème des joueurs humains de très haut niveau. Voir notamment David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm", 2017, https://arxiv.org/abs/1712.01815, consulté le 30 Novembre 2021.

l'intelligence artificielle[10]. 2019 marque une nouvelle étape, avec la victoire du programme OpenAI Five à DOTA 2, un jeu vidéo de stratégie jugé plus difficile encore en raison du grand nombre de possibilités à chaque tour, et du grand nombre d'éléments à gérer par le joueur. Il est cependant intéressant de noter que les résultats d'OpenAI Five ne se sont considérablement améliorés qu'après l'intégration de règles heuristiques définies par des joueurs dans l'algorithme - celui-ci n'a donc pas exactement appris tout seul à jouer aussi bien[11]. Des programmes capables de triomphes significatifs sur des joueurs humains à des jeux particulièrement difficiles à maîtriser voient donc régulièrement le jour depuis les prédictions de Simon et de ses collègues. Mais ces simples étapes sur le chemin de l'intelligence artificielle se réalisent bien plus tard qu'ils ne l'anticipaient, et d'autres jeux résistent par ailleurs encore toujours aux machines[12]. La prédiction d'Alan Turing concernant son test, elle, ne s'est toujours pas vraiment réalisée. En 2014, un programme de conversation automatique (chatbot) anglophone parvient bien à se faire passer pour un jeune garçon ukrainien auprès de l'un de ses interlocuteurs[13]. En 2018, le programme Duplex, fruit des recherches de Google sur la reconnaissance vocale, parvient à prendre rendez-vous chez le coiffeur sans que son interlocuteur devine qu'il ne s'agit pas d'un humain au bout du fil[14]. Mais dans les deux cas, peu de spécialistes considèrent le test comme vraiment réussi. Duplex a beau être capable de prendre un rendez-vous, il ne mène pas vraiment de conversation. Le chatbot, lui, parvient à faire excuser un certain nombre d'incohérences et d'erreurs en anglais grâce à son identité supposée de jeune garçon ukrainien. La pertinence du test de Turing est néanmoins régulièrement remise en question comme indicateur d'intelligence aujourd'hui, et moins d'efforts sont sans doute consacrés à la conception de programmes informatiques capables de flouer un humain sur ce plan. Duplex montre aussi par ailleurs que beaucoup de progrès ont été faits dans le domaine du traitement du langage. Google ou Deep L ont également récemment fait des avancées significatives dans le domaine de la traduction automatisée, livrant maintenant des textes dénués d'incohérences ou presque. Ce sont cependant des progrès faits essentiellement au cours de la dernière décennie - bien plus tard que prédit - et qui sont loin d'une démonstration complète d'intelligence. Quant à exécuter n'importe quelle tâche accomplie par un être humain ou rivaliser d'intelligence, nous en sommes encore loin, même si les machines commencent à pouvoir non seulement prendre rendez-vous chez le coiffeur, mais aussi conduire des voitures, reconnaître des cancers ou imiter Rembrandt.

---

[10] Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017). https://doi.org/10.1038/nature24270

[11] Open AI, "Dota 2 with Large Scale Deep Reinforcement Learning", 2019, https://arxiv.org/abs/1912.06680, consulté le 30 Novembre 2021.

[12] C'est par exemple le cas du jeu de cartes Magic The Gathering. Voir C. D. Ward and P. I. Cowling, "Monte Carlo search applied to card selection in Magic: The Gathering," 2009 IEEE Symposium on Computational Intelligence and Games, 2009, pp. 9-16.

[13] Martin Untersinger, " Réussite contestée d'un ordinateur au légendaire test de Turing", Le Monde, 09 Juin 2014.

[14] Yaniv Leviathan and Yossi Matias, "Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone", 2018, https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html, consulté le 30 Novembre 2021.

La quête pour l'intelligence artificielle s'avère donc bien plus difficile qu'anticipé par Marvin Minsky, Allen Newell ou Herbert Simon. Programmer des machines intelligentes pose en effet nombre de problèmes, décrits au cours des décennies qui suivent la naissance du champ. Le problème majeur est celui dit des connaissances de sens commun. Celles-ci désignent les faits du quotidien, que tout le monde est supposé connaître. Par exemple, ouvrir la fenêtre implique de savoir ce qu'est une fenêtre, de pouvoir la reconnaître et la trouver dans la pièce, de savoir comment la rejoindre dans l'espace, de trouver comment utiliser la poignée. Un humain à qui on le demande saura par ailleurs qu'ouvrir la fenêtre provoque un changement de température et évaluera donc plus ou moins consciemment son ressenti en regard de la température dans la pièce avant l'ouverture de la fenêtre, pour déterminer s'il lui paraît acceptable de la modifier. Autant de choses qu'on se formule rarement de manière explicite lorsqu'on nous demande d'ouvrir la fenêtre, mais qui sont pourtant déterminantes dans notre capacité à le faire. Et il ne s'agit là que d'une simple fenêtre - la plupart des tâches que nous exécutons au quotidien sont bien plus complexes. Hubert Dreyfus décrit le problème des connaissances de sens commun comme la question de "*comment stocker et accéder à tous les faits que les êtres humains semblent connaître*"[15]. Les connaissances de sens commun désignent donc une très grande quantité d'informations sur le monde, essentielles même si nous n'en avons que rarement pleine conscience. Or ce sont des connaissances dont il faut doter un ordinateur si l'objectif est de le voir se comporter comme un humain. Par ailleurs, un second problème est lié à la manière dont nous manipulons ces connaissances, un processus baptisé raisonnement de sens commun. Ceci désigne notre capacité à faire des suppositions, ce qui nous permet de gérer des situations au sujet desquelles nous n'avons que des informations incomplètes.

Deux autres problèmes de l'intelligence artificielle permettent de mieux comprendre ce second enjeu. Le premier se base sur la notion de cadre[16] telle que définie par Marvin Minsky[17]. Pour chaque notion dont nous nous servons, il existe un ensemble de faits qui nous permettent de la définir et d'y recourir. Par exemple, nous savons qu'un oiseau a des ailes et un bec et piaille ou chante. Nous nous servons de ces informations et de quelques autres pour identifier un oiseau quand nous en voyons un. Cet ensemble de faits peut varier d'une personne à l'autre, alors que toutes sont capables de reconnaître un oiseau : elles n'utilisent pas forcément les mêmes informations. Les cadres sont donc des ensembles de connaissances plus ou moins précises auxquelles nous nous référons dans un contexte donné, pour pouvoir agir dans ce contexte. Or nombre des concepts auxquels nous recourons sont plus ou moins

[15] *"how to store and access all the facts human beings seem to know",* Dreyfus, H. & Dreyfus, S., Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer, New York, NY: Free Press, 1987, p 78.
[16] *frame problem*
[17] Marvin Minsky. 1974. A Framework for Representing Knowledge. Technical Report. Massachusetts Institute of Technology, USA.

imprécis, et leur donner une définition précise dans le programme n'est pas un bon substitut à comprendre comment nous nous en servons. John McCarthy et Patrick Hayes emploient, pour désigner cette difficulté, la formule *problème du cadre*, puisqu'il s'agit de comprendre notre gestion du cadre de référence que nous utilisons dans un contexte donné[18]. Dans le même article, les deux chercheurs formulent le problème des réserves, qui exemplifie plus avant la question à laquelle ils sont confrontés.

*Nous avons supposé que si p cherche le numéro de téléphone de q dans l'annuaire, il le connaîtra, et que s'il compose le numéro, il entrera en conversation avec q. Il n'est pas difficile de penser à des exceptions possibles à ces affirmations, telles que*

*1. La page avec le numéro de q peut être déchirée.*

*2. p peut être aveugle.*

*3. Quelqu'un peut avoir délibérément encré le numéro de q.*

*4. L'opérateur téléphonique a peut-être fait une erreur d'inscription. 5.*

*5. q n'a peut-être obtenu le téléphone que récemment.*

*6. L'installation téléphonique peut être en panne.*

*7. q peut être frappé d'une incapacité soudaine*[19].

Les réserves auxquelles fait référence l'intitulé du problème sont les raisons possibles de ne pas pouvoir accomplir une action, comme McCarthy et Hayes en donnent ici l'exemple. Il existe de nombreuses exceptions à la règle sur laquelle ils se penchent : si quelqu'un (ici p) a connaissance du numéro de téléphone d'une autre personne (ici q), alors il peut entrer en contact avec elle. Nous partons généralement du principe que cette règle est valide, à moins d'une bonne raison - une réserve suffisamment importante. Mais il s'avère compliqué de formuler dans un programme informatique ce qu'est une bonne raison, tout autant que de lister l'ensemble des exceptions possibles à une règle donnée[20]. Comme le problème des connaissances de sens commun, le problème des réserves implique

---

[18] J. McCarthy and P. J. Hayes. 1987. Some philosophical problems from the standpoint of artificial intelligence. Readings in nonmonotonic reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 26–45.

[19] "*We assumed that if p looks up q's phone-number in the book, he will know it, and if he dials the number he will come into conversation with q. It is not hard to think of possible exceptions to these statements such as*
*1. The page with q's number may be torn out.*
*2. p may be blind.*
*3. Someone may have deliberately inked out q's number.*
*4. The telephone company may have made the entry incorrectly.*
*5. q may have got the telephone only recently.*
*6. The phone system may be out of order.*
*7. q may be incapacitated suddenly*".
Cité par Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.

[20] Le problème est causé notamment par le fait de recourir à la logique du premier ordre pour formuler ces règles, ce qui mène McCarthy et Hayes et d'autres logiciens à formaliser de nouvelles logiques.

de saisir une très vaste quantité d'informations : toutes les conséquences possibles de toutes les actions, une description qui devient vite fastidieuse[21].

Du problème des connaissances de sens commun découle donc une nuée d'autres problèmes et paradoxes, formulés par le biais d'expressions diverses par les chercheurs du champ. Hans Moravec, roboticien à Carnegie Mellon, énonce par exemple le paradoxe qui prend son nom : le plus difficile pour une machine - plus exactement un robot - est ce qui paraît le plus facile à l'humain, et inversement. Moravec fait ce constat à partir de la difficulté observée dans ses travaux à faire faire à un robot même les mouvements considérés comme les plus simples, qu'un enfant sait pourtant faire aisément. Mais ce paradoxe permet aussi d'expliquer en partie l'optimisme de Newell, Simon ou Minsky. La logique et les échecs étant perçus comme des signes d'intelligence supérieure, le fait que les ordinateurs soient capables de les maîtriser aussi bien implique pour eux que le reste, considéré comme plus simple, serait forcément rapidement à la portée des machines. Ces problèmes sont tous en lien avec la question des connaissances tacites. Le paradoxe de Polanyi, baptisé ainsi en l'honneur du philosophe auquel on doit la notion de tacite, résume les difficultés de l'intelligence artificielle dans ce domaine[22]. Celui-ci pointe le fait que la majeure partie des connaissances humaines sur le monde ne sont pas explicites. La plupart des actions que nous accomplissons sont guidées par l'intuition, sans que nous sachions formaliser les procédures qui nous font les accomplir. Comme nous l'avons vu au chapitre I, Polanyi donne dans son ouvrage *The Tacit Dimension*[23] de nombreux exemples de ces actions. Or les programmes informatiques sur lesquels repose la recherche de l'intelligence artificielle ne sont formulés qu'à partir d'opérations logiques - précisément ce que nous ne savons pas comment formaliser pour décrire nombre de nos actions. Ceci fait du paradoxe de Polanyi un des obstacles majeurs à la recherche de l'intelligence artificielle. Cela fait également des problèmes que nous avons vu en exemples des difficultés qui ont systématiquement à voir avec la question de la traduction tacite-explicite - dont la conception architecturale, par sa difficulté, est un si beau cas d'étude.

D'autres problèmes viennent par ailleurs complexifier la tâche des chercheurs. La notion de problème ingérable[24], par exemple, désigne des problèmes solubles en théorie mais pas en pratique au vu des ressources disponibles. En effet, la complexité d'un problème peut être évaluée notamment en regard du nombre de possibilités différentes que la variation des paramètres génère. Par exemple, un problème d'échecs offre plusieurs dénouements en fonction de la suite des mouvements qui sont exécutés pour tenter de le résoudre. Pour sélectionner son prochain coup, un programme doit examiner tous les coups possibles, les réponses possibles de l'adversaire à chacun de ces coups, puis

[21] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.
[22] Autor, David, "Polanyi's Paradox and the Shape of Employment Growth", NBER Working Paper Series, Cambridge, MA: National Bureau of Economic Research, pp. 1–48, 2014.
[23] Polanyi, M., *The Tacit Dimension*, Routledge, 1966.
[24] *intractable*

ce qui pourra être joué en réponse, et ainsi de suite. Il est ainsi possible de connaître les successions de coups qui mènent à la victoire ou qui mènent à la défaite et de sélectionner à chaque tour le coup offrant le plus de chances d'accéder à la victoire. Il existe donc un arbre des possibilités pour chaque problème d'échecs, que le programme doit parcourir. Le go ou DOTA sont considérés comme des jeux plus complexes que les échecs justement parce que l'arbre des possibilités est beaucoup plus vaste. L'agrandissement de cet arbre des possibilités se nomme l'explosion combinatoire, et c'est une des causes qui expliquent que certains problèmes soient ingérables - l'arbre est bien trop grand pour pouvoir être exploité efficacement par le programme. Ceci se combine dans les années 1970 à un autre problème : celui des capacités de calcul limitées. Certes elles le sont toujours aujourd'hui, et certains problèmes demeurent donc ingérables, mais les ordinateurs des années 1970 étaient bien plus faibles, limitant d'autant l'exploration permise. Or, si l'on considère la quantité de données à gérer qu'implique la prise en compte des connaissances de sens commun, le problème de l'explosion combinatoire est d'autant plus compliqué à gérer. La conséquence des nombreuses difficultés pratiques rencontrées par les chercheurs en intelligence artificielle est que même les programmes les plus impressionnants ne peuvent gérer qu'une fraction du problème réel qu'ils sont supposés traiter. Ils sont donc condamnés à évoluer dans des mondes jouets, bien loin du contexte auquel les humains, eux, sont confrontés[25].

La quête pour l'intelligence artificielle rencontre donc au fur et à mesure des expérimentations des difficultés pratiques significatives. A ces difficultés s'ajoutent les remises en questions philosophiques que nous avons évoquées au chapitre précédent. Les critiques successives contribuent, tout comme les obstacles pratiques, à faire apparaître dans le champ des complications en cascade. Le premier âge d'or[26] de l'intelligence artificielle, de 1956 à 1974, est autant marqué par des succès que par la réalisation de l'existence de ces complications. Peu des chercheurs impliqués dès les premières années s'attendent pourtant à ce que cela s'avère aussi difficile. Cela s'explique pourtant notamment par le fait que nombre des travaux portant sur la nature de la pensée datent en fait d'après les premiers essais de l'intelligence artificielle. Les nombreux blocages rencontrés au fil des années 1960 et 1970 ne sont pas sans conséquence. La frustration due au manque de résultats des programmes de recherche en dépit des annonces enthousiastes des chercheurs augmente peu à peu. Les institutions perdent graduellement espoir et intérêt pour les travaux du domaine de l'intelligence artificielle, et quelques événements particulièrement significatifs vont mener en quelques années à la disparition complète des financements de ce genre de recherche. Le premier de ces événements est le rapport de l'ALPAC, écrit en 1966. Le gouvernement américain cultive alors en raison de la guerre froide un fort intérêt pour la traduction automatique de documents russes. La National Research Commission finance donc

[25] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.
Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.
[26] Ainsi baptisé dans les histoires de l'intelligence artificielle écrites a posteriori.

plusieurs projets de recherche dédiés à cette question. Les chercheurs y prenant part ont néanmoins, comme tous leurs collègues, fait preuve d'un extraordinaire optimisme dans leurs annonces. Ils ont notamment sous-estimé un point particulier : le travail d'interprétation des traducteurs. En effet, pour pouvoir bien traduire une phrase, il faut avoir une idée de ce dont elle traite. On retrouve ici la question des connaissances de sens commun. Confrontés à ce problème, les chercheurs font peu de progrès dans la traduction automatisée. La National Research Commission se posant de plus en plus de questions au sujet du bien-fondé de ces investissements, un comité d'évaluation est formé : le Automated Language Processing Advisory Committee (ALPAC). L'ALPAC signe un rapport sans appel : malgré les années de recherche à ce sujet, la traduction par des opérateurs humains est toujours plus fiable et moins coûteuse que celle exécutée par des machines. Après vingt ans de recherche et vingt millions de dollars investis, la National Research Commission met un terme à tous les financements portant sur la traduction automatique.

Le second événement clé qui précipite la remise en question des recherches pour l'intelligence artificielle a lieu au Royaume-Uni : il s'agit du rapport Lighthill. Commandé par le parlement britannique au mathématicien James Lighthill pour évaluer les recherches nationales dans le domaine de l'intelligence artificielle, il est motivé par les critiques de plus en plus nombreuses qu'essuient alors ces travaux. Rendu en 1973, le rapport est très négatif et préconise l'arrêt complet des recherches. James Lighthill y fait l'hypothèse qu'aucun des objectifs ambitionnés par les chercheurs en intelligence artificielle ne peut en fait être atteint, en se fondant notamment sur la question des problèmes ingérables. Le parlement britannique suit ses recommandations, ce qui a pour conséquence l'arrêt presque total des recherches sur le sujet en Grande-Bretagne, et la disparition des financements associés. Aux Etats-Unis, les groupes de Stanford, du MIT et de Carnegie Mellon ont été épargnés par le rapport ALPAC car ils ne travaillent pas sur le sujet de la traduction automatique. Par ailleurs, la National Research Commission n'est pas leur source de financement principale, puisque c'est la DARPA qui encadre leurs recherches. Mais au moment du rapport Lighthill, la DARPA elle aussi commence à s'interroger sur l'intérêt des travaux sur l'intelligence artificielle. Et un épisode en particulier refroidit significativement ses ardeurs : les débouchés du programme Speech Understanding Research (SUR). Le programme SUR finance plusieurs chercheurs de Carnegie Mellon pour développer un programme informatique qui réponde aux commandes vocales des pilotes. Les deux programmes résultant de ce travail présentent une contrainte majeure : ils ne comprennent les commandes vocales que si les mots sont dits dans un certain ordre. Cette caractéristique se nomme "grammaire contrainte"[27], et peut rendre l'usage des programmes qui se reposent dessus particulièrement difficile. En effet, il peut être plus long pour les utilisateurs de comprendre ce qu'ils peuvent et ne peuvent pas dire que de prendre en main un programme classique[28]. La DARPA est

[27] *constraint grammar*
[28] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.

furieuse de cette entorse aux objectifs du programme SUR, et d'autant plus agacée que les publications du groupe de chercheurs présentent leur travail comme une percée majeure dans le domaine de la reconnaissance du langage[29]. En 1974, le programme SUR est donc brutalement arrêté, malgré l'investissement de la DARPA de 15 millions de dollars en cinq ans. L'épisode irrite tellement l'institution que les rapports sur les programmes de recherche dans le domaine de l'intelligence artificielle qui sont rédigés en interne par la suite parlent de "débâcle SUR"[30]. A ceci s'ajoute le passage du Mansfield Amendment[31], qui contraint la DARPA à ne financer que des programmes de recherche avec des objectifs précis. C'est donc la fin des financements libres sur lesquels le domaine de l'intelligence artificielle s'est tant reposé jusqu'ici. Au milieu des années 1970, la majeure partie des financements sur lesquels s'appuyer a donc disparu.

La plupart des analyses pointent l'articulation entre les attentes créées par les annonces des chercheurs et le peu de résultats des travaux effectifs comme l'une des raisons principales de cette disparition. Hans Moravec souligne dans l'ouvrage de Daniel Crevier à quel point les chercheurs de cette période de l'intelligence artificielle se sont peu à peu retrouvés enfermés dans un cercle vicieux. Se sentant obligés de promettre des résultats au moins à la hauteur de leurs premières suggestions, ils continuent à rédiger des propositions de recherche très ambitieuses, refusant malgré la réalité des résultats obtenus de prendre acte d'un contraste qui aurait pu leur coûter ces nouveaux financements[32]. Les résultats eux-mêmes sont régulièrement enjolivés dans les rapports, comme le programme SUR le montre. Autre exemple, le robot Shakey, supposé pouvoir accomplir les tâches qu'on lui donne à effectuer en les décomposant en un certain nombre d'actions. Par exemple, allumer la lumière dans une pièce implique pour Shakey de trouver l'interrupteur, de le rejoindre et d'appuyer dessus. Les vidéos livrées par les chercheurs de Stanford travaillant dessus montrent bien Shakey en train d'accomplir toute la série des actions qu'une tâche requiert de lui. Mais il s'agit d'un montage : le robot peut accomplir chacune des actions séparées mais n'a jamais réussi à les enchaîner pour mener à bien une tâche[33]. Entre annonces faramineuses, rapports enjolivés et louvoiements divers par peur de ne pas pouvoir poursuivre leurs recherches, les chercheurs en intelligence artificielle créent peu à peu une bulle d'attentes bien trop hautes au fil des années 1970, se condamnant à décevoir leurs soutiens. La période qui en résulte, nommée le premier hiver de l'intelligence artificielle et usuellement datée de 1974 à 1980, n'épargne pas les premières recherches du champ computationnel en architecture. L'expression, dérivée de celle d'hiver nucléaire, apparaît en 1984 lors d'un débat à l'American Association of Artificial Intelligence. Une partie des chercheurs du domaine souhaitent alors mettre en

[29] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.
[30] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.
[31] Voir Chapitre II.
[32] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993, p115.
[33] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.
McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

garde leurs collègues des dangers d'un trop grand enthousiasme dans leurs annonces. L'avertissement s'avère prémonitoire, puisque la suite de l'histoire du champ de l'intelligence artificielle est marquée par un renouveau que suit très vite une nouvelle chute.

Malgré les difficultés rencontrées par l'intelligence artificielle dans les années 1970, qui provoquent un changement de perception, quelques programmes de recherche continuent. Au bout de quelques années, ceux-ci débouchent sur la renaissance du champ autour d'un nouveau dispositif : les systèmes experts. Comme leur nom l'indique, les systèmes experts sont des programmes qui concernent un domaine en particulier. Ces programmes s'appuient sur les connaissances des experts de ce domaine pour répondre à des questions ou résoudre des problèmes. La leçon tirée par les chercheurs de leurs échecs des années 1970 est en effet que les comportements intelligents dépendent de la gestion du savoir[34]. Les systèmes experts sont la réponse apportée à cet enjeu au cours des années 1980 : ce sont des programmes qui encapsulent les connaissances des experts d'un domaine donné, ou plutôt leurs règles de travail. Ces programmes sont composés de deux éléments clés : la base de données et le moteur d'inférence. La base de données est une collection de faits, et le moteur d'inférence un ensemble de règles logiques. Ces règles relèvent usuellement de la logique dite de propositions, c'est-à-dire des liens logiques qui lient les différents faits et qui permettent de déterminer si une affirmation est vraie ou fausse en regard de ces liens logiques. Les systèmes experts sont aussi dotés d'une interface utilisateur, qui permet de poser des questions ou de faire vérifier une affirmation par le système. Par exemple, si le moteur d'inférence contient la règle "*tous les hommes sont mortels*", que la base de données contient l'information que "*Socrate est un homme*", et que l'on interroge le système expert pour savoir si Socrate est mortel, le programme peut en déduire que oui. La base de données des systèmes experts rassemble des faits reconnus en relation avec le domaine d'application du système. Le moteur d'inférence est cependant le lieu où le savoir des experts est traduit en règles logiques, c'est-à-dire où sont formalisées leurs règles empiriques de travail. L'objectif des systèmes experts est de parvenir à imiter la capacité des humains à prendre des décisions à partir des faits qui leur sont connus. Bien que la structure des systèmes experts soit relativement simple, ce double ensemble de faits et de règles leur permet d'exécuter un grand nombre d'activités : interprétation, prédiction, diagnostic, design, planning, débogage, réparation, instruction, contrôle[35]. Le fonctionnement des systèmes experts est basé sur le principe de l'exploration d'un arbre de possibilités. En recoupant règles et faits, le système élimine les branches qui ne correspondent pas et se rapproche peu à peu de la solution. Dans son livre, Daniel Crevier formule l'argument que la capacité de suivre plusieurs branches d'un arbre de possibilités à la fois est très importante dans nos processus de réflexion[36]. Dans les années 1980, cet argument a beaucoup fait pour le succès des

[34] McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[35] Hayes-Roth, Frederick; Waterman, Donald; Lenat, Douglas (1983). Building Expert Systems. Addison-Wesley.
[36] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.

systèmes experts, considérés comme une bonne piste pour l'intelligence artificielle puisqu'ils ont la capacité d'imiter ce processus. Selon Crevier, cela explique cependant également le succès des systèmes experts auprès des utilisateurs, auxquels le processus de fonctionnement des systèmes experts semble donc familier et compréhensible.

Si les systèmes experts sont particulièrement accessibles à des utilisateurs peu versés en informatique, c'est également parce que la barrière du langage de programmation n'existe pas : les règles énoncées par les experts le sont dans un format beaucoup plus proche du langage naturel. En effet, les systèmes experts ont des règles d'abord exclusivement au format "si… alors...", puis un peu plus variées grâce à la prise en compte d'autres propositions logiques, mais toujours formulées en langage naturel et non directement en langage de programmation. Trois systèmes experts donnent la mesure du développement de ces nouveaux programmes et de leur influence sur le champ de l'intelligence artificielle après son premier hiver. Ils vont également nous fournir des exemples des règles contenues dans ce genre de programmes, ainsi qu'une explication du cheminement intellectuel qui mène le champ de l'intelligence artificielle à les considérer comme une piste viable. Les premiers systèmes experts, DENDRAL et MYCIN, sont conçus bien avant le boom des années 1980 qui popularise ces programmes. Le premier, DENDRAL, est développé entre 1965 et 1975 par l'équipe qui se constitue pour l'occasion autour d'Edward Feigenbaum[37]. Informaticien formé à Carnegie-Mellon, où il écrit sa thèse de doctorat sous la supervision d'Herbert Simon, il propose d'abord EPAM, un modèle informatique d'apprentissage. Il est ensuite nommé directeur du centre informatique de Stanford, ce qui lui offre la possibilité de poursuivre plus en profondeur ses recherches sur l'implémentation de connaissances dans un programme informatique. Edward Feigenbaum rencontre à Stanford le biologiste Joshua Lederberg, qui est à la tête du département de génétique de l'université. Cette rencontre est le point de départ d'une collaboration pour la mise au point d'un programme informatique qui permet d'identifier des composés chimiques : DENDRAL. Parvenir à reconnaître les différents composés chimiques est en effet alors un travail long et minutieux : il faut identifier les différentes molécules, comprendre comment elles sont liées les unes aux autres, puis aller de supposition en supposition en fonction de la position des différents groupes moléculaires pour trouver à quel composé cela peut correspondre. L'identification de composés chimiques est donc longue, mais aussi très liée à l'expertise des chimistes. C'est cette expertise que DENDRAL vise à traduire en règle, pour automatiser à partir de ces règles l'identification de composés chimiques. Le programme est structuré de manière à procéder de la même façon que les chimistes qui aident à sa création : en identifiant les molécules et les sous-groupes de molécules du composé à reconnaître. DENDRAL fonctionne uniquement avec des règles au format "si… alors…", dont voici un exemple.

[37] Pour les informations qui suivent, voir Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993 et McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

*IF the spectrum for the molecule has two peaks at masses x1 and x2 such that*
*a. x1 + x2 = Molecular Weight + 28*
*b. x1 - 28 is a high peak, and*
*c. x2 - 28 is a high peak, and*
*d. at least one of x1 or x2 is high,*
*THEN the molecule contains a ketone group.*[38]

DENDRAL marque un tournant dans l'approche de l'intelligence artificielle, car il offre la démonstration qu'il est possible d'intégrer à un programme qui réalise des opérations mathématiques le savoir d'un expert, sous une forme qui permet de guider le programme vers une réponse au problème posé[39]. Il s'agit d'un important changement par rapport à l'idée à l'origine du General Problem Solver de Newell et Simon, qui part du principe que ce sont les règles logiques qui sont la source de l'intelligence. L'idée derrière DENDRAL est plutôt que c'est l'accumulation des connaissances qui forme la base des comportements intelligents, et c'est cette accumulation de connaissances que le projet cherche à reproduire via le programme pour pouvoir reconnaître les composés chimiques qui lui sont soumis.

MYCIN est développé par la même équipe au début des années 1970. L'objectif du programme est cette fois de déterminer l'origine des infections du sang. A partir de résultats de tests sanguins, de cultures bactériologiques et autres informations médicales, MYCIN émet des suggestions concernant les micro-organismes qui pourraient être responsables de l'infection et propose des antibiotiques pour traiter l'infection. Le programme fonctionne également à partir de règles au format "si… alors…", dont voici un second exemple :

*IF*
*The site of the culture is blood, and the gram strain is positive, and the portal of entry is gastrointestinal tract, and*
*[A]- the abdomen is the locus of infection, or*
*[B]- the pelvis is the locus of infection*
*THEN*
*There is strongly suggestive evidence that Enterobacteriaceae is the class of organisms for which therapy should cover.*[40]

[38] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993, p149.
[39] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993, p150.
[40] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993, p151.

En plus de confirmer l'hypothèse explorée avec DENDRAL qu'il est possible d'intégrer des savoirs spécialisés à des programmes informatiques, MYCIN apporte une distinction entre les informations et les opérateurs logiques impliqués dans la formulation des règles. C'est donc avec MYCIN qu'est mise au point la structure classique d'un système expert, que DENDRAL ne possédait pas encore. MYCIN intègre par ailleurs la capacité de prendre en compte la probabilité d'un diagnostic pour faire ses propositions. Bien que les règles soient toujours au format "si… alors…", MYCIN offre donc un niveau de complexité supplémentaire par rapport à DENDRAL, et fait ainsi la démonstration du potentiel des systèmes experts.

Le troisième système expert est plus tardif : il s'agit d'XCON, développé entre 1978 et 1979. XCON marque cependant un tournant dans la structure du champ de l'intelligence artificielle en plus de faire partie des projets qui relancent les financements de recherche après le premier hiver de l'IA. En effet, XCON[41] est un projet développé en partenariat entre Carnegie Mellon et la Digital Equipment Corporation (DEC), alors le deuxième plus grand fabricant d'ordinateurs aux Etats-Unis[42]. La DEC a une particularité : tous les systèmes informatiques qu'elle vend sont sur-mesure, conçus en échangeant avec les clients à partir de différentes pièces disponibles sur le marché. La configuration du système en fonction des besoins du client est donc pour la DEC une étape cruciale. Celle-ci se fait en trois étapes. En premier lieu, à partir d'un échange avec le client, une liste de produits DEC à intégrer dans le système est identifiée par les équipes de vente. En second, les équipes de techniciens produisent les plans du système, vérifiant au passage que la configuration est fonctionnelle. Troisièmement, un prototype du système est assemblé sur une ligne de production dédiée, pour confirmer que tout fonctionne comme prévu. A l'issue de ce travail de configuration et vérification, le système informatique peut être envoyé au client. Un tel processus de travail implique cependant de longs délais et des coûts considérables en particulier pour l'étape de prototypage. Cela implique de plus d'employer des techniciens qui maîtrisent parfaitement les applications des différents produits du catalogue DEC et qui se maintiennent constamment à jour des avancées dans le champ informatique. En effet, dès le milieu des années 1970, le catalogue de la DEC contient 50 processeurs différents et 4000 options de configuration complémentaires - une quantité conséquente d'informations à garder en tête pour conseiller un client[43]. A la fin des années 1970, les projections de DEC montrent qu'avec la croissance du secteur informatique, leur carnet de commande va beaucoup augmenter dans les années à venir. Mais les coûts liés au système de configuration en place dans l'entreprise risquent eux aussi d'exploser. L'entreprise fait donc appel à John McDermott, professeur d'informatique à Carnegie Mellon, pour concevoir XCON, un système expert dont la tâche est de détailler et vérifier les

[41] Pour eXpert CONfigurer..
[42] Sviokla, John J. "An Examination of the Impact of Expert Systems on the Firm: The Case of XCON." MIS Quarterly 14, no. 2 (1990): 127–40.
[43] Sviokla, John J. "An Examination of the Impact of Expert Systems on the Firm: The Case of XCON." MIS Quarterly 14, no. 2 (1990): 127–40.

configurations. La base de données d'XCON comprend l'ensemble des produits de la DEC et son moteur d'inférence des règles de configuration. A partir d'une feuille de commande, XCON produit pour la configuration envisagée les dessins techniques et la liste exacte des pièces, indique si certaines pièces dans la commande fournie par le service de vente sont superflues ou manquantes et fournit également des indications sur les pièces de rechange et la performance du système. Testée localement dans l'une des antennes de la DEC fin 1979, XCON est en usage dans toute l'entreprise dès le début de l'année 1981[44]. Les résultats sont éblouissants. A l'exception d'un très petit nombre de systèmes sur-mesure encore gérés manuellement, toutes les commandes passent par XCON. Les vérifications du fonctionnement des configurations sont correctes dans 95% des cas, alors que les opérateurs humains en charge de cette étape avant XCON n'atteignaient que 65%. Des estimations montrent que la simplification du processus de configuration et la suppression de l'étape de prototypage et de test permettent à la DEC d'économiser jusqu'à 25 millions de dollars par an.

Plus tardif, XCON est aussi la démonstration éclatante des bénéfices que peuvent apporter les systèmes experts aux entreprises. Le succès d'XCON contribue à relancer les recherches dans le champ de l'intelligence artificielle, mais avec des financements privés bien plus importants qu'avant. De nombreuses entreprises suivent l'exemple de la DEC et décident de développer des systèmes experts en interne pour les assister dans leurs activités. Cela fait naître entre 1980 et 1985 toute une industrie de l'intelligence artificielle, qui fournit ces entreprises en systèmes experts. IntelliCorp par exemple propose Knowledge Engineering Environment (KEE), qui permet d'implémenter des systèmes experts sur des machines LISP. Aion, autre entreprise fondée à la même période, utilise son propre moteur d'inférences pour développer ce que ses dirigeants appellent des *systèmes experts métiers*, adaptés à nombre de professions. Chacune de ces entreprises se base sur son propre moteur d'inférence pour mettre en place des systèmes experts chez les entreprises clientes. Chaque système expert est différent, mais le moteur d'inférence est universel, c'est-à-dire qu'il offre le recours au même ensemble d'opérateurs logiques pour construire les règles. Le développement du langage Prolog en France est très utile à ces moteurs d'inférence, car il permet d'exprimer facilement des règles non seulement au format "si… alors…", mais aussi avec toute la logique du premier ordre. Les nouvelles entreprises qui développent des systèmes experts sont par ailleurs étroitement associées aux entreprises de hardware qui émergent à la même période, comme Symbolics ou Lisp Machines, qui commercialisent les machines LISP développées au MIT. Ensemble, ces entreprises forment une bulle informatique. Le champ de l'intelligence artificielle s'appuie sur ce nouveau monde pour financer de nouvelles recherches. L'une des plus grosses initiatives privées est le groupement américain Microelectronics and Computer Corporation (MCC), où un grand nombre de ces entreprises joignent leurs efforts pour financer des projets d'intelligence artificielle qui développent plus avant les

[44] Sviokla, John J. "An Examination of the Impact of Expert Systems on the Firm: The Case of XCON." MIS Quarterly 14, no. 2 (1990): 127–40.

systèmes experts. La MCC dépense 850 millions de dollars au cours des années 1980 pour concevoir des machines qui puissent raisonner comme des humains, en investissant en particulier dans des projets sur la reconnaissance de langage et d'image[45].

Devant le succès des systèmes experts, les financements publics reviennent eux aussi à partir de 1980, première année de ce que les historiens identifient comme le second âge d'or de l'intelligence artificielle. En 1981, le Japon lance un projet d'ordinateur de cinquième génération, qui a pour objectif de mettre au point une intelligence artificielle en dix ans. Les systèmes experts occupent une place très importante dans la feuille de route initiale du projet. En Angleterre, le projet Alvey prévoit l'investissement de 350 millions de livres dans des recherches aux objectifs similaires. Aux Etats-Unis, la DARPA augmente de nouveau largement les financements attribués à des projets de recherche en intelligence artificielle entre 1984 et 1988[46]. Pendant la deuxième moitié des années 1980, Carnegie-Mellon travaille de nouveau sur ses programmes d'échecs, et rencontre ses premiers succès avec HiTech et Deep Thought. Le renouveau des investissements marque donc des progrès significatifs dans le domaine, grâce aux systèmes experts dont le développement se poursuit mais aussi grâce à d'autres programmes. Les années 1980 sont aussi le moment de lancement de projets de bases de données ou stocker le maximum d'informations sur le monde. L'objectif de ces recherches est de traiter le problème des connaissances de sens commun, en les accumulant dans ces bases de données. Celles-ci peuvent ensuite par ailleurs servir de base à des systèmes experts de plus en plus généralistes. Puisque c'est la manipulation des connaissances qui est désormais identifiée comme source d'intelligence, il faut numériser le maximum de connaissances pour produire des systèmes intelligents. Ces projets ont donc vocation à développer le stockage et la représentation des connaissances sur lesquelles s'appuient les systèmes experts, en même temps que les moteurs d'inférence se diversifient. Cyc, par exemple, lancé en 1984 par la MCC et mené par l'informaticien Douglas Lenat, se concentre sur la formulation des connaissances implicites que requièrent les raisonnements humains. Le projet occupe à son lancement 400 personnes et repose sur le développement d'un langage de programmation supposé permettre la formulation d'une ontologie complète et cohérente sur laquelle une intelligence artificielle puisse s'appuyer. Cyc est aujourd'hui toujours en cours, sous la supervision de Cycorp, fondée en 1995 par Douglas Lenat pour poursuivre le projet après le désengagement de MCC. Les bases de données publiées année après année par l'entreprise contiennent des millions d'informations et ont été utilisées dans divers projets de recherches, du domaine de la pharmacie à celui du contre-terrorisme. Bien que critiqué parce que ses

[45] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993. McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.
[46] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993. McCorduck, P., *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

objectifs s'ancrent dans des paradigmes de recherche dépassés depuis longtemps, Cyc est aujourd'hui une des plus grosses bases de données existantes[47].

L'une des critiques principales faites à Cyc, le travail de fourmi nécessaire pour rassembler et articuler les connaissances. est aussi faite aux systèmes experts, qui malgré leur succès vont poser des problèmes assez rapidement. En effet les systèmes experts souffrent très vite des mêmes problèmes que leurs ancêtres du premier âge d'or, et précipitent un second hiver de l'intelligence artificielle entre 1987 et 1993. Les bases de données nécessaires au fonctionnement des systèmes experts sont conséquentes. Il suffit d'imaginer la quantité d'informations relatives aux infections du sang à saisir pour que MYCIN fonctionne, ou la quantité d'informations relatives à n'importe quel composé chimique existant nécessaires au fonctionnement de DENDRAL pour se rendre compte de l'ampleur de ces bases de données. Or ces exemples de systèmes experts portent seulement sur une tâche très précise. Dominique Cardon et ses collègues utilisent dans leur article l'expression "cathédrales de règles" pour pointer ce problème, et c'est bien de cela qu'il s'agit[48]. Ces cathédrales de règles imposent par ailleurs la présence constante d'un ou plusieurs experts pour le développement du système, mais aussi, en fonction de son domaine d'application, pour son maintien. C'est le cas pour XCON, qui a besoin d'être sans cesse mis à jour pour intégrer les nouveaux produits de la DEC et les configurations associées dans la base de données. La DEC emploie ainsi au cours des années 1980 près de 30 personnes uniquement pour maintenir la base de données d'XCON[49]. De plus, les systèmes experts ne fonctionnent que dans des contextes particuliers très précis, pour lesquels leur base de données et leur moteur d'inférence a été mis au point. Malgré les tentatives de diversification des connaissances portées par les projets comparables à Cyc, les systèmes experts ne parviennent pas à monter en généralité. La somme des connaissances de sens commun est trop grande à intégrer, et très difficile à cerner puisqu'il s'agit généralement de connaissances que personne n'a vraiment conscience d'avoir[50]. Les systèmes experts ont le gros défaut d'être *friables*, c'est-à-dire que dès que les entrées fournies au système diffèrent ne serait-ce qu'un peu de ce qui a été initialement prévu, les sorties deviennent erratiques et inutilisables[51]. Hubert et Stuart Dreyfus, qui proposent dans Mind over Machine une longue critique des systèmes experts, formulent le problème comme suit : les systèmes experts "souffrent de l'impossibilité de remplacer le savoir-faire contextuel par des connaissances

[47] Cynthia Matuszek, John Cabral, Michael Witbrock, John DeOliveira, An Introduction to the Syntax and Content of Cyc, 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, Stanford, CA, March 2006.
[48]Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.
[49] Sviokla, John J. "An Examination of the Impact of Expert Systems on the Firm: The Case of XCON." MIS Quarterly 14, no. 2 (1990): 127–40.
[50] La publication de la base de données de Cyc a par exemple plus pour objectif de mettre en évidence ce qui n'y est pas pour la nourrir en utilisant les remarques faites.
[51] *brittle* en anglais.

détachées"[52]. Les principes de travail sur lesquels se fondent les règles intégrées aux systèmes experts sont applicables "toutes choses égales par ailleurs", et ce que ceci signifie dans une situation précise ne peut jamais être totalement formalisé[53]. L'objectif des systèmes experts était pourtant justement de parvenir à capturer le versant non-algorithmique de la prise de décision, voyant dans leur architecture une réponse au problème des connaissances de sens commun et un chemin vers l'intelligence artificielle.

Se concentrer sur un domaine précis permettait aux premiers systèmes experts d'éviter ce problème des connaissances de sens commun. En effet, puisque les utilisateurs qui manipulent le système expert évoluent dans le domaine concerné, ils connaissent usuellement les concepts qui sont manipulés. Ceux-ci n'ont donc pas besoin d'être définis, seulement d'être associés à un symbole qui puisse être traité par l'ordinateur et reconnu par l'expert. Nous avons vu au chapitre précédent que ceci est à la source d'une déconnection entre certains savoirs de l'expert et les entrées et sorties traitées par le programme. Nous avons aussi vu comment cette déconnection est mise en œuvre dans les premières expérimentations informatiques des architectes du champ computationnel, par exemple en opérant sur le plan en amont ou en aval du programme informatiques. Cette déconnection constitue de fait une limite des systèmes experts sur deux plans. D'une part, cela implique que le système expert ne saisit en fait pas tout le savoir de l'expert. C'est la critique faite par Dreyfus et Dreyfus. D'autre part, cela implique une relation particulière des experts aux systèmes experts. Les systèmes experts ont en effet besoin des experts pour être mis en place et mis à jour, voire même tout simplement pour être utilisés. L'exemple des systèmes experts médicaux étudiés par Marc Berg dans son livre *Rationalizing Medical Work*[54], et les raisons de leur échec, fournissent un cas d'étude intéressant concernant les implications de cette relation particulière aux experts. Les raisons avancées par Berg aux difficultés d'adoption des systèmes experts en médecine sont au nombre de trois. D'abord, comme ailleurs, les difficultés techniques, la friabilité et le besoin d'entretien des systèmes experts les fragilisent. Mais les deux autres raisons sont à chercher du côté de la perception par les praticiens de ces nouveaux outils. Les systèmes experts nourrissent en effet des inquiétudes chez les médecins, qui craignent non seulement d'être remplacés, mais qui plus est d'être remplacés par des systèmes inadéquats. Ces programmes sont initialement conçus pour transformer les prises de décision dans les domaines où ils sont implémentés de réflexions individuelles et personnalisés en processus plus rationnels[55]. Cela inquiète les médecins, qui craignent que d'autres facteurs ne prennent de l'importance dans les

[52] "*[expert systems] suffer from the impossibility of replacing involved knowing how with detached knowing that*" Dreyfus, H. & Dreyfus, S., Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer, New York, NY: Free Press, 1987.

[53] Crevier, D., AI: The Tumultuous Search for Artificial Intelligence. New York, NY: BasicBooks, 1993.; on retrouve exactement ce qu'on disait plus haut sur les raisons du premier hiver

[54] Marc Berg, *Rationalizing Medical Work. Decision Support Techniques and Medical Practices*, The MIT Press, 1997.

[55] Sviokla, John J. "An Examination of the Impact of Expert Systems on the Firm: The Case of XCON." MIS Quarterly 14, no. 2 (1990): 127–40.

décisions cliniques prises par des systèmes experts commandités par des non médecins. Les assureurs, les financeurs, les services juridiques ou comptables prendraient ainsi de l'importance dans la pratique de la médecine, au détriment des médecins eux-mêmes. L'altération de la pratique médicale induite par la rationalisation et portée par le recours aux systèmes experts braque ainsi les praticiens, qui s'y opposent. C'est cette même logique qu'on retrouve dans l'étude faite par Virginia Eubanks des systèmes experts développés par différents Etats américains pour décider de l'attribution des aides sociales. Les témoignages donnés dans son livre *Automating Inequality*[56] montrent bien que les fonctionnaires utilisateurs des systèmes et chargés de suivre les dossiers de tels systèmes ne sont pas dupes des règles d'attribution. Ils se rendent très bien compte qu'elles sont formulées non pour être justes mais pour diminuer le nombre de bénéficiaires et permettre aux Etats de faire des économies. Si les fonctionnaires de ces administrations ont peu d'influence sur le recours à des systèmes experts dans ce cadre, de manière générale la méfiance des utilisateurs joue cependant un rôle dans nombre de domaines d'applications, où les systèmes experts ne percent pas en raison de cette défiance.

Les difficultés rencontrées par les systèmes experts empêchent leurs promoteurs de remplir les objectifs annoncés. Les institutions qui financent cette seconde vague de recherches en intelligence artificielle sont à nouveau déçues, et suspendent à nouveau leur soutien. À la fin des années 1980, un nouveau directeur farouchement anti-IA et convaincu que ces recherches ne mènent à rien arrive à la DARPA,. L'agence met donc à l'arrêt tous les projets d'intelligence artificielle dans lesquelles elle est impliquée. Au début des années 1990, le Japon suspend son projet d'ordinateur de cinquième génération faute de résultats. Quant à l'industrie fraîchement éclose autour des systèmes experts, elle est devenue en quelques années une bulle économique qui explose. Les défauts d'utilisation des systèmes experts se combinent à un coût élevé d'acquisition et d'entretien, alors que les PC d'Apple ou d'IBM deviennent plus puissants, plus généralistes et moins chers au fil des années 1980. Plus personne ne veut des systèmes experts et de leur lourde machinerie, et plus personne ne souhaite non plus investir dans l'entretien des systèmes experts déjà acquis. Le marché s'effondre donc, marqué notamment par la faillite des entreprises qui commercialisent des systèmes Lisp, devenus complètement obsolètes. 1987 marque le début du deuxième hiver de l'intelligence artificielle, qui marque l'échec des systèmes experts. Ce second hiver est aussi causé par le retentissement médiatique encore plus grand qu'à la première période de développement de l'intelligence artificielle. Les chercheurs du monde universitaire ont renoué lors du second âge d'or avec des annonces d'une folle ambition, mais l'industrie qui découle de l'exploitation commerciale des systèmes experts fait aussi une publicité considérable. Ceci persuade le grand public que des progrès significatifs dans le domaine de l'intelligence artificielle sont proches, et fait gonfler d'autant les attentes, puis les déceptions à l'éclatement de la bulle. Il faut cependant noter l'hypothèse que les systèmes experts ont

[56] Eubanks, Virginia. Automating inequality: how high-tech tools profile, police, and punish the poor. First Edition. New York, NY: St. Martin's Press, 2017.

été un échec parce qu'ils ne fonctionnaient pas très bien, mais aussi parce que les éléments qui fonctionnaient bien ont été absorbés par l'industrie de l'informatique pendant que le terme de "système expert" tombait en désuétude. L'architecture contemporaine des ordinateurs intègre certaines logiques des systèmes experts, et ce sont aussi ces programmes qui ont vu les premières applications de notions de programmation orientée objet, si usitée aujourd'hui. Certains principes des systèmes experts sont donc passés dans les habitudes de conception de hardware et de software malgré leur échec.

*3.1.2 Un champ computationnel qui survit tant bien que mal*

Après son premier hiver, le renouveau de l'intelligence artificielle autour des systèmes experts s'accompagne d'une transition vers le privé. Cette nouvelle vague n'a cependant pas beaucoup plus de succès puisqu'elle se solde par un second hiver. Le champ de l'intelligence artificielle traverse donc au cours des années 1970 à 1990 plusieurs périodes de désamour. Ces deux hivers successifs touchent durement le champ, et ne sont pas sans impact sur son champ cousin de l'architecture computationnelle. Comment les pratiques y évoluent-elles ? D'abord, les praticiens à l'œuvre poursuivent leurs recherches : Charles Eastman, Bill Hillier ou William Mitchell continuent à creuser le sillon que nous avons évoqué au chapitre précédent. Ils restent néanmoins cantonnés à une petite sphère de travail pour beaucoup. Nous avons mentionné que beaucoup de praticiens de cette génération ont développé leur travail dans une petite bulle académique, au sein d'une université leur ayant donné un poste de professeur, avec un ou deux collègues et quelques élèves mais pas plus. Cette croissance limitée est liée à l'hiver procédural, dont on observe donc les effets bien que les praticiens continuent à exercer. C'est en effet notamment pour cette raison que les pionniers ont tellement évolué de manière solitaire pendant les années 1970 et 1980[57]. John et Julia Frazer, par exemple, dont le travail sur l'architecture évolutionnaire constitue un des piliers de développement du champ computationnel, y travaillent depuis la fin des années 1960. Mais le pic de leur travail se situe bien plus tard, entre 1989 et 1996, au sein de leur studio de projet à la AA. Or cette période est justement celle du renouveau du champ de l'intelligence artificielle, entre ses deux hivers. Les seuls praticiens qui accèdent à une plus grande renommée et à plus de responsabilités - non plus au sein du champ mais plus largement dans le domaine architectural - sont ceux qui ne restent pas cantonnés à des sujets qui ont trait au computationnel. Ainsi, le travail de Christopher Alexander autour des motifs architecturaux est bien plus ambitieux qu'un simple usage en programmation informatique, puisqu'il s'agit d'une description de la pratique architecturale au sens large. Le travail en théorie du design de Nigel Cross porte également sur la conception architecturale dans son ensemble. Yona Friedman se détache très vite des expérimentations de programmation menées avec le MIT. William Mitchell, lui,

[57] Voir le commentaire sur leurs carrières au Chapitre II.

promeut la conception assistée par ordinateur globalement, et non seulement le recours aux algorithmes.

La fin des financements touche aussi les expérimentations architecturales qui avaient lieu au sein du champ de l'intelligence artificielle. La recherche sur les processus de conception algorithmique entre donc dans ce que nous allons appeler l'hiver procédural, en référence aux hivers de l'intelligence artificielle. Jusque dans les années 1990, une poignée de chercheurs seulement travaillent sur la question, à peine plus qu'au début du champ, et le nombre de projets et de publications n'augmente que peu en comparaison de ce que nous avons décrit au chapitre précédent. Les articles sont publiés dans les annales de l'Association for Computer Machinery (ACM), dans le journal Automation in Construction, ou dans les publications de la Design Research Association, toujours en bordure du champ computationnel lui-même. Néanmoins, quelques nouveaux praticiens apparaissent tout de même dans le champ. Leur travail s'inscrit dans la continuité de celui de leurs aînés, abordant les mêmes problématiques. Ici et là dans des universités européennes, quelques praticiens se piquent d'intérêt pour les applications de la programmation informatique à l'architecture, à la lecture des publications de leurs collègues anglophones. C'est notamment le cas de Per Galle, architecte formé à l'Académie Royale des Beaux-Arts du Danemark, à Copenhague. Son diplôme porte sur le développement d'un algorithme de hiérarchisation d'espaces en plan, et fait l'objet d'une publication dans les annales de l'ACM[58]. L'association est alors très reconnue pour ses contributions aux sciences informatiques, et cette publication permet à Per Galle de décrocher une bourse de doctorat à l'Université de Copenhague pour poursuivre ses recherches. Il s'y consacre pendant les dix années qui suivent, bénéficiant de plusieurs financements successifs, avant de finalement se tourner vers la théorie du design et de revenir enseigner à l'Académie Royale des Beaux-Arts du Danemark, où il exerce encore aujourd'hui. Dans les pôles existants, quelques étudiants formés par des praticiens déjà en poste rejoignent les groupes de recherche à l'œuvre. C'est le cas de John Peponis, formé de sa licence à son doctorat à la Bartlett à Londres, entre 1973 et 1983. Il rejoint ensuite le Georgia Institute of Technology. Il y importe les travaux de Bill Hillier sur la Space Syntax, auxquels il a contribué au cours de son doctorat. Au Georgia Institute of Technology, John Peponis côtoie Athanassios Economou, qui y dirige le Shape Computation Lab. Athanassios Economou a d'abord été formé à l'Université Polytechnique Nationale d'Athènes, avant de suivre un master d'architecture à l'université privée de Californie du Sud. Il obtient ensuite un doctorat à UCLA. Les recherches menées dans le cadre de sa thèse portent sur la notion de symétrie en architecture et en musique, et sur les opérations algébriques qui y correspondent. Ceci devient ensuite la base de ce qu'il nomme la computation des formes, sorte de dérivé de la space syntax qui s'intéresse à la combinaison de formes grâce à des opérations algébriques diverses, le tout permettant de donner corps à des plans ou des

[58] Per Galle, "An algorithm for exhaustive generation of building floor plans", *Communications of the ACM*, Volume 24 Issue 12 Pages 813-825, 1981.

coupes. George Stiny, professeur au MIT et à UCLA où il a également été formé, s'intéresse à la même question, qui donne naissance à l'idée de grammaire de formes. Ces grammaires sont des collections de formes et de règles de combinaison de ces formes, qui peuvent être intégrées à un programme informatique pour que celui-ci génère des projets à partir de ces combinaisons. Chacun de ces praticiens est formé ou influencé au cours de ses études par des praticiens du champ déjà en exercice, comme John Peponis, formé par Bill Hillier à la Bartlett. Le lien est parfois indirect : c'est le cas avec Per Galle, sur qui le travail de Christopher Alexander a énormément d'influence bien qu'ils n'exercent pas ensemble[59].

[59] Entretien Per Galle. En témoignent aussi des textes plus récents signés par Per Galle sur le travail de Christopher Alexander : Galle, P. (2020). Christopher Alexander's battle for beauty in a world turning ugly: The inception of a science of architecture? She Ji: The Journal of Design, Economics, and Innovation, 6(3), 345-385 et Galle, P. (2020). Christopher Alexander's battle for beauty: Any prospects of victory? (Author's response.) She Ji: The Journal of Design, Economics, and Innovation, 6(3), 380-385.

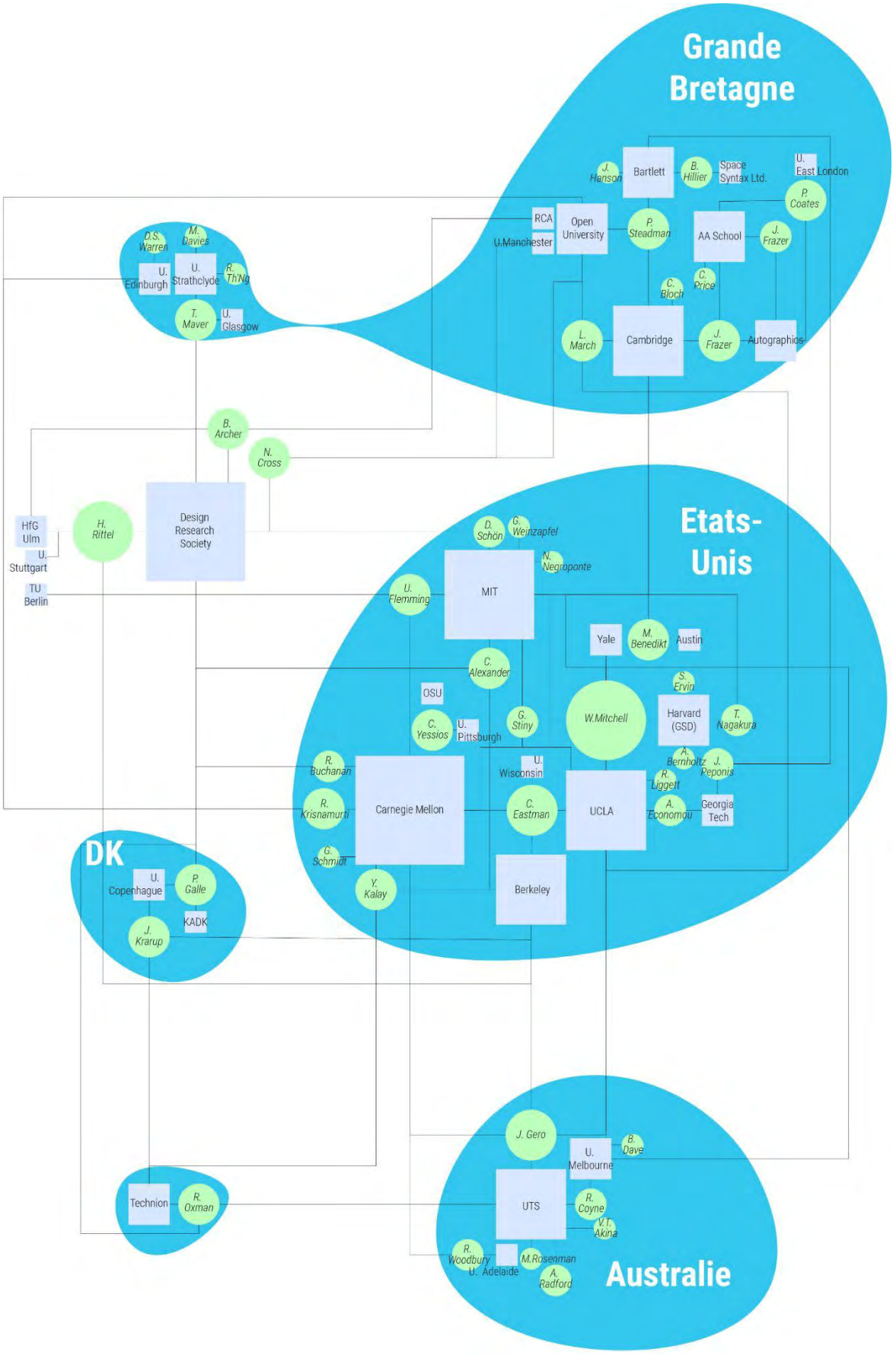

Grande Bretagne
Etats-Unis
DK
Australie
D.S. Warren
M. Davies
U. Edinburgh
U. Strathclyde
R. Th'Ng
T. Maver
U. Glasgow
RCA
U.Manchester
Open University
J. Hanson
Bartlett
B. Hillier
Space Syntax Ltd.
U. East London
P. Coates
P. Steadman
AA School
J. Frazer
C. Price
C. Bloch
L. March
Cambridge
J. Frazer
Autographics
B. Archer
N. Cross
HfG Ulm
U. Stuttgart
TU Berlin
H. Rittel
Design Research Society
D. Schön
G. Weinzapfel
N. Negroponte
U. Flemming
MIT
Yale
M. Benedikt
Austin
C. Alexander
S. Ervin
OSU
C. Yessios
U. Pittsburgh
G. Stiny
W.Mitchell
Harvard (GSD)
T. Nagakura
A. Bernholtz
J. Peponis
R. Liggett
U. Wisconsin
R. Buchanan
R. Krisnamurti
Carnegie Mellon
C. Eastman
UCLA
A. Economou
Georgia Tech
G. Schmidt
Y. Kalay
Berkeley
U. Copenhague
P. Galle
KADK
J. Krarup
Technion
R. Oxman
J. Gero
U. Melbourne
B. Dave
UTS
R. Coyne
V.T. Akina
R. Woodbury
U. Adelaide
M.Rosenman
A. Radford

*Figure 1. Réseau du champ computationnel dans sa deuxième période de développement*

Si l'on observe le développement du champ à cette période en considérant les pôles qui le composent, on constate que les pôles américain et anglais demeurent les deux principaux (Fig. X). Ils sont cependant rejoints par le pôle australien, marqué par un très fort développement dans les années 1980. En effet, ce pôle était jusqu'alors constitué presque uniquement par John Gero, professeur à l'Université de Sydney. Ingénieur de formation et donc initialement spécialisé dans l'analyse structurelle, pour laquelle il recourt à des modèles informatiques dès les années 1970, John Gero s'intéresse petit à petit à l'usage de ces outils pour la conception architecturale, en particulier pour l'optimisation de plans. A partir de la fin des années 1970, il est rejoint par plusieurs praticiens qui viennent grossir les rangs de son unité de recherche. Anthony Radford, diplômé d'urbanisme de l'université de Newcastle et passé ensuite par quelques agences de la région, entame son doctorat à l'université de Sydney en 1978. Encadré par John Gero, leur collaboration se poursuit ensuite jusqu'en 1989. Richard Coyne, diplômé d'architecture de l'université de Melbourne, rejoint l'équipe en 1982. Ayant également exercé en agence auparavant, il accomplit lui aussi sa thèse sous la supervision de John Gero, et reste au sein de l'unité jusqu'au début des années 2000, où il déménage à Edimbourg. Le développement du pôle australien est aussi marqué par une intensification des échanges avec d'autres pôles du réseau. Après un diplôme d'architecture au Technion, en Israël, Rivka Oxman rejoint le groupe à la fin des années 1970, et y travaille plusieurs années avec ses membres, avant de retourner au Technion ou elle termine en 1988 une thèse consacrée aux systèmes experts en architecture. Antony Radford collabore également sur un projet de système expert avec William Mitchell, et Gero collabore également le temps d'un projet avec Robert Woodbury, architecte formé au Canada puis à Carnegie Mellon, qui s'intéresse dès les années 1980 à l'idée de modèles paramétriques. À travers John Gero et les collaborateurs qu'il s'attache, le pôle australien est donc particulièrement actif à cette période pourtant autrement un peu ralentie pour les pratiques computationnelles. Son unité est en effet à cheval entre architecture computationnelle et ingénierie, avec des thématiques de recherche toujours à la frontière du génie civil et de la construction. Ancré dans la pratique, très concentré sur les applications constructives, avec une articulation de leurs recherches éloignées des questionnements du champ de l'intelligence artificielle, le groupe préserve ainsi ses financements et son développement.

Malgré le ralentissement des recherches en intelligence artificielle et dans le champ computationnel, des travaux continuent d'être menés par une poignée de praticiens. Leurs recherches sont consacrées à l'identification des informations nécessaires à la conception architecturale et à leur formalisation en instructions. Les programmes élaborés le sont toujours en collaboration avec des informaticiens. John Gero et ses collègues travaillent avec la Computational Application Research Unit de leur université,

Per Galle avec László Béla Kovács, programmeur à l'université de Copenhague, George Stiny avec James Dips. Une partie des recherches accomplies l'est dans une logique d'extension de l'axe de recherche vu au chapitre précédent autour de la décomposition de l'espace. Comme les isovists et la space syntax, qui continuent à être développés, d'autres systèmes computationnels de description et de génération de l'espace voient le jour. Ce sont des tentatives de systématisation, dont l'idée est de parvenir à des règles de production de l'espace applicables en tout cas. Per Galle analyse par exemple au fil de ses projets de recherche l'ensemble des conditions à examiner pour la génération d'un plan, dont l'objectif est de construire peu à peu une méthode algorithmique permettant d'évaluer l'ensemble des configurations possibles pour un nombre d'espaces donné et leurs connexions[60]. L'enjeu est de ne pas recourir à des techniques d'optimisation classiques comme celles utilisées par John Gero. Ces techniques prennent des solutions au hasard parmi celles disponibles et progressent ensuite vers les meilleurs en se basant sur les paramètres des solutions les plus intéressantes parmi celles sélectionnées au hasard. Le travail de Per Galle, à l'inverse, vise à éliminer cette sélection au hasard au profit d'une recherche systématique[61]. Charles Eastman également s'intéresse au cours des années 1980 à cette approche, proposant notamment un General Space Planner, dont le nom fait directement référence au General Problem Solver de Newell et Simon[62]. Ci-dessous, le pseudo-code d'un des programmes élaborés par Per Galle, qui montre l'ambition d'analyser toutes les solutions possibles.

*•GIVEN MODULAR COMPLEXITY m, FIND ALL PLANS":*

*• For every* $P \subseteq \{r_h \in R \mid f_h \leqq A^*\}$ *:*

*• Find a room sequence* $R' = (r'_1, r'_2, \ldots, r'_i, r'_{n+1})$*, according to Eq. (1).*

*• "GIVEN SET P OF SMALLEST ROOMS, FIND ALL PLANS":*

*(Comment: This action computes the cell distributions for a fixed P.)*

*• Let i be I.*

*• "FIND NUMBER OF GRID CELLS FOR* $r'_i$ *AND SUBSEQUENT ROOMS IN R'" :*

*(Comment: This action is recursively defined.)*

*• If* $i > n + 1$ *(Comment: A cell distribution has been found and is given by the* $f_i$ *values most recently stored) then*

*• GIVEN CELL DISTRIBUTION* $f_i$ *FIND ALL PLANS":*

*(Figure 6),*

*else*

*• Compute cell-number bounds* $f_i$ *and* $f^+_i$ *according to Eqs. (9) and (10).*

---

[60] Per Galle, "An algorithm for exhaustive generation of building floor plans", *Communications of the ACM*, Volume 24 Issue 12 Pages 813-825, 1981 et articles suivants.

[61] Galle, P , 1987c, "Branch & Sample: Systematic combinatorial search without optimization", TR 87/5, Department of Computer Science, University of Copenhagen, Copenhagem

[62] Baybars, I, Eastman, C M, 1980, "Enumerating architectural arrangements by generating their underlying graphs" Environment and Planning B 7 289–310. Charles M. Eastman, "Automated space planning", Artificial Intelligence, Volume 4, Issue 1, p. 41-64, 1973.

*• For every natural number $f_i$ for which $f^-_i \leqq f_i \leqq f^+_i$*

*(Comment: i.e., for all $f_i$ satisfying Eqs. (5), (6), and (7)):*

*• Compute cell-area bounds $a^-_i(f)$ and $a^+_i(f)$ according to defmitions (2) and (3), and store the results.*

*• If $a^-_i(f) \leqq a^+_i(f)$ (Comment: i.e., Eq. (4) is satisfied) then*

*• Store $f_i$.*

*• Increase i by 1.*

*• Find number of grid cells for r\ and subsequent rooms in R'.*

*• Decrease i by 1.*

Les grammaires de formes sont un second exemple de ces travaux. Développées par George Stiny et James Dips d'abord pour la peinture et la sculpture puis pour l'architecture, elles sont définies comme suit par les deux auteurs :

> *Les grammaires de formes sont une forme de calcul purement visuel. Les primitives des grammaires de formes sont des formes, plutôt que des symboles. Les relations et les opérations sont toutes spatiales (par exemple, la similarité, la rotation) plutôt que symboliques. On peut montrer l'équivalence formelle des calculs symboliques traditionnels*[63].

La volonté de pouvoir voir les formes manipulées et de formuler les transformations à partir de ce vocabulaire de formes plutôt que de devoir exprimer des règles de transformation entièrement par le biais de fonctions mathématiques est au cœur de ces recherches. Il s'agit de computation "purement visuelle", pour reprendre les mots d'Athanassios Economou, dont la computation des formes s'inscrit exactement dans la même idée d'utilisation du langage visuel pour l'expression des règles. Aucune de ces recherches n'a débouché sur une méthode de formalisation universelle de la conception architecturale. Néanmoins, elles sont à l'origine de nombreux dispositifs algorithmiques en usage dans le champ computationnel. Les travaux de Per Galle sont très proches des méthodes de relaxation dynamique, les travaux de Gero sur l'optimisation s'appuient sur des algorithmes génétiques, la récursivité observée dans les grammaires de formes de Georges Stiny rappelle celle des systèmes de croissance. Or tous ces algorithmes sont ceux qui sont aujourd'hui employés dans le champ, comme nous le verrons au chapitre V. C'est dans les pratiques de ce groupe de chercheurs qu'ils trouvent leurs racines.

---

[63] "*Shape grammars are a form of purely visual computation. The primitives in shape grammars are shapes, rather than symbols. The relationships and operations are all spatial (e.g. similarity, rotation) rather than symbolic. One can show the formal equivalence of traditional symbolic computations",* Gips, J. Implementing Shape Grammars with Computers, conference paper presented at The NSF/MIT Workshop on Shape Computation, 1999.

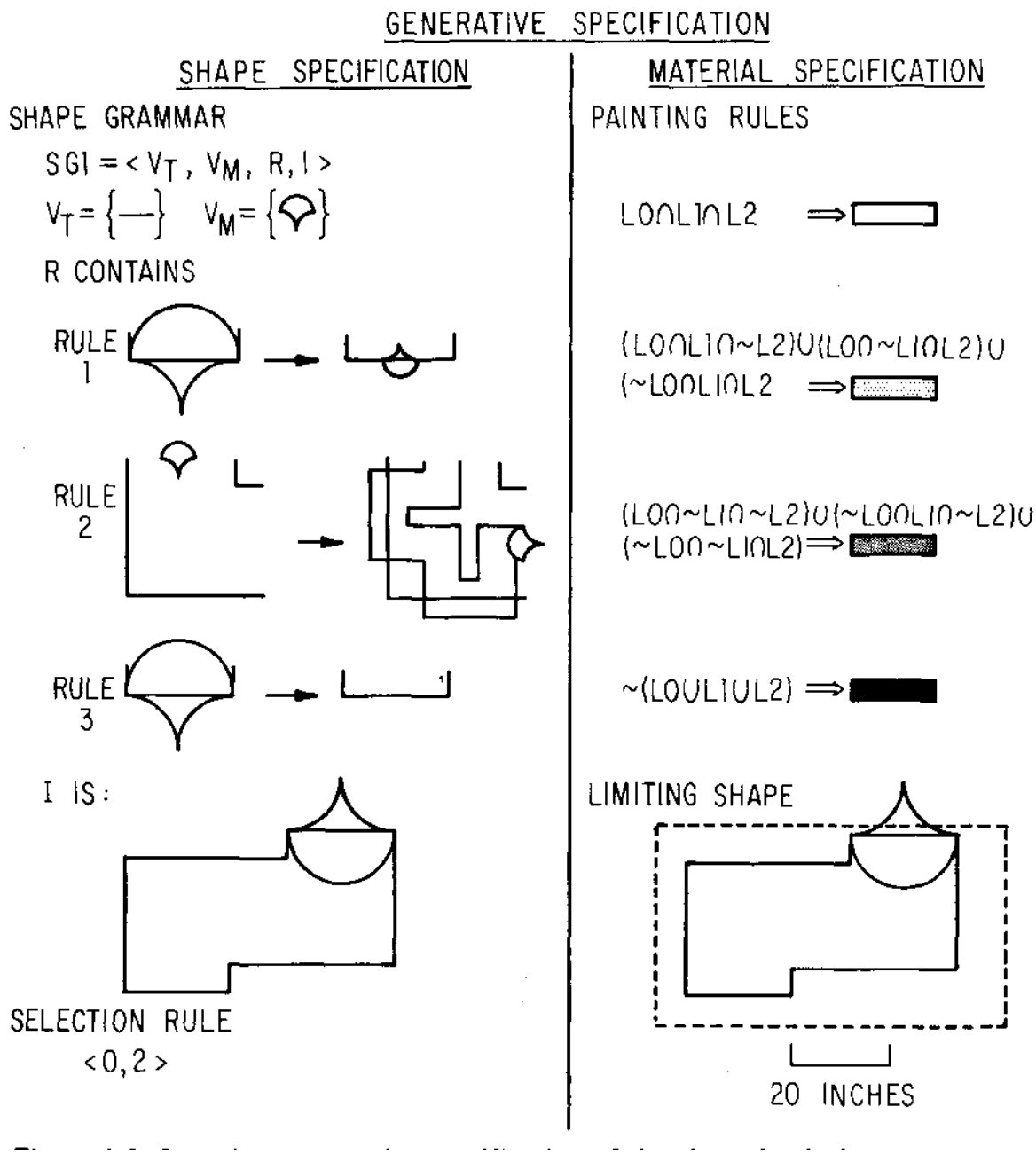



*Figure 2. Extraits des règles de composition présentées par George Stiny et James Dips*

Le second volet de travaux relève moins d'une ambition de systématisation des règles de conception employées pour pouvoir les appliquer à n'importe quel projet. Elle relève néanmoins toujours de la même ambition de formalisation, et se double d'une analyse de projets prédatant le développement de l'informatique. Cette analyse vise à trouver des règles de composition de ces projets, puis à les intégrer à un programme qui permette de générer des espaces similaires. Ces reproductions algorithmiques permettent d'interroger de près le processus de conception et son explicitation. George

Stiny développe avec William Mitchell en 1978 une grammaire formelle inspirée de Palladio[64]. Plus exactement, ayant identifié qu'une des caractéristiques majeures de l'architecture de Palladio est de suivre des règles systématiques de composition du plan, ils se lancent dans l'automatisation de ce processus[65]. Palladio a décrit dans ses Quattro Libri dell'Architettura sa méthode de composition, que Stiny et Mitchell transforment en une série d'opérations de dessin symétriques basés sur une grille. Le programme est composé d'une liste de 72 règles qui identifient chacune une de ces opérations géométriques. En combinant les règles adéquates, l'utilisateur du programme peut reproduire les villas construites par Palladio ou en inventer de nouvelles. Le projet *Possible Palladian Villas (Plus a Few Instructively Impossible Ones)*[66], développé quelques années plus tard par les historiens de l'architecture Richard Freedman et George L. Heersey, reprend exactement les mêmes principes. Le livre publié se penche cependant plus avant sur les villas imaginaires, non dessinées par Palladio mais fournissant selon les deux historiens des indications supplémentaires sur le travail de l'architecte puisqu'elles suivent les mêmes règles de composition. Les travaux de Stiny et Mitchell, eux, s'intéressent plus au savoir que ces expérimentations permettent d'obtenir sur les connaissances qu'implique le processus de conception architectural. Bien que plus tardifs, les travaux d'Athanassios Economou relèvent de la même stratégie. En collaboration avec ses étudiants, il s'intéresse aux architectes modernes et à la création de règles de conception qui puissent être intégrées à des programmes. L'un de ses projets, mené avec Edouard Din, s'intéresse à Richard Meier et aux typologies de séparation verticales qu'il emploie[67]. Le programme mis au point utilise une grille de neuf cellules dont chacune correspond à une caractéristique : ouvert, fermé, opaque, transparent. La combinaison de ces caractéristiques permet de créer cloisons, portes ou fenêtres et détails ornementaux qui rappellent la Smith House de Richard Meier, sur l'analyse de laquelle le projet s'appuie. Dans le même esprit, un autre projet, en collaboration avec James Park utilise les dessins préparatoires de l'agence de Mies van der Rohe pour la conception du palais de justice de Chicago pour programmer un algorithme qui propose des variations de palais de justice dans le style miesien[68].

---

[64] George Stiny and William J. Mitchell, 'The Palladian Grammar', Environment and Planning B, 5, 1 (1978), pp. 5-18.
George Stiny and William J. Mitchell, 'Counting Palladian Plans', Environment and Planning B, 5, 2 (1978), pp. 189-198.

[65] *As Wittkower (1952) observes, the distinguishing feature of Palladio's villas is "the systematization of the groundplan".* George Stiny and William J. Mitchell, 'The Palladian Grammar', Environment and Planning B, 5, 1 (1978), pp. 5-18.

[66] Richard Freedman et George L. Heersey, *Possible Palladian Villas (Plus a Few Instructively Impossible Ones)*, The MIT Press, 1992.

[67] James Park et Athanassios Economou, "The Dirksen variations: Towards a generative description of Mies's courthouse language", Conference: Real Time: Extending the Reach of Computation, Proceedings of the 33rd eCAADe ConferenceAt: Vienna University of Technology, Vienna, AustriaVolume: 1, 2015.

[68] Din, E., & Economou, A. (2007). Rewind–Pause–Forward: The Wall Variations of the Smith House. In Proceedings of the Sixth Conference of the International Society of Arts, Mathematics and Architecture-ISAMA (Vol. 7, pp. 135-144).

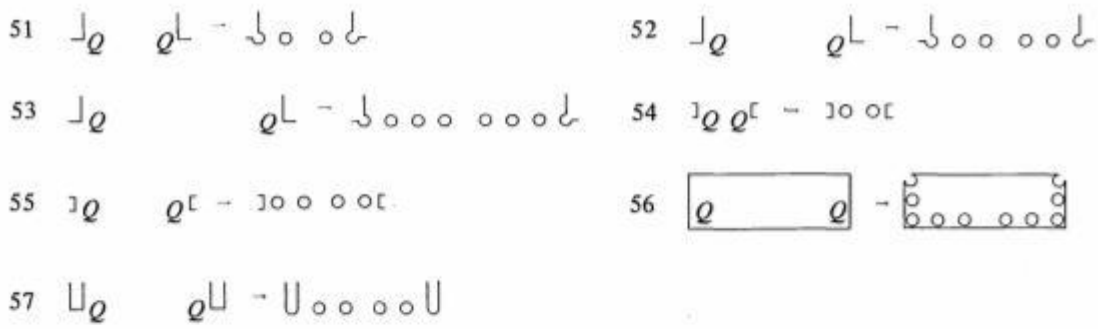


**Figure 12.** Rules for finishing porticos by the addition of columns.

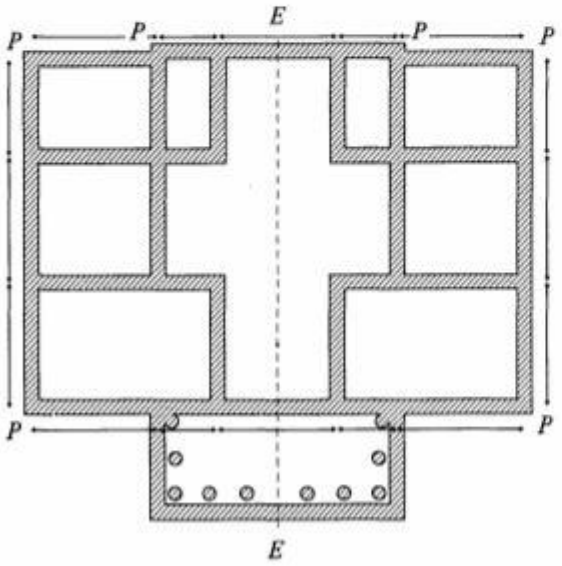



**Figure 13.** Addition of columns for the Villa Malcontenta.

*Figure 3. Exemples de règles et leur résultat dans le projet de grammaire palladienne de George Stiny et William Mitchell*

Ces travaux contribuent à explorer la formalisation des informations nécessaires à l'obtention de plans en sortie d'un programme sous forme de règles, mais pas seulement. À travers la formalisation de règles de conception telles qu'utilisées par Mies, Palladio, Meier et d'autres, il s'agit aussi de collectionner les connaissances de sens commun sur l'architecture. Le Spatial Data Management System, développé par l'Architecture Machine Group début 1978 est lui aussi une tentative de cartographier, stocker, relier des ensembles de données. Ces projets s'inscrivent donc dans une ambition plus vaste : accumuler des cas des études en un ensemble de plus en plus grand de règles architecturales. Un peu comme la base de données Cyc, mais appliqué à la conception architecturale pour s'y confronter au problème des connaissances de sens commun. L'idée est de décomposer en systèmes d'instructions similaires toutes les architectures, pour pouvoir automatiser sa conception. Il s'agit d'un travail de fourmi, mais qui peut se faire sur le principe d'une base de données générale que tout le monde alimente. Les travaux comme ceux de Stiny ou Economou peuvent être vus comme des

contributions possibles à cette base de données, bien qu'aucun projet de mise en commun de ces systèmes de règles n'ait véritablement existé. C'est aussi ce que la citation suivante de William Mitchell pointe - même s'il ne s'agit plus de traduire en systèmes de règles les travaux des grands architectes du passé, mais plutôt les bâtiments post-modernes contemporains de sa phrase.

*Nous aurons des Mac Decon's, des PoMo Huts, des Colonel Planners et des Techno Bells basés sur des logiciels, tous avec leurs ingrédients et recettes "secrets"*[69].

Le développement des systèmes experts dans les années 1980 encourage cette approche par la formation d'un catalogue architectural ultime, d'autant plus que ces systèmes posent justement la question de la représentation des connaissances et de leur manipulation par le biais de règles.

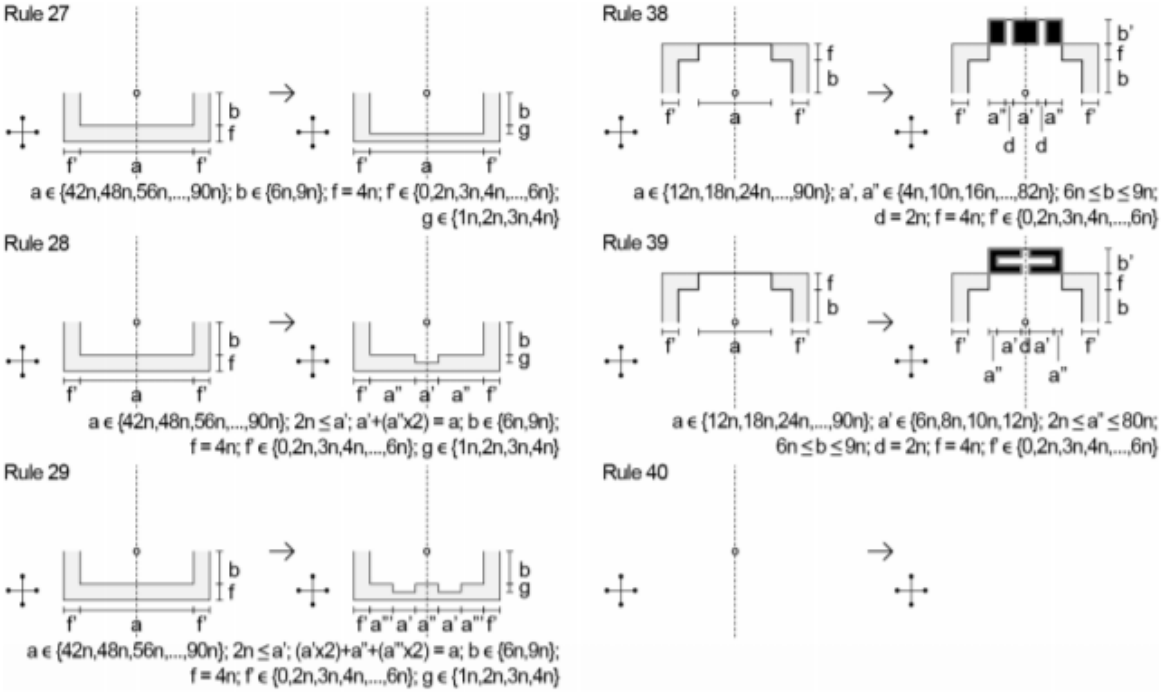



*Figure 4. Exemples de règles et leur résultat dans le projet de variations de palais de justices miesiens par Athanassios Economou et James Park*

Les travaux que nous venons de décrire qui se confrontent à la formalisation des connaissances architecturales et de leur computation sont néanmoins des programmes qui ne sont pas des systèmes experts. En complément de ces expérimentations qui se poursuivent dans la lignée des pionniers du champ, les systèmes experts se développent également au cours des années 1980. Les systèmes

[69] *"We will have software-based Mac Decon's, PoMo Huts, Colonel Planners, and Techno Bells, all with their 'secret' ingredients and recipes",* William J. Mitchell, "Reviewed Work(s): Possible Palladian Villas (Plus a Few Instructively Impossible Ones) by George Hersey and Richard Freedman: The Program" AA Files , Autumn 1993, No. 26 (Autumn 1993), pp. 97-99.

experts dédiés à l'architecture commencent à être mis au point en même temps que des systèmes experts voient le jour dans de nombreux autres domaines. Ce pic correspond à la mise à disposition d'outils de programmation de systèmes experts généralistes comme Hearsay III. Hearsay III est un outil de conception de systèmes experts[70]. Il inclut des inférences généralistes, qui peuvent être agencées pour constituer des bases de règles concernant des objets précis. Ces objets peuvent provenir de n'importe quel domaine. L'avantage offert par Hearsay III est justement sa non-spécialisation. L'outil rend donc possible l'expérimentation des systèmes experts dans le domaine architectural puisqu'il rend accessible leur construction. La plupart des systèmes experts conçus pour la conception architecturale le sont justement avec l'aide de Hearsay III. Le langage Prolog fournit néanmoins également un environnement de programmation utile, nous l'avons mentionné plus haut. Dans son livre *Knowledge-Based Design Systems*, paru en 1990, l'équipe de John Gero recense l'ensemble des systèmes experts développés par leur groupe de recherche au cours des années passées grâce à ce langage[71]. Le développement du champ computationnel continue donc de suivre celui de l'intelligence artificielle, puisqu'après un ralentissement qui voit l'activité se concentrer essentiellement autour des quelques pionniers déjà installés se produit un nouvel élan autour de la création de systèmes experts. Dans la deuxième moitié des années 1980, un très grand nombre d'ateliers sont organisés sur ces sujets, et des commissions sont créées pour financer et examiner le développement d'applications des systèmes experts à la construction[72]. En effet, l'attribution de financements publics pour des projets de ce genre reprend en même temps que le champ de l'intelligence artificielle vit son second pic de développement. Aux Etats-Unis par exemple, des systèmes experts pour l'architecture et la construction sont financés par la DARPA, par la National Science Foundation ou encore par la General Services Administration[73]. On observe également une croissance des financements privés. Des entreprises de construction, comme Steelcase, des agences d'architecture, comme Perkins+Will, des bureaux d'études comme Dar al Handasah font développer des systèmes experts qui puissent leur servir dans leur pratique quotidienne.

Néanmoins, peu de ces systèmes s'intéressent à des problèmes de conception de l'espace : la majeure partie des systèmes experts développés entre le milieu des années 1980 et le début des années 1990 l'est pour des applications dans le domaine de l'ingénierie. Le terme utilisé pour désigner ces systèmes est plutôt celui de knowledge based system, c'est-à-dire de système de connaissance, mais il s'agit bien de systèmes experts. En témoigne le recours à Hearsay III, et la notion de *shell* utilisée dans la

[70] Erman, L., London, P., & Fickas, S. (1981, August). The design and an example use of hearsay-ii. In Proceedings of IJCAI81 (pp. 409-415).
[71] R. D. Coyne, M. A. Rosenman, A. D. Radford, M. Balachandran, and J. S. Gero, *Knowledge-Based Design Systems*, Addison Wesley, Reading, Mass., 1990.
[72] Voir par exemple Report From the 1984 Workshop on Advanced Technology for Building Design and Engineering, publié par le Building Research Bord du National Research Council.
[73] Report From the 1984 Workshop on Advanced Technology for Building Design and Engineering, publié par le Building Research Bord du National Research Council.

description des programmes développés, qui fait référence au moteur d'inférences qui entre dans la structuration d'un système expert. Ces programmes sont plus dédiés à la construction, au management, et à l'évaluation financière qu'à la conception architecturale. En l'espace de quelques années sont mis au point des systèmes experts pour aider au dessin de détails architecturaux, à l'évaluation des performances structurelles, des coûts financiers, des risques d'incendie et de la consommation d'énergie, au respect des réglementation ou encore à la conception de systèmes de chauffage, ventilation et climatisation. Les annales de 1991 de la conférence *Artificial Intelligence in Design* donnent un aperçu de la part des processus d'ingénierie dans les recherches (Fig. 5)[74]. Ce tournant vers les enjeux de l'industrie de la construction plutôt que vers ceux de la conception architecturale reflètent le tournant industriel pris par les systèmes experts de manière générale à cette période. Le pôle australien, dont nous avons souligné qu'il évolue à ce moment-là à la frontière entre conception spatiale algorithmique et applications de ces mêmes algorithmes à des problématiques relevant plutôt de l'ingénierie, se préoccupe tout particulièrement du développement des systèmes experts. C'est l'objet de la thèse de Rivka Oxman, *Expert System for Generation and Evaluation in Architectural Design*, qu'elle soutient au Technion en Israël mais qui se base largement sur les travaux menés lors de son séjour en Australie[75]. Les divers systèmes experts développés par le pôle australien prennent par exemple en charge le dessin de détails de construction, comme EAVE, développé par Anthony Radford, ou aident à l'intégration de systèmes passifs de gestion de l'énergie, comme SOLAREXPERT, conçu par Richard Coyne[76]. D'autres systèmes experts visent à assister dans la conception d'hôpitaux ou de centres commerciaux, voir même ont des ambitions plus généralistes de conception architecturale de tout projet - aucun de ces derniers n'est cependant conçu dans son intégralité[77].

[74] J. S. Gero (ed.), *Artificial Intelligence in Design '91*, Elsevier, 1991.
[75] Rivka Oxman, Expert System for Generation and Evaluation in Architectural Design, Thèse de doctorat, Technion – Israel Institute of Technology, 1988
[76] Mitchell, J. R. and A. Radford. "EAVE, a Generative Expert System for Detailing." *Environment and Planning B: Planning and Design* 14 (1987): 281 - 292. Rosenman, Michael A., John S. Gero and Richard Coyne. "SOLAREXPERT : A Prototype Expert System for Passive Solar Energy Design in Housing." (1987).
[77] Karandikar, S. S.. "Expert system applications in architecture." (1989).

## Contents



v







*Figure 5. Sommaire annales de 1991 de la conférence Artificial Intelligence in Design 1991*

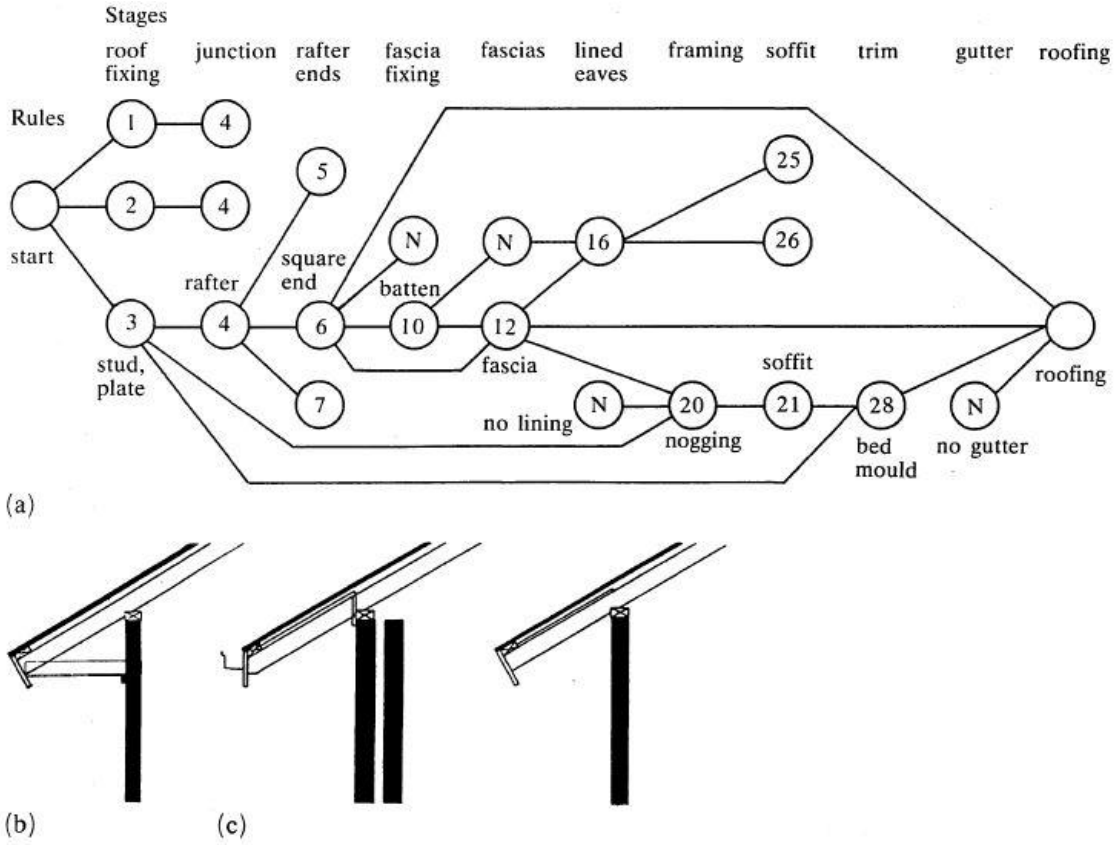



**Figure 6.** (a) Links between EAVE rules in the generation of design (b). For a rule to fire, the earlier (lower number) rules to which it is linked by a heavy line must also be fired. (c) Two other designs.

*Figure 6. Exemple de production du système expert EAVE, développé par A. Radford et J.R. Mitchell*

Entre 1986 et 1993 s'observe donc un pic de travaux dédiés aux systèmes experts, qui viennent compléter ceux déjà en cours autour de la formulation de règles de compositions. Un reproche fait au livre publié à ce moment-là par l'équipe de John Gero est que bien que le titre en soit *Knowledge-Based Design Systems*, il ne traite que de systèmes experts appliqués à des problèmes relevant essentiellement de l'ingénierie[78]. La critique, écrite par Amit Mukerjee et publiée dans AI Magazine, souligne qu'il existe de nombreuses autres approches de la conception algorithmique en architecture, que le livre de Gero et ses collègues laisse pourtant de côté. Rappelant l'existence des grammaires de formes et de plusieurs projets parmi ceux dont nous avons parlé plus haut, la critique regrette également que le livre ne prenne pas mieux en compte les nombreuses recherches de théorie du design dédiées à la nature des connaissances. Dans le cas du livre concerné, le titre, *Knowledge-Based Design Systems,* peut effectivement induire en erreur puisque cette expression peut évoquer un plus grand ensemble de systèmes que seulement les systèmes experts. L'expression *intelligent CAD* qui est alors aussi de plus en plus populaire ratisse quant à elle encore plus large en

[78] Mukerjee, A. (1991). Review of Knowledge-Based Design Systems. AI Magazine, 12(3), 122.

intégrant les logiciels de dessin assisté par ordinateur dans ce groupe de travaux[79]. La remarque faite par Mukerjee tout comme cet amalgame entre de nombreux systèmes informatiques différents soulignent cependant à quel point les travaux du champ computationnel sont encore vus comme des recherches en lien avec la nature des connaissances. Étant donné la place qu'occupe l'expertise dans leur développement, les systèmes experts appliqués à l'architecture déclenchent évidemment des questions très proches de celle que le champ de la théorie du design, notamment les membres de l'HfG Ulm et de la Design Research Society, pose alors déjà. Les questionnements se rapprochent d'autant plus que le champ de l'intelligence artificielle pivote au travers de la mise au point des systèmes experts d'un accent sur les règles à un accent sur les connaissances. Or leur nature exacte dans le champ du design est déjà un enjeu bien présent au sein de la Design Research Society.

D'un côté comme de l'autre, la notion de décision au cours du processus de design et la formalisation de ces décisions joue un rôle clé. Dans un texte de 1989, *A new agenda for computer-aided design*, William Mitchell fait justement le lien entre les deux, décrivant les inférences et les prédicats sur lesquels les systèmes experts s'appuient, expliquant le role de ces systèmes et des grammaires de forme dans la formalisation de règles de design et discutant ensuite les méthodes d'enseignement les plus adéquates pour ces nouveaux outils[80]. Il revient également sur la computation de formes, discutant l'ambiguïté de l'interprétation des formes manipulées qui existe dans la méthode. Mitchell esquisse ici l'idée que l'application du savoir-faire architectural a lieu dans une étape séparée d'interprétation, plutôt que d'être complètement intégrée à l'algorithme[81]. Le texte de Mitchell sert d'introduction à un ouvrage collectif, The Electronic Design Studio, dont nombre des articles sont des réflexions sur la structure des connaissances nécessaires à la conception architecturale et sur la manière de transmettre ces connaissances[82]. Le livre fournit par ailleurs un bon aperçu de l'état des recherches dans la partie de la théorie du design qui s'intéresse aux systèmes computationnels. La liste des contributeurs au livre contient les noms de la plupart des praticiens du champ à ce moment-là. Leurs réflexions dans cet ouvrage comme ailleurs s'ancrent dans le travail préexistant de Nigel Cross et de la Design Research Society. L'association occupe une place importante dans leurs recherches : Per Galle et Rivka Oxman en sont notamment membres. Per Galle et Rivka Oxman, mais aussi Richard Coyne, se tournent d'ailleurs au bout d'une dizaine années de recherche vers le champ de la théorie du design, délaissant le champ computationnel comme Cross avant eux.

---

[79] Voir par exemple Paul J.W. ten Hagen, Tetsuo Tomiyama (eds.), *Intelligent CAD Systems I: Theoretical and Methodological Aspects*, Springer, 1987.
[80] Malcolm McCullough, William J. Mitchell, Patrick Purcell (eds.), *The Electronic Design Studio*, The MIT Press, 1990.
[81] Voir Chapitre IV.
[82] Malcolm McCullough, William J. Mitchell, Patrick Purcell (eds.), *The Electronic Design Studio*, The MIT Press, 1990.

Néanmoins les systèmes experts architecturaux ne sont pas exempts des problèmes qui touchent ceux appliqués dans d'autres domaines. C'est ce que soulignent George Stiny et William Mitchell dans leurs textes respectifs pour l'ouvrage *Electronic Design Studio*. Georges Stiny y souligne le problème que pose le grand nombre de connaissances de sens commun sur lesquelles la conception architecturale s'appuie[83]. William Mitchell s'attache également à commenter ce problème, le disséquant un peu plus en détail. Il donne trois raisons aux problèmes rencontrés par les chercheurs qui souhaitent systématiser les règles de conception pour les intégrer à un programme informatique. Il pointe d'abord la multidisciplinarité des connaissances nécessaires en architecture et le grand nombre d'acteurs que demande la réalisation d'un projet[84]. Mitchell rappelle ensuite le problème du cadrage rencontré par les chercheurs en intelligence artificielle et indique que cela s'applique particulièrement à l'architecture, pour laquelle il est difficile de déterminer de manière systématique ce qui est important ou pas dans un projet donné. Enfin, il ajoute à ce second point l'idée que la conception architecturale se base sur un discours non monotone : l'avancement de la réflexion peut remettre en cause les conclusions précédentes n'importe quand. Il est donc difficile de construire un raisonnement logique du début à la fin qui soit fidèle au processus de conception dans son entièreté et qui puisse l'automatiser[85]. Comme pour les autres systèmes experts, la quantité d'informations à saisir pour que le système puisse fonctionner est massive, et la mise à jour des systèmes experts est constante puisque chaque nouveau projet requiert des ajustements. Même pour des opérations d'agencement du plan relativement simple, le temps et la place prise par l'explicitation des règles sont très vite conséquents. La computation des formes, approche qu'on retrouve alors chez de nombreux praticiens, implique de décrire toutes les formes et opérations possibles. Dans l'article de George Stiny *What Designers Do That Computers Should*, la simple explications des bases du raisonnement à partir d'exemples avec seulement quelques formes prends déjà presque tout l'article[86]. Chez Per Galle ou chez William Mitchell, les instructions de programmation pour l'agencement de différents espaces les uns par rapport aux autres en plans sont également très longues. Les méthodes d'évaluation systématiques visées ne peuvent être appliquées à des plans comprenant un trop grand nombre d'espaces : huit ou dix tout au plus. L'une des figures montrées par John Gero et Richard Coyne dans un article de 1985 montre à quel point l'espace des possibilités grandit vite, et à quel point l'écriture des règles d'inférence est fastidieuse (Fig. 7)[87].

---

[83] Malcolm McCullough, William J. Mitchell, Patrick Purcell (eds.), *The Electronic Design Studio*, The MIT Press, 1990.
[84] David B.T. (1987) Multi-Expert Systems for CAD. In: ten Hagen P.J.W., Tomiyama T. (eds) Intelligent CAD Systems I. EurographicSeminars (Tutorials and Perspectives in Computer Graphics). Springer, Berlin, Heidelberg.
[85] Malcolm McCullough, William J. Mitchell, Patrick Purcell (eds.), *The Electronic Design Studio*, The MIT Press, 1990.
[86] Malcolm McCullough, William J. Mitchell, Patrick Purcell (eds.), *The Electronic Design Studio*, The MIT Press, 1990.
[87] Coyne RD, Gero JS. Design Knowledge and Sequential Plans. Environment and Planning B: Planning and Design. 1985;12(4):401-418.

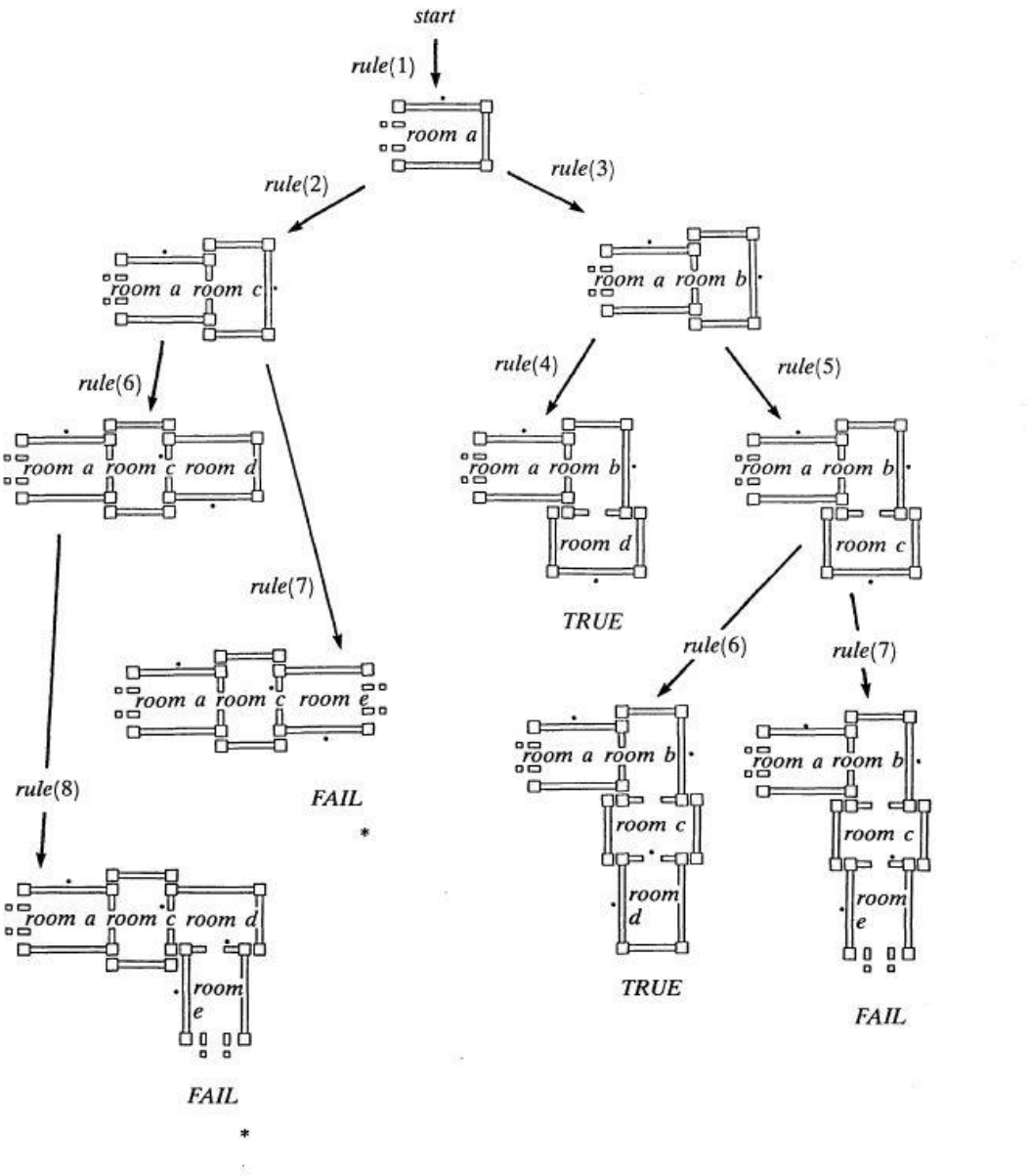



**Figure 5.** The search graph for satisfying the goal: located(room_*b*) and located(room_*d*).

*Figure 7. Logique d'application des règles étape par étape pour la création d'un plan*

Le problème des cathédrales de règles se pose donc particulièrement pour les systèmes experts appliqués à l'architecture. De plus, il nous semble que certaines des raisons avancées par Berg au sujet des systèmes experts médicaux fournissent un parallèle intéressant avec ceux en usage en architecture à la même période. Certes, les fragilités techniques des systèmes experts sont encore plus flagrantes quand il s'agit de les appliquer à l'architecture. Mais il est possible que les praticiens aient eu la même réaction que leurs collègues médecins face aux systèmes experts, freinant leur adoption. La méfiance

des praticiens à l'idée que leur savoir-faire puisse être intégralement implémenté dans une machine, mais aussi à l'idée d'un bouleversement de la hiérarchie des acteurs au cours de la conception a pu jouer dans la trajectoire des systèmes experts. Simples limitations techniques ou rejet des praticiens, toujours est-il que le catalogue architectural ultime s'avère un objectif difficile à atteindre, et les systèmes experts tombent rapidement en désuétude en architecture comme ailleurs. Mêmes faiblesses, mêmes échecs, même chronologie que pour l'hiver de l'intelligence artificielle : le champ computationnel vit alors bien un hiver procédural. Un hiver procédural plutôt qu'un hiver complet de l'architecture computationnelle. En effet, si le premier hiver de l'intelligence correspond à un hiver similaire dans le champ computationnel, le second s'y répercute différemment. Les systèmes experts sont abandonnés, et peu travaillent à de nouveaux algorithmes pour la conception spatiale. Mais d'autres choses se développent. La déchéance des systèmes experts va en effet laisser de la place à des sujets différents à partir du milieu des années 1990 : fabrication, écriture, modélisation.

### *3.1.3 Une ouverture vers de nouvelles pratiques*

L'état du champ de l'intelligence artificielle et des recherches en architecture computationnelle qui en sont le satellite favorisent une émancipation de l'architecture computationnelle. La conséquence principale de cette émancipation est l'apparition de nouveaux sujets de recherche et de nouvelles pratiques. Plutôt que de se pencher sur les problèmes de l'intelligence artificielle, mal aimés et mal financés, une partie des nouveaux venus dans le champ choisit de se tourner vers la fabrication numérique. Les équipements informatiques de bureau ne sont en effet pas les seuls à connaître un fort développement au cours des années 1980 et 1990. De nombreuses industries voient aussi apparaître des machines à commande numérique. Sous cette appellation sont regroupées diverses machines de fabrication plus ou moins spécialisées. Des machines fabriquant automatiquement des trombones à la chaîne, mais aussi des machines usinant des matériaux divers en 2.5 ou 3 dimensions. Les diverses industries qui exploitent le bois recourent notamment à des fraiseuses numériques : des fraiseuses montées sur un portique et associées à des moteurs qui déplacent l'outil automatiquement et permettent donc de découper le bois en suivant un dessin prédéterminé à l'ordinateur. En modifiant le type de fraiseuse, la même machine peut cependant néanmoins très bien servir à découper du métal ou du plastique. Les portiques le long desquels l'outil est fixé permettent dans ce cas de le déplacer selon trois translations. Mais certains outils disposent aussi d'axes de rotation, qui ajoutent des axes de déplacement possibles. En fonction du nombre d'axes dont la machine à commande numérique dispose, on parle de machine à trois, quatre, cinq axes ou plus. La diversité des mouvements de l'outil augmente avec le nombre d'axes de la machine, comme la complexité des formes usinables. Les robots six-axes, des bras articulés souvent utilisés sur les chaînes de production automobiles pour l'assemblage, la soudure ou la peinture, sont particulièrement appréciés car très versatiles : l'outil fixé au bout du bras articulé peut être changé, et le robot changer de fonctionnalité. Encore peu accessible

en dehors des usines au moment où les praticiens du champ computationnel commencent à s'intéresser à la fabrication, le six-axes deviendra cependant par la suite un de ses outils privilégiés, chaque laboratoire étant équipé d'une ou plusieurs de ces machines.

*Figure 8. Bloom House, Greg Lynn FORM*

Versatilité, complexité, guidage numérique : des qualités qui ne sont pas pour déplaire aux architectes les plus friands d'expérimentation. Parmi eux, Greg Lynn, un architecte américain né en 1964, dont

l'activité a partir des années 1990 tourne autour de l'exploration des technologies numériques. D'abord concentré sur la modélisation, Greg Lynn se pique assez vite d'intérêt pour le détournement de méthodes de fabrication utilisées ailleurs, en aéronautique, en automobile ou en construction navale. Il les emploie pour réaliser des installations et des meubles divers, mais aussi ponctuellement dans certains bâtiments. Plusieurs de ses projets utilisent par exemple de vieux jouets en plastique, redécoupés grâce à une fraiseuse numérique et empilés pour enfin être soudés entre eux. Les étranges formes créées par ces empilements de matériaux rappellent le jeu de contraste mis en œuvre par Greg Lynn dans le projet de la Bloom House. Le bâtiment, dont l'extérieur conventionnel ne laisse pas deviner les particularités intérieures, à l'exception d'une étrange lampe trouant la baie vitrée du salon, offre pourtant dans chaque pièce des parois courbes irrégulières, qui tranchent sur le reste des choix formels. Chacune de ces excroissances est réalisée grâce au recours à des fraiseuses numériques. L'un des premiers projets de Greg Lynn, l'Embryological House, est emblématique des recherches menées autour de la fabrication numérique dans le champ computationnel, à la fin des années 1990. Greg Lynn se penche sur l'utilisation de fraiseuses numériques pour la réalisation d'une maison préfabriquée. Il souhaite réinterpréter les sous-produits de la construction alors disponibles aux Etats-Unis, et permettre une nouvelle approche formelle. L'Embryological House est un objet ovoïde, que le processus de fabrication prévoit de trancher en tranches successives. Ces tranches fournissent le contour de cadres métalliques découpés puis assemblés pour composer la forme modélisée. N'importe quel ovoïde peut être décomposé et fabriqué de cette manière, il suffit de changer la trajectoire exacte de la fraiseuse pour obtenir une série de cadre différente, qui obéit pourtant exactement au même processus de fabrication. Plus encore que l'exploration des possibilités offertes par le fraisage numérique, c'est cette malléabilité qu'offre le processus de conception qu'explore Greg Lynn avec le projet de l'Embryological House[88].

[88] Lawrence Bird et Guillaume LaBelle, "Re-Animating Greg Lynn's Embryological House: A Case Study in Digital Design Preservation", Leonardo 43(3), 2010.

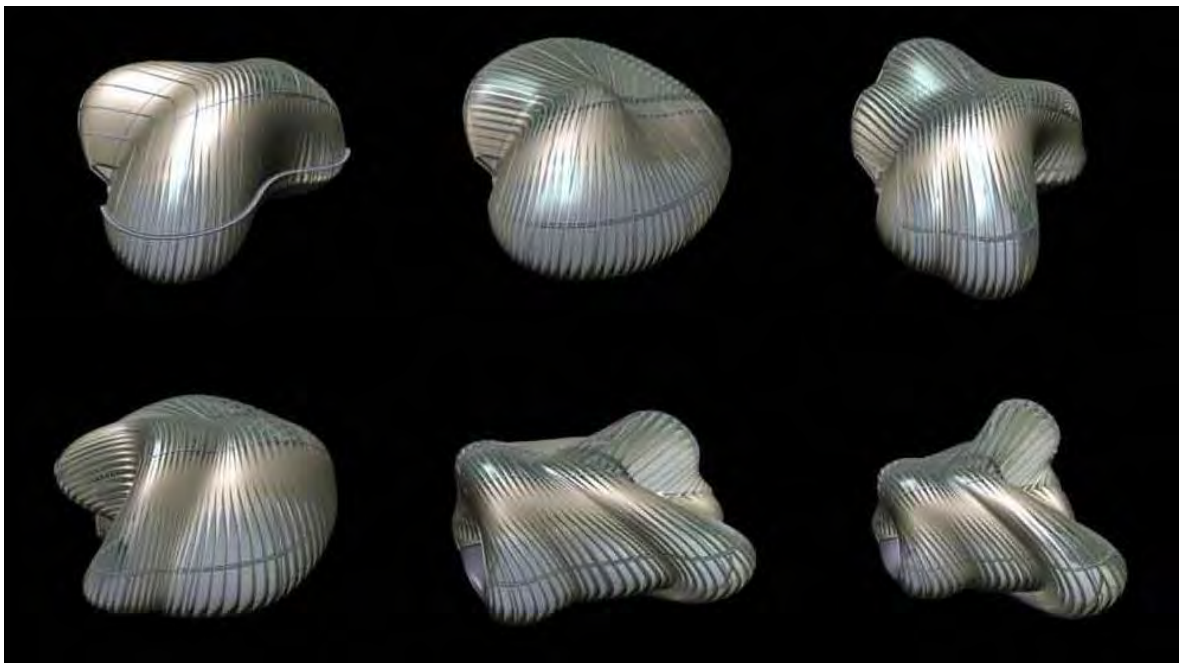


*Figure 9. Embryological House, Greg Lynn FORM*

C'est cette même idée de pièces toujours différentes qu'explorent Bernard Cache et Patrick Beaucé au sein de leur agence Objectile. Bernard Cache, né en 1958, est diplômé de l'EPFL, à Lausanne, son associé Patrick Beaucé, architecte d'intérieur, de l'école des Beaux-Arts de Valenciennes. Objectile, leur atelier de fabrication, créé en 2002, fait suite à plusieurs années d'expérimentation philosophique et numérique. Objectile n'est en effet pas seulement le nom de l'agence, mais aussi celle que les deux architectes utilisent pour désigner les objets qu'ils fabriquent. Bien que chacun des objets au sein d'une série obéisse à la même logique de fabrication, ils sont tous différents les uns des autres. L'agence produit des séries d'objets en bois, sculptés à l'aide d'une fraiseuse numérique. Ces objets sont l'occasion d'explorer ce qu'ils désignent sous le nom de production non-standard : la production d'objets toujours différents, à rebours de la standardisation de la manufacture à l'œuvre depuis longtemps dans l'industrie[89]. En effet, Bernard Cache et Patrick Beaucé, en plus de produire des séries successives d'objectiles, écrivent largement sur les idées qui sous-tendent leur processus de travail. Cette approche des outils de fabrication numérique et de leur potentiel devient très populaire dans le champ computationnel, en particulier grâce au livre *Terre Meuble*, écrit par Bernard Cache, qui condense plusieurs années de réflexion autour de la fabrication non-standard[90]. Ces réflexions ont démarré au milieu des années 1980, alors que Bernard Cache étudie sous la direction du philosophe Gilles Deleuze : c'est avec lui que l'architecte développe la notion d'objectile. Véritable théorie des pratiques de conception algorithmique et de fabrication numérique, la production non-standard s'ancre

[89] Vers un Mode de Production Nonstandard. In: Objectile Patrick Beaucé + Bernard Cache. Consequence Book Series on Fresh Architecture, vol 6. Springer, Vienna, 2007.
[90] Bernard Cache, *Terre Meuble*, Editions HYX, 1990.

largement dans le travail de Gilles Deleuze, qui développe ses propres idées autour de l'objectile dans *Le Pli*, qui sort en 1988[91]. L'ouvrage de Bernard Cache, *Terre Meuble*, sort seulement en 1993, mais il se trouve que c'est aussi l'années de traduction du *Pli* en anglais. La popularité rencontrée par les deux ouvrages au sein du champ computationnel (et un peu plus largement en architecture) est immense, comme en témoigne le texte d'Andrew Ballantyne *Deleuze and Guattari for architects*, plaidoyer pour la lecture non seulement du Pli mais de toute l'oeuvre du philosophe français[92]. Les notions de variation permanente des objets tout comme les métaphores mathématiques de Gilles Deleuze paraissent en effet susciter une grande fascination et un écho très fort chez les architectes, qui y retrouvent peut-être quelque chose de l'effort de formalisation situé fourni à chaque nouveau projet. Cette fascination reste par la suite, Deleuze étant depuis une des références principales utilisées par les architectes du champ computationnel, au point que Mario Carpo fait de la date de traduction du Pli le point de départ de son récit du tournant numérique en architecture[93]. Ainsi, l'hiver de l'intelligence artificielle débouche sur une prise d'importance de la fabrication numérique, et sur l'amorce d'une nouvelle réflexion théorique propre au champ, un passage obligatoire pour une discipline en cours de construction.

[91] Gilles Deleuze, *Le Pli. Leibniz et le Baroque*, Editions de Miniuit, 1988.
[92] Andrew Ballantyne, *Deleuze and Guattari for Architects*, Routledge, 2007.
[93] Mario Carpo, "Twenty Years of Digital Design", in Mario Carpo (ed.), *1993-2012, The Digital Turn in Architecture*, Wiley & Sons, 2012.

*Figure 10. Panneau de bois, Objectile*

## 3.2 L'avènement des modeleurs

### *3.2.1 Brève histoire de l'infographie*

Fabrication non-standard et réflexions écrites ne sont néanmoins pas les seules choses à émerger dans le vide laissé par les hivers successifs de l'intelligence artificielle. L'émancipation du champ computationnel en architecture de son parent celui de l'intelligence artificielle se fait à travers un autre ancrage, dans le champ de l'animation et de l'infographie[94] cette fois. Nous avons vu que la première phase de développement du champ computationnel est marquée par des incursions ponctuelles en dehors du monde universitaire. Parmi les plus durables de ces interactions avec le monde professionnel se trouvent celles qui ont trait à l'infographie, c'est-à-dire au dessin sur écran. C'est le cas par exemple d'Autographics, logiciel de dessin assisté par ordinateur développé par Paul Coates et les Frazer à partir de 1978. L'Architecture Machine Group collabore quant à lui avec la DEC et avec les laboratoires Bell, dans la continuité de ses travaux sur les interfaces numériques. Les points identifiés a posteriori par Nicholas Negroponte pour améliorer le fonctionnement de l'interface d'URBAN 5 sont d'ailleurs les mêmes principes que ceux qui seront ensuite utilisés dans l'industrie pour les logiciels de dessin par ordinateur[95]. La différence essentielle entre les logiciels dont nous allons traiter dans cette partie et les outils employés précédemment est en effet qu'il s'agit de dessiner et non de saisir des lignes de code. Or Negroponte pointe les insuffisances de l'interface d'URBAN 5 notamment en indiquant qu'il faut aller beaucoup plus loin dans la manipulation de représentations logiques spatiales, qui doivent remplacer l'écriture de programmes informatiques pour les architectes. C'est justement cette idée de manipulation directe de données visuelles qui est à l'origine du développement de logiciels de visualisation interactive - de dessin assisté par ordinateur. Avec la mise en évidence de ce besoin, l'Architecture Machine Group crée les prémisses d'une filiation aussi bien avec l'intelligence artificielle que avec l'animation pour le champ computationnel. En effet, les recherches amorcées par l'Architecture Machine Group sur le dessin assisté par ordinateur se font en collaboration avec des chercheurs comme Steven Coons ou Ivan Sutherland : des pionniers de l'infographie dont le travail va déboucher à terme sur des logiciels clés pour le champ de l'animation.

En effet, l'interaction avec l'ordinateur est une composante clé de la conception computationnelle, qui a longtemps occupé l'Architecture Machine Group ou les chercheurs de la AA. Mais en parallèle de leurs travaux, les chercheurs en sciences de l'informatique travaillent également au développement des

[94] Au sens de graphismes informatiques, pas au sens des diagrammes informatifs souvent désignés par ce terme aujourd'hui.
[95] Eliza Pertigkiozoglou, "1973. Nicholas Negroponte and Architecture Machine Group MIT", 17 Février 2017, https://eliza-pert.medium.com/1973-a1b835e87d1c, consulté le 30 Novembre 2021.

capacités d'affichage des informations traitées par l'ordinateur, ainsi que d'interfaces pour interagir avec les programmes. Les écrans eux-mêmes existent depuis plusieurs décennies au moment où notre récit commence - le développement de l'infographie s'ancre bien sûr dans celui de l'électronique et de la télévision qui a eu lieu au cours de la première moitié du XXe siècle. Dès les années 1950, des chercheurs tentent d'interagir directement avec ces écrans, et notamment de dessiner dessus. Vers 1955, Douglas T. Ross, un ingénieur en électronique du MIT, travaille sur les ordinateurs Whirlwind, utilisés pour le SAGE (Semi-Automatic Ground Environment), le système de surveillance et de contrôle des missiles américains pendant la guerre froide. Traiter les affichages d'information sur écran fait partie de ses tâches, et il développe à des fins d'expérimentation un petit programme qui détecte les mouvements de son doigt, dont il se sert pour écrire son nom directement sur l'écran. C'est d'ailleurs Ross qui a ensuite forgé le terme de *computer-aided design* (CAD), en français *conception assistée par ordinateur* (CAO), dont provient la variante *computer-aided drawing*, ou *dessin assisté par ordinateur* (DAO). Le programme de Ross est une des toutes premières interactions de dessin sur écran documentées. Mais plus encore que Ross, c'est Ivan Sutherland qui est crédité d'une influence majeure sur le champ du dessin assisté par ordinateur. En 1962, Ivan Sutherland est doctorant au MIT, sous la supervision de Claude Shannon et Marvin Minsky. Sutherland fait cette année-là la démonstration de son programme Sketchpad, qui permet d'interagir avec des formes sur l'écran par le biais d'un crayon lumineux. Sketchpad offre des fonctionnalités absolument remarquables pour l'époque. Il permet de dessiner des formes précises - rectangles, arcs, lignes - quelque soit la précision du trait de crayon, en indiquant des contraintes géométriques en amont du dessin tracé. Il permet aussi de combiner des éléments basiques comme des lignes en formes plus complexes. Les géométries manipulées peuvent être copiées, déplacées, pivotées ou mises à l'échelle.  Avec Sketchpad, Ivan Sutherland jette les bases de ce que peuvent être une interface utilisateur et un logiciel de dessin paramétrique. Pour ces raisons, Sketchpad est considéré comme un des programmes les plus influents de l'histoire de l'informatique.

Sutherland ne s'arrête cependant pas là. En 1968, il fonde avec David C. Evans, le directeur du département d'informatique de l'Université d'Utah, l'entreprise Evans & Sutherland. Les activités de l'entreprise sont doubles : fabriquer de l'équipement informatique pour les universités ou Evans et Sutherland interviennent en tant qu'enseignants-chercheurs, et fournir des conseils aux entreprises qui cherchent à se doter de systèmes informatisés. En effet, la commercialisation de terminaux graphiques pour les ordinateurs commence à intéresser d'autres entreprises qui fabriquent déjà du matériel informatique, comme IBM, General Electric ou Lockheed. Les années 1960 sont néanmoins le moment d'un développement bien plus large de l'infographie, partout dans le monde. Nombres d'algorithmes qui permettent d'animer des objets et de produire des vidéos sont également mis au point. Ces vidéos sont faites à partir de l'impression sur une feuille de papier de caractères ou directement affichées sur écran. L'animation de l'institut Royal de Technologie suédois, en 1960, est

réalisée sur un simple oscilloscope et montre par des lignes blanches en mouvement la trajectoire d'une voiture sur l'autoroute en vue subjective. Boeing propose un peu plus tard le premier personnage animé représenté par un groupement de lignes en mouvement. Les laboratoires Bell se servent à partir de 1963 de la même technique pour produire de petites animations de phénomènes physiques, intégrées à des vidéos pour le grand public. Le chat du mathématicien Nikolai Konstantinov, dont l'animation en 1968 permet de tester le modèle mathématique de déplacement du chat proposé par ses collègues, est un des premiers personnages animés. Enfin, les années 1960 sont aussi le moment des tout premiers jeux vidéos, Pong et Spacewar! en tête - des jeux extrêmement rudimentaires mais très importants car ils ouvrent des possibilités et des attentes considérables sur le plan de la visualisation informatique et de l'interaction avec un écran.

La description informatique d'objets géométriques de plus en plus complexe est rendue possible par les travaux notamment de Pierre Bézier, directeur des méthodes mécaniques chez Renault. Celui-ci est à la recherche d'une méthode pour parvenir à déduire du travail du designer la représentation mathématiques des courbes et non l'inverse. Les designers de chez Renault pourraient ainsi travailler librement sur la forme des automobiles, sans que trop de connaissances mathématiques soient exigées d'eux. Les courbes développées par Pierre Bézier ont en effet l'avantage de pouvoir être modifiées très facilement, simplement en déplaçant leurs points de contrôle. La position de ces points paramètre la définition mathématique des courbes, et ils peuvent être affichés et déplacés par les designers. Ceci est rendu possible par la nature mathématique des courbes de Bézier, des courbes polynomiales dont la construction mathématique rend la déformation très intuitive. Si on déplace un point de contrôle d'une courbe de bézier, elle se déforme en suivant le déplacement, la ou d'autres familles de courbes ont des relations beaucoup plus complexes entre points de contrôle et déformation, qui rendent leur transformation beaucoup plus difficile à visualiser. Pierre Bézier brevète en 1966 Unisurf, logiciel de modélisation et de contrôle de machines à commande numérique, basé sur les courbes de Bézier, que Renault met en usage à partir de 1968 dans l'entreprise. Paul de Casteljau fait le même travail presque en même temps chez Citroën, développant lui aussi les courbes aujourd'hui dites de Bézier. Son travail logiciel est cependant perdu en 1974, après le rachat de Citroën par Peugeot qui lui préfère Unisurf, devenu entre temps un logiciel très prisé dans l'industrie automobile.

Pendant ce temps David Evans et Ivan Sutherland poursuivent leurs propres recherches. Si ils ont fondé leur entreprise dans l'Utah, c'est parce qu'ils animent à l'université un groupe de travail sur l'animation. La liste des étudiants du groupe regroupe des noms parmi les plus prestigieux de l'histoire de l'infographie : Edwin Catmull, James H. Clark, John Warnock, Alvy Ray Smith, Martin

Newell, Jim Blinn, Bui Tuong Phong, Henri Gouraud ou encore Alan Kay[96]. Edwin Catmull est à l'origine de ce qu'on appelle le texture mapping, ou texturage : l'application de pixels de couleurs à un objet 3D. Ces pixels peuvent provenir de la photographie d'un matériau, et ainsi donner l'impression que l'objet 3D est fait de ce même matériau. Ceci permet de créer des animations beaucoup plus réalistes que lorsque les objets conservent l'aspect par défaut du modeleur dans lequel ils ont été formés. C'est justement en cherchant un moyen de rendre ses animations plus réalistes qu'Edwin Catmull met au point cette méthode. En collaboration avec James Clark, Edwin Catmull développe aussi plus tard l'algorithme dit de Catmull-Clark, qui permet de subdiviser des maillages pour mieux contrôler la géométrie d'objets en trois dimensions. Jim Blinn, Bui Tuong Phong et Henri Gouraud sont les développeurs des méthodes principales de calcul et d'affichage des ombres, un élément également crucial pour l'animation. Le groupe de recherche d'Evans et Sutherland développe aussi ce qu'on appelle la détermination des surfaces cachées : des algorithmes qui permettent de déterminer, pour un modèle composé de plusieurs objets, lesquels sont devant les autres et les cachent. Au fil des années 1970 et 1980, le petit groupe se sépare en sous-groupes évoluant dans divers cadres : l'Université de l'Utah, le New York Institute of Technology computer graphics laboratory, la division de Lucasfilm Graphics Group. Ils y développent nombre des bases de la synthèse d'images. Parmi les membres du groupe, on trouve aussi les fondateurs de Pixar, d'Adobe, ou de Silicon Graphics Inc. (SGI). Le premier, fondé par Edwin Catmull et Alvy Ray Smith (et financé par Steve Jobs), est un célèbre studio d'animation. La seconde, fondée par John Warnock, est l'entreprise qui édite l'une des suites logicielles les plus utilisées du monde pour le travaux de graphisme et de dessin. La troisième, fondée par James H. Clark, est l'entreprise qui fournit la majeure partie de ceux qui en ont besoin au cours des années 1980 en systèmes informatiques.

Les progrès du matériel informatique se poursuivent aussi : l'augmentation de la mémoire et de la mémoire vive permettent peu à peu de mener des travaux de plus grande ampleur. Un nombre croissant d'entreprises se dotent par ailleurs de systèmes, souvent sur mesure, pour la gestion de leurs activités. Les courbes de Bézier, qui ne sont qu'une famille particulière de courbes dont les équations permettent la déformation si intuitive, sont généralisées d'abord au sein de la familles des courbes appelées B-Splines, puis au sein de la famille des NURBS, qui sont utilisées comme base pour modéliser des objets en trois dimensions très variés. David Rogers, auteur du livre An Introduction to NURBS, les décrit ainsi :

[96] Martin Newell est le créateur de la théière de l'Utah, un objet utilisé depuis sa modélisation en 1975 comme une référence graphique dans le domaine de la synthèse d'images. Alan Kay est le créateur de la programmation orientée objet. Pour les autres, le reste du paragraphe donne une idée de leurs apports.

*Avec les NURBS, un système de modélisation peut utiliser une seule représentation interne d'un large éventail de courbes et de surfaces, depuis les lignes droites et les plans plats jusqu'aux cercles et sphères précis, en passant par les surfaces complexes sculptées par morceaux*[97].

L'invention des bornes d'arcade fait du jeu vidéo un phénomène culturel, et le cinéma fait lui aussi de plus en plus fréquemment recours à la synthèse d'images. Les premières expérimentations se concentrent sur l'ajout d'effets sur des images filmées, comme le fait le National Film Board of Canada, qui s'impose comme un centre de recherche essentiel dans le domaine. La deuxième moitié des années 1970 voit sortir les premiers films qui ont recours ponctuellement à des images de synthèse : *Great* en 1975, *Futureworld* en 1976, et surtout Star Wars en 1977. Ces expérimentations se poursuivent dans les années 1980, au cours desquelles le clip de *Money for Nothing* des Dire Straits et l'apparition d'un chevalier de vitrail animé dans le film *Le secret de la pyramide* marquent les esprits. C'est le début d'un tournant dans l'industrie du cinéma, ou la synthèse d'image va occuper de plus en plus de place.

Ce tournant n'est pas permis seulement par la popularisation des images numériques auprès du grand public et par le développement d'algorithmes permettant de créer ces animations. Il est aussi le fait du développement d'une industrie logicielle entière consacrée à l'animation. Celle-ci s'appuie notamment sur le fait que les ordinateurs qui sont commercialisés au cours des années 1980 se rapprochent bien plus de ceux qu'on connaît aujourd'hui que des énormes machines de la décennie précédente. Beaucoup plus puissants, ils sont aussi plus accessibles financièrement, et plus simples d'utilisation grâce aux interfaces logicielles comme au clavier et à la souris, désormais bien installés. La décennie voit naître de nombreuses entreprises qui commercialisent des systèmes informatiques, mais aussi des entreprises qui commercialisent des logiciels de synthèse d'image. Parmi les plus importantes, on peut notamment citer :

- Autodesk Inc., fondée en 1982 par John Walker et douze autres personnes aux États-Unis ;
- Alias, fondée en 1983 par Stephen Bingham, Nigel McGrath, Susan McKenna et David Spring au Canada ;
- Wavefront, fondée en 1984 par Bill Kovacs, Larry Barels et Mark Sylvester aux États-Unis ;
- Sogitec Audiovisuel, un département de l'entreprise Sogitec, fondée en 1984 par Xavier Nicolas en France ;
- Softimage, fondée en 1986 par Daniel Langlois, un ancien du National Film Board of Canada;
- Side Effects Software, fondée en 1987 par Kim Davidson et Greg Hermanovic au Canada.

[97] "*With NURBS a modeling system can use a single internal representation of a wide range of curves and surfaces, from straight lines and flat planes to precises circles and spheres as well as intricate piecewise sculptured surface*", David Rogers, *An Introduction to NURBS with historical perspective*, Morgan Kaufmann Publishers, 2001.

La production de ces entreprises oscille initialement entre aide à la production d'animation pour le cinéma et développement de logiciels. Alias travaille par exemple à l'animation du monstre du film *The Abyss* de James Cameron et à l'animation des dinosaures de *Jurassic Park*, réalisé par Steven Spielberg. Alias produit par ailleurs le logiciel Alias-1, Softimage le logiciel Creative Environment, Side Effects Software le logiciel PRISMS. Alors que d'autres entreprises, comme SGI, se concentrent sur la commercialisation de matériel informatique, celles-ci se concentrent sur les logiciels. Le développement de leurs outils accompagne le passage de dispositifs informatiques sur-mesure pour les entreprises à des dispositifs clés-en-main beaucoup plus légers et accessibles. La structuration de l'industrie du logiciel au cours des années 1980, qui implique que les différents acteurs cessent de faire tout en même temps mais se concentrent sur la production d'un seul élément, se combine aux progrès techniques et à la chute des prix en un très fort développement du secteur.

Parmi ces entreprises, certaines commercialisent rapidement des logiciels de dessin et de visualisation qui vont trouver un usage notamment en architecture, pour la conception et la représentation des projets. Alias, qui développe le logiciel d'animation Maya, propose en complément plusieurs suites logicielles pour le design industriel et l'architecture, notamment AliasStudio. Surtout bien sûr, Autodesk, spécialisé dans le dessin assisté par ordinateur à destination de l'industrie de l'architecture et de la construction dès le départ *via* son logiciel AutoCAD. Or, Autodesk s'emploie très vite à augmenter ses parts de marché en rachetant des concurrents. Et ses concurrents de l'industrie de l'animation ne font pas exception, puisque Autodesk les rachète petit à petit tous ou presque[98]. En effet, en 1989, l'entreprise Thomson Digital Image absorbe Sogitec. En 1993, Thomson Digital Image est à son tour rachetée par Wavefront. En 1995, Silicon Graphics Inc. achète Wavefront et Alias, et les combine en Alias|Wavefront, qui devient une division de SGI. En 2004, Alias|Wavefront, redevenue Alias, est vendu par SGI à un fond d'investissement. Sous le nom d'Alias sont alors combinées les entreprises Sogitec, Wavefront et Alias, c'est-à-dire trois sur les six principales nées dans les années 1980. En 2006, Autodesk rachète Alias, et en 2008, Softimage, passée entre-temps entre les mains de Microsoft de 1994 à 1998, puis par Avid Technology. Autodesk absorbe donc le quatrième de ses cinq concurrents initiaux, après avoir obtenu le contrôle de l'une des suites logicielles les plus utilisées dans le monde de l'animation : Maya. Au fil des années, Autodesk établit donc une position dominante dans le monde de l'architecture, mais pas seulement, puisque l'entreprise établit aussi au sein de ses produits des passerelles entre animation et conception architecturale[99].

Le développement d'outils de modélisation pour l'industrie de l'architecture et de la construction n'a cependant pas uniquement lieu dans le champ de l'animation. Les produits logiciels utilisés pour la

[98] Seuls 3DS Max et Houdini continuent à être principaux concurrents de Maya aujourd'hui.
[99] Voir Chapitre VI.

modélisation dans le domaine de l'ingénierie ont une histoire à part entière. Il s'agit de logiciels employés pour la modélisation et l'évaluation de la majeure partie des objets manufacturés. En particulier, l'aéronautique et l'automobile sont à l'origine d'outils de modélisation des pièces de moteur, dont la conception revêt une grande importance. La chronologie de de développement de ces outils est similaire à celle de l'animation. Les années 1960 et 1970 sont des décennies de recherche et de développement. Les années 1980 voient le même tournant industriel, et l'apparition de stations de travail dédiées aux ingénieurs, c'est-à-dire de groupes d'équipements informatiques et de logiciels calibrés spécialement pour leurs activités. Les entreprises à l'origine de ces logiciels ne sont pas les mêmes que dans le champ de l'animation : Lockheed, un avionneur américain, commercialise en partenariat avec IBM le logiciel CADAM. Dassault Systèmes s'associe à Renault pour transformer Unisurf en un logiciel de modélisation aux capacités étendues, CATIA. Ford propose le logiciel de dessin assisté par ordinateur PDGS. À la fin des années 1980, la Parametric Technology Corporation commercialise le logiciel Pro/ENGINEER, considéré par Daniel Davis comme un des tous premiers logiciels paramétriques[100]. Dans les années 1990, Dassault étend son offre par le biais de SolidWorks, et Autodesk prend de l'importance sur ce marché avec Inventor. David Weisberg propose dans son texte The Engineering Design Revolution un récit détaillé des recherches et du développement industriel de cette branche de la conception assistée par ordinateur[101]. Une fois installés, les grands acteurs de ce marché s'intéressent de plus près au transfert des logiciels développés de la manufacture à proprement parler vers l'industrie de l'architecture et de la construction. Une partie des outils de modélisation en usage dans cette industrie est donc issue du domaine de l'ingénierie mécanique. Le développement de l'infographie et des modeleurs 3D a donc deux origines, l'animation et la manufacture. Mais le mouvement de développement est commun, puisque le passage du laboratoire vers l'industrie se fait selon des chronologies et des dynamiques similaires.

### *3.2.2 Des outils pour les architectes*

Ce transfert a également lieu en architecture. Nous avons vu plus tôt dans le chapitre que la première tentative d'industrialisation des outils résultant des premières expériences de l'architecture computationnelle, les systèmes experts, n'a eu qu'un bref temps de vie. Les outils de dessin assisté par ordinateur et de modélisation rencontrent cependant un plus grand succès. Cependant, la plupart des éditeurs de logiciels qui s'installent durablement dans l'industrie du même nom sont les éditeurs dont nous avons déjà vu la contribution un peu plus haut, à travers les logiciels de manufacture et d'animation qu'ils produisent. Le passage à l'échelle de l'emploi des modeleurs 3D en architecture ne

[100] Au sens donné par ce mot au sein du champ computationnel - voir Chapitre I.
Daniel Davis, "A History of Parametric", 6 Aout 2013, https://www.danieldavis.com/a-history-of-parametric/, consulté le 14 Novembre 2021.
[101] David Weisberg, "The engineering design evolution", https://www.cadhistory.net/toc.htm, consulté le 30 Novembre 2021.

se fait en effet pas exactement par l'apparition d'une industrie à part entière[102]. Le transfert se fait plutôt de la recherche et de la pratique expérimentale à la pratique conventionnelle. Nous allons maintenant examiner comment cette transition se fait.

Nous avons mentionné qu'un peu de programmation informatique a lieu dans les une poignée d'agences dès les années 1960. C'est par exemple le cas chez Skidmore, Owings & Merrill (SOM), une agence américaine fondée à Chicago en 1936 par les architectes Louis Skidmore et Nathaniel Owings, rejoints en 1939 par l'ingénieur John Merrill. L'agence s'impose dans la décennie qui suit comme une agence majeure aux Etats-Unis, y concevant les premiers grands immeubles de l'International Style. Mais en parallèle d'une trajectoire certes prospère mais néanmoins plutôt conventionnelle, SOM montre très tôt un intérêt pour les technologies numériques au sens large. Dès les années 1960 et 1970, des programmes écrits en FORTRAN sont utilisés au sein de l'agence pour aider à la planification architecturale et urbaine[103]. Ces programmes sont écrits en interne par le Computer Group de l'agence, qui s'intéresse particulièrement à l'analyse structurelle et au calcul des besoins énergétiques - notamment parce que le groupe est essentiellement piloté par des ingénieurs. Cet intérêt est à l'origine du développement du logiciel Architecture Engineering System (AES), en utilisation dans l'agence dès les années 1980, qui combine un système de dessin assisté par ordinateur et du calcul des structures et des besoins énergétiques[104]. Malgré tout le travail engagé, SOM se tourne finalement au bout de quelques années vers les solutions logicielles mises à disposition par les éditeurs entre-temps, et abandonne son développement interne sur AES. Robert Aish, un des principaux développeurs d'outils pour le champ computationnel comme pour la pratique conventionnelle, estime que SOM a fait le choix d'abandonner le développement et l'usage d'AES car ils ne souhaitaient pas s'engager dans la réflexion sur l'évolution de leurs activités que cela impliquait. Si AES avait été utilisé seulement en interne, SOM aurait pu tirer des avantages de ses capacités, dont ses concurrents ne bénéficiaient pas, mais aurait dû prendre en charge toute la recherche et le développement. Vendre des licences de leur logiciel aurait pu permettre de compenser les coûts de développement, mais aurait signifié la perte de cet avantage concurrentiel. Surtout, SOM se voyant avant tout comme une agence d'architecture, ils n'auraient pas souhaité s'engager sur la voie d'un

[102] Encore aujourd'hui les éditeurs de logiciels proposent des offres qui sont rarement exclusivement dédiées à l'AEC - voir Chapitre VI.
[103]Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.
[104] Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013. Daniel Cashen et Neil Katz, "Skidmore, Owings & Merrill. Building on the legacy of technological and architectural innovation" dans Michael U. Hensel et Fredrik Nilsson (eds), *The Changing Shape of Architecture. Further Cases of Integrating Research and Design in Practice*, Routledge, 2019.

éditeur de logiciels[105].

SOM n'est d'ailleurs pas la seule agence de grande taille à faire un essai dans le développement de son propre logiciel. C'est aussi le cas par exemple de l'agence anglais Gollins Melvin Ward Partnership (GMWP), fondée en 1947 et à l'origine de nombreux bâtiments gouvernementaux et scolaires au Royaume-Uni. À la fin des années 1970, GMWP s'associe avec deux étudiants de l'université de Liverpool, John Davidson et John Watts, qui développent un système de dessin assisté par ordinateur. Ce système est un système de dessin en 2.5D, c'est-à-dire qu'il permet de tracer des plans, mais aussi de créer des modèles en trois dimensions pour obtenir des perspectives. Le système, baptisé le Really Universal Computer Aided Production System, ou RUCAPS, est d'abord utilisé à l'agence sur des projets tests. Puis GMWP fait le choix inverse de SOM : l'agence crée une filiale nommée GMW Computers, qui commercialise RUCAPS à partir de 1977. Bien que plusieurs centaines d'agences aient acheté RUCAPS dans la première moitié des années 1980, le logiciel est ensuite rapidement supplanté par des modeleurs plus avancés, et GMW Computers est racheté par Alias|Wavefront au début des années 1990[106]. Les années 1980 voient donc des logiciels de dessin assisté par ordinateur personnalisés arriver dans quelques agences. Les incursions des pratiques numériques en agence d'architecture ne sont pas donc seulement des incursions dans le monde industriel d'enseignants-chercheurs basés à l'université. L'activité au sein d'agences conventionnelles reste cependant mince, notamment en raison du prix des systèmes. Dans les années 1960, le poste de travail le plus simple est usuellement composé d'une tablette de programmation, d'un écran et de plusieurs traceurs, ce qui représente un investissement d'environ 90 000, sans compter le coût de financement de personnel formé à l'utilisation de ces outils. Le coût des postes de travail les plus avancés, des systèmes complets de dessin, se rapproche plutôt de 500 000$ de l'époque, soit près de 3,5 millions de dollars aujourd'hui. Si les années 1980 voient le coût de ces systèmes diminuer petit à petit, cela représente néanmoins toujours des investissements autour de 125 000$ dollars d'aujourd'hui. Seules des agences avec des moyens financiers conséquents peuvent donc se permettre d'investir dans de tels équipements au cours des premières décennies de développement des outils de dessin assisté par ordinateur.

Par la suite, d'autres agences encore s'essaient également au développement de logiciels en interne. Plus tardif que RUCAPS et AES, Digital Project possède une trajectoire un peu différente, qui fait du logiciel un élément plus durable du paysage de la conception assistée par ordinateur. Frank Gehry est

[105] Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.

[106] Port, Stanley (1989). The Management of CAD for Construction. New York: Springer. ISBN 9781468466058. Ruffle S. (1985) "Architectural design exposed: from computer-aided-drawing to computer-aided-design" Environments and Planning B: Planning and Design 1986 March 7 pp 385-389.
Robert Aish et Nathalie Bredella, "The evolution of architectural computing: from Building Modelling to Design Computation", Architectural Research Quarterly, vol 21, no 1, 2017.

un architecte américain né en 1929, et unanimement considéré comme l'un des architectes les plus importants de sa génération. Après une licence d'architecture à l'université de Californie du Sud et un master d'urbanisme à Harvard, qu'il ne termine pas, Frank Gehry exerce quelques années dans les agences d'autres architectes aux Etats-Unis et en Europe. En 1962, il fonde sa propre agence. À partir de la fin des années 1970, plusieurs projets le font accéder à une certaine renommée, notamment sa maison personnelle à Santa Monica. Ses travaux des années 1980, parmi lesquels ses premiers musées, l'amènent à être lauréat du Pritzker Prize en 1989[107]. Les décennies suivantes sont l'occasion de développer plus avant sa méthode de travail et son style architectural sculptural, pour lequel il est particulièrement connu. Mais le travail de Frank Gehry est particulièrement reconnu pour une autre raison : ses bâtiments sculpturaux sont considérés comme des applications pionnières de la modélisation numérique en construction. L'agence s'intéresse à partir de 1986 à l'intégration de modeleurs 3D dans ses flux de travail. Elle n'est donc pas pionnière à proprement parler, puisque nous avons vu que de nombreux praticiens s'intéressent alors déjà à la question, et que des suites logicielles utilisables pour la conception architecturale, même si elles n'ont pas forcément directement été conçues pour, sont alors déjà commercialisées depuis peu. Mais l'agence de Frank Gehry, en s'intéressant à ces nouvelles technologies de conception, contribue à les populariser au sein des pratiques conventionnelles. 1989 n'est pas seulement l'année du Pritzker pour Frank Gehry, c'est aussi l'année du recrutement de Jim Glymph, un architecte américain qui travaille depuis plusieurs années déjà à l'intégration d'outils numériques dans le processus de conception.

Jim Glymph est recruté pour une raison précise : l'agence rencontre des difficultés significatives dans la production des dessins d'exécution d'un projet de sculpture pour le village olympique de Barcelone, un poisson de 54m de long et de 35m de haut, composé d'une armature et d'une peau métalliques. Le délai de réalisation est court - dix mois - et l'agence cherche donc à créer un modèle 3D du poisson pour pouvoir en extraire les plans de construction. La maquette initiale, réalisée par Frank Gehry, est en bois et en métal. L'agence fait d'abord appel à William Mitchell, alors à Harvard, et à son étudiant Evan Smythe pour réaliser le modèle 3D. Les deux chercheurs se servent d'Alias-1, qui se révèle pourtant trop peu précis. Jim Glymph propose donc de recourir au modeleur CATIA, dont l'usage dans l'aéronautique et l'automobile lui fait estimer que moins de problèmes de précision vont se poser. En 1991, Rick Smith, consultant en informatique pour l'industrie de l'aéronautique, rejoint l'équipe et développe un modèle qui permet de visualiser les points ou le cadre métallique se connecte à la peau du poisson. Les données peuvent ainsi être transmises aux équipes de chez SOM, qui est à la fois responsable du masterplan du village olympique et bureau d'étude structure pour le poisson de Frank Gehry. Les données sont également transmises à Permasteelisa, l'entreprise responsable de la gestion du chantier, qui se dote également de CATIA pour l'occasion. Grâce à la mise en place du

[107] Le Pritzker Prize est la récompense la plus prestigieuse dans le domaine de l'architecture, un équivalent au prix Nobel en l'architecture en quelque sorte.

modèle CATIA et à la transmission des données qu'il permet, le poisson de Barcelone - El Peix d'Or - est achevé en six mois. Mais bien que l'exemple du poisson soit particulièrement marquant, car la modélisation numérique y est particulièrement utile pour réussir le projet dans les temps, ce n'est pas le seul projet qui y recourt au sein de l'agence ces années-là. À partir de la fin des années 1980, la méthode travail développée par l'agence de Frank Gehry à partir de CATIA est utilisée dans plusieurs projets. C'est le cas pour le Walt Disney Concert Hall, dont la façade était initialement prévue en pierre. Un premier modèle 3D est mis au point en 1989 pour en extraire les parcours outils à utiliser pour fraiser les panneaux de façade. Mais face au coût que représente un tel procédé de fabrication, le choix est finalement fait d'une façade métallique fixée sur des armatures en acier. Le processus de travail développé pour le poisson de Barcelone est donc de nouveau utilisé pour obtenir les dessins d'exécution. De 1991 à 1994, l'agence de Frank Gehry est chargée de la conception d'un abribus pour l'exposition universelle de Hanovre, et se saisit de l'occasion pour concevoir intégralement le projet sur CATIA. La petite échelle permet d'explorer le processus en entier, de la conception à la livraison. Enfin, le musée Guggenheim de Bilbao, conçu de 1991 à 1997, stabilise le processus de conception qui sera ensuite mis en œuvre systématiquement ensuite au sein de l'agence. Le projet est développé à la main, par le biais de croquis et de maquettes. La maquette finale est scannée pour obtenir un modèle 3D. Puis une maquette est fabriquée à partir du modèle 3D. Dans le cas du musée Guggenheim de Bilbao, elle est fraisée. Une fois la correspondance entre la maquette et le modèle 3D vérifiée, les calculs de structures et les dessins d'exécution sont effectués sur CATIA.

Le musée Weisman, à Minneapolis, réalisé entre 1991 et 1993, est le dernier projet réalisé par Frank Gehry et ses équipes intégralement à la main. Les années 1990 et les projets qui suivent sont l'occasion pour l'agence de développer une méthode de travail à partir du recours aux outils numériques - scan et modélisation - pour retranscrire dans les dessins d'exécution les géométries complexes sculptées par Frank Gehry pour ses bâtiments. Cette méthode de travail s'appuie au départ sur le logiciel CATIA de Dassault Systèmes, conçu initialement pour le secteur de l'aviation. Au fil des années l'agence de Frank Gehry développe sa propre variante de CATIA, faite non pour les avionneurs mais pour les architectes : Digital Project. Conçu en collaboration avec Dassault et doté des mêmes capacités de modélisation, Digital Project diffère de CATIA notamment par son interface, pensée pour la conception architecturale et qui se veut plus intuitive à utiliser que celle de son logiciel parent. En 2002, Frank Gehry crée avec Jim Glymph et Dennis Shelden Gehry Technologies, une filiale de l'agence qui commercialise Digital Project. Distributrice du logiciel pour Dassault Systèmes, la structure propose aussi du conseil pour l'adoption des processus de conception numérique aux agences d'architecture intéressées. Digital Project se fait au fil des années une place relativement stable parmi les solutions logicielles du secteur de l'architecture et de la construction, la notoriété de Frank Gehry contribuant notamment à sa popularisation chez d'autres architectes. En 2014, Gehry Technologies est cependant racheté par l'éditeur de logiciels Trimble, alors en pleine expansion du

domaine du positionnement satellite vers celui de la construction. Avec Digital Project et son pendant Gehry Technologies, l'agence de Frank Gehry offre cependant l'exemple d'une agence qui s'investit jusqu'au bout dans la conception d'un modeleur 3D dédié à l'architecture. Le développement du logiciel est certes plus facile que pour SOM ou GMWP, partis de rien pour développer AEC ou RUCAPS, puisque Digital Project est basé en grande partie sur CATIA. L'association avec Dassault Systèmes rends aussi plus facile la gestion de l'ouverture de l'agence à un marché différent de celui de la conception architecturale. Comme AES ou RUCAPS, Digital Project est un modeleur dédié spécifiquement à la conception architecturale, plutôt qu'à la manufacture ou l'animation. Son développement est guidé par la pratique architecturale en vigueur autour de Frank Gehry. Digital Project est donc le détournement d'un logiciel produit pour l'aéronautique pour le transformer. La raison initiale du recours à CATIA au sein de l'agence est un impératif constructif - la précision du logiciel est un atout pour obtenir les représentations précises nécessaires à l'obtention de fichiers d'exécution. Le recours à CATIA est d'ailleurs associé au départ à une méthodologie constructive, celle des armatures métalliques sur lesquelles sont fixées des feuilles de métal - un système constructif qu'on retrouve à Bilbao, au musée à Seattle et dans de nombreux projets de la période de Frank Gehry. Dans son livre Digital Gehry, qui fait le récit de l'adoption de CATIA au sein de l'agence, Bruce Lindsey note d'ailleurs qu'il s'agit d'une logique constructive de l'extérieur vers l'intérieur ("skin-in") jusqu'alors bien plus fréquente dans l'aéronautique et l'automobile que dans la construction[108]. Mais les méthodes de représentation intégrées au processus de travail de l'agence ouvrent un univers formel plus large à l'architecte, et l'adaptation de CATIA en Digital Project finalise l'intégration des possibilités offertes par le modeleur plus en amont du processus de conception.

Dernier exemple de modeleur que nous allons examiner, Form*Z va plus loin que Digital Project, puisqu'il s'agit d'un modeleur entièrement développé par un architecte pour les architectes. Il n'est donc pas détourné à partir d'un modeleur pensé pour d'autres professions, mais pensé directement pour la conception architecturale. Son concepteur, Christos Yessios, est né en Grèce en 1938. Après un master en économie, il s'inscrit en licence d'architecture à Thessalonique, puis en urbanisme à l'Université de Pittsburgh, où il suit également des cours d'informatique et de programmation[109]. Christos Yessios rencontre à Pittsburgh Charles Eastman, qu'il suit à Carnegie Mellon où il s'inscrit en doctorat sous sa supervision en 1969. Il termine son doctorat en 1973, et devient le premier docteur diplômé de Carnegie Mellon dans le domaine de l'architecture computationnelle. Sa thèse, *Syntactic Structures and Procedures for Computable Site Planning*, relate le développement du programme Siplan (Site Planning Language and System), basé sur le travail de l'urbaniste Kevin Lynch et écrit en

[108] Bruce Lindsey, *Digital Gehry*, Birkhäuser, 2001.
[109] Pierluigi Serraino, *History of Form*Z*, Birkhäuser, 2002.

ALGOL[110]. Lynch propose en effet dans son livre *The Image of a City* une décomposition de l'espace urbain en cinq éléments : chemins, bordures, quartiers, noeuds et monuments[111]. Pour Lynch, ces éléments sont ce qui permet aux personnes de se former une image mentale de la ville dans laquelle ils évoluent, mais cette décomposition peut aussi servir de base à un système algorithmique, comme le propose Christos Yessios. Pendant le reste des années 1970, Christos Yessios continue ses recherches sur la décomposition de l'espace pour l'établissement de programmes informatiques de conception architecturale. Après Carnegie Mellon et ALGOL, il enseigne à l'Ohio State University, où il programme désormais sur FORTRAN. À partir de 1978 et de son travail sur le programme ILCON (Interactive Language for 3D CONstruction), Christos Yessios commence cependant à s'intéresser de plus près à la modélisation. Son approche se concentre sur les éléments nécessaires pour donner une description complète de compositions en trois dimensions, une approche dérivée de l'ambition descriptive qu'on trouve dans ses travaux précédents.

Ses projets de recherche suivants sont tous des systèmes de dessin assisté par ordinateur, de plus en plus poussés. Au début des années 1980, IBM offre à l'université d'Ohio une bourse de recherche d'un million de dollars pour concevoir un nouveau système de conception assistée par ordinateur. Yessios, qui vient de terminer la conception de TEKTON, système utilisé en interne à l'université, entre alors en collaboration avec IBM. Le résultat de cette collaboration est ARCHIMODOS (Architectural Modeling Design and Drafting System of the Ohio State University), écrit en FORTRAN IV au sein du Computer Aided Architectural and Environmental Design Laboratory de la School of Architecture de OSU, pour fournir aux élèves un outil de modélisation numérique pour leurs studios de projet. ARCHIMODOS est l'occasion pour Christos Yessios de développer l'idée de modélisation par le vide, une logique qui s'oppose à la logique de modélisation des solides alors en cours de développement dans le domaine de la manufacture. L'ambition est de créer un système de modélisation qui permette de donner forme à des surfaces entourant du vide - comme un architecte crée une enveloppe qui contient les espaces du projet. IBM trouve le principe de modélisation à partir du vide sur lequel ARCHIMODOS est basé peu adéquat, et malgré son utilisation massive au sein de l'université, le système n'est donc pas commercialisé. Christos Yessios a cependant le temps de commencer à s'intéresser à un autre enjeu technique : le transfert du programme de modélisation d'un système informatique de dessin sur-mesure vers un plus petit ordinateur générique - c'est-à-dire la transformation du programme en un logiciel à part entière. Quand IBM abandonne le financement d'ARCHIMODOS, Christos Yessios vient tout juste de rencontrer Peter Eisenman, qui s'intéresse depuis peu de près aux outils numériques. En 1987, Peter Eisenman invite donc Christos Yessios à participer au Fractal Studio, ou il prévoit de mettre en œuvre des outils algorithmiques similaires à

[110] Christos Yessios, Syntactic Structures and Procedures for Computable Site Planning, Thèse de doctorat. Carnegie Mellon, 1973.
[111] Kevin Lynch, *The image of a city,* The MIT Press, 1960.

ceux utilisés par John Frazer à la AA[112]. L'année suivante, le duo décide de développer un logiciel de dessin assisté par ordinateur pour Macintosh avec l'aide de quelques étudiants : Form*Z. Début 1990, Christos Yessios fonde AutoDesSys avec son collègue de l'Ohio State University Richard Parent, professeur d'informatique, et avec son ancien élève David Kropp. Quelques mois plus tard, l'entreprise présente le logiciel Form*Z à l'occasion d'un salon de la construction à Atlanta. Enfin, en 1991, la première version de Form*Z sort. Le succès est au rendez-vous, notamment parce que les opérations booléennes, qui servent à unir plusieurs formes tridimensionnelles en une seule, sont particulièrement performantes dans Form*Z, et parce que le logiciel est particulièrement peu cher comparé aux autres offres existantes. En 1995, Christos Yessios quitte l'Ohio State University pour se concentrer sur la poursuite du développement de Form*Z, ce qui débouche en 1996 sur une version pour Windows du logiciel.

Le succès de Form*Z est vraisemblablement aussi lié à sa conception spécifique pour l'architecture. Celle-ci ne se résume pas à des opérations booléennes particulièrement performantes. La collaboration de Christos Yessios avec Peter Eisenman résulte d'abord en une série d'algorithmes intégrés à Form*Z pour que les étudiants puissent s'en servir par le biais du logiciel. Mais cette collaboration se poursuit ensuite, et plusieurs des fonctionnalités du logiciel, comme la fonction de déroulement des faces d'une géométrie, sont le résultat d'une demande directe de Peter Eisenman et de son agence. Christos Yessios accorde également beaucoup d'attention à l'interface de Form*Z. Plutôt que de proposer comme dans les autres programmes alors disponible une vue des objets modélisés régie par des paramètres numériques à entrer manuellement, il met en place le cône de vision. Celui-ci est un dispositif beaucoup plus proche de la visualisation contemporaine dans les modeleurs, qui permet de tourner plus facilement autour de l'objet et de se déplacer plus facilement dans l'espace du modèle - un avantage indéniable pour les architectes. Son travail avec les étudiants de l'Ohio State University et avec Peter Eisenman, mais aussi ses programmes de modélisation précédents permettent à Christos Yessios de disposer d'une solide expérience des habitudes de travail des architectes et des interfaces qui rendent les logiciels de modélisation plus proches de ces habitudes. Enfin, AutoDesSys ne se contente pas de commercialiser Form*Z. L'entreprise édite également le journal *The Fantastic and the Constructible*, qui rassemble cas d'usage, conseils d'utilisation et nouvelles du développement du logiciel. La publication est une plateforme pour l'entreprise, qui permet de populariser le logiciel et de fédérer une communauté autour de son usage. Et bien que Form*Z aie été remplacé par des modeleurs plus récents au sein de la majeure partie des agences, la communauté fédérée autour du logiciel continue d'exister et de louer les avantages de ce modeleur si intuitif.

[112] Pierluigi Serraino, *History of Form*Z*, Birkhäuser, 2002.

Autographics, AES, RUCAPS, Digital Project, Form*Z… Peu à peu, des architectes s'emparent non seulement des outils numériques que les sciences de l'informatique en plein développement contribuent à mettre au point. Mais surtout, peu à peu, des architectes se confrontent directement au travail de développement de programmes et d'outils logiciels. Leur expérience en tant que praticien leur permet de créer des outils adaptés spécifiquement à la conception architecturale. Les logiciels de modélisation sont caractérisés notamment par leur kernel géométrique, ou noyau : l'ensemble des définitions mathématiques sur lesquelles la description des géométries est basée. Les possibilités offertes par les modeleurs diffèrent en fonction de ce kernel. Les modeleurs du champ de l'animation et ceux de la manufacture sont très différents les uns des autres. L'un des champs demande de pouvoir représenter les plis d'un tissu ou le fouillis d'une chevelure, quand l'autre nécessite une grande précision et une sobriété formelle en accord avec les processus de fabrication. Les architectes aussi ont leur propre champ formel, développer des modeleurs propres à la conception de l'espace ne fait que refléter cette spécificité - d'où l'idée de produire des outils spécifiques au champ, par les praticiens eux-mêmes, les mieux placés pour connaître leurs exigences géométriques. Certes, les modeleurs 3D que les années 1980 voient apparaître et prendre une place croissante dans le paysage architectural ne relèvent pas d'un recours direct à la programmation informatique. Nous avons pourtant indiqué qu'il s'agit du principal élément de définition du champ computationnel. Mais la création de modeleurs 3D dédiés à la conception architecturale est un moment de développement d'outils par et pour les architectes. C'est aussi un moment où le champ computationnel n'est guère éloigné des cercles ou se développent ces outils : SOM a un pied dans ce champ depuis les années 1960, William Mitchell contribue aux réflexions sur le numérique au sein de l'agence de Frank Gehry. Avec le développement d'outils au sein des agences d'architecture plutôt que dans des laboratoires de recherche en informatique, le rapport de force entre programmateurs et architectes s'inverse, autorisant une émancipation du champ parent. Le recours aux modeleurs et le recours aux algorithmes vont par la suite suivre des trajectoires différentes. Mais l'instant partagé de l'émergence de ces outils impose une idée dans le champ computationnel : les architectes peuvent et doivent créer leurs propres outils informatiques. L'abandon par SOM d'AES et le fait que somme toute peu d'agences se lancent dans le développement de leurs propres systèmes de dessin assisté par ordinateur montre que l'idée de concevoir ses propres outils ne perdure pas dans les pratiques architecturales conventionnelles. Mais alors même que le champ computationnel continue à se construire en marge de ces pratiques conventionnelles, faire ses propres outils devient un marqueur clé du domaine, inspiré par ce moment d'appropriation par les architectes de la modélisation 3D à travers la création d'outils dédiés.

### *3.2.3 Un biais de représentation*

Le tournant industriel des années 1980 auquel nous nous intéressons depuis le début du chapitre peut s'envisager sous deux angles. Il s'agit d'une part d'observer la création d'une industrie du logiciel,

plus exactement de l'essor du développement et de la production de certains types de logiciels, au sein d'une industrie de l'informatique qui se développe rapidement et sur plusieurs plans : équipements informatiques, logiciels informatiques, conseil et personnels spécialisés. Il s'agit donc de comprendre si les systèmes experts, les logiciels de modélisation pour l'animation, la manufacture ou l'architecture sont produits et commercialisés à grande échelle, avec des acteurs spécifiques à leur production et un marché. L'existence de ce marché, nécessaire à l'industrialisation, implique que l'industrie de la construction et de l'architecture utilise ces logiciels massivement, puisque ce sont les clients qui composent ce marché. Le tournant industriel doit donc s'observer sur deux plans : les programmeurs et les utilisateurs. Dans le cas des systèmes experts, un tournant industriel a bien lieu sur le premier plan : une industrie de production des systèmes experts se met en place. Mais elle repose sur l'existence d'autres utilisateurs, et l'architecture conventionnelle ne se saisit au final pas de ces outils. Dans le cas des modeleurs cependant, le tournant s'effectue sur les deux plans, avec d'une part la naissance de l'industrie du logiciel et d'autre part l'adoption de ses produits par des praticiens de tous bords pour la représentation du projet architectural. Les modeleurs 3D et les ordis des années 1980 vont en effet vite devenir accessibles aux architectes à la suite du développement de l'infographie, et de plus en plus indispensables à leur pratique. Si l'on peut parler d'hiver procédural dans le champ computationnel, parce que son développement est marqué par des affaiblissements des expérimentations de programmation, c'est aussi le cas parce qu'en comparaison de la trajectoire d'autres outils numériques, les algorithmes prospèrent peu. Notamment en comparaison des modeleurs 3D, dont nous avons vu qu'ils n'impliquent pas que leur utilisateur aie recours directement à la programmation, mais qui prospèrent sur plusieurs plans : leur développement comme industrie, leur développement par des architectes pour des architectes, mais leur adoption par les praticiens. Cette adoption s'accompagne de l'idée d'un style numérique propre aux projets conçus avec ces outils, marqueur de son époque comme l'avaient été d'autres styles et mouvements avant. Cette idée devient phare au cours des années 1990, et place la question du recours aux modeleurs sur le devant de la scène architecturale. Ce style propre est reconnaissable aux géométries complexes qui composent les projets, et à des formes courbes qui caractérisent en partie le virage numérique pour beaucoup dans les années suivantes. La popularisation des recours aux modeleurs devient ainsi le socle de l'émergence d'une esthétique digitale en architecture.

Quelques architectes célèbres contribuent notamment avec leurs réalisations à fixer ce qu'est ce style numérique. Frank Gehry contribue bien sûr à l'établissement de cette patte en combinant des choix formels très reconnaissables à un recours précoce à la modélisation. De nombreux commentateurs associent le travail sculptural de Frank Gehry à son recours aux outils numériques, en s'appuyant notamment sur les témoignages des spécialistes en chef de ces technologies de l'agence, Jim Glymph

et Dennis Shelden[113]. Mais Frank Gehry lui-même tient pourtant souvent le discours inverse : le recours au numérique n'impliquerait aucun changement dans sa pratique architecturale. La coexistence de ces deux discours au sein de l'équipe s'expliquent notamment par l'intérêt que les différents praticiens en retirent. Jim Glymph et Dennis Shelden valorisent leur travail au sein de l'agence en soulignant l'importance des outils numériques dans sa pratique. À l'inverse, Frank Gehry met en scène son génie architectural en diminuant cette importance[114]. Une anecdote certainement très exagérée, mais racontée à plusieurs reprises par Frank Gehry et reprise partout le montre : il raconte refuser de jeter le moindre regard aux écrans avant que le projet ne soit finalisé, pour ne pas polluer sa vision initiale[115]. Les deux discours sont par ailleurs complémentaires puisqu'ils sont vendeurs chacun pour une partie des activités de l'équipe. Valoriser l'apport des outils numériques fait sens pour les activités commerciales de Gehry Technologies, appuyer le talent de Frank Gehry fait sens pour les activités architecturales de l'agence.

Ces deux discours sont des narrations certainement exagérées de part et d'autre. Pourtant, examiner les réalisations architecturales de Frank Gehry permet de se faire une idée de la réalité du poids du numérique dans ses pratiques de conception. Au vu de la chronologie, formellement et architecturalement, le processus de Frank Gehry semble peu évoluer au moment du tournant numérique de l'agence. Les marqueurs principaux de son processus de conception ne bougent qu'à une période qui prédate ce tournant : entre ses premières productions et sa maison. Les principes qui émergent - hiérarchisation des espaces puis conception de l'enveloppe qui les englobe, tout en laissant ce morcellement des espaces apparents - sont ceux qu'on retrouve aussi dans la série de projets au début de recours au numérique (musée Weisman, Walt Disney Concert Hall, musée Guggenheim de Bilbao), puis dans les projets ensuite solidement ancrés dans le recours au numérique (fondation Louis Vuitton, tour Luma Arles). Ce changement peut être considéré comme une simple évolution de son travail, qui n'a rien à voir avec l'arrivée du numérique puisqu'il date d'avant. Même si le numérique permet des réalisations techniques différentes, les principes architecturaux qui sous-tendent la pratique de Gehry ne bougent pas après le tournant numérique de son agence mais avant. Jim Glymph indique que le recours aux outils numériques, uniquement utile sur un plan technique au départ, a cependant modifié la pratique de Frank Gehry au cours du temps[116]. Le rôle des techniciens du numérique et des ingénieurs ne change pourtant pas entre la période d'arrivée du numérique à l'agence et les projets plus tardifs. Pour les voiles en verre de la Fondation Louis Vuitton par exemple, le rôle de la maquette n'a pas du tout changé. Elle sert toujours de vérification du respect de l'intention architecturale initiale

[113] Bruce Lindsey, *Digital Gehry*, Birkhäuser, 2001.
[114] Un épisode des Simpson tourne même en dérision son travail sculptural, le mettant en scène froissant une boule de papier avant de la jeter sur la table et de déclarer le bâtiment terminé.
[115] Bruce Lindsey, *Digital Gehry*, Birkhäuser, 2001. Smith, R., *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace, 2017.
[116] Bruce Lindsey, *Digital Gehry*, Birkhäuser, 2001.

de Frank Gehry, alors que le processus computationnel de définition de leurs géométries est bien plus évolué qu'un simple scan pour la retranscription des volumes d'une maquette.

*Figure 11. a. Walt Disney Concert Hall, Gehry Partners ; b. Fondation Louis Vuitton, Gehry Partners ; c. Fondation Luma, Gehry Partners.*

Il n'est donc pas aisé de saisir vraiment ce qui est caractéristique du recours aux modeleurs chez Frank Gehry, puisqu'ils ne semblent pas induire de changement architectural profond. Le milieu des années 1980 est d'ailleurs marqué à l'agence par plusieurs refus nets de Frank Gehry de recourir à des outils numériques. Il n'accepte qu'au moment ou des problèmes techniques sévères se posent avec la réalisation des plans d'exécution ou des fichiers de fabrication. Le processus de Frank Gehry reste un processus de travail qui cantonne les outils numériques à des travaux techniques de rationalisation pour la construction et d'exécution. Le changement est purement technique, même s'il permet en effet de réaliser des formes différentes. Comparer le Weisman Art Center, dernier projet réalisé sans outils numériques à l'agence, et le musée Guggenheim de Bilbao, premier projet réalisé avec leur appui constant, permet de s'en rendre compte. Les principes architecturaux sont identiques, le processus de conception de Frank Gehry aussi, mais le recours à la modélisation se saisit dans les détails formels. Le Weisman Art Center est composé d'un ensemble de solides platoniciens, avec une façade métallique faite de pièces rectangulaires identiques. Le musée Guggenheim de Bilbao présente les mêmes volumes éclatés et la même façade métallique. Mais les éléments qui composent la façade sont tous différents, et les courbes des volumes ne sont pas du tout celles de solides platoniciens, ce sont des courbes que seule la manipulation d'équations dans un modeleur permet de documenter suffisamment précisément pour pouvoir les construire. Malgré le discours de l'agence qui qualifie les outils numériques comme indispensables et transformants, examiner de près leur rôle montre qu'il s'agit surtout d'une contribution technique. Même si ce n'est pas négligeable, ces outils ont sans doute moins d'impact sur l'architecture que l'agence ne le laisse parfois croire. C'est néanmoins un discours utile à l'agence, et la combinaison des deux discours ensemble contribue à fixer certaines des

caractéristiques formelles du travail de Frank Gehry comme les marqueurs du recours aux modeleurs, preuve de leur impact sur la pratique architecturale.

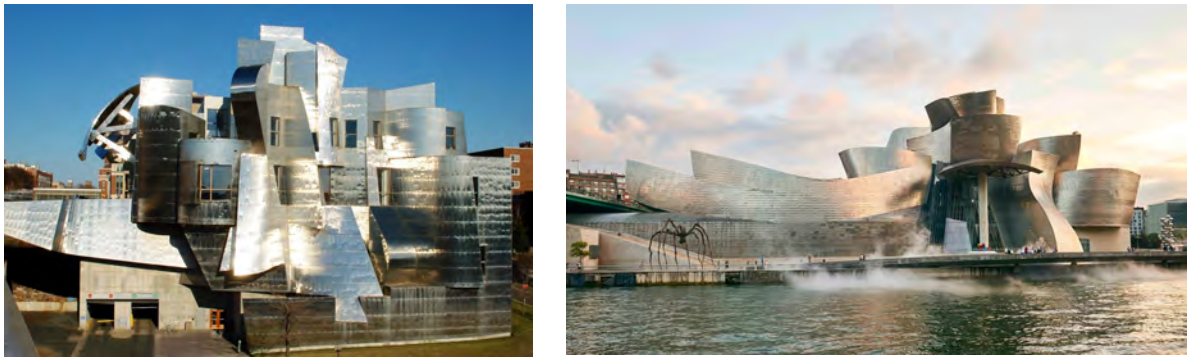


*Figure 12. a. Weisman Art Museum, Gehry Partners ; b. Musée Guggenheim de Bilbao, Gehry Partners*

Au cours des années 1990, Frank Gehry n'est pas le seul architecte célèbre à utiliser des modeleurs : d'autres vont contribuer à établir une connexion entre ces outils et des géométries courbes signatures. Zaha Hadid, architecte anglo-irakienne né à Baghdad en 1950, étudie d'abord les mathématiques à l'université américaine de Beyrouth, puis l'architecture à la AA, auprès de Rem Koolhaas. Elle fonde l'agence Zaha Hadid Architecture (ZHA) en 1980 et participe en 1988 à l'exposition Deconstructivist Architecture au MoMa de New York[117]. Cette exposition met son travail en lumière, ce qui lui offre l'occasion de participer à de nombreux concours, et d'atteindre une grande renommée au fil des années qui suivent : elle reçoit le Pritzker Prize en 2004. Avant d'atteindre ces sommets, Zaha Hadid est très remarquée dès son passage à la AA, ou elle est invitée à enseigner dès ses études terminées. Son travail pictural est saisissant. Les peintures qu'elle crée à la place des dessins de projet traditionnels sont considérées comme la marque d'un processus de conception innovant, qui s'ancre dans le déconstructivisme - d'ou sa participation à l'exposition du MoMa. Bien qu'elle soit basée sur un travail pictural plutôt que sculptural, la pratique de Zaha Hadid peut ainsi être comparée à celle de Frank Gehry pour sa relation à l'abstraction et pour son processus d'élaboration formelle, très lié à la création d'un objet particulier. Mais leurs pratiques sont aussi comparables pour le rôle qu'y occupe les outils numériques. Le premier bâtiment construit par Zaha Hadid est la caserne de pompiers de l'usine Vitra, en Allemagne. La première usine Vitra ayant été détruite par un incendie, le fabricant de mobilier fait reconstruire ses infrastructures par de grands noms de l'architecture. À terme, l'endroit devient d'ailleurs un musée d'architecture et de design. En 1993, Vitra commande à Zaha Hadid une caserne de pompiers, destinée à protéger le lieu d'une répétition du drame. Bien que la caserne n'ait au final que peu servie en tant que telle, elle est désormais au même titre que les autres bâtiments du site un classique de l'architecture. Zaha Hadid dispose de moins d'un an pour concevoir et construire le

[117] Philip Johnson et Mark Wigley, *Deconstructivist architecture*, MoMa, 1988.

bâtiment. Elle le conçoit comme une série de plans de béton qui s'intersectent, définis à partir des lignes directrices du site et représentés dans une série de peintures. La question se pose néanmoins de la transformation de ces peintures abstraites en un espace constructible. L'enjeu est d'autant plus grand qu'il s'agit du premier projet construit de Zaha Hadid, jusqu'ici connue comme une architecte aux propositions seulement spéculatives. Pour ce passage du pictural à l'objet architectural, l'agence réalise un modèle numérique, également utilisé pour produire une animation des différents plans de béton qui le composent[118]. Il s'agit donc, comme pour Frank Gehry, de mettre un logiciel de modélisation au service de la réalisation des dessins d'exécution. Zaha Hadid utilise la modélisation numérique à un stade un peu différent de Frank Gehry, qui a lui construit de nombreux bâtiments avant. Mais ce n'est pas pour autant le début de la carrière de Zaha Hadid quand elle saute le pas. Elle partage par ailleurs la méfiance initiale de Frank Gehry à l'égard de ces outils, que ses collaborateurs la poussent à utiliser. Zaha Hadid partage aussi la conviction de Frank Gehry de l'intérêt de ces outils sur le plan technique, acquise après les premières utilisations. Contrairement à Frank Gehry, Zaha Hadid ne se sert cependant pas de Digital Project, ni d'un modeleur issu de la manufacture. Dès la sortie de Maya, c'est ce logiciel que l'agence installe au cœur de son processus. La dynamique, le mouvement jouent un rôle clé dans le travail de Zaha Hadid, et le recours à un l'animation permet de montrer d'autant mieux ce rôle. L'aide apportée par le modeleur tient plus de la formalisation d'une peinture abstraite vers un bâtiment aux espaces clairement définies et représentés que d'un recours constructif. Dans le cas de Zaha Hadid, la transition faite au sein de l'agence de Frank Gehry de Maya vers CATIA puis Digital Project est donc moins utile.

[118] https://www.zaha-hadid.com/architecture/vitra-fire-station-2/, consulté le 30 Novembre 2021.

*Figure 13. a. Peinture préparatoire pour le projet de la caserne Vitra, Zaha Hadid ; b. Caserne Vitra, Zaha Hadid Architects*

Il existe chez Zaha Hadid comme chez Frank Gehry une corrélation entre le style architectural cultivé et le recours aux outils de modélisation numérique. Les premiers bâtiments de Zaha Hadid, construits dans les années 1990, sont marqués par le déconstructivisme, angulaires, éclatés, comme la caserne de Vitra ou le tremplin de Bergisel, en Autriche. Puis, à partir des années 2000, ses travaux s'emparent d'un vocabulaire beaucoup plus arrondi, et sont aussi beaucoup plus massifs. On le voit dans le Phaeno Science Center, ou au musée Ordrupgaard, dont la coque en béton ne présente aucun angle et donc la masse arrondie préfigure les travaux suivants de l'architecte. Ces courbes et cette massivité deviennent caractéristiques d'un style aussi marquant et reconnaissable visuellement que celui de Frank Gehry. Le centre Heydar Aliev ou l'Abu Dhabi Performing Art Center en sont des exemples, et on retrouve le même vocabulaire dans le mobilier aussi conçu par l'agence. Néanmoins, la

comparaison avec Frank Gehry ne fonctionne que dans un premier temps. En 1988, l'agence embauche un élève de Zaha Hadid, Patrik Schumacher. Diplômé de la AA en 1990, il fait toute sa carrière au sein de l'agence. Surtout, c'est Patrik Schumacher qui convainc Zaha Hadid de recourir à des outils de modélisation numérique. Mais les modeleurs ne sont pas les outils qui intéressent le plus Patrik Schumacher. Celui-ci porte bien plus d'intérêt à la programmation informatique et à son usage pour la conception architecturale qu'à la modélisation en soi. Le recours à Maya au sein de l'agence n'est pas privilégié seulement pour l'animation, mais aussi pour la bibliothèque d'algorithmes que le logiciel permet d'utiliser, et pour MEL, le langage de programmation intégré au logiciel. Patrik Schumacher devient vite le bras droit de Zaha Hadid et occupe donc une place très importante à l'agence, au sein de laquelle il fait largement explorer et adopter les outils algorithmiques au fil des années. Sans abandonner la modélisation, ce sont les pratiques computationnelles que ZHA place au cœur de ses activités. Plutôt que de suivre une trajectoire similaire à celle de Frank Gehry, l'agence développe en interne des outils algorithmiques, un travail qui culmine en 2007 avec la création de l'unité interne ZHA Code[119]. Schumacher insiste néanmoins lui aussi sur l'idée que les outils numériques sont associés à un style architectural particulier, qu'il nomme le paramétricisme[120]. Son discours contribue lui aussi à associer le recours à des modeleurs, en particulier issus de l'animation, à un style formel architectural particulier, qui renforce la corrélation déjà créé par Frank Gehry auparavant. Or, la caractéristique formelle que partagent les réalisations de Frank Gehry et de Zaha Hadid, c'est la courbe, si facile à décrire et à manipuler avec les modeleurs. L'un comme l'autre contribuent donc à établir et à populariser l'idée d'un lien entre numérique et courbe.

[119] Voir Chapitre IV.
[120] Patrik Schumacher, "Parametricism - A New Global Style for Architecture and Urban Design", in: *AD Architectural Design - Digital Cities*, Vol 79, No 4, July/August 2009. Voir aussi Chapitre I.

*Figure 14. a. Tremplin de Bergisel ; b. Musée d'Ordrupgaard ; c. Centre Heydar Aliev ; d. Abu Dhabi Performing Art Center, , Zaha Hadid Architects*

Nous avons vu un peu plus haut que Greg Lynn est un habitué de l'expérimentation avec des outils venant d'autres industries en ce qui concerne les processus de fabrication numérique. Mais son travail de conception n'est pas en reste, car on y retrouve le même mécanisme de détournement. L'Embryological House dont nous avons vu le processus de fabrication s'appuie en effet sur le recours à Maya pour modéliser les formes ovoïdes qui la caractérisent. Le projet n'est en effet pas seulement l'occasion d'examiner une méthode de manufacture novatrice pour l'architecture, mais aussi un jeu d'exploration du modeleur et de ses performances. L'Embryological House s'inscrit par ailleurs dans une plus vaste entreprise de conception architecturale à l'aune des performances des logiciels de modélisation. En témoigne l'étrange projet de la Stranded Sear Tower, proposé par Greg Lynn en 1992 à l'occasion d'une exposition dont le thème est la réinterprétation de monuments de Chicago en de nouvelles propositions architecturales. Déplacés au bord du fleuve, les rigides parallélépipèdes se transforment avec Greg Lynn en un paquet de cylindres avachis et emmêlés, caractéristiques de l'approche ludique, presque cartoonesque que revendique l'architecte. Détourner des modeleurs 3D issus de l'industrie de l'animation ne relève pas pour Greg Lynn simplement d'une question technique, il s'agit aussi d'une passerelle vers l'affirmation de son approche architecturale, qui s'ancre à la fois dans une certaine dynamique visuelle et dans une forme de personnification de ses propositions architecturales. Les jouets découpés utilisés dans plusieurs de ses projets de mobilier, les variantes de l'Embryological House présentées comme autant de personnages ou de versions d'un même personnage animé[121], le titre de sa monographie *Animate Form*[122] sont autant de signes de cette approche. Le projet Blob Wall, parmi les plus connus, rassemble la plupart des caractéristiques du travail de Greg Lynn en un seul objet, ludique, coloré, courbe, support d'une réflexion sur la fabrication comme sur la morphologie de notre environnement bâti. Greg Lynn articule en effet dans son travail expérimentation et théorisation. Il est à l'origine dans l'essai Blobs, or *Why Tectonics is*

[121] Lawrence Bird et Guillaume LaBelle, "Re-Animating Greg Lynn's Embryological House: A Case Study in Digital Design Preservation", Leonardo 43(3), 2010.
[122] Greg Lynn, *Animate Form*, Princeton Architectural, 1997.

*Square and Topology is Groovy* du terme *blob*, contraction selon lui des termes Binary Large Object[123]. Le mot désigne les formes molles et courbes caractéristiques de l'architecture de Greg Lynn, mais est vite adapté par nombre de ses lecteurs en un terme qui désigne l'ensemble des formes courbes et complexes considérées comme caractéristiques du recours à des modeleurs numériques. Entre texte et expérimentations, Greg Lynn se crée au cours des années 1990 l'image d'un des pionniers de ce style numérique, et ses réflexions sur les blobs contribuent à forger un imaginaire formel propre au numérique.

*Figure 15. a. Stranded Sear Tower ; b. Blobwall, Greg Lynn FORM*

Loin des intrigants morceaux de jouets de Greg Lynn ou des spectaculaires bâtiments de Zaha Hadid et Frank Gehry, le recours aux modeleurs et aux outils algorithmiques sait pourtant aussi se faire discret. Skidmore Owings & Merrill en constitue un bon exemple : cette agence, présente dans le champ depuis le début, n'en a encore jamais raté une étape. Des premiers programmes en FORTRAN a ses propres modeleurs en interne, puis aux algorithmes plus complexes développés plus tardivement - nous le verrons aux chapitres suivants -, SOM a toujours gardé un œil sur les outils en usage au sein du champ. Ces outils sont systématiquement expérimentés au sein de l'agence, avec plus ou moins de succès et sans être toujours généralisés à tous les salariés. Pourtant, leur pratique est bien plus conventionnelle, les bâtiments réalisés bien plus sobres que les travaux de leurs homologues de chez ZHA ou Gehry Partners. Seule trace de cette expérimentation intensive avec les outils numériques, l'expertise de l'agence dans la conception de bâtiments de très grande hauteur : c'est notamment dans ce but que SOM s'intéresse tant aux outils algorithmiques. Malgré l'attention de Skidmore, Owings & Merrill, qui surveille constamment l'apparition d'outils qui pourraient améliorer leur quotidien d'architectes et les possibilités de conception accessibles, les outils numériques restent relégués au second plan. L'abandon de leur système de dessin assisté par ordinateur AES le montre, il s'agit pour

[123] Greg Lynn, *Folds, Bodies & Blobs. Collected Essays*, La lettre volée, 1998.

SOM de mettre les outils au service d'une pratique architecturale à part entière, qui n'a pas besoin d'être bouleversée par les outils numériques. Ces derniers offrent une simple extension du domaine technique. Malgré les discours et les coups d'éclats, la modélisation 3D n'impose donc pas systématiquement un style courbe et complexe, et le recours aux modeleurs croissant dans les agences s'ancre aussi dans une pratique plus sobre, comme celle de SOM.

*Figure 16. a. One World Trade Center ; b. 100 Mount Street ; c. Guoco Tower, Skidmore, Owings & Merrill*

Alors que les modeleurs s'installent dans le paysage architectural au cours des années 1990, les praticiens construisent des relations variées à ces outils numériques. Les modeleurs occupent une place plus ou moins importante dans le processus de conception. Il y a donc des nuances dans la relation aux modeleurs, dans la manière des praticiens de se l'approprier, à partir d'une même ambition de formalisation et de constructibilité des projets conçus. Ces nuances s'observent aussi dans la place réservée aux logiciels en provenance du champ de l'animation, que tout le monde ne garde pas comme logiciel de référence pour ses travaux de modélisation. Néanmoins, Maya constitue le point de départ du travail pour beaucoup de ces architectes, même si cela ne reste pas le logiciel en usage au sein de toutes les agences. Or Maya et les techniques de modélisation issues du champ de l'animation contribuent à ouvrir aux architectes un champ formel nouveau, même s' il n'est pas exploité et mis en scène par tous. Cela contribue à populariser cette approche de la modélisation, plutôt que l'approche proposée dans les logiciels de modélisation pour la manufacture. La liberté géométrique que permet cette approche, ou les contraintes physiques sont moindres que dans son alternative, donne naissance à d'autres expérimentations architecturales, qui vont encore plus loin. C'est le cas par exemple des travaux de Marcos Novak, architecte vénézuélien qui s'intéresse au cyberespace et à l'architecture virtuelle qui peut y prospérer. Novak décrit cette architecture comme

extraterrestre, affranchie de toute contrainte et de toute habitude que nous en avons dans le monde physique[124]. Cet affranchissement, qui s'appuie sur les géométries rendues possibles par le recours aux logiciels d'animation, se retrouve dans ses propositions architecturales. Comme d'autres, Novak s'empare de l'objet géométrique qu'est la surface pour mieux décrire ses ambitions. Plus exactement, de l'*hypersurface* : un objet qui se transforme sans arrêt - une transformation permanente qui est la continuité tant du travail d'animation que des idées proposées par Deleuze dans son ouvrage Le Pli. Cette transformation permanente est une idée que l'on retrouve aussi chez Lars Spuybroek et Kas Oosterhuis, deux architectes néerlandais dont les projets explorent la notion d'interactivité avec le bâtiment. Les *hyperbodies* de Kas Oosterhuis se transforment en permanence, mais en fonction de l'utilisateur, de ses besoins et de ses déplacements[125]. Cette transformation permanente est rendue possible dans le cyberespace, et dans le monde physique par les extensions de la modélisation: réalité virtuelle, réalité augmentée ou projections, comme en témoigne les projets jumeaux de Kas Oosterhuis et Lars Spuybroek pour le pavillon de l'eau, deux structures aménagées sur l'île de Neeltje Jans en 1997, pour sensibiliser les visiteurs à la place de l'eau dans nos écosystèmes.

[124] Marion Roussel, "A la couture des mondes… Transarchitecture et hypersurfaces : une introduction" https://dnarchi.fr/analyses/a-la-couture-des-mondes-transarchitecture-et-hypersurfaces-une-introduction/, consulté le 30 Novembre 2021.
[125] Kas oosterhuis, *HYPERBODY, First Decade of Interactive Architecture*, Jap Sam Books, 2012.

*Figure 17. AlloExoBio, Marcos Novak*

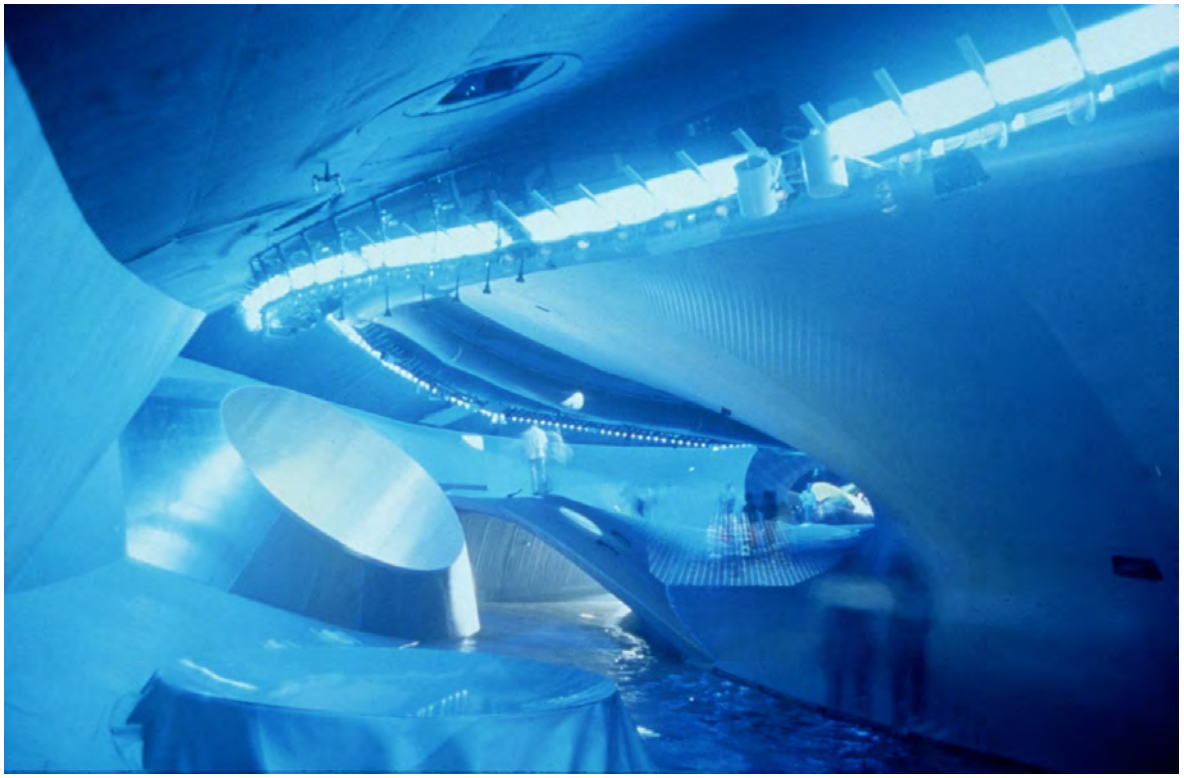


*Figure 18. a. Saltwater Pavilion, ONL ; b. FreshWater Pavilion, NOX*

Justement, les modeleurs et leurs extensions sont avant tout des outils de représentation. Si ce sont bien des outils numériques, leurs interfaces font passer les algorithmes qui leur permettent de fonctionner au second plan, les éloignant des utilisateurs. La chronologie de l'infographie nous a permis de voir les développements techniques qui ont permis l'avènement des modeleurs, mais aussi comment une transformation s'est peu à peu faite de recherches basées sur l'écriture de programmes

informatiques à la manipulation de logiciels de modélisation. Cette transition n'est pas sans conséquences sur la manipulation des outils. En donnant l'impression à travers la structure de l'interface logicielle et à travers celle de l'interface physique qu'il s'agit de dessin sur écran plutôt que de la formulation d'instructions de calculs, les modeleurs cachent les instructions qui président pourtant à leur fonctionnement et à la modélisation d'objets. Perdre de vue ces algorithmes contribue ensuite à oublier l'existence des kernels géométriques, qui diffèrent d'un modeleur 3D à un autre et engendrent des différences dans les géométries possibles avec ces modeleurs. L'interface des modeleurs contribue ainsi à la création de biais techniques et de boîtes noires dans leur manipulation. Ces nouveaux outils numériques font donc apparaître un biais de représentation : les algorithmes sur lesquels repose la modélisation sont dissimulés par les explorations géométriques que le logiciel permet, et les processus procéduraux au sens large que ces algorithmes permettent aussi d'explorer sont eux aussi relégués en arrière. La popularisation de la modélisation au cours des années 1990 dans l'industrie de la construction n'est donc pas suivie par celle des outils algorithmiques. Au contraire, une bifurcation se crée entre les deux familles d'outils au sein du champ.

## Conclusion

### Une filiation dans le champ de l'animation

Cette bifurcation a lieu entre modélisation et programmation, et entre recherches et applications dans l'industrie. Mais nous l'avons vu, la conception architecturale peut avoir recours à deux types de modeleurs différents, ceux issus de l'animation et ceux issus de la manufacture. S'il s'agit bien pour ces deux types de logiciels de dessin assisté par ordinateurs, ceux de la manufacture s'ancrent dans un autre paradigme de modélisation que ceux issus de l'animation : le *solid modelling* (modélisation solide). Cette approche met l'accent sur la précision de la description des formes et la fidélité de la modélisation aux caractéristiques physiques de l'objet représenté. En effet, puisque ce sont des logiciels pour la manufacture, l'objectif est de fabriquer l'objet, et donc de pouvoir évaluer avec certitude la possibilité de sa fabrication. Pour les mêmes raisons, les objets modélisés sont de plus des solides et non des surfaces, c'est-à-dire des objets fermés : la description et la correction des frontières de ces objets est donc également très précise. Les logiciels basés sur le principe de la modélisation solide fonctionnent de plus avec des attributs assignés aux différents objets, c'est-à-dire des informations à leur sujet. L'utilisateur peut indiquer des informations relatives à l'objet lui-même, comme sa dimension, mais aussi des informations relatives aux relations entre l'objet et d'autres objets. Ceci permet de créer des objets paramétriques dans le sens ou ces attributs peuvent être modifiés si besoin. Si l'objet a une hauteur, une largeur et une longueur assignées, modifier l'une fera changer l'objet. Si l'objet est construit en relation avec d'autres objets, et que ces autres objets changent de dimension, alors il s'adapte également. Cette série de logiciels est également dotée d'un historique de modélisation qui permet d'enregistrer l'ensemble des actions effectuées, et

éventuellement de les répéter. Les logiciels issus de la manufacture offrent donc nombre de possibilités auxquelles les architectes ne s'intéressent que bien après le développement de ces logiciels pour leur marché initial. Au vu de ces caractéristiques, pourquoi les architectes ont-ils un temps largement privilégié les outils de dessin assisté par ordinateur issus du champ de l'animation ? D'abord les kernels géométriques ne sont pas basés sur les mêmes principes dans les deux séries de logiciels. Les logiciels pour l'animation privilégient un kernel géométrique qui leur permet de modéliser et manipuler des formes complexes, quand ceux pour la manufacture privilégient des standards de fabrication qui demandent au contraire des jeux de formes moins délicats. Par exemple, les logiciels pour l'animation doivent permettre d'animer des cheveux ou de modéliser les plis d'un tissu, ce qui requiert des caractéristiques géométriques qui ne sont pas nécessaires pour fabriquer des pièces de moteur. Les modeleurs issus de l'animation offrent donc un nouvel horizon formel aux architectes. Les caractéristiques de leur interface privilégient aussi les logiciels d'animation, plus faciles à utiliser avec leurs courbes déformables intuitivement que les logiciels de la manufacture, avec leurs arbres de relations entre éléments. Les objets peuvent presque être sculptés directement sur écran, plutôt que d'établir une longue liste d'instructions de construction et de transformations géométriques. Enfin, les possibilités de visualisation offertes par les logiciels d'animation pèse sans doute également : ils permettent d'obtenir des représentations beaucoup plus riches et réalistes que leurs contreparties du domaine de la manufacture, un atout incontestable pour les architectes qui souhaitent donner à voir leurs projets.

La richesse de ces représentations contribue aussi à populariser le recours aux modeleurs issus de l'animation. Cette période est en effet aussi celle d'architectes prescripteurs, comme Greg Lynn, Zaha Hadid ou Frank Gehry, dont la notoriété contribue à établir la complexité formelle comme une caractéristique de l'architecture numérique. De plus, des écrits offrent une proposition théorique architecturale en relation avec cette complexité : c'est le cas des textes de Gilles Deleuze sur le pli, des textes de Marcos Novak ou de Kas Oosterhuis sur les hypersurfaces, ou des textes de Bernard Cache sur le non-standard. Le non-standard et les machines à commande numérique établissent un processus de fabrication qui non seulement offre un rôle intéressant à cette diversité de forme, mais garantit qu'il est possible de les fabriquer. Théoriser le processus de fabrication théorisé à l'aide de la notion de non standard fait de plus émerger l'idée d'une chaîne de production numérique totale, continue du début de la modélisation jusqu'à la fabrication, elle aussi guidée numériquement, de l'objet. Cette idée devient un premier marqueur essentiel dans le champ computationnel. Le second, qui émerge également à travers le développement et le recours aux modeleurs, c'est l'impératif de faire ses propres outils. Par la suite, développer ses propres outils conserve un sens très large dans le champ : algorithmes, plug-ins, logiciels, robots, outils de fabrication robotique - la notion d'outils recouvre toutes ces possibilités sans distinction, mais l'impératif demeure très fort. Or, nous avons vu avec SOM que faire ses propres logiciels, ses propres modeleurs intégralement est un processus long

et lourd à porter pour une agence. Faire ses propres algorithmes l'est moins, et l'impératif se maintient donc plus facilement dans les activités de programmation que dans celles de modélisation. Or Maya, qui provient de l'industrie de l'animation, offre un autre avantage : MEL, le langage de programmation qui y est associé, est bien plus facile d'utilisation que les langages associés aux logiciels de modélisation de la manufacture. Faire ses propres algorithmes est donc plus simple au sein de Maya, d'où un ancrage dans l'animation plutôt que dans la manufacture renforcé. Pendant que le champ de l'intelligence artificielle est confronté à des difficultés significatives, celui de l'animation décolle. Le champ computationnel y construit donc un second ancrage. Et malgré la bifurcation technique qui se produit ensuite, la période de chevauchement entre programmation et modélisation dont nous venons d'examiner les production débouche cependant sur des marqueurs que le champ computationnel va conserver bien qu'il se concentre plus sur la programmation que sur la modélisation.

# CHAPITRE IV.
# -
# Un savoir-faire propre au champ

## 4.1. Un réseau à l'équilibre

### *4.1.1 Evolution du réseau*

Après le ralentissement qu'ont vu les décennies précédentes, causé tant par l'hiver procédural que par la prise d'importance du dessin assisté par ordinateur, le champ computationnel connaît de nouveau un fort développement à partir de la fin des années 1990. Le travail autour du non-standard et de la réalité augmentée contribuent à redorer le blason des expérimentations architecturales autour du numérique. La présence de plus en plus incontournable d'ordinateurs dans tous les domaines contribue également à normaliser l'idée qu'il s'agit d'un domaine digne d'exploration. L'arrivée à maturité d'une nouvelle génération de praticiens contribue également fortement à cette croissance renouvelée. Pour ces praticiens, nés dans la seconde moitié des années 1960 ou la première moitié des années 1970, l'ordinateur n'est plus une technologie de pointe encore en développement dans une poignée de centres de recherche, mais un outil bien plus facilement à disposition. Ces praticiens sont formés dans les pôles historiques du réseau, et à mesure que leur carrière progresse, ils essaiment dans d'autres universités. Ils promeuvent ainsi le recours aux outils algorithmiques à une bien plus grande échelle que jusqu'alors. Le réseau voit donc à la fois les pôles existants se développer pour accueillir des étudiants et des chercheurs de plus en plus nombreux, et de nouveaux pôles émerger.

Le pôle londonien est déjà depuis ses débuts un pôle multi-têtes, composé de plusieurs institutions majeures qui collaborent régulièrement[1]. La majeure partie des recherches se concentre à Londres, autour de trois écoles : l'Université d'East London (UEL), la Bartlett School of Architecture et l'Architectural Association (AA). Paul Coates, qui mène ses recherches au sein de l'UEL depuis les années 1970[2], y forme successivement Robert Thum (diplômé en 1994), Pablo Miranda Carranza (2000) et Christian Derix (2001). Tous trois restent ensuite à ses côtés, formant ce qui prend alors le nom de Centre for Evolutionary Computing in Architecture (CECA). A la mort de Paul Coates en 2013, c'est Christian Derix qui prend la direction du CECA, poursuivant les recherches dans la continuité de ce que son mentor avait initié dans les années 1970. La décomposition de l'espace en éléments ensuite recombinés à l'infini se poursuit, tirant parti de l'évolution des technologies

[1] Voir Chapitre II.
[2] Voir Chapitre II.

numériques pour visualiser les résultats produits en 3D et pour combiner de bien plus grands nombres d'éléments. Au gré des choix algorithmiques, hexagones, cubes et particules s'empilent, s'envolent ou se drapent de surfaces courbes rendant hommage aux blobs de Greg Lynn[3] (figure 2). Au fil des expérimentations, les chercheurs visent toujours à trier les règles de composition les plus efficaces de celles sans guère de potentiel, pour garantir que les programmes créent des espaces de qualité. Derix est par ailleurs chargé en 2004 par Aedas, une agence d'architecture britannique parmi les plus importantes du monde, d'animer un pôle R&D. Il y développe des programmes variés en soutien aux activités de l'agence, en particulier des outils de conception volumétrique à l'échelle urbaine, notamment avec l'aide de Pablo Miranda Carranza. A la Bartlett, Bill Hillier poursuit jusqu'à sa mort en 2019 ses recherches sur la space syntax[4], les appliquant à l'échelle urbaine et articulant une théorie de l'architecture autour[5]. Laura Vaughan, formée par Bill Hillier, prend sa suite à la tête du Space Syntax laboratory, qui anime un master dédié et participe également au master Architectural Space & Computation. L'école propose en complément de ces deux formations trois masters autour des questions computationnelles : l'un spécialisé dans le bio-design, le deuxième dans les processus de fabrication numérique et le troisième dans les théories du numérique. La Bartlett compte en dehors du Space Syntax laboratory une poignée de praticiens du champ computationnel. Marcos Cruz et Marjan Colletti, tous deux diplômés de l'école, y poursuivent leur carrière en explorant le rôle des outils computationnels dans le bio-design. Phil Ayres, Nick Callicott, Emmanuel Vercruysse et Bob Sheil sont eux aussi formés à l'école, puis exercent ensemble, tout en enseignant notamment à la Bartlett. Bob Sheil en particulier y occupe diverses positions jusqu'à devenir directeur de l'école, et cofonder Fabricate, série de conférences dédiées à la fabrication robotisée.

[3] Paul Coates, Tom Appels, Corinna Simon et Christian Derix,'Current work at CECA', *Proceedings of the 4th Generative Art Conference (GA2001)*, Milan: Generative Design Lab Milan Polytechnic University, Italy, 2001.
[4] Voir Chapitre II.
[5] Bill Hillier, *Space is the Machine: A Configurational Theory of Architecture*, Cambridge University Press, 1996.


*Figure 1. Réseau de la troisième génération du champ computationnel*

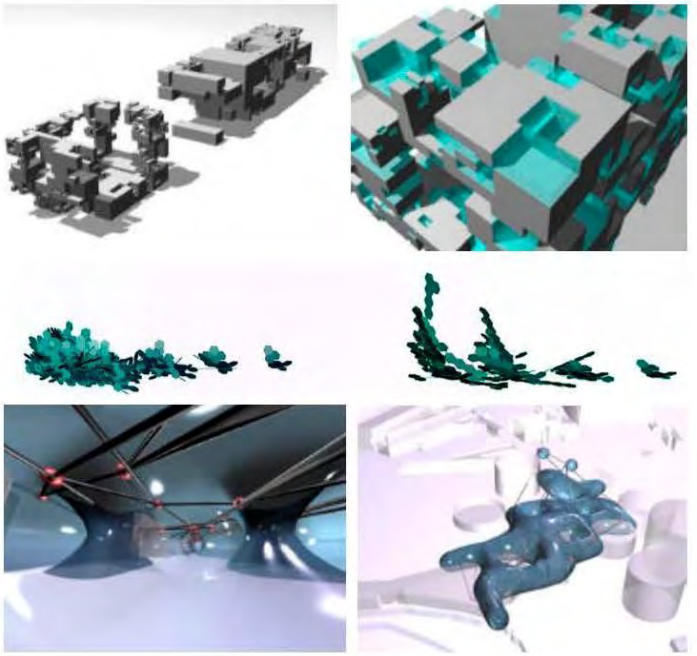


*Figure 2. Productions de l'University of East London*

Quant à la AA, au sein même de l'école, différents espaces de recherche se développent. Les unités de diplôme (*diplomas units*), réservées aux élèves de master, étaient jusque là le lieu d'exercice privilégié du champ computationnel dans l'école. Celles-ci sont adoptées par l'école dans les années 1960 et prennent leur forme actuelle dans les années 1980[6]. Les succès critiques de Rem Koolhaas, de Bernard Tschumi ou de Zaha Hadid, anciens étudiants de l'école, apportent beaucoup de crédibilité à l'école et à son système d'unités, alors unique. L'objectif des unités est d'offrir une grande liberté aux élèves, qui ont le choix entre une grande diversité de spécialités et de manières d'envisager l'architecture. Cette diversité est supposée encourager un esprit critique chez les étudiants, mais donne également beaucoup de liberté aux enseignants dans la structuration des cursus des unités. Pendant un temps, aucun d'entre eux n'a de poste fixe : la direction des unités et les cours sont assurés par un kaléidoscope de profils divers, en fonction des besoins de l'instant[7]. C'est ce cadre qui permet à Gordon Pask d'intervenir ponctuellement sans véritable poste d'enseignant, mais aussi à John Frazer d'officier au sein de la Diploma Unit 11 à partir de 1989, et d'y développer comme bon lui semble ses architectures évolutionnaires. Au départ de John Frazer en 1996, c'est Michael Hensel, lui-même ancien étudiant de la AA, qui prend la relève au sein de la Diploma Unit 4, qui poursuit les ambitions

[6] Irene Sunwoo, "From the "Well-Laid Table" to the "Market-Place:" The Architectural Association Unit System", *Journal of Architectural Education*, 65: 24-41

[7] C'est toujours le cas en théorie, même si de nombreux enseignants voient leurs postes renouvelés d'une année sur l'autre.

de Frazer, sous le nom d'*architecture morphogénétique* cette fois. Michael Hensel est ensuite rejoint par Achim Menges en 2003, juste après l'obtention de son diplôme de l'école. Entre 2002 et 2008, George Legendre est également directeur d'une unité, la Diploma Unit 5, où il explore surtout des géométries paramétriques avec ses étudiants. Enfin, Christopher Lee et Sam Jacoby, eux aussi formés à la AA, dirigent entre 2004 et 2006 la Diploma Unit 6, où ils se penchent sur l'utilisation d'outils algorithmiques pour la conception urbaine.

Les activités du champ computationnel à la AA se déplacent néanmoins par la suite peu à peu vers deux entités : le AA Design Research Laboratory (AADRL) et le programme EmTech. Le premier est lancé en 1997 par Patrik Schumacher. Celui-ci, ancien étudiant de la AA, a été recruté dès le master par Zaha Hadid et est ensuite rapidement devenu son bras droit au sein de l'agence. Il est crédité de l'implémentation des pratiques de modélisation 3D et des pratiques computationnelles au sein de l'agence, pratiques qui sont désormais au cœur des activités de l'agence. Le DRL est un programme de master professionnel qui fait aussi office de laboratoire, puisque les étudiants y sont poussés à développer des projets de recherche dans le cadre de la seconde phase du cursus. Theodore Spyropoulos, de retour après les années d'exercice dans d'autres institutions qui ont suivi son diplôme à la AA en 2001, prend la direction du groupe en 2005 après y avoir exercé deux ans en tant que tuteur. Shajay Bhooshan, lui aussi diplômé de la AA et qui a cofondé le groupe spécialisé dans les pratiques computationnelles CODE chez Zaha Hadid, rejoint l'équipe en 2013. Chacun des trois praticiens anime aujourd'hui un studio distinct au sein du DRL. Patrik Schumacher y développe *Parametric Semiology*, un projet à long terme dont l'objectif est d'intégrer dans les pratiques computationnelles des mesures de l'apport social de l'architecture[8]. Le studio développe des outils de simulation du comportement humain dans l'espace, que ce soit à l'échelle de la ville ou à celle de l'interaction d'un corps humain avec du mobilier. Les étudiants mettent ensuite ces outils au service de la conception de vastes projets architecturaux dont les géométries courbes ne sont pas sans rappeler celle de l'agence Zaha Hadid Architects (Figure 3). Theodore Spyropoulos fait également appel à des simulations de comportements, mais aussi à un vaste ensemble de ressources électroniques diverses. Le tout est mis au service de la conception d'environnements adaptatifs, constitués d'une multitude de petits robots ou de cloisons articulées, dans la lignée des productions de son agence Minimaforms. Enfin, Shajay Bhooshan propose un programme d'enseignement et de recherche autour de la construction robotique modulaire, explorant le potentiel de la production non-standard et de la customisation de masse pour l'industrie de la construction. Michael Hensel et Achim Menges, rejoints

[8] Patrik Schumacher, "Parametric Semiology – The Design of Information Rich Environments", dans Pablo Lorenzo-Eiroa et Aaron Sprecher (eds), *Architecture In Formation – On the Nature of Information in Digital Architecture*, Routledge, Taylor and Francis, New York, 2013. Nous reviendrons plus en détail sur ce projet au Chapitre V.

par Michael Weinstock, lui aussi ancien étudiant de la AA et enseignant à l'école de longue date, fondent en 2001, en complément du DRL, le programme Emergent Technologies (EmTech). Fonctionnant sur le même principe, il accueille lui aussi des professionnels cherchant à se former à la pratique computationnelle, et situe ses activités à cheval entre enseignement et recherche. Pendant des unités de diplôme où enseignent ses fondateurs, EmTech leur survit au fur et à mesure que ces dernières sont remplacées par d'autres, et est encore aujourd'hui héritier des recherches évolutionnaires et morphogénétiques des pionniers de la AA, comme en témoigne l'arrivée dès 2002 de Georges Jeronimidis, spécialiste du biomimétisme. Non seulement les étudiants se pressent dans les formations de la AA consacrées au champ computationnel, mais nombre d'entre eux y interviennent ensuite également en tant que tuteurs. Ils s'y font la main à la suite de leur formation, dans le cadre de leurs premières expériences d'enseignement avant d'essaimer ailleurs. On les retrouve à l'Université de Stuttgart, à l'Israel Institute of Technology ou encore à l'école d'architecture d'Aarhus. Ceci est permis grâce à la structure fluctuante des enseignements mais aussi par un certain entre-soi que la AA cultive, et qui lui permet de maintenir sa position dominante dans le champ computationnel.

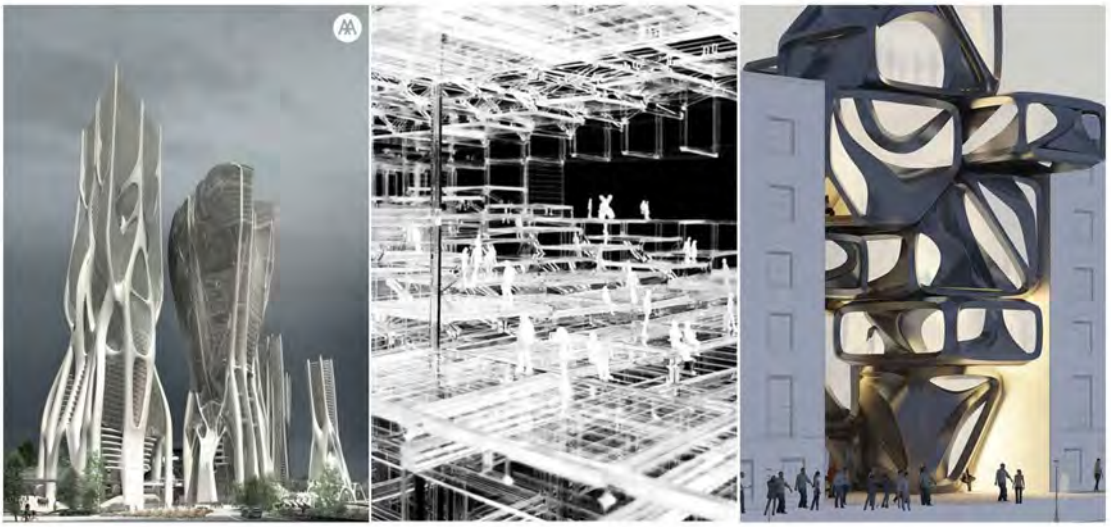


*Figure 3. Productions du AA Design Research Laboratory*

Les pôles américains du réseau privilégient un temps, plutôt que la création de groupes de recherche à part entière, l'intégration des pratiques computationnelles dans un cursus plus généraliste. Les différentes facultés d'architecture qui avaient accueilli les premiers praticiens du champ computationnel - Ohio State University, Georgia Tech, Carnegie Mellon, Berkeley - se remettent à

recruter des enseignants, et constituent des équipes plus conséquentes. Elles sont rejointes peu à peu par le reste des cursus architecturaux majeurs des Etats-Unis : le Rensselaer Institute à New York, Princeton University, le Pratt Institute, le GSAPP à Columbia, Cornell University ou encore le GSD à Harvard. Les universités se partagent les praticiens basés aux Etats-Unis les plus versés dans les pratiques computationnelles, et invitent régulièrement leurs homologues européens pour des séjours plus ou moins courts - jurys, workshops, séjours de recherche, semestres d'enseignements. Il s'agit donc d'une reprise du fonctionnement d'avant l'hiver procédural[9], mais avec beaucoup plus de praticiens. La deuxième vague des institutions qui se lancent dans l'intégration des pratiques computationnelles dans les programmes de leurs facultés d'architecture recrute par ailleurs des praticiens un peu plus jeunes, comme Matias del Campo à l'Université du Michigan, Dana Cupcova ou Joshua Bard à Carnegie Mellon. Si l'entre-soi est moins marqué qu'au sein de la AA, le réseau des institutions reste relativement restreint : les praticiens font leurs études et interviennent dans le même ensemble d'universités américaines, noyaux du pôle américain. Cette génération de praticiens voyage dans un premier temps beaucoup entre les différentes institutions des pôles du réseau, avant de peu à peu se fixer au fur et à mesure que ces institutions donnent de l'importance aux pratiques computationnelles et créent des postes dédiés. Leur carrières arrivent désormais à maturation, les plaçant en position de prendre la suite de figures comme William Mitchell, qui avaient en leur temps beaucoup contribué à intégrer dans le paysage universitaire architectural américain les outils informatiques.

Comme Bob Sheil à la Bartlett, Hernan Diaz Alonso, dont l'agence Xefirotarch est un des fers de lance des pratiques computationnelles aux Etats-Unis, accède à la direction de Sci_Arc en 2015, après y être arrivé en 2001 comme enseignant et avoir fait partie de la direction des programmes depuis 2007. Au fil de ses mandats, il redessine les programmes de l'école en intégrant largement au cursus le recours aux outils algorithmiques. En témoignent les cours disponibles des *studios de design computationnel* de licence au master professionnel Architecture Technologies, et les membres des équipes pédagogiques, nombreux à avoir un parcours au sein du champ computationnel. Casey Rehm, qui dirige Architecture Technologies, est diplômé de Carnegie Mellon et de Columbia, a travaillé chez kokkugia et dans plusieurs universités américaines, et compte parmi les premiers praticiens qui ont beaucoup utilisé Processing pour modéliser des systèmes d'agents. Elena Manferdini, qui dirige les programmes de licence, est auparavant passée par UCLA ainsi que par le studio de Greg Lynn. Peter Testa et Devyn Weiser, formés respectivement au M.I.T et au GSAPP de Columbia, ont animé pendant plusieurs années le groupe de recherche Emergent Design Group au M.I.T. Ils s'y associent avec Una May O'Reilly, spécialiste des algorithmes génétiques avec laquelle ils s'investissent eux

[9] Voir Chapitre II. Les stratégies de carrière décrites dans le Chapitre II sont aussi de nouveau à l'œuvre.

aussi dans des recherches sur les architectures évolutionnaires, s'inspirant de John Frazer. Leur agence commune, Testa & Weiser, créée en 2002, est l'une des premières à se spécialiser dans les pratiques computationnelles. A Sci_Arc, Devyn Weiser coordonne de 2009 à 2013 les programmes de licence et Peter Testa fait partie du corps enseignant, de même que Tom Wiscombe, formé à Berkeley et UCLA et passé par chez Coop Himmelb(l)au. Le choix d'intégrer les pratiques computationnelles à un cursus généraliste peut donc tout de même teinter fortement l'ADN de l'école, comme c'est le cas pour Sci_Arc, dont les projets se placent toujours quelque part entre l'héritage déconstructiviste américain, les complexités formelles permises par les modeleurs 3D et le recours aux outils algorithmiques.

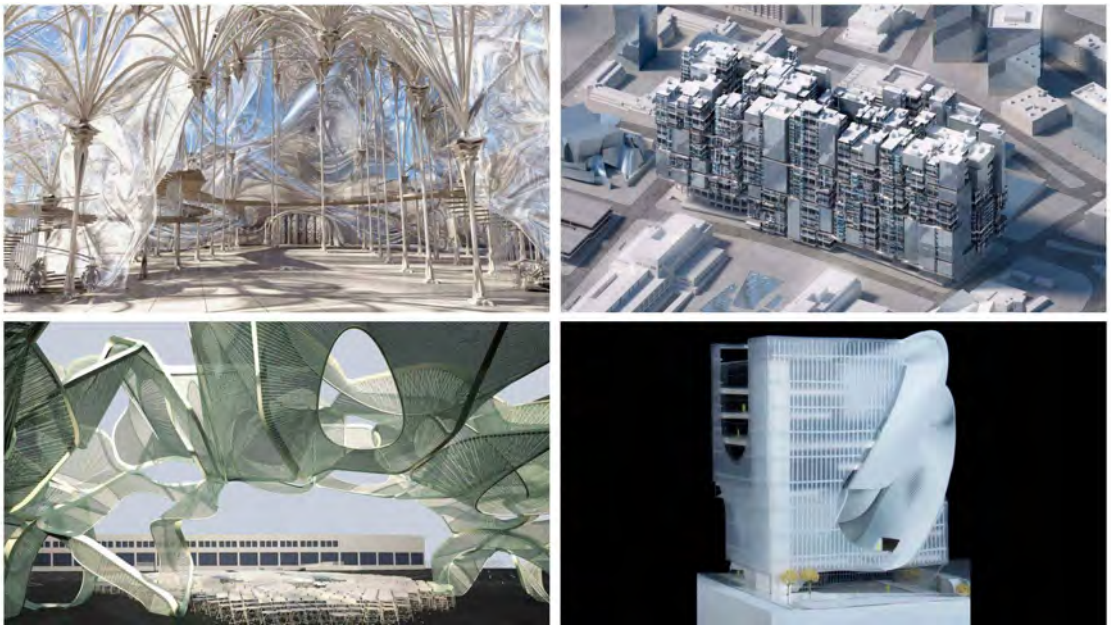


*Figure 4. Productions de Sci_Arc*

Un peu plus tard émergent également dans l'écosystème américain des lieux dont l'activité est intégralement consacrée à la conception computationnelle. Au Medialab, le M.I.T crée en 1996 un premier groupe, Architecture Representation Computation, qui devient plus tard le Design and Computation Group, dirigé par Takehiko Nagakura. Au fil des années, de nombreux autres groupes dédiés à divers aspects du champ sont ajoutés au sein du medialab. Ils y partagent l'espace avec de nombreux autres groupes dont le travail porte sur des sujets très différents, avec pour mission de mettre l'innovation technologique au service des objectifs les plus divers, depuis la conquête spatiale

jusqu'à l'opéra du futur[10]. Les pratiques computationnelles s'y combinent à l'exploration de sujets les plus divers, pour des applications parfois encore lointaines. Là où le Design and Computation Group s'intéressait au recours aux outils algorithmiques de manière très large, ces nouveaux groupes se spécialisent. Au Self-Assembly Lab, Skylar Tibbits et son équipe s'intéressent à la "*matière programmable*"[11], développant des textiles s'adaptant à la chaleur ambiante, cloisons déployables et systèmes modulaires se déformant en fonction des conditions climatiques. Le Design and Fabrication Group, mené par Larry Sass, s'intéresse au flux de travail permis par les outils de conception et fabrication numérique, extrayant directement de modèles 3D des parcours-outils, dans la lignée des travaux de Bernard Cache et Patrick Beaucé sur le non-standard. Le Mediated Matter Group, fondé en 2010 et dirigé par Neri Oxman, est aujourd'hui le plus connu des groupes de recherche consacré aux pratiques computationnelles. Ses membres s'inspirent de diverses observations de la nature pour élaborer des géométries, des outils et des processus de fabrication. Leur pratique s'accompagne d'une théorisation d'une "*écologie matérielle*"[12], où les pratiques computationnelles permettent d'orchestrer une continuité entre les objets, les bâtiments, et les écosystèmes qui les accueillent. A la suite du M.I.T, mais aussi des universités européennes qui créent des groupes de recherche dédiés aux questions computationnelles en architecture, les universités américaines du réseau établissent elles aussi, enfin, des groupes spécifiques, sans plus se contenter d'intégrer ces questions à leurs cursus d'enseignement. Ainsi apparaissent le groupe CODE à Carnegie Mellon, le Sabin Design Lab à Cornell University, ou le Stuckeman Center for Design Computing à Penn State.

Les pôles existants continuent de se développer dans la lignée de leurs fondateurs, tout en prenant de l'ampleur. Par ailleurs, de nouveaux pôles apparaissent également dans le réseau. En Allemagne, d'où sont originaires un certain nombre des praticiens qui évoluent à la AA ou aux Etats-Unis, plusieurs universités voient revenir d'anciens étudiants, qui commencent à y installer les pratiques computationnelles. Patrik Schumacher et Achim Menges ont fait leurs études respectives d'abord à Stuttgart et à Darmstadt, en Hesse, avant de s'inscrire à la AA. Quelques décennies plus tôt, l'HfG Offenbach avait déjà accueilli les premières réflexions de la Design Research Society à l'occasion de conférences[13] et l'Université de Stuttgart celles de Frei Otto. Le bureau d'études Bollinger +

---

[10] Pour la liste des différents groupes, voir https://www.media.mit.edu/research/?filter=groups (consulté le 17/08/2021). Pour l'opéra du futur, voir https://www.media.mit.edu/groups/opera-of-the-future/overview/ (consulté le 17/08/2021).

[11] "programmable matter" en anglais. Skylar Tibbits, *Self-Assembly Lab: Experiments in Programming Matter*, Taylor & Francis, 2016.

[12] "*material ecology*" en anglais, défini comme suit : "*Notre domaine de recherche, intitulé "écologie matérielle", intègre des stratégies informatiques de recherche de formes à une fabrication inspirée de la biologie*.". Texte original : "*Our research area, entitled Material Ecology, integrates computational form-finding strategies with biologically inspired fabrication*." https://mediatedmattergroup.com/about (consulté le 17/08/2021).

[13] Voir Chapitre II.

Grohmann, dont une grande partie de l'activité s'appuie sur le recours à des outils computationnels, se trouve également dans la région, à Francfort. Si Schumacher continue d'exercer à la AA, Achim Menges revient dans la région après avoir passé le début de sa carrière à la AA. Après trois ans à l'HfG Offenbach, il fonde en 2008 à l'université de Stuttgart l'Institute for Computational Design and Construction (ICD), qui s'impose très vite comme une unité majeure du réseau. Plus récemment, d'autres universités se lancent aussi : la Digital Design Unit (DDU) fondée à l'Université de Darmstadt en 2015 par Oliver Tessmann, auparavant passé chez Bollinger + Grohmann, continue à densifier le réseau en Hesse. La région accueille aussi l'équipe de l'Experimental and Digital Design and Construction (EDEK) à l'Université de Kassel. L'Autriche aussi voit un pôle se développer autour de la Angewandte à Vienne, qui accueille post-modernistes, déconstructivistes, et pionniers de l'architecture computationnelle dès le début des années 2000. Greg Lynn et Zaha Hadid y animent notamment des studios. Bollinger + Grohmann a par ailleurs également un bureau dans la ville. Des échanges nourris s'établissent entre Vienne et la Hesse, tout comme le pôle londonien voit la AA, la Bartlett et l'UEL se partager jurys, workshops et cours.

En Australie également se développe un nouveau pôle. Celui-ci n'est pas tout à fait nouveau puisque John Gero avait dès les années 1960 abordé les pratiques computationnelles dans son travail[14]. Mais les années 2000 voient une poignée de praticiens supplémentaires contribuer à l'établissement de ces pratiques dans un plus grand nombre d'institutions. La trajectoire de ces praticiens relie par ailleurs le pôle australien au reste du réseau du champ. Les parcours de Iain Maxwell, Dave Pigram, Robert Stuart-Smith et Roland Snooks en sont particulièrement représentatifs. Tous les quatres étudient d'abord à l'Australian National University (ANU), avant de partir suivre un master à la AA pour Maxwell et Stuart-Smith et à Columbia pour Pigram et Snooks. Les quatre praticiens partagent un intérêt pour la programmation dès l'ANU, ou le cursus dit pour l'*environnement bâti*[15] comprend une formation aux bases de la modélisation des phénomènes physiques entrant en jeu dans la conception de bâtiments. Leur séjour à la AA et à Columbia les expose aux pratiques computationnelles qui se développent alors dans ces deux pôles et les fait basculer vers une spécialisation dans le domaine. Les quelques années qui suivent les voient commencer à collaborer en deux binômes : Robert Stuart-Smith et Roland Snooks d'une part, Iain Maxwell et Dave Pigram de l'autre. Ces deux binômes donnent naissance à deux agences, kokkugia en 2004 et supermanoeuvre en 2006. En parallèle de premières expérimentations de programmation à leur compte, les quatre praticiens programment des outils pour d'autres agences du champ, comme biothing ou A_LA. Ils se forment également dans des agences plus classiques, comme Grimshaw ou Arup, où ils sont recrutés comme spécialistes de la

[14] Voir Chapitre II.
[15]En Australie, la licence d'architecture est dite pour l'environnement bâti (*Built Environment*) et regroupe des enseignements de formation à l'architecture, mais aussi de solides bases en génie civil.

programmation pour la conception architecturale. Leur activité d'enseignement démarre dans les institutions qui les ont accueillis pour leurs masters, puis dans diverses institutions du pôle américain du réseau : Penn, Sci_Arc, Pratt Columbia, Michigan. Enfin, les quatre reviennent prendre des postes en Australie, à l'University of Technology Sydney (UTS) pour Pigram et Maxwell, qui y enseignent ensemble en complément de leur pratique commune, et au RMIT pour Snooks et Stuart-Smith. supermanoeuvre et kokkugia développent au cours de leurs premières années d'activité des projets dont les algorithmes comptent parmi les plus complexes du champ. Leur pratique très prospective, nourrie de leurs expériences d'enseignement, et les bibliothèques qu'ils créent pour asseoir leur pratiques contribuent largement au développement d'un paysage algorithmique propre au champ computationnel en architecture. Tim Schork à l'UTS, Matthias Hank Haeusler et Nicole Gardner au sein de l'unité Code de l'Université de New South Wales et Mark Burry, qui a auparavant travaillé chez Arup et avec dECOI, et qui évolue avec Jane Burry entre le RMIT et l'Université de Swinburne à Melbourne, complètent le pôle australien. Ailleurs, enfin, le Center for Information Technologies in Architecture à Copenhague, au Danemark, l'Institute for Advanced Architecture of Catalonia en Espagne, mais aussi les activités dans le domaine à la Hong Kong University ou au Technion en Israël contribuent tous à l'extension du réseau du champ computationnel. Ces nouvelles unités vont à leur tour devenir le point de départ de pôles plus larges. CITA est ainsi rejointe au Danemark par l'Université d'Aarhus et l'Université du Danemark-Sud, tandis que les universitaires du Technion, notamment Eran Neuman, Rivka et Robert Oxman contribuent largement aux expositions et publications qui popularisent les pratiques du champ computationnel.

### *4.1.2 Diversification et popularisation du champ par le prisme d'AD Magazine*

Alors que dans les décennies précédentes, les pôles historiques du réseau offraient des espaces de pratique computationnelle en architecture au sens large, à partir des années 2000, les pôles se diversifient et se spécialisent progressivement. Les nouveaux pôles qui apparaissent dans le réseau assurent certes la formation computationnelle des étudiants des institutions au sein desquelles ils sont fondés, mais ils concentrent également leurs activités dans des sous-domaines du champ. CITA se penche sur le biodesign et ses matériaux, l'ETHZ sur la fabrication robotisée, l'ICD sur des sujets hybrides entre ces deux thèmes, avec pour objectif de parvenir à construire des structures biomimétiques à l'échelle architecturale. L'étude du magazine Architectural Design (AD)[16], particulièrement représentatif des pratiques computationnelles en architecture, et des sujets auxquels ses différents numéros sont consacrés, montre également l'émergence de sous-champs spécialisés[17].

[16] À ne pas confondre avec le magazine Architectural Digest.
[17] L'étude consacrée au magazine AD qui suit a été publiée dans l'article suivant : Nadja Gaudillière-Jami, "AD Magazine. Mirroring the Development of the Computational Field in Architecture 1965-2020", dans Brian

Fondé en 1930, il a depuis été continuellement édité, d'abord par Academy, puis par Wiley. Douze numéros par an ont été publiés jusqu'à ce que le magazine passe à six numéros par an en 1987, chaque numéro étant l'occasion pour AD d'inviter un rédacteur en chef à se pencher sur un enjeu de la production architecturale de l'époque. Le magazine AD a représenté pendant des décennies une référence clé pour de nombreux architectes, en particulier pour les développements technologiques de la discipline. Depuis les débuts du champ computationnel en architecture, il est également reconnu comme une référence pour ce domaine spécifique. Par rapport à d'autres titres de la presse architecturale, AD a l'avantage d'avoir un contenu plus approfondi que Architectural Digest, Dezeen ou Archdaily, avec un processus de sélection éditoriale des contenus plus rigoureux que ces deux derniers. Il constitue un intermédiaire entre les titres axés sur des contenus théoriques comme Log et d'autres axés sur les questions pratiques de la construction, comme Detail. En outre, la ligne éditoriale de AD présente un état d'esprit particulièrement ouvert à l'innovation par rapport aux autres titres du champ, et ce dès le début du tournant numérique. Au cours de cette période, le contenu de AD a légèrement évolué, mais reste constitué d'une sélection d'articles et de projets présentés par le rédacteur en chef invité, en rapport avec le thème du numéro, ainsi que d'une série de sections fixes avec des rédacteurs réguliers, tels que Craig Kellogg, Will McLean, Valentina Croci ou Neil Spiller. L'équipe éditoriale, dirigée par Helen Castle depuis 1999, a également évolué au fil des ans, et comprend depuis les années 2000 plusieurs figures clés du champ computationnel[18].

La figure 5 présente les thèmes des questions liées au numérique au fil des ans. Huit catégories ont été identifiées. Les numéros thématiques sur le biodesign traitent du processus de conception morphogénétique, du biomimétisme, des algorithmes évolutionnaires ou des biomatériaux. Douze d'entre eux ont été publiés au cours des vingt dernières années, par exemple le numéro de 2004 intitulé *Emergence : Morphogenetic Design Strategies*[19] ou celui de 2015 intitulé *Material Synthesis : Fusing the Physical and the Computational*[20]. Les numéros thématiques sur la rationalisation examinent les pratiques orientées vers la construction. Onze ont été publiés, par exemple le numéro de 2002 *Versioning : Evolutionary Techniques in Architecture*[21] ou celui de 2013 *The Innovation*

Slocum, Viola Ago, Shelby Doyle, Adam Marcus, Maria Yablonina, Matias del Campo (éds.), *Distributed Proximities : Proceedings of the ACADIA 2020 Conference*, Association for Computer Aided Design in Architecture, p. 150-160, 2021.

[18] Parmi les membres du comité éditorial pour la période : Will Aslop, Denise Bratton, Paul Brislin, Mark Burry, André Chaszar, Nigel Coates, Peter Cook, Teddy Cruz, Max Fordham, Massimiliano Fuksas, Edwin Heathcote, Michael Hensel, Anthony Hunt, Charles Jencks, Helen Castle, Jayne Merkel, Mark Robbins, Deborah Saunt, Leon van Schaik, Patrik Schumacher, Ken Yeang, et Alejandro Zaera-Polo.

[19] Michael Hensel, Achim Menges, Michael Weinstock (éditeurs), *Emergence : Morphogenetic Design Strategies*, Wiley & Sons, 2004.

[20] Achim Menges (éditeur), *Material Synthesis : Fusing the Physical and the Computational*, Wiley & Sons, 2015.

[21] SHoP/Sharples Holden Pasquarelli (éditeurs), *Versioning : Evolutionary Techniques in Architecture*, Wiley & Sons, 2002.

*Imperative : Architectures of Vitality*[22]. Les numéros théoriques s'intéressent aux clés globales possibles pour la compréhension du paradigme computationnel en architecture. Deux Readers et onze numéros ont été publiés, par exemple le numéro de 2011 *Mathematics of Space*[23] ou le numéro de 2019 *Discrete : Reappraising the Digital in Architecture*[24]. Les numéros consacrés à la fabrication examinent les nouvelles méthodes de fabrication, telles que les processus robotiques, ainsi que les flux de travail et les cadres de fabrication non standard. Un Reader et six numéros ont été publiés, par exemple le numéro de 2014 *Made by Robots*[25]. Les numéros consacrés à l'urbanisme traitent des questions relatives aux villes intelligentes et aux outils d'analyse et de conception pour la ville. Six numéros ont été publiés, par exemple le numéro de 2020 *Urban Futures : Designing the Digitalised City*[26]. Les numéros sur le thème de l'esthétique examinent le potentiel des techniques de conception informatique pour un renouvellement de la complexité formelle de l'architecture. Cinq numéros ont été publiés, par exemple le numéro de 2010 *Exuberance*[27]. Les numéros globaux (trois et un Reader) offrent une vue d'ensemble du domaine computationnel dans son ensemble au moment de la publication, et les numéros à thème (huit et un Reader) regroupent des sujets sans rapport avec les thèmes majeurs identifiés.

[22] Pia Ednie-Brown, Mark Burry, Andrew Burrow (éditeurs), *The Innovation Imperative : Architectures of Vitality*, Wiley & Sons, 2013.
[23] George L. Legendre (éditeur), *Mathematics of Space*, Wiley & Sons, 2011.
[24] Gilles Restin (éditeur), *Discrete : Reappraising the Digital in Architecture*, Wiley & Sons, 2019.
[25] Fabio Gramazio et Matthias Kohler (éditeurs), *Made by Robots: Challenging Architecture at a Larger Scale*, Wiley & Sons, 2014.
[26] Mark Burry (éditeur), *Urban Futures : Designing the Digitalised City,* Wiley & Sons, 2020.
[27] Hernan Diaz Alonso (éditeur), *Exuberance*, Wiley & Sons, 2010.

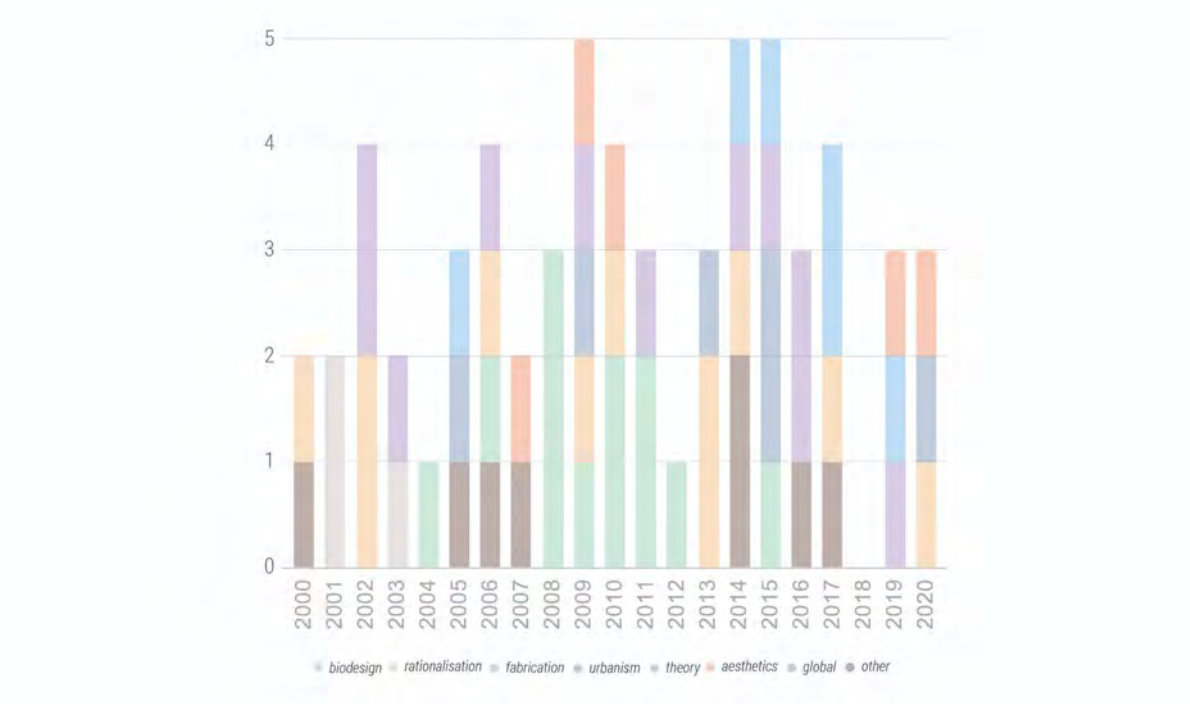



*Figure 5. Thèmes des numéros de AD sur la période 2000-2020*

La filiation entre générations dans le champ computationnel devient particulièrement claire avec les praticiens qui terminent leurs études et poursuivent leur parcours professionnel au début des années 2000. Neri Oxman fait ses études à la AA et travaille un temps chez KPF avant de faire son doctorat auprès de William Mitchell puis de fonder le Mediated Matter Group au M.I.T ; Christian Derix étudie avec Paul Coates avant de prendre sa suite à l'UEL ; nombre des enseignants du GSD, du GSAPP et des autres institutions américaines du réseau ont été formés dans l'une ou l'autre de ces mêmes institutions. L'entre-soi entretenu à la AA est particulièrement prégnant : les enseignants des unités computationnelles sont, on l'a vu, presque exclusivement des anciens élèves de l'école. Les praticiens du pôle allemand et du pôle australien du réseau sont eux aussi très nombreux à être passés à la AA, d'Achim Menges à Iain Maxwell ou Robert Stuart-Smith. L'étude des contributeurs d'AD montre aussi une forme d'entre-soi qui se perpétue au-delà des institutions. Ce n'est plus exactement le réseau de la AA, mais certains contributeurs sont particulièrement présents, quand d'autres praticiens de la génération qui n'ont pourtant pas moins contribué à l'extension du réseau et des pratiques sont quasiment absents du magazine. La figure 6 présente les institutions et les entreprises les plus souvent citées dans les numéros de 2000 à 2020 consacrés au design computationnel - de 3 mentions pour les cercles les plus petits à 111 pour le plus cité (la AA). Le tableau 2 montre les contributeurs les plus fréquemment invités, le nombre d'articles dont ils sont les auteurs dans des

numéros qu'ils n'ont pas édités et leurs antécédents. Le tableau 3 montre la répartition de tous les contributeurs dans les numéros à thème informatique. Le tableau 4 montre les éditeurs invités les plus fréquemment invités, ainsi que le nombre de mentions de projets qu'ils ont dirigés dans tous les numéros sur le champ computationnel, y compris ceux qu'ils ont eux-mêmes édités. Il montre que certains rédacteurs invités ont été invités pour plusieurs numéros, alors que d'autres praticiens ayant joué un rôle déterminant dans la constitution du domaine n'ont jamais été invités. Si l'on tient compte à la fois des numéros édités et des contributions, sur 905 contributeurs et éditeurs, 15 ont été invités pour un article important dans un numéro sur 10, et 3 ont été invités pour un article important dans un numéro sur 5. En outre, certains rédacteurs invités ont tendance à mettre en avant leurs propres œuvres ou celles de leurs proches collaborateurs dans les numéros qu'ils éditent, comme le montre la colonne du nombre de mentions. Ceci met en évidence l'existence d'un réseau AD de praticiens, ce qui laisse entendre que le fait d'être présenté dans le magazine AD ou d'être invité en tant que rédacteur en chef est guidé par un système de cooptation - un phénomène qui n'est pas rare dans le monde de l'édition et de la conservation, mais qui n'en souligne pas moins l'existence d'un certain entre-soi dans le champ computationnel.

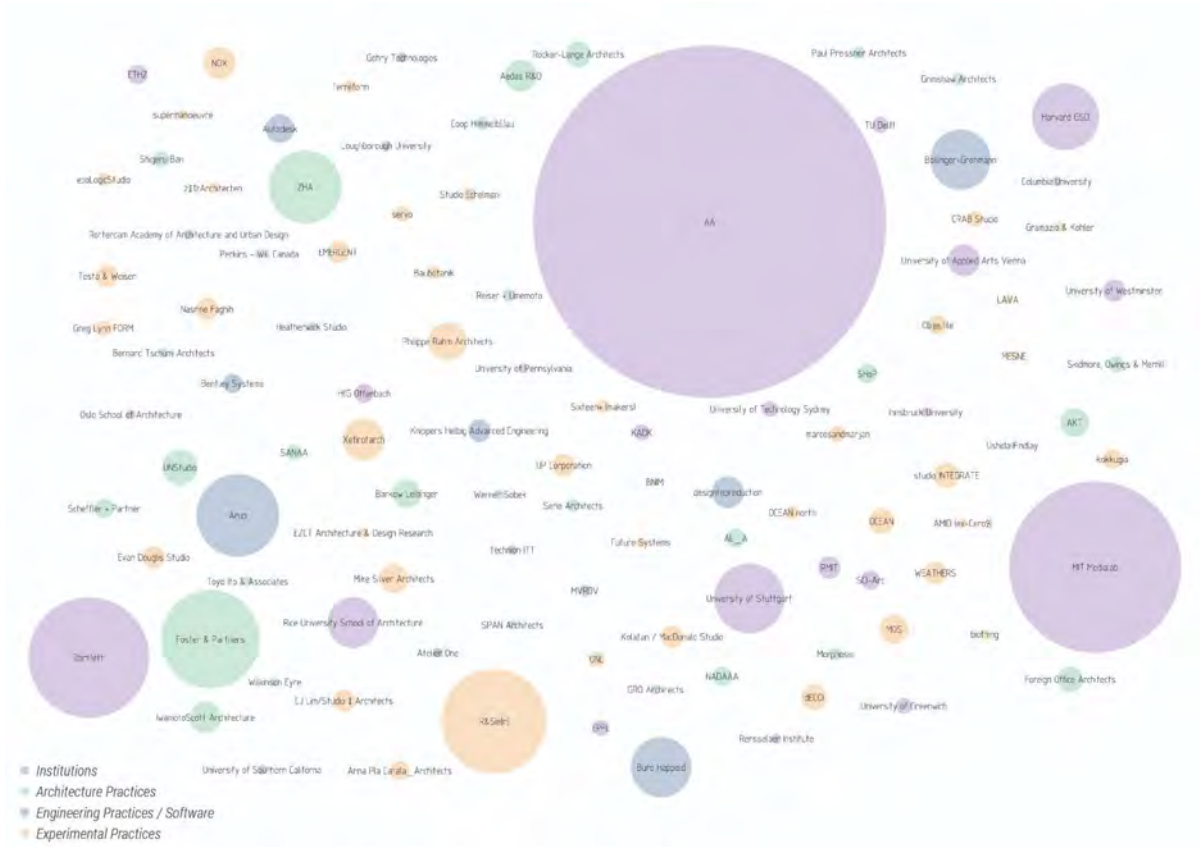



*Figure 6. Praticiens cités dans AD*

| Contributor | Number of contributions*** | Background |
|---|---|---|
| Mark Burry | 9 | Architect |
| Patrik Schumacher | 9 | Architect |
| Michael Weinstock | 8 | Architect / Academic |
| Greg Lynn | 7 | Architect / Academic |
| Neri Oxman | 7 | Academic |
| Ben van Berkel | 6 | Architect |
| Mario Carpo | 6 | Academic |
| Peter Cook | 6 | Architect |
| Francois Roche | 6 | Architect |
| Dennis Shelden | 6 | Architect / Academic |
| Benjamin H. Bratton | 5 | Academic |
| Manuel DeLanda | 5 | Academic |
| Hernan Diaz Alonso | 5 | Architect / Academic |
| Mark Garcia | 5 | Academic |
| Achim Menges | 5 | Architect / Academic |
| Antoine Picon | 5 | Academic |
| Philippe Morel | 5 | Architect / Academic |
| Leon van Schaik | 5 | Academic |
| Philip F Yuan | 5 | Architect / Academic |

| Number of contributions | Number of contributors |
|---|---|
| 1 | 735 |
| 2 | 97 |
| 3 | 31 |
| 4 | 11 |
| 5 | 10 |
| 6 | 7 |
| 7 | 3 |
| 8 | 1 |
| 9 | 2 |
| 10 | 3* |
| 11 | 1* |
| 16 | 2* |
| 19 | 1* |
| **Total contributors**** | 905 |

* Contributors for a fixed section
** Including contributors co-authoring papers
*** In issues other than the ones edited by the contributor

| Editor | Computational issues edited | Total number of issues edited | Number of mentions | Background | Part of the editorial board |
|---|---|---|---|---|---|
| Achim Menges | 7 | 7 | 51 | Architect / Academic | No |
| Michael Hensel | 5 | 7 | 36 | Architect / Academic | Yes |
| Neil Leach | 4 | 4 | 7 | Curator | No |
| Bob Sheil | 4 | 4 | 7 | Academic | No |
| Ali Rahim | 4 | 4 | 17 | Architect | No |
| Lucy Bullivant | 3 | 4 | 0 | Curator | No |
| Mark Garcia | 3 | 4 | 0 | Academic | No |
| Neil Spiller | 3 | 5 | 4 | Academic | No |
| Michael Weinstock | 3 | 3 | 28 | Architect / Academic | No |
| Mark Burry | 2 | 2 | 13 | Architect | Yes |
| Richard Garber | 2 | 2 | 3 | Architect | No |
| Christopher Hight | 2 | 2 | 6 | Designer / Academic | No |
| Hina Jamelle | 2 | 2 | 4 | Architect | No |
| Leon van Schaik | 2 | 2 | 3 | Academic | Yes |

L'étude du corpus AD permet par ailleurs de mettre en évidence la popularisation au cours de ces années des sujets liés au champ computationnel. Bien que l'intérêt pour les questions technologiques fasse partie de la ligne éditoriale du magazine dès le début, l'augmentation de la quantité de numéros dédiés aux questions computationnelles montre l'espace croissant qu'ils occupent dans le débat architectural. Le corpus étudié pour établir la base de données est constitué de 456 numéros d'AD, du n°1 de 1965 au n°6 de 2020 et de 8 numéros de la série AD Reader. Sur ces 464 numéros, 134 contenaient des articles pertinents pour l'étude proposée du champ computationnel en architecture. Sur la période 2000-2020, 126 numéros du magazine AD ont été publiés, ainsi que 8 numéros de la collection AD Reader. Ces numéros dédiés représentent donc à eux seuls près de la moitié des numéros d'AD contenant des articles en lien avec le champ computationnel depuis 1965. Parmi ces numéros, 62 ont été consacrés à des thèmes liés au tournant computationnel, ainsi que 4 Readers, comme le montre la figure 7. Cela représente près de la moitié des numéros d'AD sur la période, avec une légère augmentation au fil des années. Ceci souligne l'importance accordée par le magazine aux enjeux de la pratique computationnelle en architecture.

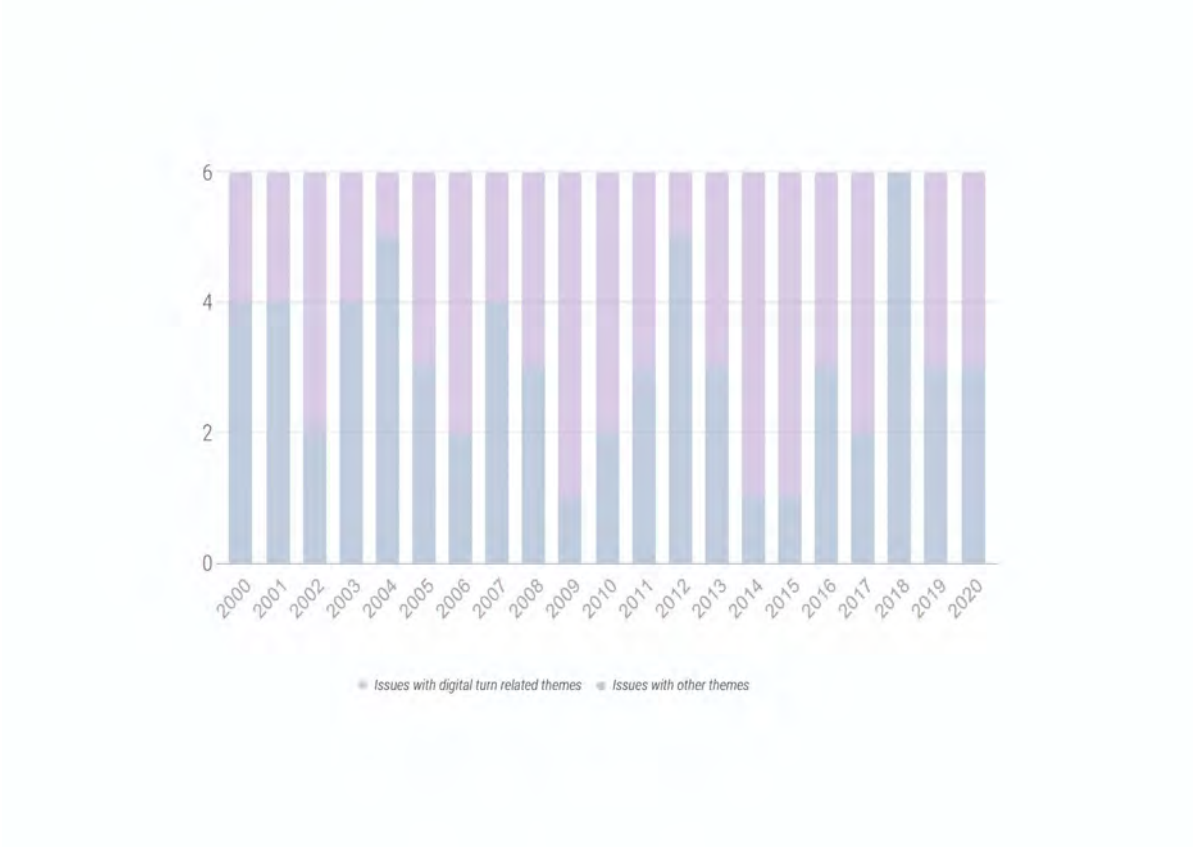



*Figure 7. Nombre de numéros consacrés à des thèmes du champ computationnel par an*

De nombreuses expositions dédiées à ces questions sont également organisées tout au long des années 2000 : *Blobjects & Beyond: The New Fluidity in Design*[28] au San Jose Museum of Art en 2005, *The Gen(h)ome Project*[29] au MAK de Los Angeles et *future city*[30] au Barbican en 2006, *scriptedbypurpose* [31] à la FUEL Gallery de Philadelphie en 2007, *Performalism*[32] au Tel Aviv Museum of Art en 2008.

Les thèmes de ces expositions sont d'abord à la frontière entre architectures expérimentales et digitales, et les praticiens du champ computationnel y partagent donc l'affiche avec divers déconstructivistes et postmodernistes, de Claude Parent à Archigram. Un glissement se fait ensuite vers des expositions exclusivement consacrées aux pratiques computationnelles. S'y retrouve toujours le même groupe de praticiens du champ computationnel, dont l'un ou l'autre en charge de la

[28] Steven Holt et Mara Holt Skov (éditeurs), *Blobjects & Beyond: The New Fluidity in Design*, San Francisco : Chronicle Books, 2005.
[29] Peter Noever, Kimberli Meyer et Open Source Architecture (éditeurs), *The Gen(h)ome Project,* Los Angeles : MAK, 2006.
[30] Jane Alison (éditrice), *Future city : experiment and utopia in architecture,* London : Thames & Hudson, 2006.
[31] https://scriptedbypurpose.wordpress.com/, consulté le 01 Septembre 2021.
[32] Yasha Jacob Grobman et Eran Neuman (éditeurs), *Performalism: Form and Performance in Digital Architecture*, London ; New York : Routledge, 2012.

scénographie voire du commissariat. Achim Menges, Michael Hensel, Aaron Sprecher ou Fabio Gramazio et Matthias Kohler y exposent leurs prototypes de toutes sortes, Theodore Spyropoulos ou Mark Goulthorpe leurs dessins et machines. Ces participations sont ensuite suivies dans les années 2010 par l'acquisition de pièces par le Centre Pompidou et le Centre Canadien d'Architecture, qui marquent une certaine reconnaissance de ces recherches. Le développement du pôle français, un peu plus tardif et un peu moins conséquent qu'en Australie ou en Allemagne, fait écho à cette popularisation. Une nuée d'expositions voit d'abord le jour, en écho aux événements internationaux organisés dans d'autres pôles du réseau. En 1999, Marie-Ange Brayer, conservatrice, et Frédéric Migayrou, critique d'art et conservateur, créent *ArchiLab*, une exposition qui a vocation à devenir régulière, au FRAC Centre, à Orléans, ou doivent être présentées les avant-gardes architecturales contemporaines. Huit éditions sont organisées entre 1999 et 2006, puis une neuvième en 2013, *Naturaliser l'architecture*[33]. En 2003, la maison de l'architecture de Marseille accueille *L'architecture au-delà des formes*[34]. Plus tard dans l'année, c'est au tour du Centre Pompidou d'accueillir *Architectures Non-Standard*[35]. En 2012, son catalogue est revisité à l'occasion d'une nouvelle exposition au FRAC Centre[36], qui accueille entre-temps au sein de ses collections nombre des travaux exposés jusqu'alors, devenant ainsi précurseur dans la conservation de ces architectures d'avant-garde. Le Centre Pompidou expose par ailleurs cette année-là *Multiversités Créatives*[37], qui présente les productions les plus récentes du champ computationnel. A la suite de cette exposition, le Centre Pompidou lance en 2017 le cycle *Mutations / Créations*, dédié aux pratiques computationnelles. La première édition s'intitule *Imprimer le Monde*[38], la seconde *Coder le Monde*[39], et la troisième *La Fabrique du Vivant*[40], chacune se penchant sur un aspect en particulier de ces pratiques, tout en cherchant à donner une perspective chronologique du développement de ces travaux.

Pour presque toutes ces manifestations, c'est le même groupe de praticiens et de commissaires qui se retrouve aux commandes : Frédéric Migayrou et Marie-Ange Brayer, en poste d'abord au FRAC Centre puis au Centre Pompidou, et l'agence EZCT Architecture & Design Research, dont l'un des fondateurs, Philippe Morel, a enseigné dans plusieurs institutions du réseau, en particulier à la Bartlett

---

[33] Marie-Ange Brayer et Frédéric Migayrou (éditeurs), *Naturaliser l'architecture : Archilab 9e édition*, Orléans : HYX, 2013.
[34] Maison de l'Architecture Marseille, *L'architecture au-delà des formes. Le tournant computationnel*, du 22 Février au 20 Avril 2007.
[35] Frédéric Migayrou (éditeur), *Architectures Non-Standard*, Paris Centre Pompidou, 2003.
[36] Marie-Ange Brayer (éditrice), *Architecture expérimentales 1950-2012*, Orléans : Éditions HYX : FRAC Centre, 2013.
[37] Centre Pompidou, *Multiversités Créatives*, du 3 Mai au 6 Août 2012.
[38] Marie-Ange Brayer (éditrice), *Imprimer le Monde*, Orléans, France : Éditions HYX, 2017.
[39] Frédéric Migayrou (éditeur), *Coder le monde : mutations, créations*, Paris : Éditions du Centre Pompidou, 2018.
[40] Marie-Ange Brayer et Olivier Zeitoun (éditeurs), *La Fabrique du Vivant*, Paris : Editions du Centre Pompidou, 2019.

et à la AA. Mais d'autres praticiens contribuent également à ouvrir le champ en France, comme François Roche, à cheval entre France et Thaïlande, ou Philippe Rahm, sans qu'ils se réclament du champ computationnel avec autant de force que d'autres. Bien qu'ils fassent tous deux appels à des outils algorithmiques et qu'ils cultivent une esthétique qui s'aligne sur celle du champ, ils articulent leur travail l'un autour de questions de pouvoir et l'autre autour de la perception des climats et des atmosphères. La filiale parisienne de Bollinger + Grohmann, dirigée par Klaas de Rycke, et les activités du laboratoire Navier de l'ENPC complètent le paysage computationnel parisien sur le plan de l'ingénierie. Une poignée d'écoles parisiennes se penchent sur ces enjeux dans le sillage de ces praticiens : à l'ENSA Paris Malaquais où enseigne Philippe Morel, à l'ENSA Versailles où enseigne Klaas de Rycke, à Marne-la-Vallée, proche de Navier, ou encore à Val de Seine. Enfin, quelques agences conventionnelles cherchent à expérimenter avec ces nouveaux outils de conception, et s'associent aux praticiens déjà versés dans la programmation pour les explorer. L'agence Hubert & Roy, par exemple, fondée en 1986 par Michel Roy et Bruno Hubert, porte à travers ce dernier depuis ses débuts un certain intérêt aux questions informatiques. Bruno Hubert a effectué juste avant la fondation de l'agence un séjour de recherche à l'université de Pennsylvanie et conseille à son retour à Paris la direction de l'architecture au ministère sur l'informatisation du processus de conception, tout en animant un séminaire à ce sujet à l'ENSA Paris La Villette. En 2002, Hubert & Roy conçoit un projet de concours de 1000 logements avec cours pour Tianherenjia, en Chine. Un programme permettant d'automatiser le placement des différentes cours en fonction de leurs dimensions et de leurs relations aux différents appartements est intégré au projet, dans l'objectif d'explorer des variations de la volumétrie d'ensemble du projet et de sa densité. Celui-ci est programmé par Pierre Vincent, architecte parisien qui s'intéresse lui aussi depuis longtemps à ces questions et appelé en renfort par l'agence. Bollinger + Grohmann entraînent quant à eux Dominique Perrault et son agence sur la voie du recours aux algorithmes, en fournissant à plusieurs reprises aux architectes des programmes de recherche et d'optimisation de forme pour les projets dont ils sont le bureau d'étude structure. Entre 2004 et 2006, ils mettent par exemple au point un programme qui permet de définir la géométrie des poteaux soutenant la canopée de la station de métro de la Piazza Garibaldi, à Naples. Celui-ci permet de jouer avec le placement de ces poteaux à travers la place et leur inclinaison, comme souhaité par les architectes qui souhaitent faire dialoguer ces "arbres métalliques" avec l'espace paysager qui occupe le reste de la place[41].

[41] Voir Klaus Bollinger, Manfred Grohmann and Oliver Tessmann, "Form, Force, Performance Multi-Parametric Structural Design", dans Michael Hensel et Achim Menges (eds.), *Versatily and Vicissitude*, AD Magazine Vol. 78, no. 2 (Mar./Apr. 2008), p. 20-26 et http://www.perraultarchitecture.com/fr/projets/2497-piazza_garibaldi.html (consulté le 17/08/2021).

*Figure 8. 1000 logements avec cours pour Tianherenjia, Hubert & Roy*

L'embryon de pôle français donne ainsi un exemple des stades de popularisation des expérimentations du champ : des praticiens qui s'investissent dans le recours aux outils algorithmiques, des expositions qui donnent à voir leurs recherches, des praticiens plus conventionnels qui expérimentent, des écoles qui commencent à s'y intéresser. En France, le succès est cependant plutôt mitigé. Si un noyau dur de praticiens poursuit ses travaux et si de nouvelles structures émergent dans le sillage du master Digital Knowledge À l'ENSA Paris-Malaquais, les agences les plus conventionnelles se désintéressent rapidement de ces outils et de ces questions. Après le concours pour Tianherenjia, Hubert & Roy abandonnent le recours aux algorithmes, peu convaincus de leur plus-value, et Bruno Hubert s'éloigne peu à peu également de ses activités académiques sur le sujet, leur privilégiant d'autres champs. Chez Dominique Perrault Architecture, malgré des collaborations régulières avec Bollinger + Grohmann qui eux continuent à développer leurs pratiques computationnelles, le désintérêt est total. Malgré la démonstration faite par ces derniers sur les différents projets communs, l'agence ne se saisit pas vraiment des outils, n'y voyant que peu d'intérêt en regard du processus de conception auquel Dominique Perrault et ses équipes sont habitués. Bollinger + Grohmann montre à plusieurs reprises les algorithmes programmés pour les projets réalisés avec Dominique Perrault comme des exemples de la pratique computationnelle du bureau d'étude. Du côté des architectes, néanmoins, aucune mention n'est jamais faite de l'existence de ces algorithmes, le projet étant toujours présenté par un discours architectural classique. Dans quelques écoles, des enseignants s'investissent bien dans le développement de pratiques computationnelles, dans l'établissement d'un pôle de fabrication numérique ou dans l'organisation de quelques expositions et conférences. Mais hormis cette poignée d'enseignants, l'accueil fait aux initiatives relevant de l'architecture computationnelle est plutôt froid au sein du reste des équipes pédagogiques, qui n'y voient que peu d'intérêt. La réception critique dans la presse architecturale généraliste lorsque des travaux du champ computationnel sont montrés est

plutôt fraîche également. Les publications dans CREE[42], D'Architectures[43] ou A+U[44], qui rendent compte d'une exposition de 2007 présentant les résultats d'un concours d'architecture dédié aux pratiques computationnelles à la Maison Rouge, sont assez sceptiques.

Malgré la timidité rencontrée en France, qui s'explique peut-être par le rapport particulièrement conflictuel aux techniques de la formation beaux-arts des architectes[45], les outils algorithmiques amorcent bien une phase de popularisation dans les années 2000. Expositions, enseignements et publications leur donnent de la visibilité. Ils commencent par ailleurs aussi à être mis en pratique dans des projets construits, et plus seulement dans des projets de papier. Recherches universitaires, projets expérimentaux, prototypes, ou projets qui n'avaient jamais vu le jour, comme le Generator Project, sont rejoints par un petit ensemble de projets construits. Cela reste certes des projets d'exception, avec des budgets conséquents, des ambitions formelles complexes, des équipes d'ingénieurs nombreuses et des processus de fabrication sur mesure. Mais ces projets représentent néanmoins une incursion du champ computationnel dans le monde de la construction, en dehors du monde universitaire. Leur statut de projets d'exception contribue par ailleurs là encore à populariser les algorithmes dans le monde architectural, à les donner à voir à un plus grand nombre de praticiens. Au-delà des praticiens qui évoluent directement au sein du champ computationnel, d'autres architectes exerçant en agence commencent à s'investir. Certaines de ces agences gardent un oeil sur le développement de l'informatique de longue date, comme Skidmore, Owings & Merrill, qui avait déjà collaboré avec des chercheurs dès les débuts de l'infographie[46]. A partir des années 2000, ils mobilisent également modélisations paramétriques et évaluations structurelles, en particulier pour différents projets de tours : Al Hamra Tower, Cyan Tower, ou encore White Magnolia Plaza. Ces projets sont l'occasion de développer une expertise au sein même de l'agence, dans leur bureau de Chicago, avec le BlackBox Studio, nom qu'ils donnent à l'unité spécialisée dans les recherches computationnelles qu'ils y fondent[47]. D'autres poussent cette ambition encore plus loin : en compléments de leurs activités de conception classiques, elles créent au sein de l'agence de petits groupes dont les membres se dédient à plein temps à la recherche computationnelle. C'est le cas chez Foster & Partners, qui crée le Specialist Modelling Group en 1997, chez Aedas, dont le groupe Computational Design Research voit le jour en 2004, ou encore chez Zaha Hadid Architects, où CODE naît en 2007. D'autres agences vont

[42] Anonyme, A.: 2007, Le pavillon d, Architecture intérieure CREE, 332, 0294-8567.
[43] Scoffier, R.: 2007, Architecture, rayon passementerie, six propositions pour le pavillon Seroussi, D, 167, 1145-0835.
[44] Anonyme, A.: 2008, Xefirotarch / Hernan Diaz Alonso, Maison Seroussi – Paris, France, A+U, 455, 0389-9160.
[45] Voir Conclusion.
[46] Voir Chapitre III.
[47] SOM Blackbox, https://static1.squarespace.com/static/542319abe4b0b8239434a0e8/t/544c14abe4b0420e8842209a/1414272171706/08-SOM+Blackbox.pdf - consulté le 01 Septembre 2021.

simplement faire appel directement aux praticiens du champ pour leur faire programmer des algorithmes sur lesquels baser des projets. C'est le cas de Coop Himmelb(l)au, de SANAA ou de Reiser + Umemoto. Même David Greene, l'un des fondateurs d'Archigram, s'intéresse un temps à la question[48], ce qui montre la place que gagne peu à peu le champ computationnel comme espace d'expérimentation reconnu. Nombre d'agences de grande taille[49] saisissent au cours des années 2000 que quelque chose est en train de se produire, et s'aventurent dans le domaine de la conception algorithmique. Cette incursion dans le domaine computationnel est plus ou moins marquée. ZHA s'y investit jusqu'à en faire un élément essentiel de la pratique de l'agence, OMA s'y essaie brièvement le temps d'un concours pour une tour, UNStudio ou SOM en font un simple outil supplémentaire à intégrer à un processus qui reste somme toute conventionnel. La pratique de ces dernières agences est alors souvent à la lisière entre processus algorithmiques et simple numérisation. Ces différents degrés d'investissement dans ces expérimentations prennent différentes formes. Cela peut être une simple demande de programmation d'un algorithme, comme SANAA ou Archiglobe avec Bollinger + Grohmann. Cela peut aussi être commencer à faire travailler ponctuellement des gens en interne dessus, comme chez OMA, ou démarrer un groupe dédié, comme chez Foster + Partners. D'autres choisissent aussi de collaborer dans le cadre d'un workshop dans une école d'architecture, comme SOM avec biothing au GSAPP. Cela contribue non seulement à la popularisation des pratiques computationnelles en dehors des universités, mais aussi à l'établissement d'une multitude de formats où s'exerce l'expérimentation avec les outils algorithmiques.

### *4.1.3 Deux objectifs distincts : industrialisation et prospective*

Au cours des années 2000, le champ computationnel croît jusqu'à former un réseau universitaire et professionnel vaste et solide. S'y croisent universitaires et praticiens, architectes, ingénieurs et informaticiens. Mais ce réseau reste un réseau abrité des contraintes de l'industrie de l'architecture contemporaine. Les praticiens du champ computationnel évoluent pour beaucoup dans la sphère universitaire, où les recherches sont certes conditionnées à l'obtention d'un poste ou de financements, mais libres par ailleurs de choisir leur sujet et leur forme. Les praticiens du monde professionnel qui les rejoignent à cette génération, bien qu'ils exercent en agence, disposent eux aussi d'une certaine liberté, puisque les projets pour lesquels le recours aux outils computationnels se fait en premier en agence sont des projets d'exception. Le musée Guggenheim de Bilbao, l'opéra de Sydney ou le musée BMW impliquent certes le respect d'un certain nombre de contraintes constructives, mais nombre des

[48] Pour la biennale d'architecture de Londres en 2006, il collabore notamment avec Marcos Cruz, Marjan Colletti et Samantha Hardingham pour le projet *Invisible University*, qui fait se rencontrer situationnisme et robotique. Majan Colletti et Marcos Cruz, *Output 3 (Design) Invisible University*, University of Westminster, 2008.
[49] Plus de 200 employés.

injonctions financières ou même réglementaires usuelles sont bien moins fortes dans leur cas. Les budgets sont suffisamment élevés pour passer plusieurs années à chercher la forme la plus pertinente pour une coque[50], pour produire des éléments préfabriqués *via* des processus de fabrication exceptionnels ou pour financer des autorisations réglementaires pour ces méthodes de fabrication expérimentales. Dans les deux cadres, cette protection permet aux praticiens de se confronter à deux objectifs sans être pollués par ces contraintes qu'ils évitent : recherche architecturale prospective, ou industrialisation des nouveaux processus computationnels.

Malgré le pas de côté qu'ils représentent en regard des conditions de production usuelles, les projets réalisés nécessitent quand même de mettre au point des processus de fabrication suffisamment robustes pour produire un très grand nombre de pièces. La façade de la fondation Louis Vuitton compte ainsi 19 000 panneaux de béton à ultra-haute performance, la philharmonie de Hambourg 10 000 panneaux de plâtre fraisés pour l'intérieur de son auditorium. Les processus de fabrication robotisés qui permettent d'obtenir un grand nombre de pièces aux formes variables doivent donc passer de l'atelier de praticiens qui expérimentent avec en produisant des pavillons de petite échelle à une échelle industrielle qui exige robustesse, rapidité et fiabilité. Une partie du champ se prend d'intérêt pour le passage à l'échelle nécessaire des méthodes et outils développés au sein du champ. Ceci autorise à terme l'intégration des outils algorithmiques et les prouesses qu'ils permettent de réaliser dans la palette des outils de conception et fabrication classiques de l'industrie de la construction. Ce qui permet d'y recourir plus régulièrement et d'en diminuer le prix. Il s'agit par ailleurs aussi, *via* cette rationalisation du processus, de simplifier un travail parfois fastidieux pour les ingénieurs. En effet, les modélisations fournies par les architectes, souvent pensées dans un premier temps comme des explorations spatiales, puis dans un second temps comme des représentations permettant de s'imaginer le projet final, sont souvent peu précises géométriquement : les erreurs de maillages ou d'alignement sont parfois légion dans ces fichiers initiaux[51]. Quand ce ne sont pas des erreurs de modélisation, ce sont parfois simplement des difficultés de fabrication : c'est le cas par exemple des voiles en verre de la fondation Louis Vuitton. Malgré l'insistance de Frank Gehry à conserver les formes initialement produites par son agence pour ces voiles, la seule solution aurait été de les couler à chaud pour les fabriquer telles quelles, une technique aussi coûteuse que difficile à mettre en œuvre à cette échelle. C'est donc finalement un assemblage de portions de cylindre à simple courbure qui sera privilégié, avec cependant comme conséquence là-aussi de devoir redessiner les voiles pour qu'elles correspondent à cet impératif géométrique. Il s'agit donc pour les acteurs intervenant à la fin du processus de conception, lors de la production des fichiers d'exécution, de

[50] C'est le cas pour les coques de l'Opéra de Sydney. Françoise Fromonot, *Jorn Utzon : The Sydney Opera House*, Phaidon Press, 2002.
[51] Hugh Dutton, conférence donnée à Volumes dans le cadre du master Design by Data, 9 Décembre 2016.

s'éviter de devoir redessiner intégralement un bâtiment à partir des esquisses de l'architecte. Devoir fabriquer et assembler ces pièces exige par ailleurs une grande exactitude géométrique. À l'échelle expérimentale, des erreurs de mesure peuvent être corrigées en fabriquant de nouvelles pièces, ou en bricolant un peu pour résoudre le problème. A l'échelle industrielle, s'adapter ainsi est bien plus problématique. La production des fichiers de fabrication - le dessin comme les instructions machines - doit donc elle aussi suivre un processus de rationalisation. En complément du développement des procédés de fabrication, des outils pour rendre plus faciles les opérations de dessin voient donc le jour. Ils intègrent directement le procédé de fabrication choisi à la définition de la géométrie, puis incluent la production des fichiers de fabrication dans le programme, mettant en œuvre un flux de travail qui se veut continu des premières esquisses à la fabrication des pièces. Rationalisation géométrique et industrialisation sont donc deux enjeux qui se nourrissent l'un de l'autre, dans le contexte de pratiques mobilisant les outils computationnels pour la construction.

*Figure 9. a. DFAB House (G&K ETHZ) ; b. Armadillo Vault (Block Research Group) ; c. Noeud rotoformé (DDU) ; d. Chai Gantenbein (G&K ETHZ)*

Le pôle austro-allemand du réseau des années 2000 offre un bon exemple de la forme que prennent ces recherches dans le monde universitaire. La Angewandte à Vienne s'oriente certes plutôt vers la prospective, mais le bureau de Bollinger + Grohmann qui y est situé se plonge dans le développement

de techniques de modélisation et d'outils computationnels pour la fabrication numérique. Le bureau de Francfort exerce essentiellement en tant que bureau d'études classique, mais celui de Vienne collabore avec de multiples praticiens du champ computationnel, ainsi qu'avec des artistes comme Tomas Saraceno, et développe Karamba 3D, plug-in Grasshopper pour l'évaluation structurelle. A Stuttgart, l'ICD d'Achim Menges ne fonctionne qu'en produisant des prototypes, et propose à partir de 2010 chaque année un pavillon. Le groupe a deux matériaux de prédilection : le bois, assemblé sous forme de plaques pour former des structures courbes ou sous forme de feuilles pour jouer directement avec la courbure de l'élément, et la fibre de carbone. Cette dernière est enroulée sur elle-même ou étirée d'un point à un autre de l'espace et combinées avec des fibres polymères, voire plaquées contre une structure gonflable qu'elles permettent de rigidifier. Les pièces ainsi fabriquées sont ultra-légères et résistantes, peu gourmandes en matériaux et déclinées par le groupe dans de multiples géométries. La Digital Design Unit de Darmstadt, bien que créée plus tardivement, complète le réseau et se concentre notamment sur l'impression 3D, la fabrication de modules courbes et creux en béton ou l'assemblage réversible robotisé. Les voisins suisses ne sont pas en reste : l'ETHZ s'investit lourdement dans ces recherches, en particulier autour de la fabrication. L'enjeu d'industrialisation des techniques de construction robotisées est affiché dès le début, et les équipes de recherche contribuent très tôt à des projets construits. Fabio Gramazio et Matthias Kohler développent dès 2006 un processus de placement de briques robotisées. Cette technique permet de faire varier le placement des briques et notamment de les faire pivoter différemment les unes des autres au sein du mur. Elle est mise en œuvre la même année au chai Gantenbein, en Suisse, ou Gramazio et Kohler conçoivent un espace de stockage aux murs ajourés. Ces murs sont composés de pans de briques assemblées préfabriqués et favorisent une température et un taux d'humidité optimaux pour la conservation du vin. Les années qui suivent, Gramazio et Kohler poursuivent au sein de leur groupe de recherche le développement de techniques de construction robotisées, d'abord continuant leur travail avec des briques, les assemblant cette fois-ci à l'aide de drones, puis en se tournant vers le béton. Le recours au robot six axes qui leur avait permis d'assembler les briques à Gantenbein leur permet cette fois-ci de mettre en place un coffrage glissant déformable. Ils créent à l'aide de ce dispositif des colonnes de forme variable. Le même robot leur permet quelque temps plus tard de déformer un treillis métallique sur lequel est ensuite projeté du béton, créant un système de béton armé de forme variable. Enfin, le groupe s'essaye également à l'impression 3D, qui leur permet là encore de produire des formes non-standard à l'aide d'un robot six-axes. La maison DFAB house, projet lancé en 2016 et soutenu par le National Center of Competence in Research (NCCR) suisse, rassemble en une seule construction des démonstrateurs des différentes techniques mises au point par le groupe au cours des dix années écoulées. Non loin de là, sur le même campus, le Block Research Group (BRG) se spécialise dans la construction de voûtes, en pierres assemblées comme pour

l'Armadillo Vault, à partir d'un réseau de câbles drapé de tissu ou d'une membrane servant de moule à une coque en béton. Pour ses projets, le BRG développe plusieurs outils. RhinoVault permet de modéliser des coques. eQUILIBRIUM est conçu pour apprendre la statique graphique facilement, à travers un site interactif. COMPAS enfin regroupe l'ensemble des outils du BRG programmés sur Python au cours de leurs projets, les rendant accessibles sous forme de bibliothèques. Comme Gramazio & Kohler Research, le Block Research Group fait par ailleurs partie du NCCR Digital Fabrication, qui soutient sa participation à diverses biennales et expositions. C'est donc presque un pôle suisso-austro-allemand qui se penche sur l'industrialisation de la fabrication robotisée, dont le NCCR, Bollinger + Grohmann mais aussi l'agence Scheffler & Partners à Stuttgart sont l'extension vers le monde professionnel.

Dans le monde professionnel, on l'a vu, nombre d'agences font une incursion dans le champ computationnel. Certes, quelques agences existaient déjà auparavant. Nous avons vu par exemple Space Syntax Ltd., mise en place en 1989 par Bill Hillier pour encadrer la mise en application de la space syntax développée à la Bartlett. Mais les années 2000 sont marquées par l'irruption d'un plus grand nombre d'agences. Plutôt que la création de structures professionnelles par des praticiens universitaires, ces agences ont déjà une pratique classique, et y ajoutent une approche computationnelle. Certaines d'entre elles ont par ailleurs un apport conséquent sur le plan des outils algorithmiques : il ne s'agit pas toujours juste d'une mise en application de programmes développés par d'autres. Par ailleurs, en plus des agences d'architecture, d'autres structures professionnelles de l'industrie de l'architecture font aussi leur apparition : les bureaux d'études techniques (BET). C'est la rencontre du champ computationnel tel qu'on l'a vu jusqu'ici, avec ses racines dans les domaines de l'IA et de l'architecture, avec une autre filiation informatique : celle du génie civil. Celle-ci commence avec les ingénieurs Frei Otto, Cecil Balmond ou Ove Arup, qui dès les années 1970 avaient commencé à s'intéresser à la numérisation des modèles de calcul des structures. Comme dans le champ architectural, l'université est le lieu de développement de programmes d'évaluation ensuite mis en application ponctuellement dans des projets d'exception. Les années 2000 voient ensuite la création de groupes internes aux BET, dédiés aux pratiques computationnelles, sur le même modèle que dans les agences d'architecture. En 2000, Cecil Balmond, ingénieur chez Arup, le BET fondé par Ove Arup en 1938[52] et devenu entre-temps l'un des plus influents du monde, y fonde l'Advanced Geometry Unit (AGU). Celle-ci a vocation à explorer les applications de la programmation pour la construction, et rassemble ingénieurs et architectes, mais aussi artistes, mathématiciens et programmeurs. Le groupe développe des outils à usage interne, comme FABWIN, qui permet de

[52] L'Arup d'aujourd'hui est plus exactement constitué d'un ensemble de plusieurs entreprises fondées au fil des années par Ove Arup avec plusieurs groupes d'architectes et d'ingénieurs - en 1938, en 1946 et en 1963.

modéliser des membranes, apporte son expertise pour des projets comme le siège de la télévision chinoise à Pékin ou le pavillon de Toyo Ito pour la Serpentine Gallery. L'AGU apporte également son aide à nombre d'artistes pour réaliser leurs œuvres, notamment l'Arcelor Mittal Orbit réalisée par Anish Kapoor pour les jeux olympiques de Londres en 2012. En 2003, Buro Happold, fondé en 1976 par Edmond Happold et également l'un des BET les plus renommés, lance le SMART Group, aux occupations très similaires à celles de l'AGU. Ce dernier développe notamment la suite d'outils SMART, qui regroupe un ensemble d'outils algorithmiques divers développés au fil des années d'abord à usage interne mais ensuite rendus publics. A la suite de ces unités spécialisées, des bureaux d'études techniques spécialisés voient le jour. designtoproduction, par exemple, fondé en 2007 par Fabian Scheurer et Hanno Stehling, exerce à cheval entre l'Allemagne et la Suisse. Le BET se spécialise dans la rationalisation géométrique et la production de fichiers de fabrication numériques, ainsi que dans la gestion de la production d'éléments non-standard, en particulier pour des structures bois. Parmi leurs projets les plus spectaculaires, la canopée du Centre Pompidou Metz, dessinée par Shigeru Ban, ou la façade en bois ondulée du théâtre de Kilden, en Norvège.

Chacune de ces unités spécialisées de BET emploie, en plus d'ingénieurs en génie civil, de nombreux architectes. Au sein des unités spécialisées des agences, les pratiques sont par ailleurs très proches de celles de leurs homologues des BET. Les praticiens s'y dédient à des questions techniques pour faciliter la construction à l'aide d'outils computationnels, tous les groupes exerçant sur le même modèle. Cette fluidité entre professions, entre pratiques, est un vecteur d'intégration dans le champ computationnel. Ces nouvelles structures s'intègrent par ailleurs également en favorisant les échanges non seulement entre professions, mais aussi entre le monde universitaire et l'industrie de l'architecture. En effet, Happold, Arup, Aedas ou Foster ne se contentent pas de créer des groupes de réflexion et de recherche en interne. Il s'agit aussi de faire la promotion de ces outils - l'objectif étant de les faire passer dans la pratique des architectes. Pour ce faire, il faut établir des espaces d'échange avec les praticiens du champ computationnel en architecture, dont les expérimentations se font parfois sans beaucoup de considérations pour les réalités physiques de la construction. C'est ce que vise la création de Smart Geometry en 2001 par Lars Hesselgren, alors directeur de la R&D chez KPF, J. Parrish, alors chez Arup et Hugh Whitehead, fondateur du Specialist Modelling Group chez Foster & Partners. Smart Geometry est une plateforme qui permet de rassembler tous les spécialistes pour échanger à l'occasion d'ateliers. C'est l'occasion pour chacun de présenter ses avancées, mais aussi de comparer les possibilités offertes par les différents outils en ascension dans le champ. Smart Geometry devient aussi rapidement le lieu de workshops de conception d'outils et d'échanges entre utilisateurs et concepteurs des programmes pour les adapter aux besoins[53]. En particulier, Smart Geometry est

[53] Conversation Oliver Tessmann.

l'endroit où se développe l'usage de Generative Components, un logiciel de programmation visuelle conçu spécialement à destination du champ computationnel par Bentley Systems, qui sponsorise initialement l'évènement. Le nom de Smart Geometry, tout comme celui de l'Advanced Geometry Group de Buro Happold ou le slogan de designtoproduction "*nous maitrisons la complexité*"[54] reflètent bien le cœur de cet enjeu de rationalisation et industrialisation auquel cette partie du champ a choisi de se confronter. Créée par des praticiens exerçant en agence et en bureau d'étude, sponsorisée par Bentley Systems, Smart Geometry est plus fortement ancrée dans l'industrie de l'architecture que les organisations historiques du champ. Cela renforce le poids de cette dernière dans les enjeux, lui donne de la visibilité et équilibre *in fine* un champ qui était jusqu'ici très académique, offrant un espace aussi aux professionnels pour développer des pratiques computationnelles plus proches de leur quotidien. Les premières réunions de Smart Geometry sont de petits événements, réunissant les quelques dizaines de praticiens qui veulent se frotter à ces questions. S'y retrouve rapidement tout le monde dans le champ ; les praticiens venus des agences qui commencent à s'intégrer au champ computationnel mais aussi des praticiens universitaires. A partir de 2008, ces rencontres prennent la forme d'une conférence annuelle, qui est aujourd'hui encore l'un des principaux évènements du champ computationnel. En 2009, la liste des intervenants compte aussi bien Bentley Systems, Foster & Partners, KPF, Arup, Boeing ou Morphosis que Jenny Sabin, Achim Menges, Mark Burry, Martin Tamke, Sean Ahlquist ou Axel Kilian. A ce titre, Smart Geometry est emblématique du fait que cette période est un moment d'équilibre très fort entre les différentes professions qui évoluent au sein du champ computationnel. Les espaces de pratique sont alors partagés, quand bien même tout le monde ne recourt pas aux outils computationnels dans le même objectif.

L'une des branches du champ se penche donc sur l'intégration à l'industrie de l'architecture des pratiques computationnelles. L'autre choisit une démarche prospective, à la recherche de l'innovation architecturale permise par les outils algorithmiques. Cette innovation s'exprime d'abord dans de nouveaux vocabulaires formels, des coquilles des surfaces minimales de theverymany aux motifs et structures bio-inspirés de l'ICD ou de biothing en passant par les volutes des géométries de Xefirotarch. L'inventivité formelle qu'offrent les divers dispositifs algorithmiques employés s'accompagne du développement d'un certain goût pour le hasard. Réguler la succession des instructions informatiques sans forcément avoir de géométrie précise en tête permet aux praticiens de se laisser surprendre par le résultat produit par le programme. La variabilité des résultats incite les praticiens à jouer avec les paramètres pour observer ensuite les formes résultantes, à se prendre au jeu

[54] "*we master complexity*", https://www.designtoproduction.com/ (consulté le 17/08/2021).

de résultats formels inattendus[55]. Mais la recherche de nouvelles hiérarchies spatiales comme on les voyait chez Bill Hillier ou William Mitchell sous-tend aussi toujours ces pratiques computationnelles. Avec le développement de moyens techniques plus vastes, de nombreux autres aspects font leur apparition dans cette hiérarchisation des composants de la conception architecturales. Il ne s'agit plus simplement d'agencer plusieurs pièces les unes par rapport aux autres dans le plan ou dans l'espace, mais d'articuler forme, matériau, structure en une logique unifiée. Le même phénomène de surprise à la visualisation des résultats s'instaure ici, étendant ce jeu avec les outils au-delà de simples questions formelles. Il s'agit donc bien de partir en quête de nouveaux dispositifs spatiaux dans ces démarches prospectives - forme et fonction sont après tout toujours entrelacées dans la démarche architecturale.

La branche du champ computationnel qui s'intéresse à cette approche prospective est caractérisée par une activité avant tout universitaire. Les praticiens sont néanmoins nombreux à combiner cette activité avec une petite agence expérimentale, souvent en duo. C'est le cas de Maxwell et Pigram au sein de supermanoeuvre et de Stuart-Smith et Snooks au sein de kokkugia, mais aussi de Mirco Becker et Oliver Tessmann chez f_u_r, de Tim Schork et Paul Nicholas au sein de MESNE. Créent également des structures similaires George L. Legendre avec IJP Corporation, Philippe Morel, Félix Agid et Jelle Feringa avec EZCT Architecture & Design Research ou Marcelyn Gow et William Mohline avec servo. Ces structures permettent aux praticiens d'avoir un pied dans le monde professionnel. Les agences leurs fournissent une structure juridique qui leur permet de participer à des expositions ou à des concours, mais aussi de se créer une identité repérable au-delà du monde de la recherche. Ces structures sont parfois plus grandes, comme dECOI, dirigée par Mark Goulthorpe mais faisant intervenir un grand nombre d'autres architectes et ingénieurs du champ, ou sixteen*(makers) qui rassemble Phil Ayres, Nick Callicott, Chris Leung, Bob Sheil et Emmanuel Vercruysse. Le cas d'OCEAN est également intéressant : plus qu'une agence, il s'agit d'un réseau mettant en relation un grand nombre de praticiens autour de Michael Hensel, leur offrant une plateforme professionnelle à partir de 1994. OCEAN permet notamment à ses membres de collaborer pour participer à un grand nombre de concours et d'expositions. Alors que nombre d'autres agences prospectives ne survivent pas aux variations des carrières universitaires de leurs membres, sa forme d'association d'un grand nombre de praticiens permet à OCEAN de survivre en adoptant plusieurs formes successives. Sa première version, OCEAN net, est fondée par Michael Hensel, Ulrich Königs, Tom Verebes et Bostjan Vuga. En 1998, l'association se sépare en deux branches. La première, OCEAN UK, est menée par Tom Verebes et ancre ses activités à Londres. La seconde, OCEAN NORTH, est menée par Michael Hensel reste, rejoint d'abord par un groupe d'enseignants-chercheurs de Helsinki et d'Oslo, rencontrés

[55] Léda Dimitriadi, "Algorithmique du hasard et créativité : de la nature à la computation", intervention dans le cadre de la conférence *Projeter l'architecture, aux carrefours du numérique et du vivant*, 27 janvier 2020, ENSA Paris Val-de-Seine.

à l'occasion de séjours là-bas. Au fil des années qui suivent, cette seconde branche est rejointe par une quinzaine d'autres praticiens du champ computationnel, parmi lesquels Achim Menges, Jeffrey Turko ou Daniel Cool i Capdevila. C'est la période la plus prolifique du groupe[56] qui *via* ses deux branches participe à de nombreux concours et est invitée à réaliser des scénographies d'expositions, du mobilier ou des installations un peu partout dans le monde. OCEAN NORTH en particulier est aussi commissionné par diverses institutions en Finlande, en Norvège ou en Allemagne pour des propositions de projets pour des musées, des schémas directeurs urbains ou autres bâtiments officiels, commandés par la ville ou l'État. Ces projets restent néanmoins au stade d'esquisses puisqu'aucun ne sera construit. Les membres d'OCEAN saisissent en fait les occasions qui leur sont offertes dans leur milieu professionnel et renvoient vers l'association les propositions qui leur sont faites en tant qu'individus. Michael Hensel, qui a une activité régulière en tant qu'architecte en Allemagne, ou Kivi et Tuuri Sootama qui exercent en Finlande, choisissent ainsi de répondre collectivement avec OCEAN NORTH aux invitations qu'ils reçoivent de la part d'institutions. OCEAN se crée ainsi un espace de pratique professionnelle particulièrement vivace pour le champ computationnel, proposant des bâtiments pour des concours beaucoup plus consensuels que la plupart des projets accomplis par ces petites agences expérimentales. Là où OCEAN cultive un ancrage dans une sphère professionnelle plutôt classique, les autres agences se créent un espace de pratique professionnelle à part entière, à mi-chemin entre la pratique artistique et la pratique architecturale, et donc toujours un peu déconnectées des contraintes classiques de l'industrie de la construction.

En effet, ces structures servent bien de cadre à une pratique en dehors du champ universitaire, mais cela reste surtout des projets de papier, c'est-à-dire des espaces conçus jusqu'au stade de l'esquisse, voire de l'avant-projet détaillé, ou des prototypes, mais pas des bâtiments construits. Une partie des expérimentations du champ computationnel se limite même à un bouillonnement de formes sur écran, laissant le spectateur y chercher ce qui relève d'une configuration spatiale quelle qu'elle soit. D'autres, notamment en réponse à des concours, vont jusqu'au bout de l'exercice de l'esquisse, fournissant plans, coupes, axonométries et vues en perspective. Enfin, le prototype prend une place particulière dans les productions du champ : vu la complexité des formes conçues grâce aux algorithmes, il s'agit de convaincre de la faisabilité des propositions. La conception spatiale s'associe donc souvent à la production grâce à des procédés de fabrication numérique de prototypes qui donnent à voir comment le bâtiment serait fabriqué s'il venait à être réalisé. Le champ computationnel fait donc de plus en plus la part belle aux pavillons comme aboutissement des recherches architecturales.

[56] Par la suite, en 2008, OCEAN redevient une seule entité, OCEAN Design Research Association, dont l'objectif est essentiellement de promouvoir l'activité passée d'OCEAN. En 2018, toujours sous l'impulsion de Michael Hensel, celle-ci devient OCEAN A|E et reprend son statut d'association-agence.

Ceux-ci se multiplient dans les écoles et les biennales au fur et à mesure que le champ gagne suffisamment de crédibilité pour être non seulement exposé, mais aussi invité à produire des pièces spécialement pour ces occasions, qui sont financées par les institutions. Ces pavillons sont envisagés par les praticiens comme des démonstrateurs de la capacité du champ computationnel à produire de l'architecture pour de bon, c'est-à-dire à faire le saut du papier au construit, à faire le saut d'échelle que demande la construction, répondant ainsi aux critiques que leur opposent les praticiens tenants d'une pratique plus classique. Les projets de papier du champ sont produits dans le cadre d'expositions, de recherches financées par l'université, de travaux menés en collaboration avec les étudiants au sein des enseignements, ou en réponse à des concours. Les membres du réseau cultivent dans ce contexte une certaine forme de solidarité : chacun invite les autres à ses jurys, à ses expositions ou ses conférences, ou encore à collaborer pour des concours. C'est le cas de Philippe Morel à la maison de l'architecture à Marseille, de Marc Fornes pour *scriptedbypurpose*, d'Eran Neuman pour *Performalism* et *The Gen(h)ome Project*, toutes des expositions qui rassemblent les praticiens les plus emblématiques de cette vague prospective des années 2000 et du début des années 2010.Se retrouve dans les listes de participants à ces expositions et panels l'entre-soi du réseau, un peu élargi par rapport à celui de la AA ou celui d'AD mais bien là. Ces listes marquent aussi, bien que le champ cultive un certain équilibre entre les professions comme on l'a vu, l'existence de deux branches aux objectifs différents. Si certains praticiens naviguent les deux champs, d'autres se cantonnent à l'industrialisation en marquant peu d'intérêt pour la prospective et inversement, réservant leurs activités à des collaborations avec seulement certains praticiens et dans certains espaces.

Le format des projets de papier est de fait ce qui permet l'exploration débridée qui caractérise cette branche prospective. Les projets de papier donnent une certaine marge aux praticiens qui les conçoivent, puisqu'ils évoluent dans un cadre qui ne sanctionne aucune prise de risque. Non seulement la liberté de format est complète, mais les explorations spatiales n'ont jamais vraiment besoin d'être structurellement viables, étanches ou même physiquement réalistes. Le seul budget à respecter est celui de la fabrication des prototypes, celui du bâtiment entier entre rarement en ligne de compte. Les programmes eux aussi sont rarement très précis - musées ou bibliothèques présentent certes des espaces d'exposition ou de lecture, mais le détail des toilettes ou du vestiaire n'apparaît pas. Toutes les pistes sont explorables à l'écran. Une liberté qui explique le potentiel perçu par les praticiens dans les outils algorithmiques et le bouillonnement au sein du champ computationnel à cette période. C'est en effet un moment de rencontre entre accessibilité d'une plus grande diversité d'algorithmes et de machines plus performantes, maturation du champ, popularisation des pratiques et augmentation des opportunités, et développement d'un savoir-faire. La Cliff House de kokkugia ou les

projets de Xefirotarch témoignent de cette combinaison de liberté de pratique, de recherche de forme et d'exploration des possibilités du computationnel (Figure 10). Et puisque ces expérimentations prennent la forme de projets de papiers, ce sont des dessins, des vidéos, des machines, des prototypes d'échelles diverses, mais aussi du code qui sont donnés à voir. Les images mises en scène cherchent d'abord à démontrer la variabilité et la complexité des formes obtenue, parfois jusqu'à perdre le lien avec la conception spatiale, ce qui n'est pas pour déranger particulièrement les praticiens de cette branche. L'architecture devient ainsi un terrain de jeu beaucoup plus vaste que la simple conception et construction de bâtiments, ou toute incursion dans une discipline annexe sur laquelle s'appuie la conception architecturale devient valide, toute incursion dans une technologie en devenir également. Désormais, ce qui compte c'est le processus plus que le résultat final, et les productions visuelles du champ computationnel s'alignent sur ce nouvel impératif[57].


*Figure 10. a. Babi Yar Memorial, Kokkugia ; b. Xefirotarch ; c. Projet The Very Many*

La variété des productions du champ computationnel rend parfois difficile de parler d'une esthétique commune dans ces images. Cyberspace, projections, réalités augmentées et virtuelles en sont un pan, courbes et blobs un autre, motifs du monde naturel et complexité des structures encore un autre. Ils partagent néanmoins une certaine esthétique de la technique, plus exactement l'utilisation du code

[57] Pour observer un plus grand nombre des productions de cette période, le lecteur peut consulter le cahier de projets en annexe.

comme un étendard visuel de leurs pratiques. Ce nouveau point de repère visuel permet aux praticiens du champ computationnel de s'identifier comme faisant partie de cet espace de pratique. Le code devient au fil des expositions et des magazines un élément visuel à part entière lorsqu'il s'agit de présenter le projet. Les codes classiques de la représentation architecturale perdent du terrain. Portes, fenêtres ou mobilier sont rarement présents sur les dessins. Épaississeur des traits, pointillés ou flèches qui permettent usuellement de comprendre l'espace à partir du dessin s'effacent également. Coupes, élévations et plans ne sont plus obligatoires pour présenter un projet. L'enjeu n'est plus tant de parvenir à lire l'espace créé, mais plutôt d'en comprendre la logique de conception. Font donc irruption dans la présentation des projets des diagrammes permettant de saisir cette logique. Cette représentation diagrammatique hérite des formes de représentation développées par les déconstructivistes dans les années 1980. Les pratiques computationnelles se placent en effet en partie sous le patronage de Peter Eisenman ou de Bernard Tschumi, architectes chez qui le diagramme occupe une place majeure. Les transformations géométriques successives documentées chez ces derniers deviennent dans le champ computationnel des séries d'instructions informatiques, illustrées par des exemples des transformations géométriques en résultant. Les diagrammes cherchent à saisir des ensembles d'opérations mathématiques et à les donner à comprendre au spectateur, comme en témoignent les illustrations de biothing pour *Genware*, une bibliothèque de scripts à combiner entre eux en fonction des effets recherchés. Les différents scripts sont conçus notamment pour modifier la géométrie de surfaces, les subdivisant pour leur appliquer des modules variables localement. Les diagrammes de *Genware* expliquent donc fonction par fonction le code qui permet d'obtenir ces résultats. En complément de ces diagrammes, le code s'invite dans les catalogues et sur les murs des expositions. Il se dévoile sous forme d'encarts de texte, de lignes de codes superposées à une vue du projet - l'un ou l'autre servant de toile de fond, opposant ses austères successions de caractères à l'exubérance des formes qu'il produit. Les variations qu'un programme permet d'obtenir, elles-mêmes devenues un élément à part entière de la présentation du projet, s'accompagnent d'une petite légende qui les identifie par un nom et / ou un numéro, mais aussi par les variables qui ont permis leur obtention, petit morceau de code suivant chaque forme générée. L'exposition *scriptedbypurpose* en est un exemple emblématique, son propos même est de donner codes et espaces à voir ensemble. Son manifeste est le suivant : "*1. toutes les entrées affichées doivent faire appel à des techniques de code. 2. afin d'éviter que la génération précédente ne soit régie par des exposés génériques sur les "techniques", tous les codes et outils personnalisés doivent être affichés à côté de l'œuvre*"[58]. Le titre de l'exposition - "*codé avec intention*" - et l'usage du mot "*entrée*" - qui fait référence à une entrée de code - pour désigner les projets dans le manifeste, complètent cette esthétique avec un vocabulaire

[58] *"1. all entries displayed should involve scripting techniques 2. in order to avoid a previous generation ruled by generic talks on "techniques", all codes and custom tools must be displayed next to the work"*. https://scriptedbypurpose.wordpress.com/ (consulté le 17/08/2021).

volontairement ancré dans la pratique de la programmation. La retranscription partielle du code dans expositions et magazines ne permet pourtant pas vraiment de comprendre véritablement les algorithmes présentés ou de les reproduire. Sur les panneaux d'exposition s'affiche souvent à peine quelques lignes, ou tout juste la retranscription complète d'une ou deux des fonctions clés. Mais ni la relation entre ces fonctions, ni les quelques centaines, voire quelques milliers de lignes du code complet ne sont montrées. Il s'agit donc bien essentiellement d'une esthétique, d'un marqueur qui permets aux praticiens de s'identifier au sein d'un territoire commun[59].

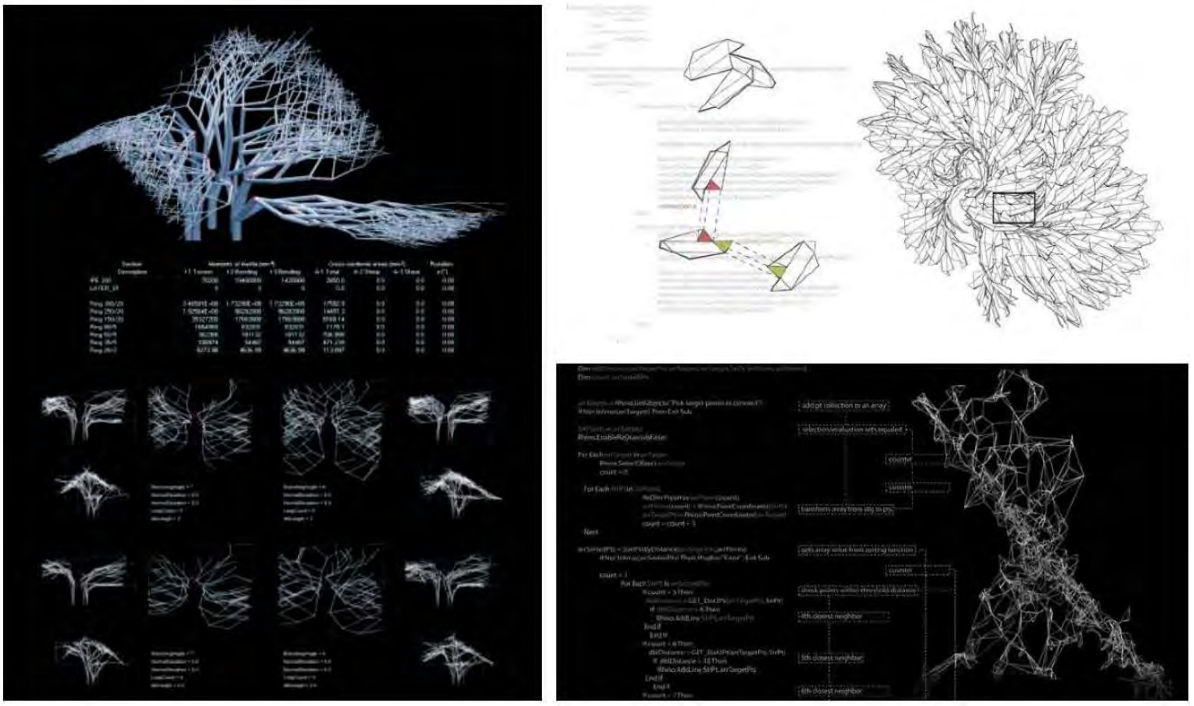


*Figure 11. a. Archiglobe avec Bollinger + Grohmann ; b. supermaneuvre ; c. kokkugia*

### 4.2 Une approche commune : la programmation heuristique

Ce qui caractérise cette recherche prospective, c'est aussi qu'elle explore précisément ce qui caractérise les pratiques computationnelles en architecture : une approche heuristique de la programmation. En philosophie des sciences, l'heuristique est ce qui a pour objet les procédés de

[59] On peut y voir un clin d'œil au partage open-source qu'on trouve en ligne et qui est aussi un marqueur de la pratique de cette branche du champ, comme on va le voir plus loin.

recherche et les règles qui les guident[60]. Par extension, la méthode heuristique désigne une méthode au cours de laquelle il s'agit de procéder par étapes successives pour voir ce qui fonctionne le mieux, plutôt que d'examiner le problème dans son ensemble dans un premier temps et de déterminer en amont la marche à suivre la plus pertinente, qui garantira un résultat. La notion de méthode heuristique fait donc appel aux connaissances tacites de la personne qui la met en œuvre. Lorsqu'un praticien accumule un certain savoir-faire dans sa discipline, il devient capable de faire appel à ce savoir-faire pour prendre des décisions au fil de l'eau, sans forcément pouvoir expliciter sa méthode pour avancer dans la tâche en cours. Il est généralement considéré qu'architectes et designers font systématiquement appel à la méthode heuristique au cours de leur processus de conception, procédant par tâtonnements et versions successives afin de déterminer la forme idéale de l'objet ou de l'espace en cours d'élaboration[61]. La méthode heuristique combine donc la mobilisation de connaissances tacites à un processus de travail par essais et erreurs. Les praticiens du champ computationnel font justement usuellement appel à une méthode heuristique pour déterminer les programmes et les valeurs exactes des variables qui vont leur permettre de donner forme à leur intention architecturale. C'est dans cette capacité à réadapter les algorithmes pour les orienter vers des résultats pertinents pour la conception spatiale que se niche le potentiel des outils computationnels pour l'architecture. Bien que cela soit particulièrement clair chez les praticiens qui s'intéressent à la dimension prospective, car nombre des règles de la conception spatiale de l'industrie de l'architecture ne s'appliquent pas à leur production, c'est une dimension qu'on trouve en fait chez tous les praticiens, y compris ceux dont l'objectif est plutôt d'industrialiser les outils algorithmiques. Cette dimension préside à l'établissement d'un véritable savoir-faire sur ce que c'est que programmer l'architecture, savoir-faire qui est une caractéristique essentielle du champ computationnel en architecture. Avant de tenter de décrire ce qui fait exactement ce savoir-faire et les modalités de cette pratique heuristique de programmation de l'architecture, un projet en particulier va nous servir de cas d'étude pour cette question : le concours du pavillon Seroussi.

### *4.2.1 Le concours du pavillon Seroussi*

Se déroulant en 2006-2007 - depuis le moment où les organisateurs ont commencé à le planifier jusqu'à la fin de l'exposition - en France, le concours est organisé à un moment clé du tournant computationnel en architecture[62]. Il accompagne la série d'expositions évoquées plus haut, en France

[60] ref
[61] ref
[62] L'étude consacrée au Pavillon Seroussi qui suit a été publiée dans Gaudillière, Nadja Gaudillière, "Towards an History of Computational Tools in Automated Architectural Design - The Seroussi Pavilion Competition as a Case Study", dans M. Haeusler, M. A. Schnabel, T. Fukuda (eds.), *Intelligent & Informed – Proceedings of the*

et ailleurs, l'augmentation du nombre de praticiens et leur structuration en champ de recherche à part entière, mais aussi les incursions en agence des pratiques computationnelles. L'intérêt du concours du Pavillon Seroussi en tant qu'étude de cas, au-delà du fait qu'il se déroule à ce moment clé, réside également dans le fait que les six agences participantes sont alors particulièrement renommées au sein du champ. Elles font par ailleurs toutes partie de cette même génération de praticiens dont ce chapitre fait le récit, de ces petites agences expérimentales qui caractérisent la branche prospective. Seule Gramazio/Kohler s'inscrit dans l'autre branche. Le concours est néanmoins emblématique de la pratique de cette génération à travers ce qu'il montre du savoir-faire de ces praticiens, quelle que soit leur branche de pratique. En outre, le concours représente un cas d'étude idéal puisque son cahier des charges offre un environnement de travail particulièrement adapté pour développer la recherche sur la conception architecturale computationnelle. Les agences invitées sont toutes assez jeunes, avec des profils professionnels particulièrement homogènes, des dates de fondation similaires, des similitudes dans la production antérieure et dans les moyens d'intégrer la recherche sur les outils informatiques dans la pratique, ainsi que dans leur relation aux outils. Les architectes dirigeants de chaque agence ont également des trajectoires similaires, avec une implication dans les environnements académiques et dans l'enseignement de l'architecture - parfois ensemble.

Formée à l'Université de Zagreb et à l'Université de Columbia, d'où elle sort en 2000, Alisa Andrasek fonde biothing en 2001. Elle enseigne en parallèle à Columbia et à l'Institut Pratt, puis à la AA. biothing est un "*laboratoire de conception/computation dont les recherches se concentrent sur le potentiel génératif des systèmes computationnels physiques ou numériques",* au sein duquel Alisa Andrasek fait régulièrement appel à d'autres praticiens, notamment pour l'écriture des algorithmes[63]. C'est le cas pour le pavillon Seroussi, dont le code est développé par Ezio Blasetti et Tobias Schwinn. EZCT Architecture & Design Research a elle été fondée en 1999 par Philippe Morel, Félix Agid et Jelle Feringa. Les deux premiers, architectes, enseignent à l'ENSA Paris-Malaquais, et, pour Philippe Morel à la AA, à la Bartlett et à la TU Delft. Le troisième, artiste, intervient également à la TU Delft. L'algorithme du pavillon Seroussi est programmé à trois, entre Mathematica et Python, avec l'aide de Marc Schoenauer, un mathématicien qui a déjà collaboré avec l'agence au cours des précédents projets. Fabio Gramazio et Matthias Kohler se sont rencontrés pendant leurs études d'architecture à l'ETH Zurich, et commencent à y enseigner en 1996. L'école devient à partir de la le lieu de toutes leurs expérimentations sur les processus de fabrication numérique. Leur proposition pour le pavillon Seroussi, extension architecturale de ces expérimentations, est faite au sein de l'agence qu'ils créent

*24th CAADRIA Conference - Volume 2*, Victoria University of Wellington, Wellington, New Zealand, pp. 581-590, 2019.

[63] Elias Guenoun (éditeur), *Pavillon Seroussi : biothing, DORA (Design Office for Research and Architecture), EZCT Architecture & Design Research, Gramazio & Kohler, IJP - George L. Legendre, Xefirotarch*, Orléans : Éditions HYX, 2007.

en 2000 pour accompagner leur pratique professionnelle. DORA, devenue labDORA, est l'agence de l'architecte Peter Macapia depuis 2001. Diplômé de Columbia, il y enseigne ainsi qu'à l'Institut Pratt. L'agence présente ses projets comme relevant essentiellement de l'intégration dans les pratiques computationnelles de modélisations numériques issues du génie civil, et sa vision de l'architecture comme relevant d'une "*collection de techniques*"[64]. Les projets, notamment celui du Pavillon Seroussi, font néanmoins la part belle aux recherches formelles et aux références à des architectures antiques. IJP Coroporation a été fondée par George Legendre, architecte français, en 2003, après deux années de pratique et huit d'enseignement à Harvard, à Princeton et à la AA. En complément des projets de l'agence, George Legendre publie beaucoup, en particulier le numéro d'AD *Mathematics of Space* et l'ouvrage *The Book of Surfaces*, qui présente la méthode employée par l'agence dans ses projets. Celle-ci se sert d'équations mathématiques simples pour définir des surfaces, et construire ensuite tout le projet à partir de cette simplicité mathématique. Enfin, Xefirotarch est l'agence de Hernan Diaz Alonso, architecte argentin, directeur de Sci_Arc et intervenant régulier à l'Université de Columbia. Connue pour les formes torturées des structures qu'elle imagine, Xefirotarch est à l'origine de plusieurs objets de design et de mobilier urbain. Le projet proposé pour le Pavillon Seroussi s'inscrit dans le développement de cette patte visuelle, qui vaut à Hernan Diaz Alonso d'être le rédacteur en chef invité d'AD pour le numéro *Exuberance* en 2010. Les six agences font partie de celles régulièrement invitées dans les expositions mentionnées plus haut, et leurs fondateurs évoluent depuis le début de leurs carrières dans les institutions du réseau. Il est également intéressant de noter la présence d'autres architectes du champ dans les équipes, comme Ezio Blasetti, Roland Snooks ou Dave Pigram, car cela indique une forte insertion du concours du pavillon Seroussi dans le réseau de la recherche architecturale computationnelle des années 2000-2020. En effet, ces architectes, un peu plus jeunes, ont par la suite joué un rôle clé dans le champ, et le concours Seroussi compte parmi leurs premières incursions dans le domaine.

L'objectif du concours est de construire un pavillon qui abrite une partie de la collection d'art de Natalie Seroussi, galeriste et collectionneuse parisienne, dans des espaces d'exposition dédiés ainsi que des espaces de vie. Outre ce double programme architectural, l'autre particularité du concours est le site du projet, un contexte particulièrement fort et spécifique : le pavillon doit être construit dans le jardin de la maison d'André Bloc, alors propriété de Natalie Seroussi. La villa, située à Meudon, a été conçue et construite en 1949 par l'architecte et ingénieur pour lui-même, et le jardin accueille également deux des sculptures-habitacles de Bloc, ajoutées en 1964-1966 (figure 12). Une zone spécifique du domaine a été sélectionnée pour le pavillon et est mentionnée dans le dossier du

[64] Elias Guenoun (éditeur), *Pavillon Seroussi : biothing, DORA (Design Office for Research and Architecture), EZCT Architecture & Design Research, Gramazio & Kohler, IJP - George L. Legendre, Xefirotarch*, Orléans : Éditions HYX, 2007.

concours, avec des commentaires sur les spécificités du site, à prendre en compte par les cabinets d'architectes participants. Le dossier comprend également un programme détaillé :

"*Le pavillon doit abriter un appartement entièrement équipé tout en offrant des espaces d'exposition pour la collection d'art contemporain de Natalie Seroussi. Les architectes [sont] invités à porter une attention particulière à la question de la présentation des œuvres d'art contemporain, notamment à la diversité des supports concernés (peinture, sculpture, installation et vidéo). Ces dispositifs d'exposition devront être répartis dans l'ensemble du pavillon. Une telle organisation spatiale devra permettre de multiplier les modes d'appréciation des objets et de créer une véritable intimité avec les œuvres exposées.*

*Le projet comprendra un espace intérieur et un espace extérieur. La surface habitable totale du projet sera d'environ 350 m².*

*L'intérieur comprendra 2 chambres, chacune avec un dressing, 1 salon/salle à manger, 1 cuisine, 1 salle de bain, 1 bureau (optionnel - peut être inclus dans une des chambres), 1 toilette (à l'extérieur du sous-sol), 1 sous-sol aménagé pour des projections vidéo, 1 espace de stockage pour des œuvres d'art entre 20 et 30 m². Chacun de ces espaces comprendra, dans la mesure du possible, un espace de stockage.*

*Bien que défini avec précision, le programme reste ouvert, permettant ainsi aux participants de modifier certains aspects en fonction des intentions de leur projet*"[65].

Deux propositions, la première par EZCT Design & Architecture Research et la seconde par IJP, sont désignées comme gagnantes par le jury, présidé par Claude Parent et composé de Niccolo Baldassini, Christian Benilan, Andrea Branzi, Marcel Brient, Alain Bubleix, Bart Lootsma et Teri When Damish. Mais ces deux propositions gagnantes ne sont en fait pas construites comme prévu initialement, Natalie Seroussi ayant renoncé à son projet. EZCT Architecture & Design Research est néanmoins invité l'année suivante à fabriquer une cloison pour la résidence principale de la galeriste. Peu après le concours, de juin à septembre 2007, une exposition est cependant organisée à l'espace parisien La Maison Rouge, présentant les six propositions. Cette exposition voyage ensuite dans plusieurs lieux en Europe et aux Etats-Unis.

---

[65] Elias Guenoun (éditeur), *Pavillon Seroussi : biothing, DORA (Design Office for Research and Architecture), EZCT Architecture & Design Research, Gramazio & Kohler, IJP - George L. Legendre, Xefirotarch*, Orléans : Éditions HYX, 2007.

*Figure 12. Jardin de la Villa Bloc*

biothing crée pour l'occasion le programme FlowerPower, écrit en Python comme un module complémentaire de Rhinocéros. Ce programme trace des courbes qui correspondent à la rencontre de champs magnétiques émanant de différents points placés dans le plan. Ces courbes, qui forment des sortes de fleurs autour des points, sont ensuite transformées en arches qui se déploient en 3D. Le point défini préalablement, qui joint toutes les courbes, devient le plus haut de l'arche. Le programme permet ainsi de former des voûtes qui couvrent les espaces du pavillon. Les arches des fleurs sont ensuite reliées les unes aux autres par des arêtes permettant de former une couverture continue. Le pavillon offre un plan ouvert, avec les différents espaces demandés par le brief répartis autour d'une cour intérieure. Le placement des points se fait en fonction de l'emplacement des espaces dans le plan, déterminé en amont. Une fois ceci fait, les paramètres définissant la force des différents champs magnétiques et la hauteur des arches peuvent également être ajustés pour modifier la géométrie de la couverture du pavillon.

EZCT Architecture & Design Research prend pour point de départ de sa proposition une répartition équitable de la lumière pour que les œuvres, réparties un peu partout dans le pavillon, soient toutes dans de bonnes conditions d'exposition. Le programme écrit pour l'occasion part d'un bloc parallélépipédique divisé en volumes plus petits. Pour créer les espaces du pavillon, le bloc est creusé

en supprimant certains de ces volumes, des briques à la géométrie particulière, dite *voronoi*. D'autres de ces volumes, aux quatre coins et en bordure, ne peuvent être touchés au cours du processus, pour garantir que les descentes de charge puissent se faire correctement. D'autres, à l'intérieur du bloc, sont également verrouillés, pour permettre l'implémentation du plan prévu à l'intérieur, et garder des partitions entre les espaces. Une fois le bloc creusé, la répartition intérieure de la lumière est évaluée à partir d'un modèle de ciel de la région de Meudon. Le programme génère un grand nombre de blocs creusés différents, cherchant la configuration permettant d'obtenir la meilleure répartition de lumière[66] . Une fois celle-ci trouvée, la géométrie finale du bloc est affinée : plutôt que de conserver les briques voronoi, le volume est transformé en un ensemble de voiles imbriquées les unes dans les autres. Les vides également sont transformés en surfaces courbes, mais plutôt que des surfaces pleines, ce sont des surfaces vitrées, pour laisser passer la lumière. Ces vitres sont en fait des vitraux, aux motifs inspirés de modèles d'évaluation structurelle.

Le pavillon de Gramazio & Kohler est un parallélépipède constitué une dalle de toit en béton entourée d'une façade de verre ponctuée de colonnes, et des cloisons de brique ajourées en retrait de cette façade, qui permettent d'enclore l'espace et de soutenir la dalle. Les cloisons intérieures ondulent dans le plan, formant un cœur en brique au centre du pavillon de verre. Les deux architectes parlent de ce dispositif comme une référence à Ludwig Mies Van Der Rohe et à sa maison de verre. Le plan du pavillon est lui aussi un classique de l'architecture moderne puisqu'il concentre les espaces techniques – salle de bain, cuisine, espaces de stockage - au centre, à l'intérieur du cœur de briques, et les espaces de vie entre les cloisons du cœur et la façade de verre. Cette répartition fonctionnaliste n'est pas sans rappeler Mies Van Der Rohe, Le Corbusier ou Louis Kahn. La dalle du toit est ajourée et éclaire le cœur en créant des jeux de lumière entre ses perforations et celles des cloisons de brique. Elle est aussi ondulée au contour des cloisons de briques, trahissant l'intérieur du pavillon lorsque vue de haut, et conçue pour être visible depuis le haut de la sculpture-habitacle d'André Bloc qui se trouve à quelques mètres du site ou le pavillon doit être construit. Le pavillon est donc formé d'un ensemble de géométries très simples, habillées par les briques ajourées dont l'assemblage avec un robot six-axes est au cœur des expérimentations de Gramazio & Kohler.

IJP Corporation place la manipulation d'équations mathématiques à la racine de sa pratique. L'agence met également en avant la notion d'idéogrammes, de symboles manipulés et assemblés. On retrouve les deux idées dans le pavillon conçu pour le concours Seroussi, une composition par collage de résultats mathématiques. Le projet repose sur une seule famille d'équations, dont la manipulation

[66] Il s'agit d'un algorithme génétique, dont nous décrirons le principe de fonctionnement en détail un peu plus loin dans le chapitre.

permet de définir des formes diverses - plans, portions de cylindre ou surfaces à simple ou double courbure. Cette bibliothèque de formes est ensuite placée dans l'espace et combinée par des opérations booléennes pour former le pavillon. La famille de fonctions manipulées est la famille des fonctions mathématiques dites périodiques, dont les plus connues sont les fonctions sinusoïdales. Pour placer ces formes dans l'espace, le projet s'appuie sur deux repères. Le premier est le plan du pavillon, prédéterminé. Le second est une grille verticale fournissant des repères de dimension. Cette grille verticale se traduit ensuite dans la formalisation du pavillon par une stratégie de fabrication qui repose sur l'empilement de pans d'acier découpés pour reconstituer les formes issues des fonctions périodiques. Selon les espaces du plan, le jeu avec les fonctions périodiques permet de créer des formes plus ou moins complexes selon que le projet propose d'utiliser l'espace pour exposer des œuvres ou comme espace de vie. En particulier, la complexité d'un assemblage de cloisons courbes dans le salon, baptisé "fleur périodique", est mise en dialogue avec des murs droits concus selon le principe muséographique du "cube blanc" dans la salle à manger. L'ensemble du pavillon est issu de fonctions périodiques, et la variété des formes en résultant est justement au centre de la proposition, qui ambitionne d'offrir l'exploration d'un continuum mathématique du simple au complexe.

La proposition de labDORA vise à articuler un processus en deux phases, l'une générative et l'autre analytique, qui permettent de manipuler seulement des éléments très simples - points, lignes et plans -, et d'en tirer une forme de coque. Le processus algorithmique démarre avec la création d'un nuage de points répartis au hasard dans l'espace. Ces points sont ensuite reliés les uns les autres au hasard pour former des triangulations. Au hasard ou presque, puisque une partie des points est reliée pour former deux plans flottant au centre du nuage de points. Le maillage obtenu grâce à l'opération de triangulation est évalué en recourant à une analyse structurelle. Diverses configurations sont évaluées pour identifier la plus performante. Le processus permet ainsi d'obtenir un maillage triangulaire continue qui forme l'enveloppe et les sols du pavillon. Ce maillage triangulaire est le support d'une coque ajourée au sein de laquelle les planchers flottent. Les points sont donc le nuage des particules dans l'espace, les lignes qui les relient sont les segments qui composent le maillage, et les plans sont les niveaux de sol du pavillon proposé. labDORA a néanmoins fait le choix, plutôt que de programmer l'ensemble de l'algorithme imaginé, d'en expliquer les principes par le biais de schémas explicatifs, puis de dessiner le pavillon comme un résultat possible de cet algorithme. La géométrie proposée du pavillon n'est cependant pas le résultat direct d'un processus algorithmique. Ayant vérifié en consultant des collègues informaticiens que les principes envisagés fonctionnaient, labDORA a préféré passer du temps sur la production de dessins plutôt que sur l'implémentation de l'algorithme. En particulier, la forme donnée à la coque du pavillon sur les dessins est le résultat d'une référence au dôme de Florence, que Peter Macapia a préféré développer plutôt que se perdre dans un travail de

programmation trop fastidieux à ses yeux. L'idée de cette double coque qui enferme les deux planchers vient d'une analogie avec la structure du dôme de Florence, et sur la coupe du pavillon proposé par labDORA, le placement de ces différents niveaux rappelle en effet les contreforts plans visibles entre les deux coques sur les coupes du dôme.

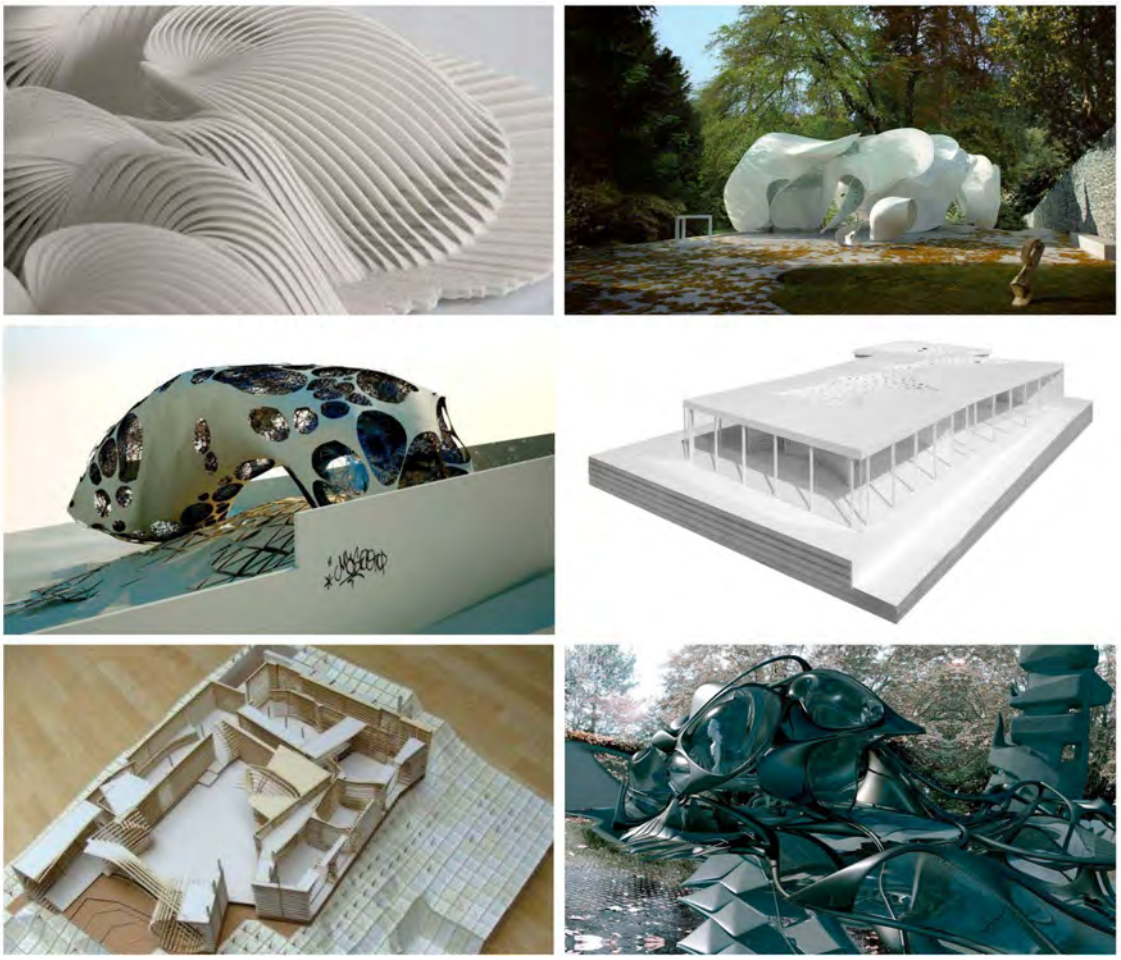


*Figure 13. Les six réponses au concours du pavillon Seroussi*

Enfin, Xefirotarch propose un projet composé d'un noyau central et de trois espaces répartis autour. Le noyau regroupe les espaces de vie : salon, salle à manger, cuisine, terrasse au rez-de-chaussé, chambres à l'étage. Les espaces annexes sont des galeries d'exposition. Chacun de ces quatres éléments est formé par un espace courbe, et des filaments les relient les uns aux autres. La géométrie de ces filaments est définie par un algorithme qui fait se déplacer des points d'un espace vers un autre. La trajectoire de ces particules fournit une courbe qui sert de base au dessin de ces filaments. L'algorithme employé fait partie d'une famille d'algorithmes qui permettent d'obtenir des formes

particulièrement alambiquées. En recourant à cet algorithme, Xefirotarch vise précisément à exploiter la complexité formelle rendue accessible par la manipulation d'outils de programmation. La proposition faite par l'agence pour le pavillon Seroussi s'inscrit de ce point de vue parfaitement dans sa pratique habituelle : un exercice formel à la recherche d'une nouvelle esthétique permise par le numérique.

*4.2.2 Les outils de programmation, des outils comme les autres*

Les processus de conception des propositions montrent tous une étape à laquelle les éléments architecturaux de la proposition n'ont pas été générés par l'ordinateur mais sont conçus et dessinés par l'équipe. Dans plusieurs cas, c'est le plan de la proposition qui a été dessiné en amont (Tab. 1). A ce moment-là, l'algorithme est déjà en cours de développement, et les morphologies qu'il génère sont prises en compte lors du dessin du plan. Mais le plan est cependant dessiné sans recours à des outils algorithmiques. Par la suite, dans une deuxième phase, le plan est introduit dans l'algorithme comme point de départ. C'est le cas pour le projet de biothing. L'autre possibilité est qu'une série de formes soient générées à l'aide de l'algorithme et ensuite insérées manuellement dans le plan précédemment dessiné pour créer la proposition architecturale elle-même. Le projet d'IJP Corporation en est un exemple. La mise en relation du plan dessiné et du processus principal de conception algorithmique est par ailleurs suivie dans chaque cas d'une série d'étapes complémentaires visant à affiner la proposition architecturale jusqu'à son dessin final. C'est par exemple ce qui se passe dans le projet d'EZCT. Les processus de conception mis en place par les praticiens pour le pavillon Seroussi montrent la place qu'occupent les outils de programmation au sein d'une démarche de conception architecturale. Celle-ci est plus large que la simple exécution d'un programme. Les outils algorithmiques ne sont donc pour ces praticiens qu'un élément parmi d'autres mobilisés pour le dessin d'une proposition finalisée. Une part importante de la prise de décision relative à la conception de l'espace se produit séparément dans le cas des propositions du pavillon Seroussi. Un autre exemple est la description des bibliothèques de kokkugia et leur utilisation en deux étapes dans la pratique : d'abord, concevoir des bibliothèques d'algorithmes en explicitant les instructions et ensuite, choisir la bibliothèque en fonction des résultats connus du comportement et d'une intuition concernant un projet spécifique, pour construire un programme sur mesure basé sur celle-ci. Cette imbrication du développement de l'algorithme et de la prise de décision architecturale externe est en fait caractéristique des expérimentations des praticiens de cette génération.

| | | Practices | | | | | |
|---|---|---|---|---|---|---|---|
| | | biothing | labDORA | ESCT Architecture & Design Research | IJP Corporation | Gramazio & Kohler | Xefirotarch |
| Algorithm | Family | Formal Research | Performance-driven design | Performance-driven design | Formal Research | Fabrication-driven design | Formal Research |
| | Type | Attractors (agent-based; magnetic fields) | Finite Element Analysis; Particle System Optimization | Genetic Algorithm | Periodic Equation | Robotic Brick Placing | Growth |
| | Whole or split | Whole | Split / Not related | Split / Related | Whole | Split / Related ?? | Whole ?? |
| | Main Technical Objective | Global Shape Generation | Structural Optimisation | Light Optimisation | Global Shape Generation | Detailed Shape Generation | Global Shape Generation |
| | Secondary Technical Objectives | / | Global Shape Generation, Detailed Shape Generation | Structural Optimisation, Global Shape Generation, Detailed Shape Generation | Mathematical Sobriety | Fabrication files generation | / |
| | Output | "Flower" Shells (Roofing) | Shell Shape and Structural Elements | Main Shell Shape, Structural Pattern for Surface Subdivision | Modules | Wall Shape and Bricks Placement | ? |
| | Significant Variables | Position of Attractors | Double Shell Structure | Shape of retrieved blocks | Periodic Equation / Cylinder | Robotic Placing Constraint | ? |
| Technical Set-Up | Software | Rhinoceros + FlowerPower | Rhinoceros, Finite Element Analysis Software | Rhinoceros, Mathematica, Radiance, Maya, Blender, VTK, Polytrans, Getfem++, 3Dmax, Vray, EO, Catia | MathCAD | ? | ? |
| | Programmation Language | C#* | C#* | Python, C++, Wolfram | Mathematica | ? | ? |
| | Interface type | Plug-in with slider parameters | / | Scripting | Scripting | ? | ? |
| | Graphic Representation Tool for Base Geometry Visualisation | Rhinoceros | Rhinoceros | Rhinoceros | AutoCAD | ? | ? |
| | Final Output Type | 3D + 2D Drawings | 2D Drawings | 3D + 2D Drawings | 2D Drawings | 3D + 2D Drawings + Fabrication File | 3D + 2D Drawings |
| Client | Date of Office Creation | 2001 | 2001 | 2000 | 2000 | 2003 | 2001 |
| | Number of people (Team + Consultants) | 7 + 1 | 17 + 7 | 11 | 10 | 12 | 10 |
| | Background of principal | Architecture | Architecture | Architecture | Architecture | Architecture | Architecture |
| | Background of Team | Architecture, Computer Science | Architecture | Architecture, Computer Science, Mathematics, Daylight Modelling, Structural Engineering, Visual Rendering | Architecture | Architecture | Architecture |
| | Background of Consultants | Mathematics, Visual Rendering | Architecture, Computational Design, Structural and Environmental Design | / | Landscape Design | Structural Engineering | / |
| Architectural Design | Drawn Architectural Design Independant of the Algorithmic Output | Yes (plan)* | Yes (structural principle) | Yes (plan)* | Yes (plan) | Yes (plan) | |
| | Site-Related items (verbal mention, written mention, site plan, sculptures present on drawings) | 2 | 2 | 4 | 3 | 3 | 4 |
| | Brief compliance (program) | Partial | No | Partial | Yes | Partial | Partial |
| Available documents | Site Plan | Y | Y | X | Y | Y | Y |
| | Ground Plan | Y | X | Y | Y | Y | Y |
| | Drawings (Sections, Renders) | Y | Y | Y | Y | Y | Y |
| | Code | X | X | X | Y | X | X |
| | Diagram/illustration of algorithmic method | Y | Y | Y | Y | X | Y |
| | Credits | Y | Y | Y | Y | Y | Y |
| | Interviews | Y | Y | Y (informal) | Y | Y | X |

*Tableau 1. Analyse des projets du concours Seroussi*

Plusieurs commentaires formulés au cours des entretiens soulignent cet aspect. Au cours de l'entretien avec Jose Sanchez notamment, il décrit un écart existant encore dans la conception algorithmique architecturale, en opposition à l'objectif souvent décrit d'unir chaque étape de la conception au moyen d'outils numériques[67]. Cet écart est décrit comme le moment de figer le projet, de choisir parmi les variations autorisées par les paramètres de l'algorithme un résultat à finaliser[68]. Mais plus largement, Sanchez décrit également la difficulté à faire correspondre complètement l'intention architecturale initiale et le résultat produit par l'algorithme. C'est bien la difficulté de la traduction qu'il souligne à travers la description de cet écart, de ce saut à effectuer à un moment donné du processus[69]. Nous retrouvons ici la question de la traduction des contraintes architecturales et de la prise de décision en

[67] Entretien Jose Sanchez

[68] Entretien Jose Sanchez

[69] Rappelle ce saut, cette difficulté du passage à la réalité que nous avons vu dans le Chapitre I.

langage algorithmique, dont nous avons émis l'hypothèse qu'elle est au cœur des pratiques du champ computationnel. Il est d'ailleurs intéressant de noter que dans l'introduction du catalogue de l'exposition du Pavillon Seroussi, Elias Guenoun fait également mention de la notion de "*traductibilité générale*" comme clé de la conception computationnelle[70]. À l'inverse de Jose Sanchez, d'autres praticiens sont beaucoup plus à l'aise avec l'existence de cet écart, et présentent ce saut comme constitutif de leur pratique. L'imbrication de la programmation et de la décision architecturale externe est chez eux beaucoup plus fluide. Cette fluidité s'observe en particulier dans la réutilisation des programmes. Ceux-ci sont développés comme des outils techniques de génération de l'espace, et donc utilisables dans d'autres projets. C'est le cas de FlowerPower, développé par biothing pour le pavillon Seroussi mais ensuite intégré à Genware, puis reprogrammé en Java pour l'utiliser dans d'autres projets. C'est aussi le cas pour le programme de DORA, dont on retrouve l'habillage ajouré de l'enveloppe dans plusieurs autres projets de l'agence dans les années qui suivent sa participation au concours[71]. Cette réutilisation des algorithmes est aussi observée chez supermanoeuvre ou chez kokkugia. Dans le même esprit que biothing avec son projet Genware, ils constituent au fil des années des bibliothèques d'algorithmes, ou plutôt de scripts, c'est-à-dire de bouts de codes. Ces bouts de codes sont ensuite réassemblés à l'envi en nouveaux programmes pour chaque nouveau projet dans lequel l'agence se lance. Iain Maxwell ou Roland Snooks décrivent précisément leurs bibliothèques comme des outils à mobiliser en fonction des projets et des « effets » recherchés pour le projet[72]. Dans l'explication qu'ils donnent de leur usage des outils computationnels, ils assument largement cette sélection comme relevant d'un processus heuristique guidé par leur savoir-faire architectural. La combinaison de décisions architecturales et de la programmation et la réutilisation des outils participent ainsi d'une intégration des outils computationnels à un processus de conception plus large comme n'importe quel autre outil de l'arsenal des architectes, particulièrement visible dans le concours du pavillon Seroussi.

Le dossier architectural, et plus particulièrement le programme du pavillon, n'est pas complètement respecté dans les propositions, voire dans le cas de DORA pas du tout. Cette négligence s'explique par la rareté de l'occasion offerte à l'époque par le concours d'expérimenter la conception computationnelle. Chaque agence s'en saisit pour favoriser l'exploration des possibilités formelles et architecturales qui peuvent en découler. Cette négligence explique cependant aussi que les propositions gagnantes n'aient pas été construites, Natalie Seroussi ayant été aussi surprise que déçue

[70] Sans aller plus loin. Elias Guenoun (ed.), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.
[71] Par exemple la Combinatorial Tower de 2012.

[72] Entretien Roland Snooks. Entretien Iain Maxwell.

de ce mépris pour les besoins exprimés dans le brief[73]. Malgré cette indifférence, une profonde préoccupation architecturale transparaît dans chaque proposition et dans les commentaires faits sur le concours au cours des entretiens. L'importance du site, qui abrite la villa d'André Bloc et surtout ses *sculptures-habitacles*, est mentionnée par tous à plusieurs reprises[74]. La force visuelle et historique de ces œuvres est soulignée par la plupart des architectes au cours des entretiens, et elles apparaissent sur la plupart des dessins. La proposition d'EZCT est particulièrement intéressante à cet égard, car la forme des blocs manipulés dans l'algorithme initial, qui informe la géométrie du pavillon final, semble une référence formelle directe aux sculptures environnantes. Le travail sur les vitres, avec ses motifs structurels, peut par ailleurs être compris comme une référence aux pratiques du champ computationnel, formant avec l'esthétique des blocs un doublé intéressant. Chez Gramazio et Kohler, le choix des briques s'explique par l'ambition de créer un dialogue formel entre leurs briques et la maçonnerie des sculptures-habitacles. Si la relation entre ces formes est moins puissante que chez EZCT, moins évidente dans les dessins du projet, l'ambition est clairement attestée dans le texte qui accompagne la proposition. Ces sculptures qu'on voit apparaître au détour de la plupart des dessins attestent du fait que malgré l'importance prise par le code comme élément de la mise en scène esthétique du projet, les plus marquants des projets du champ computationnel font rarement l'impasse sur la représentation architecturale consensuelle. C'est par ces représentations classiques qu'est mise en évidence l'importance du contexte offert par la Villa Bloc dans le cas du concours Seroussi. Ce retour aux représentations classiques de l'architecture s'explique par l'exercice somme toute lui-même classique de concours qu'est le projet Seroussi. Mais cela témoigne aussi de l'attention à cette dimension portée par les praticiens, à rebours des reproches souvent fait au champ computationnel de s'en éloigner au profit d'expérimentations technologiques peu ancrées[75]. Ces représentations classiques sont observées plus généralement chez les praticiens qui cherchent à sonder les configurations spatiales autorisées par la conception computationnelle. Ce sont celles qui permettent le mieux de les visualiser pour un praticien formé à l'architecture à la lecture de ces plans, coupe et élévations. A l'inverse, elles tendent à disparaître chez ceux qui choisissent de se concentrer sur le processus technique plutôt que sur son résultat architectural.

En complément du rôle joué par le contexte géographique, le contexte historique conserve lui aussi une certaine importance, à travers les références intégrées par les praticiens. EZCT choisit de proposer

[73] Entretien Félix Agid.

[74] Citations entretiens Seroussi.

[75] D'autres projets du champ méritent néanmoins cette critique ; mais justement pas ici – une différence qui sera commentée plus avant dans la suite du texte. On retrouve par ailleurs l'affection portée par Christopher Alexander à cette question du contexte dans *Notes on the Synthesis of Form*.

en complément de la prise en compte du site une référence aux pratiques computationnelles, mais d'autres praticiens choisissent d'intégrer à leur projet des références plus conventionnelles, témoins de l'importance de l'histoire de l'architecture dans leurs pratiques. Le projet de labDORA est un clin d'œil au dôme de Florence et à sa prouesse technique permise par la double coque qui le constitue. Cette référence peut être lue comme un rappel de l'idée de Peter Macapia que l'architecture est une « *collection de techniques* ». La proposition en combine en effet deux entre elles, cette double coque et l'usage d'outils computationnels pour parvenir au dessin de ses motifs ajourés. Chez Gramazio et Kohler, la façade, le plan et la toiture intègrent des références à Mies et à l'architecture moderne. La même référence aux mouvements modernes est faite par supermanoeuvre, dans le cadre du projet Living Morphologies. Celui-ci est une réinterprétation de l'Unité d'Habitation de Le Corbusier à Marseille, dans laquelle l'agence réinterroge la répartition des circulations dans le bâtiment à l'aide d'un algorithme permettant de modéliser la trajectoire des habitants à l'intérieur.

La prise en compte du contexte géographique et historique que l'on observe dans le projet Seroussi et ailleurs est un indice de plus de la manière d'aborder la conception architecturale des praticiens du champ computationnel à cette génération. Ce sont par ailleurs deux éléments considérés comme constitutifs de la pratique architecturale classique, dont l'importance est soulignée par de nombreux praticiens en dehors du champ computationnel. Leur importance perdure cependant au sein du champ, rarement mise en avant puisque ce sont plutôt les pratiques algorithmiques qui sont au centre du discours, mais bien présentes néanmoins. Plusieurs aspects de l'analyse menée mettent en évidence l'expertise architecturale mise en œuvre, de manière parfois plus traditionnelle qu'on pourrait l'attendre. Au-delà de la tension entre l'exploration des outils algorithmiques et la conception d'une proposition architecturale, ce qui apparaît également, c'est l'évolution de la profession. Les différentes équipes présentent des acteurs traditionnels, comme un chef de projet et des bureaux d'études. Mais elles comptent aussi d'autres acteurs spécifiques au développement de la conception computationnelle, comme des programmeurs ou un spécialiste de la modélisation de la lumière du jour. En outre, les architectes impliqués ont tous développé des compétences spécifiques à la conception algorithmique. Les organisations de groupe, dans le cas du pavillon Seroussi mais aussi dans les autres projets du champ computationnel, alternent donc entre des configurations classiques et innovantes.

L'observation de la démarche des architectes dans le cadre du pavillon Seroussi dévoile des caractéristiques qui relèvent d'une pratique somme toute classique de la conception architecturale. Les architectes y emploient des modes de représentations, de référencement et d'organisation usuels, et la mobilisation de l'expertise architecturale est elle aussi standard. Ces caractéristiques sont aussi présentes chez d'autres praticiens, qui accompagnent les projets les plus aboutis architecturalement, et

semblent donc constitutives de la démarche architecturale avec des outils computationnels. Cette pratique est classique dans le sens ou elle est similaire aux démarches de conception qui existent en dehors du champ computationnel. Elle est aussi classique au sens proposé par Peter Eisenman dans son article *The End of the Classical: The End of the Beginning, the End of the End*[76]. Il y définit la pratique classique non comme celle du mouvement dit classique en architecture, mais comme celle partagée par tous les architectes depuis l'existence de la profession. Cet article a eu un certain retentissement parmi les déconstructivistes et leurs héritiers du champ computationnel parce qu'il clame la nécessité d'aller au-delà de ces pratiques classiques. Force est de constater que le champ computationnel maintient cependant tout de même une part de classicisme dans ses processus de conception. Cette dimension n'est d'ailleurs pas toujours assumée par les praticiens de cette génération, chez qui s'observe un fort écart entre discours et pratiques à ce sujet. À l'écrit et en conférence, la parole se concentre sur les aspects techniques du recours aux algorithmes, les résultats qu'ils permettent d'obtenir, les champs extérieurs dans lesquels ils permettent de s'ancrer. Après tout, le recours aux algorithmes est un parti pris très fort, qui s'oppose à un certain conservatisme technique de la discipline. Les architectes du champ computationnel clament d'autant plus fort que ce recours a un intérêt puisqu'il est supposé révolutionner la pratique architecturale. Leur discours, articulé notamment pour répondre aux critiques auxquelles ils font face, met donc d'autant plus fort l'accent sur les particularités de leur pratique. Cela a cependant pour effet d'évacuer complètement le propos architectural. Le discours est donc dénué de tout commentaire sur la dimension classique de la pratique que nous avons mise en évidence. Ancré dans la technique, il n'a plus rien d'un discours architectural traditionnel qui parlerait d'espace sur le mode de la notice de concours ou de la critique.

Dans les entretiens, sollicités justement pour parler de la dimension architecturale de leur pratique, les praticiens sont pourtant un peu plus réalistes sur leurs pratiques. Ceci va de la reconnaissance d'un accroc comme chez Jose Sanchez à une plus grande franchise sur la mobilisation de leur expertise en tant qu'architectes, comme chez Iain Maxwell et Dave Pigram. Si ces derniers sont plutôt francs aussi dans la sphère publique, certains tiennent parfois un double discours entre entretiens et prises de paroles publiques, comme Roland Snooks. L'aisance ou non avec le saut au sein du processus computationnel pointé par Sanchez dépends d'ailleurs notamment de la capacité à reconnaître cette dimension de pratique classique ou non. Chez EZCT, la distance est actée, mais la prise de décision externe à l'algorithme, qui reconnue comme arbitraire et impossible à supprimer, est considérée comme totalement secondaire, presque négligeable[77]. Ce qui compte dans le projet c'est le

[76] Peter Eisenman, "The End of the Classical: The End of the Beginning, the End of the End", *Perspecta*, Vol. 21, pp. 154-173, 1984.

[77] Conversations Philippe Morel.

développement du processus algorithmique, non les choix finaux qui permettent de sélectionner et finaliser une proposition architecturale. L'écart entre pratiques et discours qui se creuse depuis les années 1990 et l'écriture des textes fondateurs du champ se poursuit donc à cette troisième génération du champ computationnel[78].

*4.2.3 Du parti pris aux limites techniques : une maîtrise de la programmation poussée*

Malgré les discours, les pratiques computationnelles seraient donc des pratiques comme les autres, ou les algorithmes auraient la même place que n'importe quelle autre ressource dans la boîte à outils des architectes. Oui, si ce n'est que les pratiques computationnelles font aussi intervenir une très bonne connaissance des outils de programmation, que les architectes n'ont usuellement pas, pour la gestion de l'étape de traduction. Les entretiens mettent en évidence un parcours commun et des convergences dans la découverte et l'apprentissage des outils algorithmiques, souvent dans un environnement avec une forte émulation intellectuelle et une exposition à d'autres disciplines que l'architecture. Ils sont plusieurs à décrire leur arrivée sur le campus d'une des institutions du pôle américain et à mentionner la proximité avec les départements de génie civil, d'informatique ou d'autres disciplines. Ces derniers recourant déjà à des ordinateurs pour la modélisation de certains phénomènes, la curiosité des praticiens alors en formation est piquée. Ils s'engagent alors dans un parcours d'apprentissage de la programmation qui se nourrit autant de la formation au sein d'une institution du réseau que d'échanges avec des chercheurs d'autres disciplines, plus versés dans la programmation. Ils bénéficient de cette formation, du retour d'expérience des générations précédentes et d'une pratique régulière permise par le développement du champ. Les praticiens de cette génération ont donc une maîtrise particulièrement fine des outils de programmation. Ceci leur donne les connaissances nécessaires pour explorer pleinement leur potentiel en matière de génération de formes, d'assimilation de contraintes ou d'optimisation des performances. Ce qui caractérise leur pratique est le parti pris technique du recours systématique aux algorithmes, mais aussi la capacité à jouer avec la formulation des instructions, à adapter les règles des algorithmes qu'ils emploient en fonction de leurs ambitions architecturales.

L'un des marqueurs des connaissance accumulées dans le domaine de la programmation par les praticiens de cette période est la capacité généralisée à programmer ses propres outils[79]. Celle-ci est

[78] À l'exception d'Objectile dont nous avons vu au Chapitre III que la définition du non-standard et de ses enjeux se retrouve dans nombre de pratiques autour de la construction.

[79] Pas l'idée ou la revendication, qui existent depuis la génération précédente (voir chapitre III), mais la capacité à le faire systématiquement.

bien plus poussée que dans le cas des générations précédentes, ou le travail se faisait soit en association avec des informaticiens, soit à des niveaux beaucoup plus simple vu les capacités techniques de l'époque. Programmer ses propres outils algorithmiques se fait à différentes échelles. Certains algorithmes sont programmés intégralement de zéro pour un projet. D'autres projets sont conçus à partir de l'assemblage d'algorithmes déjà existants en un nouveau programme. D'autres projets enfin sont réalisés en faisant appel à des programmes déjà codés auparavant par l'agence et simplement exécutés une nouvelle fois avec des paramètres adaptés au nouveau projet. Chaque praticien ou agence se construit ainsi un univers algorithmique dans lequel évoluer au fil de la conception architecturale des différents projets menés. Le point commun des différents outils qu'ils utilisent, c'est le fait qu'ils sont sur-mesure, par opposition aux logiciels prêts à l'emploi, auxquels les praticiens ont très peu recours, si ce n'est des modeleurs 3D utilisés pour visualiser les projets conçus. Leur maîtrise des outils facilite de fait leur intégration au sein d'un processus de conception plus large, en leur permettant de jouer sur la traduction en instructions de leur savoir architectural. Les compétences en programmation observées se traduisent également par une forte conscience des biais potentiels des algorithmes. La combinaison des connaissances en programmation et de cette conscience des préjugés se traduit par une négociation maîtrisée entre connaissances tacites et instructions explicites.

Le parcours des praticiens favorise l'émergence des capacités techniques de cette génération, mais c'est aussi le cas du setup technique de la période. Si la configuration n'est plus celle ou un ordinateur est une pièce entière manœuvrée par un spécialiste, la conception architecturale algorithmique n'est pas encore aussi simple que du traitement de texte sur Word. La variété des logiciels utilisés dans plusieurs des projets du concours Seroussi montre la complexité technique en jeu à l'époque pour mettre en œuvre les algorithmes, assurer les passerelles entre des logiciels spécifiques et différents formats de fichiers ou tracer les géométries. Les documents publiés par EZCT comprennent un tableau des différents logiciels et formats de fichiers sur lesquels s'appuie leur algorithme pour le concours, dont 12 logiciels et 5 formats de fichiers[80]. Bien qu'il s'agisse de la seule équipe à avoir détaillé la configuration technique de cette manière dans des documents écrits, d'autres entretiens font état d'une densité d'échelle identique. La méthode de définition séparée du plan choisie par les praticiens trahit également les difficultés techniques existant alors encore pour combiner différents aspects dans un même programme, et pour afficher rapidement la 3D. Ce dernier point se retrouve aussi chez supermanoeuvre pour leurs projets du début des années 2000. Ils décrivent ainsi pour leur projet Living Morphologies le recours à trois programmes différents pour la gestion de l'affichage des

[80] Elias Guenoun (éditeur), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.

géométries. Le premier permet d'obtenir les points, le second de tracer à partir de ces points une coupe du bâtiment, et le troisième d'assembler une succession de coupes pour former la 3D[81]. L'interopérabilité qu'il est nécessaire de mettre en place pour obtenir le programme complet que les praticiens cherchent à coder devient, lorsqu'ils s'y confrontent, source de maîtrise des outils.

Un des enjeux essentiels du recours aux outils algorithmiques est la gestion de cette limite technique. La sélection d'une seule direction d'exploration à la fois en est une conséquence directe. Chacune des propositions illustre une stratégie différente pour gérer cette limite et produire une réponse architecturale au brief. Pour la proposition de Gramazio & Kohler, la constitution d'un ensemble de contraintes de fabrication spécifiques permet de développer une conception architecturale basée sur celles-ci, ce qui en fait un exemple de conception axée sur la fabrication. EZCT et DORA conçoivent tous deux un algorithme axé sur la performance, mais l'extrême difficulté à l'époque de mettre en œuvre un tel algorithme dans son intégralité les amènent à l'explorer de manière différente. EZCT choisit de compléter l'implémentation de son algorithme d'optimisation de la lumière, gérant ainsi la configuration technique la plus complexe. DORA établit la possibilité d'un tel algorithme en définissant le flux de travail et en testant certaines parties de celui-ci, mais passe ensuite à une réflexion sur la référence architecturale et la manière de l'intégrer dans la conception d'un projet algorithmique. Comme cela est exprimé au cours de l'entretien, ce qui a motivé la conception de la proposition IJP semble être une tentative à la fois de rester fidèle à une idée mathématique unique - comme dans la plupart des autres projets conçus par le bureau, tels que les Vagues d'Henderson[82] - et de répondre aussi étroitement que possible au briefing architectural. L'idée de développer un projet autour d'une seule idée mathématique va de pair avec l'éloignement du problème des limites techniques ainsi que des biais techniques potentiels et permet de se concentrer sur le développement du projet architectural. Xefirotarch et biothing ont en commun l'exploration d'un potentiel esthétique et spatial prédéfini, dans le sens où l'architecture de leur proposition est une variation basée sur les formes produites par l'algorithme. Dans les deux cas, l'algorithme développé offre un vocabulaire formel spécifique, qui est ensuite adapté à la proposition architecturale. Ces différentes stratégies de gestion de la limite technique sont encore une preuve d'une maîtrise du processus, qui devient vecteur d'intégration de la prise de décision architecturale dans le processus algorithmique.

*4.2.4 Des pratiques heuristiques communes à une majorité de praticiens de la période*

[81] Entretien Iain Maxwell.

[82] I. Hwang (éditeur), *Verb Natures : What is Nature ? How Can we Modify Nature? What is the Nature of Nature*, ACTAR, 2006.

La maîtrise de la programmation est une maîtrise technique : elle implique de connaître la syntaxe des langages de programmation, les fonctions d'un logiciel, son API pour pouvoir étendre ses fonctionnalités. Mais elle implique aussi un savoir tacite, associé à ces connaissances explicites. C'est ce que Michel Polanyi souligne dans *The Tacit Dimension* lorsqu'il affirme que toute connaissance explicite possède un pendant tacite qui permet à un individu donné de se servir de ces connaissances explicites en les adaptant à la situation dans laquelle il se trouve[83]. L'exemple de la programmation est similaire à celui qu'il donne de l'apprentissage d'une langue - après tout, il s'agit bien de communiquer des instructions à la machine. Pour Polanyi, si l'usage d'une langue relève de règles grammaticales et syntaxiques, son apprentissage peut se faire de manière complètement tacite, sans jamais en apprendre les règles mais en la maîtrisant néanmoins parfaitement, à force d'écouter les autres ou de lire. Ce processus d'apprentissage par imitations, par tâtonnements, on le retrouve aussi chez les praticiens du champ computationnel. La récupération, le découpage et le collage de scripts au fil des intuitions, l'ajustement des variables, le bricolage et l'adaptation au fil de l'eau qui caractérisent leurs pratiques le montre. Le mot même de script – bout de code - très utilisé dans le champ pour décrire les objets algorithmiques manipulés, en est aussi un signe. Un nombre croissant de praticiens et d'enseignants-chercheurs du domaine de l'informatique associent d'ailleurs même la programmation à un artisanat dans leurs analyses.

La nature de l'activité de programmation est une discussion en cours depuis un certain nombre d'années. Les programmeurs questionnent leur pratique, débattant de s'il s'agit d'un art ou d'une science, d'un artisanat ou d'une industrie. En 2009, un groupe de programmeurs s'associe pour publier le *Manifeste pour un artisanat du logiciel*. Celui-ci présente quatre principes fondamentaux[84] de pratique de la programmation, et s'inscrit dans une réflexion sur sa nature qui démarre dans les années 1990. Cette réflexion prend notamment la forme de diverses listes de principes visant à garantir la qualité des codes produits par les programmeurs[85]. Le mouvement pour l'artisanat du logiciel fait suite à celui de l'Agile Alliance. L'Agile Alliance émerge en réaction à la structuration croissante de l'industrie du logiciel et à certaines pratiques de management douteuses qui découlent de

---

[83] Voir Chapitre I. Michel Polanyi, *The Tacit Dimension*, University of Chicago Press, 1966.

[84] "*En tant qu'aspirants Artisans du Logiciel, nous relevons le niveau du développement professionnel de logiciels par la pratique et en aidant les autres à acquérir le savoir-faire. Grâce à ce travail, nous avons appris à apprécier :*
*Pas seulement des logiciels opérationnels, mais aussi des logiciels bien conçus.*
*Pas seulement l'adaptation aux changements, mais aussi l'ajout constant de la valeur.*
*Pas seulement les individus et leurs interactions, mais aussi une communauté de professionnels.*
*Pas seulement la collaboration avec les clients, mais aussi des partenariats productifs.*
*C'est à dire qu'en recherchant les éléments de gauche, nous avons trouvé que les éléments de droite sont indispensables*."
https://manifesto.softwarecraftsmanship.org/#/fr-fr, consulté le 01 Septembre 2021.

[85] Par exemple SOLID, DRY ou Reuse.

la bulle informatique. L'artisanat du logiciel qui prend sa suite tourne donc d'abord autour de la fonctionnalité et de la facilité de maintenance des logiciels. Il s'agit de prioriser la qualité des logiciels plutôt que des coûts bas, des délais rapides et des profits maximisés. Le débat est aussi issu d'un mouvement de défense de leur savoir-faire par des programmeurs entraînés face au développement de langages rendant beaucoup plus accessible la pratique à des amateurs. La notion d'artisanat leur permet de mettre en avant la maîtrise d'un certain savoir-faire nécessaire pour programmer. Le mouvement pour un artisanat du logiciel définit l'artisanat comme une "*branche d'une profession qui exige un type particulier de travail qualifié. Dans un sens historique, particulièrement en ce qui concerne l'histoire médiévale et antérieure, le terme est généralement appliqué aux personnes qui produisent des biens à petite échelle*"[86]. La pratique d'un artisanat requiert donc des compétences spécifiques. Richard Sennett, fréquemment cité par les artisans du logiciel[87], rappelle que selon beaucoup d'estimations, il faut plus de 10 000 heures d'entraînement pour acquérir des compétences de l'ordre de celles nécessaires à un artisanat.

Le mouvement se base sur trois points pour soutenir sa comparaison. En premier lieu, l'appréciation par les praticiens de l'esthétique du code, non comme les architectes au sens du visuel que ses lignes de texte offrent mais au sens où certains scripts sont plus élégants que d'autres dans les instructions qu'ils mettent en œuvre. Richard Sennett assimile ceci à un amour du travail bien fait, et définit justement l'artisan comme un praticien montrant un investissement émotionnel dans son travail. Cette capacité à l'efficacité autant qu'à l'appréciation du travail sont selon lui par ailleurs usuellement l'apanage des praticiens les plus doués[88]. D'autres programmeurs, s'opposant au mouvement pour un artisanat du logiciel, dénoncent néanmoins ceci comme une élégance d'esthète sans lien avec les enjeux réels du métier de programmeur[89]. Le second élément de l'association de la programmation avec un artisanat est la méthode d'apprentissage du code par l'observation et la pratique. Les programmeurs du mouvement pour l'artisanat du logiciel font un parallèle avec les guildes du Moyen Âge et leurs apprentis[90]. L'open-source comme mode de transmission joue également un rôle important dans la pratique de ces artisans du logiciel. L'Agile Alliance et le mouvement pour l'artisanat du logiciel sont en effet aussi mues par une lutte contre les effets de l'avènement de la propriété intellectuelle dans l'industrie du logiciel. Enfin, troisièmement, les praticiens impliqués dans

---

[86] "*a craft is a branch of a profession that requires some particular kind of skilled work. In historical sense, particularly as pertinient to the Medieval history and earlier the term is usually applied towards people in small-scale production of goods.*". Il s'agit de la définition Wikipedia, donnée en définition par Ade Oshineye dans l'article "Software Craftsmanship: More than just a manifesto" - http://blog.oshineye.com/2011/01/software-craftsmanship-more-than-just.html, consulté le 01 Septembre 2021.
[87] Comme ils se nomment eux-mêmes.
[88] Richard Sennett, *The Craftsman*, Yale University Press, 2008.
[89] Dan North "Programming is not a craft", 11 Janvier 2011, https://dannorth.net/2011/01/11/programming-is-not-a-craft/, consulté le 01 Septembre 2021.
[90] Voir en particulier Pete McBreen, *Software Craftsmanship : The New Imperative*, Addison Wesley, 2001.

la discussion soulignent le fait que pour écrire le code les programmeurs ont recours à la méthode heuristique, contrairement aux sciences qui ont recours à des méthodes systématiques. La programmation tiendrait donc plus d'un artisanat que d'une science. Freeman Dyson pointe dans un article également cité fréquemment la variabilité des applications logicielles, des conditions de développement et des objectifs auxquels sont confrontés les programmeurs[91]. Ceci nécessite de faire appel à une méthode heuristique de travail plutôt qu'à l'application systématique de méthodes prédéfinies. Sennett souligne de son côté dans son commentaire sur l'artisanat que deux des différences principales d'organisation entre communautés d'artisans et entreprises capitalistes sont l'absence de mesure de la performance rigide et l'importance accordée aux connaissances tacites. Celles-ci y sont largement reconnues comme une composante essentielle de la pratique, comme un savoir à proprement parler[92].

La position du programmeur et la méthode de travail que le mouvement pour l'artisanat du logiciel cherche à décrire sont en fait assez proches de celles de l'architecte telle que nous les avons discutées au chapitre I, celle aussi décrite par Christopher Alexander dans *Notes sur la Synthèse de la Forme*[93]. Au-delà des standards de la profession que les programmeurs devraient ou non être tenus de respecter, la discussion autour de la compréhension de la programmation comme un artisanat plutôt qu'une science ou une ingénierie relève en fait d'une question épistémologique. Un article de Jack Reeves, texte fondateur pour le mouvement, le montre particulièrement bien. Il y associe la programmation à une pratique de conception. Une fois ceci posé, il esquisse un ensemble d'interrogations sur la relation de la programmation à l'ingénierie, sur la nature de l'expertise et la compartimentation des savoirs tout en cherchant à formuler des principes fondateurs pour la discipline qu'est la programmation[94]. Richard Sennett rejoint également ces interrogations sur la nature lorsqu'il souligne, dans son étude sur Linux et les outils de CAO, que si les artisans sont caractérisés par un amour du travail bien fait, encore faut-il définir ce que c'est que faire du bon travail[95]. Dans le domaine de la programmation, c'est précisément ce que soulignent les témoignages du livre *Coders at Work*, ou les praticiens se rejoignent tous autour de la question de ce que c'est que de la bonne programmation[96]. Ces réflexions rejoignent là ce que nous avions examiné au chapitre I autour de la notion d'architecture pertinente. A ce titre, la discussion sur la programmation comme un artisanat fait fortement écho aux interrogations du présent travail sur la nature des pratiques computationnelles dans le champ architectural, et sur les interrogations qu'elles déclenchent à leur tour sur la nature de la pratique architecturale en elle-même.

---

[91] Freeman Dyson, "Science as a craft industry", *Science*, Vol 280, Issue 5366, pp. 1014-1015, 1998.
[92] Richard Sennett, *The Craftsman*, Yale University Press, 2008.
[93] Christopher Alexander, *Notes on the Synthesis of Form*, Harvard University Press, 1964
[94] Jack Reeves, "What is Software Design?", *C++ Journal*, 1998.
[95] Richard Sennett, *The Craftsman*, Yale University Press, 2008.
[96] Peter Seibel, *Coders at Work : reflections on the craft of programming*, Apress, 2009.

La perception du code comme artisanat permet de fait de souligner les savoirs tacites sur lesquels la programmation se repose. Envisager le code ainsi permet aussi de prendre en compte cette pratique de l'algorithme sur-mesure dont nous venons de voir qu'elle joue un rôle clé dans les pratiques du champ computationnel. Enfin, percevoir la programmation comme un artisanat implique un savoir-faire de la programmation, au-delà de l'accumulation des connaissances techniques. Ce savoir-faire de la programmation, les praticiens de la génération dont nous faisons ici le récit des pratiques le possèdent bien. En complément, les praticiens du champ formés à l'architecture possèdent également des connaissances propres à la conception spatiale. Nous avons posé l'hypothèse au début de ce travail qu'il existe bien des savoirs propres aux architectes, et que certains de ces savoirs, clés de la discipline, sont tacites[97]. Si le recours aux outils algorithmiques change la pratique, il ne fait pas pour autant disparaître la question de la conception de l'espace et l'expertise qui y est associée. L'enjeu n'est pas de perdre l'architecture au milieu des algorithmes, mais de faire absorber à la pratique architecturale les outils algorithmiques. C'est ce dernier mécanisme qu'on observe chez les architectes du champ dont les projets conservent une dimension classique. Ces architectes possèdent donc deux savoir-faire distincts : programmation et architecture. La spécificité des pratiques computationnelles est à la lisière entre ces deux savoirs. Tout en mobilisant le savoir tacite des architectes, elles les combinent à celui de la manipulation des outils algorithmiques. C'est la rencontre de ces deux savoir-faire qui fait le propre des pratiques computationnelles en architecture. Cette rencontre débouche sur un troisième savoir-faire : celui qui combine les deux premiers pour doter les praticiens d'une capacité de traduction des savoirs tacites de l'architecture vers les formes explicites de la programmation. Les praticiens sont donc au carrefour de trois savoirs : savoir-faire de la programmation, savoir-faire de l'architecture, et savoir-faire de la pratique computationnelle.

Nous avons décrit l'isolement des praticiens du champ computationnel au sein d'une bulle protégée des contraintes de l'industrie de la construction. Il s'agit en partie d'un isolement volontaire dans le monde académique, qui leur permet de travailler sur des choses qui ne débouchent que sur des projets de papier ou des prototypes sans que cela ne soit un problème. Cet isolement offre un espace d'existence pour l'expérimentation architecturale alors que la pratique classique de l'architecture se rigidifie de plus en plus autour des enjeux de rationalisation, *via* la réglementation, mais aussi *via* des contraintes matérielles notamment financières de plus en plus fortes. Cet isolement permet aussi enfin un détournement de l'enjeu initial d'automatisation porté par le champ informatique et par les premières générations du champ computationnel[98]. Ce détour souligne d'autant plus le nouvel espace à explorer offert par les outils algorithmiques en architecture, et débouche sur une des périodes les

[97] Voir Chapitre I.

[98] Voir Chapitre II et III.

plus riches du champs, dont les productions vont démonter le potentiel architectural de ces outils et contribuer à les installer dans le paysage de la discipline. Chez les praticiens de la branche prospective, la combinaison de ces trois savoirs est particulièrement apparente, puisque l'enjeu de conception architecturale revêt une importance particulière chez eux. Le détournement de l'enjeu d'automation, l'intérêt qu'ils portent aux qualités spatiales que permettent de générer les outils algorithmiques leur fait accorder une place différente à ces derniers, en comparaison des praticiens engagés dans l'industrialisation de leur usage. Ces derniers accordent moins d'importance aux particularités du résultat et plus à la performance technique permise par les outils algorithmiques. La mobilisation des trois savoirs dans leurs projets est donc moins sensible dans les projets qu'ils produisent. Cette branche d'industrialisation se repose pourtant néanmoins aussi sur des décisions heuristiques dans ses pratiques. En effet, si le débat chez les programmeurs oppose souvent art et science comme deux disciplines aux méthodes parfaitement étrangères. Harry Collins ou John Pickstone montrent pourtant dans leur travail que les scientifiques aussi mobilisent autant l'explicite que le tacite dans leur travail[99]. Les décisions qu'ils prennent d'appliquer telle ou telle méthode, d'observer telle ou telle variable, sont dépendantes de leur expérience et de leur observation du contexte. Elles relèvent donc de choix tacites. C'est aussi le cas des ingénieurs. Mais si on peut lire les pratiques des ingénieurs comme intégrant des décisions heuristiques, la méthode dont ils se réclament ne l'est pas, au contraire. L'une des différences entre architectes et ingénieurs réside précisément dans le fait qu'une des deux disciplines se définit par le recours à une méthode heuristique quand l'autre se définit par l'emploi d'une méthode scientifique systématique. Nous retrouvons dans cette opposition la trace des deux objectifs, des deux arsenals théoriques que nous avions déjà abordés plus tôt[100]. Par certains aspects les pratiques sont pourtant similaires, et c'est ce que nous observons en étudiant le rapport aux outils computationnels, qui brouillent les limites entre les savoirs. Il existe donc un savoir-faire propre au champ computationnel chez tous les praticiens du champ, mais c'est un savoir-faire qui n'occupe pas toujours la même place. Les différents courants au sein du champ computationnel ne donnent pas tous à voir avec la même force, dans leurs pratiques et leurs discours, ce savoir-faire. Mais nous postulons que ce sont néanmoins bien ces modalités de pratique qui font le propre du computationnel en architecture. Nous proposons pour elles l'appellation de programmation heuristique[101], nom par lequel nous désignerons dans la suite du texte les usages des outils algorithmiques au service de la conception spatiale lorsqu'ils mobilisent l'ensemble des savoir-faire

[99] Voir Chapitre II. John Pickstone, *Ways of Knowing : A New History of Science, Technology and Medicine,* University of Chicago Press, 2004. Harry Collins, *Tacit and Explicit Knowledge*, The University of Chicago Press, 2013.

[100] Voir Chapitre II.

[101] Dans la littérature consacrée à la programmation informatique, notamment chez Newell et Simon ou chez Marvin Minsky, les programmes heuristiques sont ceux qui utilisent des règles heuristiques prédéfinies par leurs programmeurs

que nous venons de décrire. Il s'agit d'une appellation qui sonne redondante en regard de la description faite de la pratique de la programmation au sein du corpus des artisans du logiciels. Elle nous semble néanmoins nécessaire pour permettre de souligner l'existence de pratiques spécifiques dans le champ computationnel. Cette dimension heuristique des pratiques est trop souvent mise de côté dans les discussions du champ computationnel en architecture, or nous souhaitons pouvoir la remettre sur le devant de la scène. Reconnaître, nommer, souligner cette dimension heuristique nous paraît essentiel, car il s'agit d'une composante fondamentale des pratiques computationnelles, qui se repose sur les savoir-faire que nous venons de mettre en évidence.

## Conclusion
## Transmettre son savoir-faire

Nous avons vu dans ce chapitre que le réseau que constitue le champ computationnel s'est agrandi, diversifié et équilibré pendant cette troisième phase de son développement. Agrandi, parce que le nombre de praticiens qui s'investissent dans le développement de projets et d'outils computationnels est en nette augmentation, après la stagnation de l'hiver procédural. Diversifié, parce que les praticiens sont désormais issus à parts équivalentes de milieux différents : université, agences et bureaux d'études. Équilibré, parce que si ces praticiens exercent selon des modalités différentes en fonction de leurs milieux d'exercice, et ambitionnent des applications différentes pour les outils computationnels en fonction de ces milieux, ils influencent à parts égales les travaux du champ. La diversification des pratiques computationnelles est néanmoins à l'origine d'une nouvelle bifurcation entre industrie et pratiques dans le champ. Ceci se traduit par deux objectifs distincts d'application des outils computationnels : spéculation et rationalisation. En effet, si les pratiques computationnelles restent l'apanage d'un petit nombre de praticiens, ils ne sont plus cantonnés au monde universitaire, et les praticiens qui évoluent dans la sphère des agences d'architecture et des bureaux d'étude n'ont pas les mêmes attentes que leurs collègues chercheurs. Enfin, à cette troisième génération, les outils algorithmiques sont plus développés et mieux maîtrisés par les praticiens. Ceci leur permet de constituer un savoir-faire à part entière, entre conception architecturale et programmation informatique. Nous avons défini ce savoir-faire propre au champ computationnel à travers l'idée qu'il s'agit de la capacité à traduire le savoir tacite des architectes non en une forme explicite classique de la représentation architecturale, mais en une forme explicite computationnelle : l'instruction de programmation. Ceci requiert aussi le savoir tacite d'un programmeur. Cette génération de praticiens est donc caractérisée à la fois par l'ancrage de leur pratique dans un triple savoir-faire, chez tous, et une différenciation des enjeux malgré des espaces d'échange communs.

Cependant, un enjeu reste partagé par tous les praticiens : transmettre le savoir-faire accumulé. Ce savoir-faire partagé constitue le point commun de ces praticiens malgré leurs milieux et ambitions qui diffèrent. La conscience que les praticiens ont de ce savoir-faire, de la particularité qu'il représente, les amène à se poser la question de comment faire vivre ce savoir-faire, de comment le transmettre aux générations suivantes. Le milieu universitaire encourage évidemment cette interrogation, puisque les praticiens s'y retrouvent en position d'enseigner. Par ailleurs, cette troisième génération de praticiens du champ computationnel hérite déjà d'un certain nombre de textes, questionnements, projets et outils des générations précédentes. Les praticiens ont conscience de cet héritage, et le mouvement de transmission est donc déjà amorcé. Mais le déploiement d'un savoir-faire à part entière pour le champ qui caractérise cette génération renforce d'autant plus le poids de cet enjeu. La question de l'apprentissage des outils de programmation n'est cependant pas nouvelle à cette génération, pas plus que celle de l'apprentissage des savoir-faire du design si chère au champ des design studies. Le corpus scientifique existant est déjà riche d'interrogations sur la transmission des savoirs architecturaux, et sur la meilleure manière de mobiliser les outils numériques au sein de cet apprentissage. Nigel Cross et ses collègues de la Design Research Society sondent depuis longtemps ce champ. Nicholas Negroponte s'appuie depuis les débuts de l'Architecture Machine Group sur l'idée que l'apprentissage de la programmation fournit aux étudiants en architecture un cadre de pensée intéressant pour leur travail de conception. Dans les années 1990, alors que le recours aux modeleurs 3D se généralise, la question de l'enseignement des outils de dessin assisté par ordinateur se pose également. Le tournant vers les enjeux éducatifs dans le champ numérique effectué par Richard Coyne, Rivka Oxman ou, dans une moindre mesure, William Mitchell, témoigne de ce questionnement. La place des outils numériques dans le cursus est alors le moyen d'interroger ce que les étudiants apprennent de l'architecture en elle-même, plus qu'un enjeu technique.

Ceci fait que pour les premières générations du champ, la question de la formation technique aux outils de programmation n'est pas l'enjeu de premier plan dans les interrogations autour de l'apprentissage. Pour la troisième génération de praticiens pourtant, dont le set-up technique et la maîtrise de la programmation se sont largement développés, cette question de la formation technique se pose avec beaucoup plus d'acuité. En effet, le triple savoir-faire mobilisé dans les pratiques computationnelles implique une triple transmission. Parce que les praticiens de cette génération exercent usuellement au sein de facultés d'architecture, le savoir-faire de la conception architecturale est primordial. C'est celui-ci que les étudiants doivent acquérir pour pouvoir exercer ensuite. Le savoir-faire de la pratique computationnelle en lui-même est tout aussi crucial, puisqu'il s'agit d'armer les étudiants pour ces pratiques à leur tour. Mais il faut également un certain degré de maîtrise de la programmation pour parvenir à l'acquérir. C'est ce que démontrent les capacités de

programmation des praticiens de cette troisième génération. Les outils de programmation deviennent donc des pivots de la transmission du savoir-faire du champ, d'autant plus que la capacité à mettre au point ses propres outils de programmation est toujours un marqueur important. Les cadres et les outils de transmission du champ atteignent par ailleurs à cette période une certaine maturité, qui contribuent à lui donner sa forme, comme nous allons le voir dans les chapitres suivants.

# CHAPITRE V.
# -
# Des outils pour transmettre

## 5.1 Typologies

La génération de praticiens dont ce chapitre fait le récit se dote par ailleurs d'un autre type d'outils pour permettre cette transmission. Le champ voit alors un passage des algorithmes sur-mesure à des typologies préétablies. Les typologies d'algorithmes sont des familles d'algorithmes à la structure commune. Toutes les variantes sont donc basées sur un même ensemble d'instructions. Les divers manuels de programmation à disposition des praticiens dans le champ computationnel s'appuient largement sur ces familles d'algorithmes, sans pourtant les identifier comme des formats d'outils à part entière. Les manuels d'apprentissage de programmation visuelle et de programmation orientée objet pour l'architecture et les manuels de design génératif offrent tous des chapitres consacrés à différentes catégories d'algorithmes et aux géométries qu'ils permettent d'obtenir. L'apparition et la multiplication de ces ouvrages à partir des années 2000 n'est pas un hasard : elle coïncide justement avec l'apparition de ces typologies d'algorithmes, qui facilitent l'apprentissage. De la même manière que bibliothèques et plugins permettent de créer une boîte à outils algorithmiques à disposition au fil des projets, les structures d'algorithmes offrent une base prédéterminée pour explorer les configurations spatiales. Quatre typologies sont utilisées dans le champ computationnel : systèmes de croissance, systèmes multi-agents, simulations physiques et algorithmes génétiques. À partir du moment où elles apparaissent dans le champ, elles couvrent presque la totalité des usages d'algorithmes recensés dans le présent travail, en comptant les cas où elles sont combinées entre elles. Ces typologies sont mobilisées pour différents objectifs : génération de forme, optimisation, évaluation. Certaines se prêtent mieux à certains objectifs que d'autres. Pour la génération de forme, les praticiens recourent aux systèmes de croissance ou aux systèmes multi-agents mais aussi aux simulations physiques, dont les ressources sont également souvent mobilisées pour la rationalisation géométrique. Enfin, une dernière famille d'algorithmes est particulièrement prisée : les algorithmes génétiques, des méthodes d'optimisation qui vont de ce fait de pair avec des outils d'évaluation divers.

Ces typologies prennent racine pour certaines dans le développement informatique d'autres champs. Des structures algorithmiques développées dans le domaine de la biologie ou de l'animation sont ainsi récupérées et transformées par les praticiens du champ computationnel. D'autres structures sont basées sur des travaux des générations précédentes du champ : les travaux de Per Galle s'appuient par

exemple sur des algorithmes très similaires aux simulations physiques dites de relaxation dynamique qui se répandent plus tard dans le champ. Les typologies en usage dans le champ sont donc les héritières d'algorithmes-ancêtres, répliqués et stabilisés au fil de leur partage et de leur ajustement. Se pencher sur l'origine de ces typologies permet de retracer l'histoire du champ par un autre fil. Les typologies sont intéressantes pour comprendre les méthodes de modélisation et de transmission dans le champ computationnel, mais aussi pour saisir l'ancrage post-IA dans le monde de l'informatique. Elles sont aussi pour nous l'occasion de montrer d'autres exemples des pratiques du champ computationnel des années 2000, à travers les exemples sélectionnés pour les illustrer. On retrouve au fil de leurs histoires des noms déjà apparus, de John von Neumann aux pionniers de l'industrie de l'animation. Ces filiations offrent une meilleure compréhension de l'ancrage technique du champ. En complément de la tension tacite-explicite, les typologies contribuent largement à structurer le développement technique du champ computationnel. Par ailleurs, elles sont le reflet de sous-réseaux qui se constituent en parallèle les uns des autres, accompagnant le développement institutionnel du champ. Les réseaux qui propagent l'usage des différentes typologies montrent notamment l'existence des deux branches décrites plus haut, l'une attachée à des enjeux d'industrialisation quand l'autre poursuit ses expérimentations prospectives.

### *5.1.1 Systèmes de croissance*

Sous l'appellation systèmes de croissance sont regroupés ici des algorithmes permettant de simuler l'évolution de systèmes biologiques au cours du temps. Les L-systèmes, par exemple, sont des structures mathématiques datant du début du XX$^{ème}$ siècle, développées par plusieurs mathématiciens jusqu'à leur forme actuelle, proposée par Lindenmayer en 1968[1]. Utilisés notamment pour simuler la croissance des plantes, leur typologie de règles comprend : la forme de l'initiateur et du générateur, le placement du point de départ, les règles de branchage (angle, longueur, nombre) et le nombre d'itérations (figure 1). Les L-systèmes sont régis par une grammaire de règles très simple, qui peut être représentée par une syntaxe de seulement quelques signes. Le plug-in Rabbit par exemple, premier à introduire les L-systèmes dans Grasshopper, utilise les quatorze signes F, f, +, -, \, / ,^, &, |, J, “, !, [, ] pour retranscrire l'ensemble des règles de branchage possibles. La figure 2 montre quelques exemples d'utilisation directe de ces algorithmes dans des projets architecturaux. Le bar et restaurant The Tote, construit en 2011 à Bombay par Serie Architects, par exemple, utilise un L-système pour créer une série d'arbres faisant office de structure soutenant la couverture. Les branches de chacun de ces arbres-poteaux se subdivisent pour soutenir le toit en différents endroits, l'une des branches de chacun s'étendant à travers l'espace pour rencontrer celle d'en face et reconstituer ainsi une arche

[1] Prusinkiewicz, P. & Lindenmayer, A., “Graphical modeling using L-Systems”, in Prusinkiewicz, P. & Lindenmayer, A., *The Algorithmic Beauty of Plants*, pp. 1-50, Springer, 1990.

classique[2]. Autre exemple, l'utilisation par ArchiGlobe, en 2007, d'un programme écrit par Bollinger + Grohmann avec l'aide de Fabian Scheurer, alors à l'ETHZ. Ici le L-système est exploité un peu plus avant, puisqu'il s'agit de faire varier localement les règles de branchage en respectant la charge structurelle, et est mis en œuvre par ArchiGlobe pour créer des structures de support d'une toiture similaire à celles de Serie Architects. La toiture est cependant celle d'une maison, donc un projet plus petit, et surtout la géométrie des structures-arbres est plus anarchique puisqu'elle tire partie des variations permises par le programme de Bollinger + Grohmann d'un embranchement à l'autre du système. Le projet proposé en 2006 pour la bibliothèque nationale tchèque par OCEAN North repose sur un système similaire à celui programmé par Bollinger + Grohmann. Simplement, plutôt que de servir uniquement de support d'une toiture séparée, les branches du L-système s'enroulent tout autour du bâtiment pour former son enveloppe. Celle-ci est donc fusionnée avec le système structurel - il s'agit d'un porte-à-faux depuis la partie centrale du bâtiment, d'où l'épaisseur des embranchements qui diminue au fur et à mesure qu'ils s'éloignent du cœur.

Les L-systèmes sont en fait une catégorie de systèmes parmi d'autres dans un ensemble mathématique plus grand : les fractales. Celles-ci sont des objets mathématiques dont la caractéristique principale est leur similarité à toutes les échelles. Qu'on les observe de près, de loin, ou à n'importe quelle échelle, on voit les mêmes motifs apparaître - c'est ce qu'on observe nombre de motifs naturels, comme par exemple celui du chou romanesco. On peut donc les construire en appliquant un principe de récursivité, c'est-à-dire en répétant le même motif à chaque changement d'échelle - ce qu'on voit dans les différentes étapes de construction du L-système de la figure 1. C'est en étudiant cette notion de récursivité mathématique à partir du XVIIe siècle que des fractales diverses ont été identifiées. Parmi les plus connues : le triangle de Sierpinski ou le flocon de Koch, décrites à la fin du XIXe siècle par les mathématiciens Wacław Franciszek Sierpiński et Helge von Koch. Sur la figure 4, on peut voir un exemple d'utilisation d'une de ces courbes. Le pavillon Bloomberg, réalisé par Akihisa Hirata en 2011, est formé d'un plan subdivisé en suivant les motifs récursifs du triangle de Sierpinski, puis replié sur lui-même pour donner du volume et créer la toiture. A partir de la fin des années 1960, le mathématicien Benoît Mandelbrot, qui s'intéresse justement à la propriété d'autosimilarité, élabore une théorie mathématique unifiée pour décrire les fractales - c'est lui qui forge le terme en 1975[3]. Il collabore très vite avec IBM pour modéliser des fractales sur ordinateur, un travail qui lui permet de visualiser ce à quoi les structures mathématiques qu'il étudie ressemblent et de mieux observer leur répétitivité à toutes les échelles. De cette collaboration résulte la méthode dite des "Escape Time Fractals", qui permet de les représenter en deux dimensions. Celle-ci devient rapidement très utilisée

[2] Sur le plan structurel. Michael Hensel et Achim Menges (éditeurs), *Versatility and Vicissitude Performance in Morpho-Ecological Design*, Architectural Design March April 2008, Vol. 78 No. 2, 2008.
[3] Benoît Mandelbrot, *Les Objets fractals - Forme, hasard et dimension*, Flammarion, 1975.

dans le monde de l'animation, ou les technologies de l'informatique sont alors en plein essor. Les fractales sont en effet particulièrement efficaces pour représenter des paysages, comme en témoignent l'animation pionnière *Vol Libre* réalisée par Loren Carpenter en 1980 ou le logiciel Terragen, qui permet depuis les années 2000 de générer des paysages aléatoires. En plus de continuer encore aujourd'hui à être utilisées dans l'industrie de l'animation, les fractales sont aussi prisées des artistes numériques. Un très grand nombre de logiciels de modélisation de fractales en 2D et en 3D est aujourd'hui disponibles et utilisés par une communauté d'artistes aux pratiques diverses.

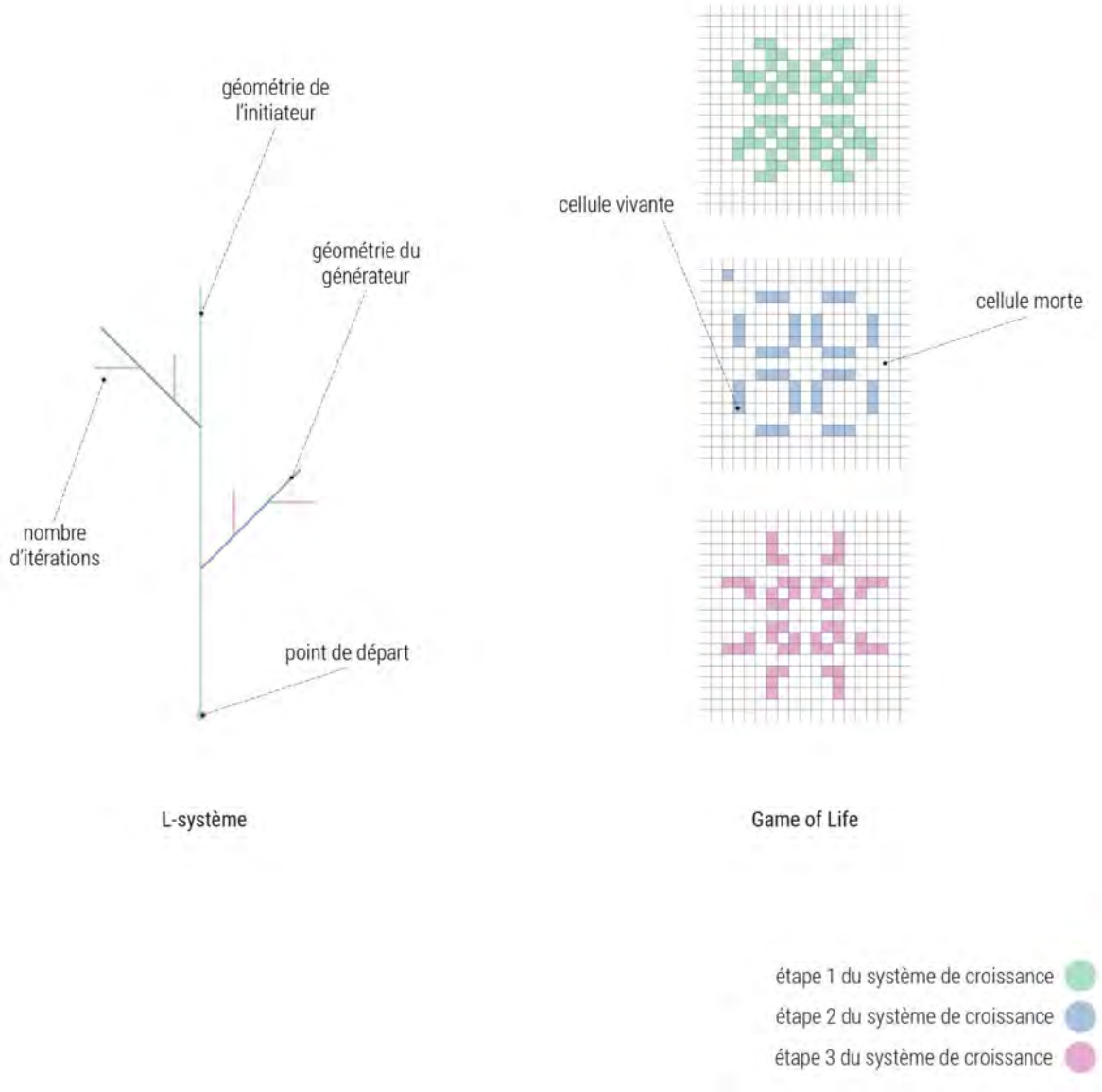



*Figure 1. Règles typologiques des systèmes de croissance*

*Figure 2. a. The Tote, Serie Architects ; b. Bibliothèque Nationale Tchèque, OCEAN North ; c. Bibliothèque Nationale Tchèque, OCEAN North*

*Figure 3. a. chou romanesco ; b. Vol Libre, Loren Carpenter*

Les systèmes de croissance incluent aussi d'autres systèmes mathématiques, obéissant à une logique similaire d'évolution au cours du temps, comme on peut le voir sur la figure 1. Les automates cellulaires sont des modèles de systèmes dynamiques constitués d'une grille de cellules. Chacune des ces cellules peut adopter différents états, et l'adoption par une cellule de l'un ou l'autre est régie par une série de règles. Dans les exemples d'automates cellulaires les plus simples, comme le *Jeu de la vie*, les cellules sont soit vivantes, soit mortes - elles ne peuvent adopter que l'un de ces deux états. Dans le Jeu de la vie, une cellule prend vie si trois de ses voisines sur la grille sont également vivantes. Une cellule meurt si elle a moins de deux voisines vivantes ou plus de trois voisines vivantes. Pour faire fonctionner l'automate cellulaire, il faut indiquer une configuration de départ, et il est ensuite possible d'observer comment l'état des cellules varie au cours du temps. La figure 1 montre trois étapes d'évolution du Jeu de la Vie à partir d'une configuration locale spécifique, dite "pulsar", qui fait partie d'une catégorie plus vaste de configurations qu'on appelle *oscillateurs* puisqu'elles alternent entre une série finie de figures. D'autres configurations classiques du Jeu de la Vie sont dites *natures mortes*, puisqu'elles sont figées, ou *vaisseaux spatiaux*, puisqu'elles se déplacent sans fin sur la grille. Les automates cellulaires permettent une grande variété de configurations locales et générales dont les paramètres sont étudiés en détail par les mathématiciens qui se spécialisent dans ce champ, et dont la variété formelle est appréciée des architectes du champ computationnel.

*Figure 4. Pavillon Bloomberg, Akihisa Hirata*

Dans les années 1940, John von Neumann, sur le conseil d'un collègue, commence à s'intéresser aux propriétés auto-réplicatives des automates cellulaires, et en formalise un qui a en effet la capacité de s'auto-reproduire. Dans les deux décennies qui suivent, nombre de mathématiciens spécialisés dans l'étude des systèmes dynamiques contribuent à ce qui deviendra un modèle mathématique unifié pour les automates cellulaires. En 1969, Konrad Zuse, Alvy Ray Smith et Gustav A. Hedlund publient trois ouvrages qui compilent connaissances mathématiques sur cette classe d'objets et commentaires sur les automates cellulaires comme modèle du fonctionnement physique du monde[4]. En 1970, le Jeu de la vie de Conway, décrit par Martin Gardner dans un article pour le Scientific American suscite nettement plus d'attention que les automates cellulaires précédents, notamment dans la communauté informatique, et contribue également beaucoup à populariser ces objets mathématiques au-delà de la branche universitaire qui leur est dédiée[5]. Si les automates cellulaires ont été développés depuis le début à mi-chemin entre les domaines des mathématiques et de l'informatique, cet article permet de les faire passer dans le grand public : nombre d'amateurs s'essayent alors à leur programmation. Comme pour les fractales, ceci résulte en de nombreuses expérimentations et codes partagés au sein d'une communauté de codeurs amateurs. Au début des années 2000, Stephen Wolfram propose une notation unifiée pour les automates cellulaires, et creuse l'idée que leur portée comme modèle physique universel a du sens, en raison de la grande complexité des comportements qu'on voit émerger à partir de quelques règles très simples. Celles-ci permettent aux automates cellulaires d'être fréquemment appliqués pour la modélisation de systèmes dynamiques divers, comme par exemple la propagation des incendies.

[4] Konrad Zuse, *Rechnender Raum*, Braunschweig: Friedrich Vieweg & Sohn, 1969. Alvy Ray Smith, *Cellular Automata Theory*, PhD Dissertation, Stanford University, 1969. Gustav A. Hedlund, "Endomorphisms and automorphisms of the shift dynamical system". *Math. Systems Theory*. 3 (4): 320–3751, 1969.
[5] Stephen Wolfram, *A New Kind of Science*, Wolfram Media, 2002.

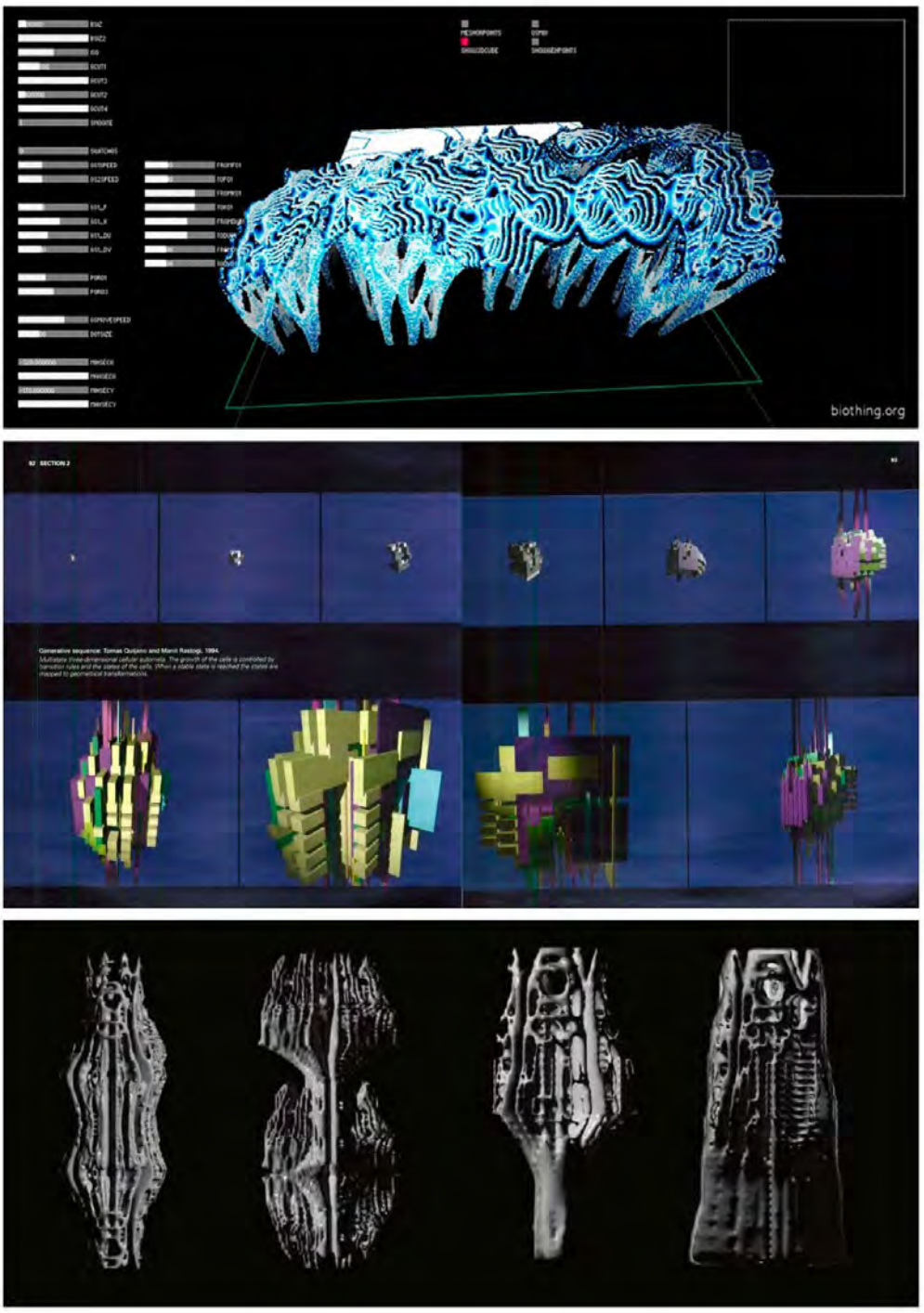



*Figure 5. a. Pavillon Turing, biothing ; b. Evolutionary Architecture, John Frazer ; c. Unprintable Forms, Marjan Colletti et al,*

La figure 5 montre quelques exemples d'usage des automates cellulaires dans le champ computationnel. Ils sont directement appliqués par Marjan Colletti et un groupe de chercheurs de la Bartlett et d'Eindhoven pour simplement générer des formes en empilant des générations d'automates cellulaires 2D - une méthode qu'on retrouve souvent utilisée pour la génération de volumétries à l'échelle urbaine[6]. Dans le cas de Colletti et al., cependant, plutôt que d'assimiler les cellules vivantes à des volumes cubiques et de les empiler directement, elles deviennent les points de contrôle d'une courbe. Cette courbe devient ensuite le contour de la forme en cours de génération, et l'empilement des contours constitue une structure qui peut ensuite être imprimée en 3D[7]. Plus tôt, John Frazer s'était également fréquemment servi des automates cellulaires avec ses étudiants de la AA, explorant non seulement la variété formelle offerte par ces modèles, mais en les utilisant aussi de manière indirecte. Plusieurs projets avec ses étudiants de 1993 et 1994, présentés dans *An Evolutionary Architecture*, utilisent les automates cellulaires comme un moyen de répartir des transformations géométriques : les états correspondent à ces transformations, et les cellules à un ensemble d'éléments géométriques auxquels ces transformations ont vocation à être appliquées, déformant l'objet initial en lui donnant une nouvelle forme. Les automates cellulaires peuvent aussi être utilisés pour modéliser des équations de réaction-diffusion[8], qui permettent de décrire la répartition dans l'espace de deux substances distinctes au moment de leur rencontre : elles réagissent donc l'une à l'autre en plus de se diffuser spatialement, et ce sont ces deux phénomènes qui sont modélisés. Ceci fait apparaître des motifs prisés des praticiens du champ computationnel, en particulier le motif dit de Turing, qu'on voit sur la figure 5 être utilisé dans un projet de 2011 par biothing, en collaboration avec D-Shape : le pavillon Turing. Celui-ci exploite la continuité entre les étapes d'évolution des motifs de Turing, qui permet de créer des colonnes s'élargissant de plus en plus en superposant les étapes les unes au-dessus les autres, sur le même principe que le travail de Marjan Colletti mentionné précédemment.

---

[6] Ahmet Emre Dincer, Gülen Çağdaş et Hakan Tong, "A Digital Tool for Customized Mass Housing Design", dans Aulikki Herneoja; Toni Österlund; Piia Markkanen (éditeurs), *eCAADe 2016 : proceedings of the 34rd International Conference on Education and Research in Computer Aided Architectural Design in Europe, 24.-26. August 2016, Oulu, Finland. Vol. 1, Complexity & simplicity*, Oulu : eCAADe : Oulu School of Architecture, University of Oulu, 2016.

[7] Yota Adilenidou, Zeeshan Yunus Ahmed, Freek Bos et Marjan Colletti, "Unprintable Forms. Complexity as a Robustness Factor for Robotic Fabrication and 3DCP Constraints through Error Elimination and Reinsertion", dans Kory Bieg, Danelle Briscoe et Clay Odom (éditeurs), *ACADIA 19:Ubiquity and Autonomy. Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture*, pp.168-177, The University of Texas at Austin School of Architecture, Austin, Texas 21-26 October, 2019.

[8] Voir par exemple Mikio Murata, "Reaction–Diffusion Equations and Cellular Automata", dans Anderssen R., Broadbridge P., Fukumoto Y., Kajiwara K., Simpson M., Turner I. (éditeurs), *Agriculture as a Metaphor for Creativity in All Human Endeavors. FMfI 2016. Mathematics for Industry*, vol 28. Springer, 2018, ou
Jörg R. Weimar, "Cellular automata for reaction-diffusion systems", *Parallel Computing*, Volume 23 Issue 11, pages 1699-1715, 1997.

De manière générale, les systèmes de croissance sont très prisés par les praticiens du champ computationnel pour les motifs et les formes organiques qu'ils permettent d'obtenir. Mais au-delà de cette attirance visuelle, les praticiens qui s'emparent de ces techniques se saisissent aussi d'un discours sur la nature et sa modélisation. Les praticiens du champ développent ainsi un certain goût pour la maîtrise de la nature que ces systèmes algorithmiques supposent autant que pour la variété et l'exubérance des formes qu'ils permettent. Ceci accompagne dans les années 2000 une passion pour la notion de morphogenèse, qui devient un pilier des pratiques à cette période. Celle-ci désigne l'ensemble des lois qui régissent le développement des formes naturelles et leur adaptation à long terme aux conditions dans lesquelles elles évoluent[9]. Cette affection pour la notion de morphogénèse était déjà visible dans la prédominance du thème du biodesign dans les numéros d'AD du moment, comme nous l'avons vu un peu plus haut. Elle s'accompagne parfois d'un fantasme d'une adéquation parfaite entre l'architecture produite à l'aide de ces typologies d'algorithmes et son environnement. La portée des différents est mise en exergue à partir de la notion d'auto-organisation : les algorithmes sont supposés permettre de modéliser des écosystèmes complets et de trouver un état d'équilibre, qui s'exprime dans une forme qui devient ensuite celle de l'espace en cours de conception. Le catalogue de l'exposition *Naturaliser l'Architecture*[10] rassemble ces questionnements en interrogeant la relation de l'architecture à la nature, et la médiation que permettent les algorithmes pour en examiner les processus de génération. Jencks décrivait cette approche comme la biologie devenant la métaphore principale guidant la conception architecturale[11]. Les praticiens glissent cependant parfois de la métaphore à une volonté de voir ces modèles comme une clé de compréhension littérale et totale du monde et de sa logique de création. Ceci n'est pas sans rappeler les propos de Stephen Wolfram ou Konrad Zuse sur la portée de ces différents modèles, des algorithmes tout puissants décrivant le monde et ses processus dans leur intégralité. Fractales et automates cellulaires y sont considérés comme des objets mathématiques fondamentaux de la modélisation.

Cet intérêt pour la modélisation des processus naturels est un héritage des premières générations du champ computationnel de la AA et du CECA, de Gordon Pask et de la cybernétique[12]. John Frazer développe alors d'abord ce qu'il nomme architecture évolutionnaire. Celle-ci plonge dans un discours aux racines proches ; il s'agit d'apprendre des lois de la nature pour les intégrer aux programmes informatiques et produire des architectures pertinentes. L'usage des systèmes de croissance dans le champ computationnel prend racine dans les générations les plus anciennes. La figure 6 présente la

[9] Les références fréquentes à D'Arcy Thompson, biologiste et mathématicien auteur de *On Growth and Form*, témoignent de cette fascination. D'Arcy Thompson, *On Growth and Form*, Cambridge University Press, 1917.
[10] Marie-Ange Brayer et Frédéric Migayrou, *Naturaliser l'architecture : ArchiLab 9e édition*, Orléans : HYX, 2013.
[11] Charles Jencks, *Architecture 2000: Predictions and Methods*, New York, Praeger, 1971.
[12] Voir Chapitre II.

partie du réseau dans laquelle cet usage émerge. A partir de la AA, ces typologies se propagent dans d'autres pôles au fil des déplacements des praticiens qui y ont été formés : Michael Hensel, Achim Menges, Marjan Colletti, Marcos Cruz ou Alisa Andrasek. Ceux-ci importent la typologie dans les institutions où ils enseignent, mais aussi en agence, notamment grâce à OCEAN. Les systèmes de croissance prennent aussi racine au MIT, qui y consacre entre 2005 et 2015 plusieurs programmes de recherche[13] et forme à son tour des usagers qui continuent à faire voyager la typologie. Le rôle joué par les deux institutions dans l'expansion des systèmes de croissance est un marqueur supplémentaire de l'influence de la AA et du MIT sur l'écosystème computationnel. Un autre groupe d'usagers se développe autour de Bollinger + Grohmann, qui tire parti de la facilité de manipulation des paramètres pour l'importer dans d'autres agences moins versées dans le recours aux outils computationnels. Enfin un pôle se constitue à partir des outils prêts à l'emploi permettant d'utiliser les systèmes de croissance. En témoignent les plugins grasshopper qui pullulent, programmés notamment par des étudiants de master. La facilité de programmation des systèmes de croissance fait par ailleurs beaucoup pour leur popularité. Les praticiens les plus aguerris y recourent sensibles notamment au discours autour duquel leur utilisation s'articule, mais aussi les plus jeunes aussi, séduits d'abord par l'exubérance des formes et la facilité d'emploi. Les systèmes de croissance prennent donc racine dans un pôle spécifique avant de se répandre plus loin plus tard.

Ce pic d'intérêt pour la question morphogénétique dans les années 2000 débouche par la suite sur la formation du champ du biodesign, qui se caractérise aujourd'hui notamment par la combinaison dans les projets de l'usage d'algorithmes pour générer des formes et d'expérimentations de fabrication. En effet, l'autre thème de travail principal du champ à ce moment est, comme nous l'avons vu, l'exploration de méthodes de fabrication pour les formes générées à l'aide des algorithmes. Il s'agit aussi d'étudier comment les processus de fabrication numérique défrichés à partir des années 1990 peuvent être mis au service non seulement d'une complexité formelle mais aussi de l'usage de nouveaux matériaux. Cellulose, soie ou mycélium sont mis en œuvre à l'aide de processus robotisés dans des structures modélisées à l'aide de programmes informatiques, et accompagnés d'une réflexion sur la place des constructions humaines au sein des écosystèmes dans lesquels elles s'installent[14]. Le champ du biodesign se structure donc à mi-chemin entre les méthodes de morphogénèse offertes par

[13] Le Medialab l'identifie comme sa troisième période de recherche : "*The third decade (2005–2015) layered in biological sciences and technologies. Lab researchers have been at the forefront of bringing the human-machine interface onto and into the body; innovating new materials and methods for fabrication; and exploring new tools and technologies for health and wellbeing"*. M.I.T Medialab, "About the Lab. History", https://www.media.mit.edu/about/history/#:~:text=When%20the%20MIT%20Media%20Lab,emerging%20field%20of%20digital%20technology., consulté le 02 Septembre 2021.

[14] Voir par exemple le travail du Jenny Sabin Studio, de l'ICD, de CITA, ou du Mediated Matter group. L'ouvrage suivant en analyse les réflexions : Christina Cogdell, *Toward a Living Architecture?: Complexism and Biology in Generative Design*, University of Minnesota Press, 2019.

les systèmes de croissances et des préoccupations environnementales donnant lieu à l'exploration de procédés potentiellement plus soutenables que les procédés de constructions actuellement en usage dans l'industrie. Autre caractéristique, le retour dans le discours de la notion de biomimétisme, qui désigne l'imitation d'éléments naturels. Cette approche est notamment affectionnée par le pôle allemand, et se fait le pendant technique de la réflexion philosophique proposée entre autres par le pôle français autour de la place de la nature à partir de la notion de morphogénèse. Ces deux pôles du discours complètent un usage esthétique de ces algorithmes ailleurs dans le champ.

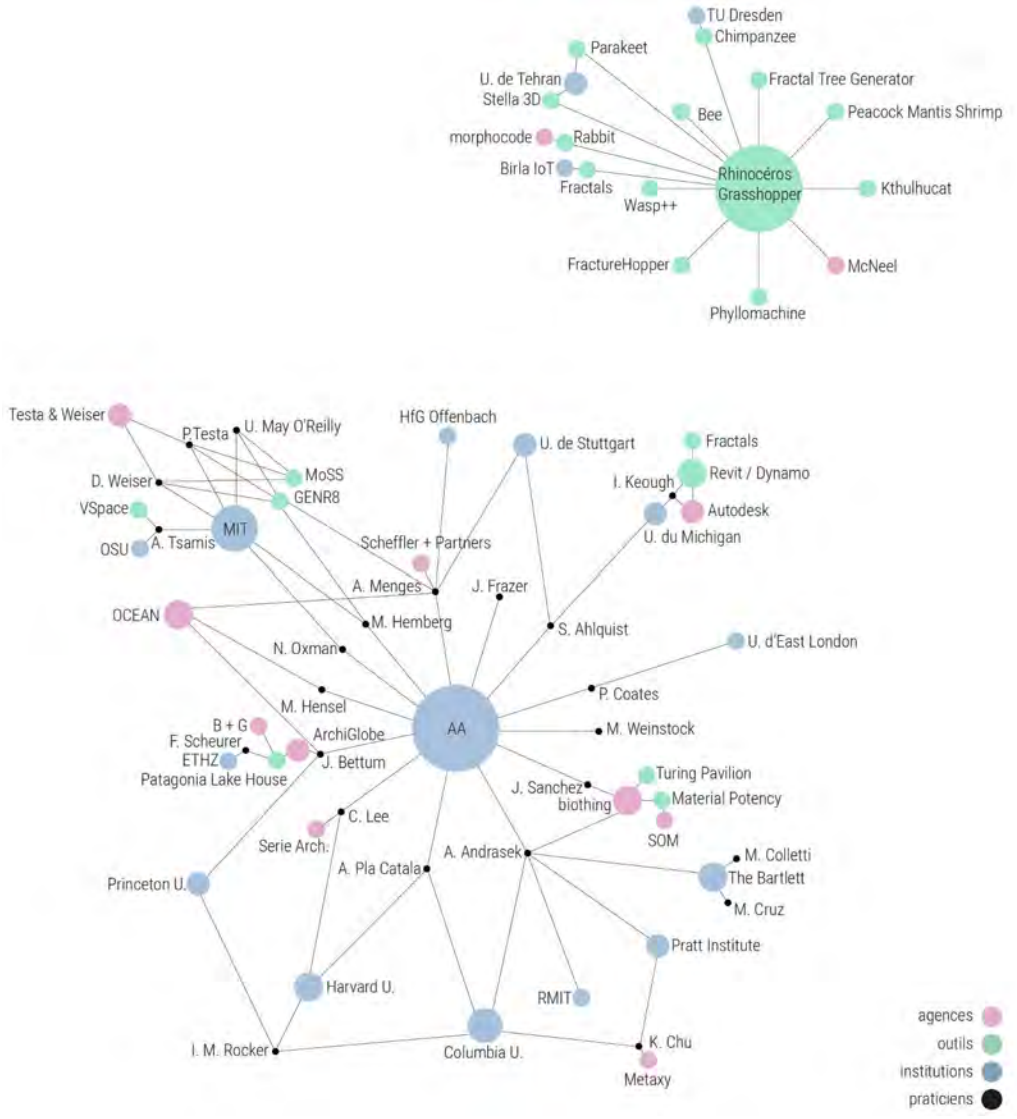



*Figure 6. Réseau de développement des systèmes de croissance*

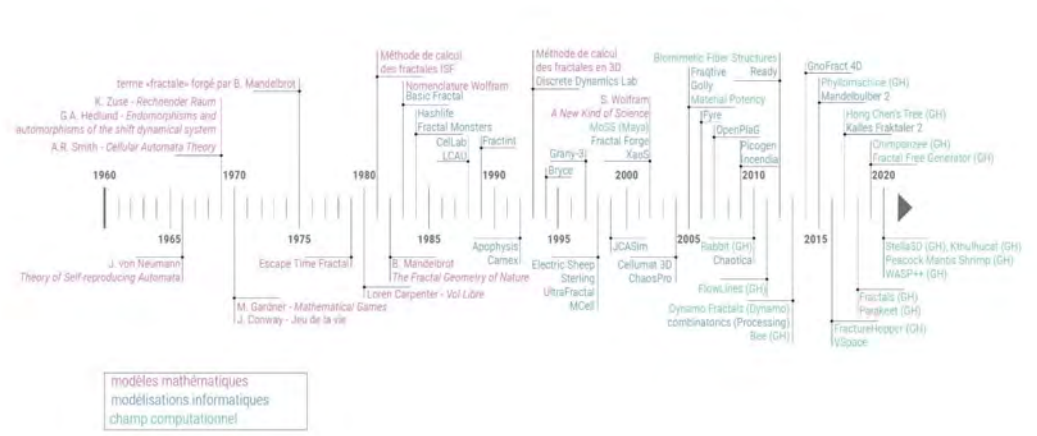



*Figure 7. Chronologie de développement des systèmes de croissance*[15]

### *5.1.2 Systèmes multi-agents*

Les systèmes multi-agents (SMA), souvent également appelés *swarms* en anglais (essaims), sont des systèmes algorithmiques qui modélisent le comportement et les interactions d'ensembles de particules. Chaque particule, ou agent, obéit à une série de lois données, et leurs interactions génèrent ce que l'on appelle un comportement émergent : un comportement de l'ensemble qui diffère du comportement spécifique donné à chaque élément. Les composants d'un système multi-agents sont les suivants : un *environnement* - généralement un espace mesurable -, des *objets* peuplant cet environnement et pouvant être modifiés par les *agents*, des *relations* liant les objets et les agents entre eux, et des *opérations* permettant l'interaction entre les objets et les agents[16]. Les systèmes multi-agents sont en fait une des typologies d'algorithmes principales de la programmation orientée objet. Il s'agit d'un paradigme de programmation reposant sur la définition d'entités, de leurs propriétés et de leurs interactions avec les autres objets. La programmation orientée objet émerge dans les années 1960, notamment grâce au travail du programmeur Alan Kay, qui forge le terme en 1967[17] (figure 8). Suite à son apparition, de nombreux langages spécifiques à ce paradigme de programmation sont développés, comme SmallTalk, C++ ou Java. C'est au sein de ces langages que sont ensuite développés les

[15] Comme pour les suivantes, les outils listés sur la chronologie ne sont pas totalement exhaustifs. Les pratiques diverses, notamment privées, rendent difficile de lister l'intégralité des outils apparus dans le champ computationnel à cette période. De plus, le développement des modèles mathématiques et informatiques de la typologie n'est ici pas intégralement documenté, il s'agit plutôt de fournir des points de repère.

[16] Ferber, J., *Les systèmes multi-agents*, InterEditions, 1995.

[17] T. Elrad, R. E. Filman, and A. Bader. Aspect–Oriented Programming. Communications of the ACM, 44(10):29–32, October 2001.

systèmes multi-agents. Les systèmes multi-agents regroupent les algorithmes de programmation orientée objet dont l'objectif est de modéliser le déplacement dans l'espace d'entités en fonction de leurs mouvements et du contexte spatial.

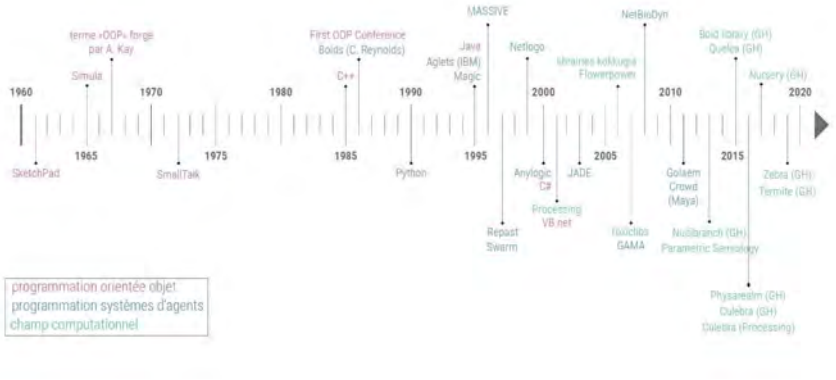



*Figure 8. Chronologie de développement des systèmes multi-agents*

Parmi les premiers exemples de systèmes multi-agents se trouvent les *boids* de Craig Reynolds, créés en 1986[18]. Craig Reynolds est un informaticien et graphiste qui travaille à l'époque sur l'animation de grands ensembles d'éléments pour l'industrie cinématographique. Il programme les boids pour simuler le comportement d'une volée d'oiseaux à l'écran. Tout en étant l'un des premiers systèmes multi-agents développés, l'algorithme de flocage[19] conçu par Reynolds constitue également un exemple du type de règles qui sont appliquées aux agents dans les systèmes multi-agents. Les boids obéissent à trois règles : la séparation - respect d'une distance minimale par rapport aux autres particules - l'alignement - tête vers la direction de déplacement moyen du troupeau - et la cohésion - respect d'une distance maximale par rapport aux autres particules[20]. Suite à cette percée, un grand nombre de langages et bibliothèques ont été développés pour permettre la programmation avec des systèmes multi-agents à partir de 1995. Ces outils ont depuis été régulièrement utilisés, principalement dans les industries du film et des jeux vidéo. MASSIVE, développé en 1996 pour l'animation de foules de milliers de personnes dans la série de films *Le Seigneur des Anneaux*, ou Golaem Crowd Maya, développé en 2011, sont encore utilisés dans l'industrie[21]. Actuellement, le recours aux systèmes multi-agents se fait dans trois domaines principalement : simulations de foules

[18] Craig Reynolds, "Flocks, herds and schools: a distributed behavioral model", *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, p. 25-34, 1987.
[19] Le flocage désigne le comportement des oiseaux en vol en groupe.
[20] Craig Reynolds, "Flocks, herds and schools: a distributed behavioral model", *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, p. 25-34, 1987.
[21] https://www.massivesoftware.com/about.html. Consulté le 02 Septembre 2021.

pour l'urbanisme ou l'animation, auto-organisation des flux de travail et pilotage des décisions. Puisque les systèmes multi-agents peuvent être identifiés comme une typologie spécifique d'algorithme, ils impliquent le recours à un ensemble prédéfini d'instructions, identique pour tous les algorithmes (figure 9). La typologie des systèmes multi-agents telle qu'utilisée dans le champ computationnel comporte trois niveaux de règles. Premièrement, des règles relatives à l'état initial du système, telles que la position ou le point d'entrée des agents ou l'emplacement des objets. Ensuite, des règles de comportement des particules sont mises en œuvre pour dicter leur mouvement - règles de direction, telles que la séparation, l'alignement et la direction des boids - et leurs interactions avec les objets - attraction ou répulsion par exemple. Enfin, des règles relatives à l'exploitation géométrique de la modélisation des mouvements, afin de la convertir en données pertinentes pour l'objet architectural. Ceci se fait par exemple en utilisant la position finale des agents, les traces de leur déplacement ou des opérations, et potentiellement en ajoutant de nouveaux éléments géométriques.

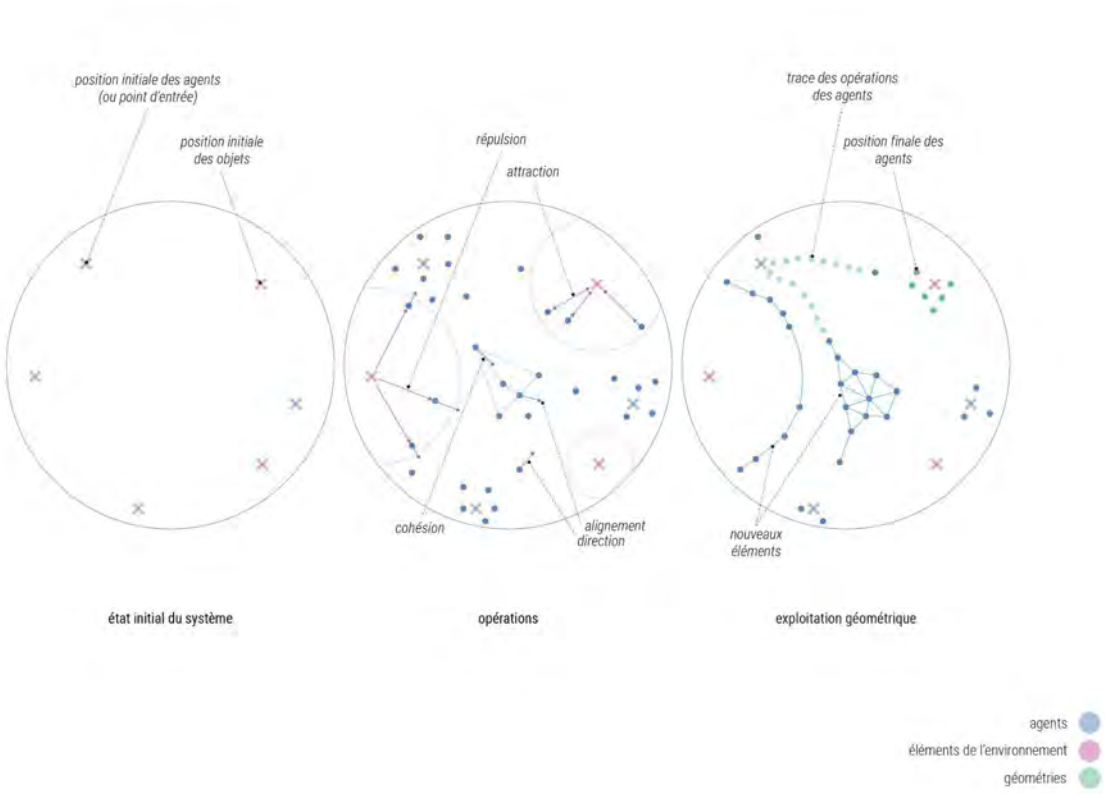



*Figure 9. Règles typologiques des systèmes multi-agents*

A la suite du fort développement de la typologie dans les années 1990, les architectes commencent également à recourir aux systèmes multi-agents[22], et ces algorithmes ont pris un nouvel essor dans le champ computationnel au cours des années 2000. Les utilisateurs ont conçu diverses applications basées sur des essaims se comportant selon des règles variées, allant de simples algorithmes de flocage à des simulations de champs magnétiques[23] ou de données comportementales[24]. Des utilisations plus complexes sont également développées, comme des algorithmes combinant plusieurs essaims ensemble, ou avec d'autres typologies algorithmiques. Le champ magnétique de Flower Power, l'outil sur lequel biothing base sa proposition pour le pavillon Seroussi, est codé à partir d'un système multi-agent. Autre projet développé par biothing, le projet Mesonic Fabrics associe ce même système multi-agents à des automates cellulaires afin de générer un terrain et des structures de toiture. Mesonic Fabrics donne ainsi à voir la conjugaison de plusieurs typologies, et le fonctionnement de la bibliothèque de typologies développées au fil des années par biothing. Le projet Cliff House, de kokkugia, programme un système multi-agents à partir de l'intention d'utiliser pour la construction une structure composite, formées de fibres qui renforcent une coque en résine. C'est la géométrie de ces fibres que le système multi-agents doit permettre d'obtenir. Le projet est par ailleurs une maison sur une topologie de falaise extrême, et doit donc intégrer l'ancrage structurel du système dans la roche. Un troisième exemple est le projet Trabeculae / Protosynthesis de supermanoeuvre, qui réimagine une tour de bureaux et son atrium en mettant en œuvre deux systèmes multi-agents combinés. L'un obéit à un ensemble de règles de suivi de la lumière, et sert à définir une géométrie qui garantit la transmission de la lumière jusqu'au cœur du bâtiment. Le second prend en compte les résultats du premier et crée autour un treillis tridimensionnel, structure principale du bâtiment. Living Morphologies, autre projet de supermanoeuvre dont nous avons vu qu'il revisite l'Unité d'Habitation de Le Corbusier, le fait aussi avec un système multi-agents. Ses particules obéissent à des règles de déplacement basées sur une rétro-ingénierie de la logique de circulation imaginée par l'architecte suisse. Patrik Schumacher, avec sa série de projets "*Agent based parametric semiology*" que nous avons également mentionné, explore également les usages de la simulation de foules *via* un système multi-agents pour la conception architecturale.

---

[22] Krause, J., "Agent Generated Architecture", *Design and Representation : Proceedings of the 1997 Association for Computer Aided Design in Architecture (ACADIA)*,p. 63-70, 1997. Coates, P. and Schmid, C., "Agent Based Modelling", *Architectural Computing from Turing to 2000 : Proceedings of the 1999 eCAADe*, p. 652-661, 1999.

[23] Alisa Andrasek, *biothing*, HYX, 2009.

[24] Patrik Schumacher, "Freedom via Soft Order : Architecture as A Foil for Social Self-Organisation", dans O. Hopkins (éditeur), *AD Special Issue - Architecture and Freedom : Searching for Agency in a Changing World*, Wiley & Sons, 2018.

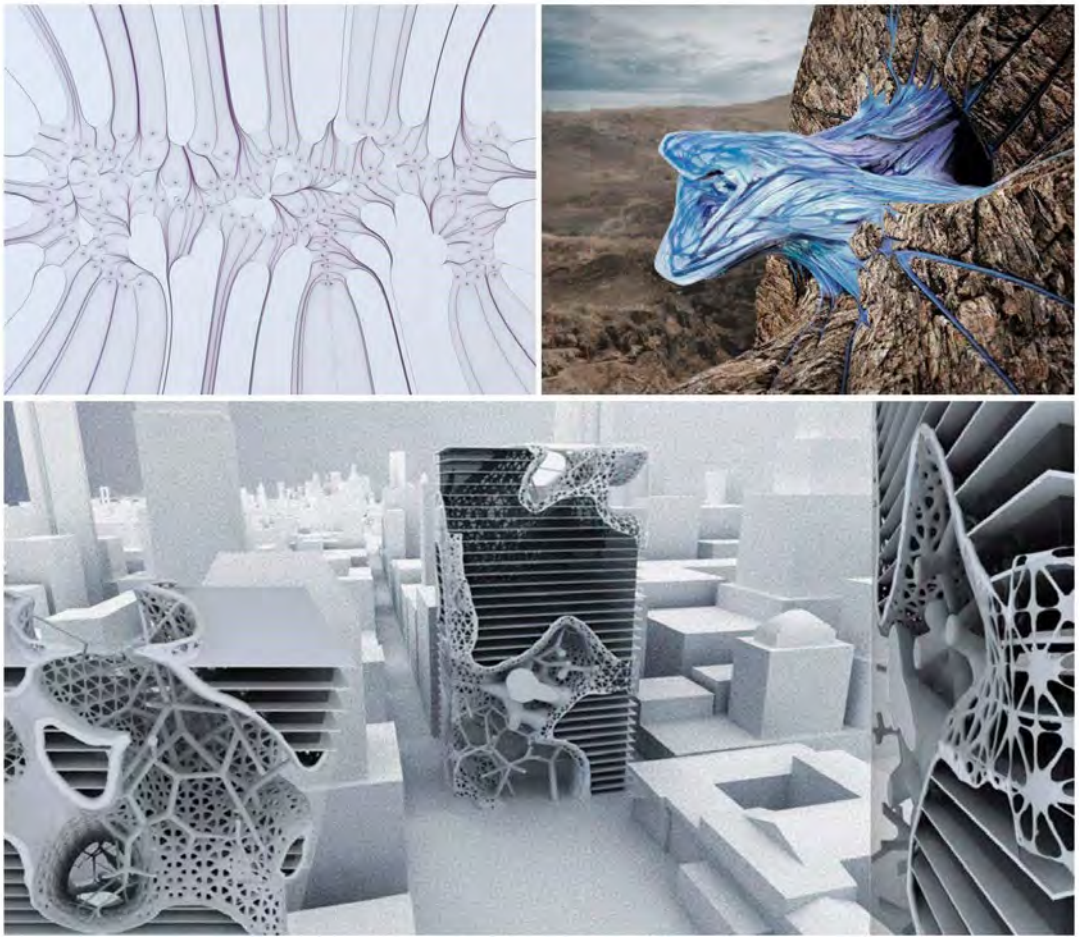


*Figure 10. a. Mesonic Fabrics, biothing ; b. Cliff House, kokkugia ; c. Trabeculae / Protosynthesis, supermanoeuvre*

Le comportement émergent des essaims et leur capacité à s'auto-organiser en font des outils populaires parmi les architectes du champ, tout comme leur capacité à gérer la complexité et la possibilité qu'ils encapsulent de programmer la matière. Le comportement émergent des systèmes multi-agent contribue à produire de l'imprévu, et pousse donc les praticiens à jouer avec à la recherche de cet effet de surprise dont nous avons mentionné qu'il est prisé dans le champ computationnel. Nombre de praticiens apprécient aussi le détail géométrique qu'autorise la gestion d'un grand nombre de particules, et la délicatesse et la complexité des ornements qui en découlent. Par ailleurs, la notion d'émergence renvoie à une capacité d'auto-organisation dans les systèmes multi-agents qui poussent les praticiens à leur prêter des capacités de gestion et d'optimisation. Laisser les particules d'un système multi-agents tournoyer dans l'espace en réaction à leurs règles de déplacement et leur environnement permettrait au cours du temps d'atteindre un état d'équilibre du système, et donc une géométrie idéale en réponse aux règles intégrées au système. Les particules dont

sont composés les systèmes multi-agents feraient figure d'unités computationnelles fondamentales. Les règles de comportement auxquelles elles obéissent peuvent être écrites sur-mesure, et donc intégrer n'importe quel impératif de conception. Propriétés physiques liées au comportement des matériaux, réaction à la présence d'une autre particule ou d'un objet quelconque, intentions esthétiques - tout peut être formulé comme une règle dictant leur comportement aux agents. C'est ce que supermanoeuvre mets en place dans le projet Trabeculae / Photosynthesis, qui combine des impératifs lumineux et structurels. De la même manière, kokkugia intègre dans le système multi-agents de la Cliff House les impératifs structurels, mais aussi sa volonté esthétique[25]. Il s'agit donc de programmer la matière au sens où ces unités computationnelles sont à charger de sens, via les instructions algorithmiques, en fonction de ce qui paraît pertinent à l'architecte[26]. Les praticiens du champ qui s'en saisissent prêtent aux systèmes multi-agents la possibilité de gérer dans son entièreté la complexité du monde[27]. Derrière ceci se cache une fois encore la recherche d'une formule algorithmique universelle, qui ferait office de réponse à la complexité de la pratique architecturale, déjà observé dans le rôle assigné aux systèmes de croissance.

Les explorations du potentiel des systèmes multi-agents se sont accompagnées du développement d'algorithmes personnalisés, ainsi que de bibliothèques développées par des pratiques emblématiques du domaine, en particulier supermanoeuvre, Kokkugia et biothing[28]. En raison des différentes étapes qu'elle demande de mettre en place, des nombreux paramètres et de la liberté qu'elle offre, la typologie des systèmes multi-agents demande beaucoup d'investissement et un certain savoir-faire pour programmer des systèmes complets. Les agences qui s'y consacrent sont donc aussi parmi les plus investies dans le développement d'algorithmes pour la conception architecturale. Plus largement

---

[25] "*Designing through agent-based behavioral strategies encodes design intent within individual elements that interact at a local scale to give rise to the emergence of complex order at the macro-scale. Applying this methodology to a composite fiber house enables the local scale to be reduced to a sub-material level. This increase in the population of agents generates greater intricacy and intensive emergent affects. The geometry of the Cliff House is not discrete or reducible - instead, geometry negotiates complex behaviors such as structure and ornament, generating emergent characteristics that shift throughout the project".* https://www.kokkugia.com/filter/research/cliff-house, consulté le 02 Septembre 2021.

[26] Igor Pantic et Someem Hahm, "Isomorphic Agency, Emerging Experience in Past, Present and Future of Digital Architecture", *Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA*, p. 178-188, 2015.

[27] Intervention d'Alisa Andrasek à la conférence Deep City le 26 Mars 2021, Ecole Polytechnique Fédérale de Lausanne.

[28] Alisa Andrasek, *biothing*, HYX, 2009. Robert Stuart-Smith, "Behavioural Matter", Conférence á la faculté d'architecture, Université du Costa Rica, 7 Octobre 2011. Robert Stuart-Smith, "A Matter of Organization : Re-emphasizing Organization", dans Neil Leach et Philippe F. Yuan (éditeurs), *Scripting the Future*, Tongji University Press, p. 157-162, 2012. Iain Maxwell et Dave Pigram, "Inorganic Speciation: Matter, Behaviour and Formation in Architecture", dans D. Kottas (éditeur), *Contemporary Digital Architecture Design & Techniques*, Links, p. 208-227, 2010. Iain Maxwell et Dave Pigram, "Algorithmic Typology : Towards in Operational Model", dans Neil Leach et Philippe F. Yuan (éditeurs), *Scripting the Future*, Tongji University Press, p. 107-112, 2012.

que les exemples donnés ci-dessus, toute la pratique des premières années de biothing, de supermamoeuvre et de kokkugia se base sur des systèmes multi-agents. Le réseau des systèmes multi-agents est ainsi un exemple de praticiens associés à une typologie particulière et alimentant son développement en en créant les outils principaux, plutôt que d'institutions. La typologie prend néanmoins racine dans un pôle spécifique. Mais ce n'est pas un pôle déjà existant comme pour les systèmes de croissance, c'est le pôle australien, qui émerge en même temps qu'il contribue à asseoir le recours aux systèmes multi-agents dans le champ. Iain Maxwell, Dave Pigram, Roland Snooks et Robert Stuart-Smith, formés par Alisa Andrasek et ayant travaillé plusieurs fois pour elle, font beaucoup non seulement pour mettre en place des bibliothèques permettant l'usage des systèmes multi-agents, mais aussi pour les populariser au fil de leurs interventions un peu partout dans les pôles du réseau. Si Paul Coates s'est intéressé un temps aux systèmes multi-agents à l'UEL, ceux-ci ne rencontrent pas le même succès au sein du pôle londonien. La typologie n'y a une place qu'à partir du moment dans les années 2010 ou Patrik Schumacher développe ses travaux au sein du DRL. Alisa Andrasek importe cependant avec elle et ses étudiants la typologie au GSAPP, à Sci-Arc et ensuite plus largement aux Etats-Unis, où elle travaille notamment avec Ezio Blasetti et Luis Quinones sur des extensions des bibliothèques existantes. Le réseau des systèmes multi-agents est donc un réseau d'individus plutôt que d'institutions. Plutôt qu'un pôle principal et une dissémination ailleurs ensuite comme pour les systèmes de croissance, ce réseau est fait de praticiens répartis entre plusieurs institutions, et qui en change au fur et à mesure que la typologie se répand avec aux

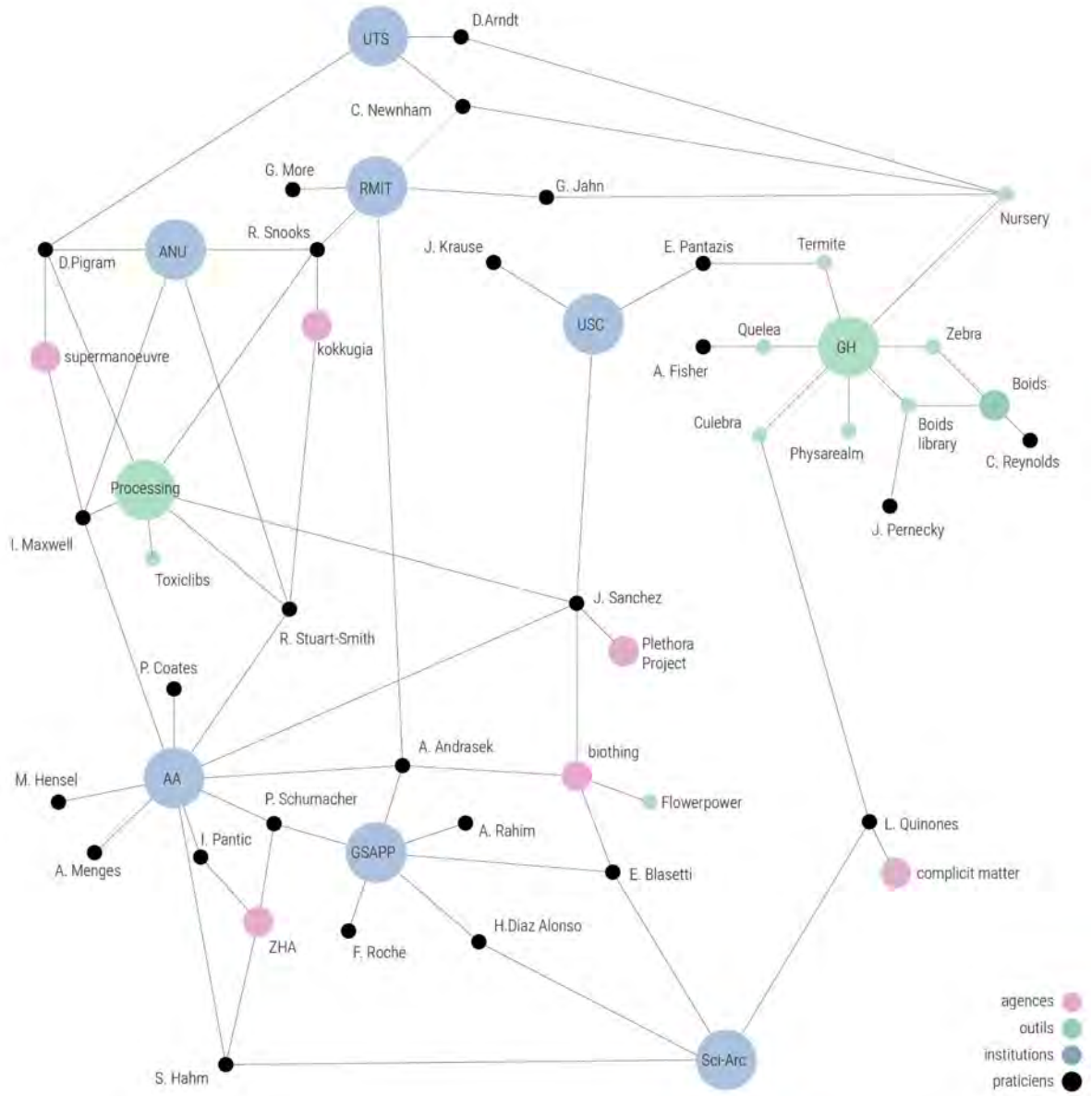



*Figure 11. Réseau de développement des systèmes multi-agents*

### *5.1.3 Simulations physiques*

*Figure 12. a. Modèle inversé de la Sagrada Familia, Antonio Gaudi ; b. Bulles de savon, Frei Otto; c. Modèles de coques inversées, Heinz Isler*

Parmi les typologies les plus usitées dans le champ computationnel se trouvent un ensemble de simulations physiques, qui permettent de mettre au point des programmes de recherche de forme. Ces techniques prennent racine dans les pratiques d'ingénieurs parmi les plus notables du XIX$^{e}$ et du XX$^{e}$ siècle. Les outils mathématiques à leur disposition commencent à permettre le calcul des efforts dans des structures de plus en plus complexes pour en permettre l'analyse. Se développent alors des techniques de recherche de forme analogues pour optimiser la répartition des efforts dans ces structures au moment de leur conception. Antoni Gaudi, architecte espagnol de la seconde moitié du XIX$^{e}$, particulièrement connu pour l'exubérance de ses réalisations, met à partir de 1883 à profit ce qu'il a observé dans les ponts suspendus pour mettre au point la géométrie de la Sagrada Familia. Cherchant à obtenir des séries d'arcs caténaires, il construit avec ses équipes une maquette composée de ficelles et de poids. La gravité donne aux ficelles des formes d'arcs, qui, inversés, deviennent les structures porteuses de la nef de la Sagrada Familia. Le même système de câbles suspendus est ensuite utilisé par Frei Otto pour développer la famille de structures dites gridshells[29], des nappes de membrures de bois qui combinent des géométries de caténaires à un système constructif qui permet de

[29] En allemand "*Gitterschalen*" - littéralement "coques en grille".

réaliser les plus complexes de ces formes, comme en témoigne le gridshell de Mannheim, dont il supervise la construction en 1975. Frei Otto développe également des méthodes de recherche de formes pour les surfaces minimales en travaillant avec des bulles de savon : les surfaces minimales sont en effet définies par leur limite, et permettent à l'intérieur de cette limite d'obtenir une répartition des contraintes idéales tout en minimisant la quantité de matière. Heinz Isler, ingénieur en génie civil suisse, mène pendant toute sa carrière des expériences sur les coques, qui lui permettent de développer une méthode de recherche de formes à partir de tissus humidifiés et suspendus, qu'il gèle ou enduit de ciment pour en figer les forme avant de transposer leurs géométries à l'échelle de couvertures entières de bâtiments. Les coques sont ensuite fabriquées en béton, matériau qui là encore permet d'explorer des géométries plus difficiles à réaliser en bois ou en acier. À partir des années 1950 se créent des institutions qui rassemblent les constructeurs qui se consacrent alors à ces recherches, offrant des lieux d'échange pour porter plus loin les travaux du domaine. Eduardo Torroja, ingénieur espagnol qui contribue lui aussi largement au développement des coques en béton armé, fonde en 1959 l'International Association for Shell Structures, qui devient en 1970 l'International Association for Spatial and Shell Structures. Celle-ci anime une conférence annuelle - ou Isler présente dès 1959 sa méthode de recherche de formes pour les coques - et publie le Bulletin of IASS, qui devient en 1995 le Journal of IASS, publication scientifique parmi les plus reconnues du domaine. Frei Otto, qui s'intéresse à l'ensemble des méthodes de modélisation et de construction de structures légères de grande portée, fonde à l'Université de Stuttgart en 1964 l'Institut für leichte Flächentragwerke (institut pour les structures légères). L'institut publie un grand nombre de comptes-rendus de ses recherches et s'investit dans la construction dans la région de plusieurs structures emblématiques de cette période et de ces travaux : le pavillon allemand de l'exposition universelle de 1967, la volière du Zoo de Munich, ou encore son stade olympique.

*Figure 13. a. Couverture de la cour du British Museum, Foster & Partners ; b. Stade Olympique de Munich, Frei Otto; c. Opéra de Sydney, Jorn Utzon*

Avec les progrès de l'informatique, la seconde moitié du XX$^{e}$ siècle voit le développement de simulations physiques qui permettent d'imiter ces processus en version numérique. Les processus de recherche de forme analogues en maquette sont accompagnés de travaux mathématiques qui permettent d'établir des descriptions formalisées des surfaces et courbes aux propriétés les plus intéressantes pour la construction. Ces descriptions peuvent ensuite être utilisées dans des programmes de modélisation qui rendent possible la recherche de forme sur écran, et le calcul des structures avant construction. C'est ce que John Argyris et Klaus Linkwitz mettent en œuvre pour la construction des trois nappes de câbles qui forment les couvertures des bâtiments du stade olympique de Munich en 1972. Celles-ci drapent les tribunes du stade, les piscines et les pistes, trois ensembles de grande taille qui nécessitent des couvertures de grande portée. Confrontés au besoin de donner forme à ces couvertures, qu'ils imaginent comme de gigantesques tentes de cirque, les architectes Günter Behnisch et Fritz Auer font appel dès la phase du concours à Frei Otto pour réaliser la

maquette du concours. Maquette qui devient immédiatement, suivant l'habitude de Frei Otto, un prototype à petite échelle de la structure. A terme, le processus ne permet cependant pas d'obtenir des mesures assez précises des composants des nappes de câbles, ni d'établir les plans d'exécution exacts. Klaus Linkwitz, l'ingénieur en charge du calcul de ces dimensions, propose de recourir à un modèle de calcul qu'il vient d'établir, et celui-ci est programmé à l'université de Stuttgart par John Argyris, qui travaille déjà depuis plusieurs années au développement de méthodes de calcul aux éléments finis [30]. Avant même la mise en application d'un processus de calcul informatisé par Frei Otto et ses équipes, c'est le bureau d'études d'Ove Arup qui recourt à un processus de calcul informatisé, dès le début des années 1960, pour parvenir à la définition géométrique exacte des voûtes de l'opéra de Sydney et pouvoir ensuite les évaluer structurellement[31]. Le recours à l'informatique permet aussi le prolongement du travail de Gaudi à la Sagrada Familia : Mark Burry, arrivé de Cambridge en 1979, encore étudiant, travaille jusqu'en 2010 à la finalisation de l'intérieur de la cathédrale, inachevée depuis la mort de son architecte, et contribue ensuite aux travaux extérieurs encore en cours. D'abord chargé de trouver la logique inhérente aux géométries produites par Gaudi, Burry a ensuite recours rapidement à des logiciels de modélisation 3D de l'industrie de l'aéronautique pour concrétiser ces formes[32].

Plus récente, la couverture d'une des cours intérieures du British Museum par Norman Foster s'inscrit elle aussi dans le développement de programmes de simulations physiques pour de la recherche de forme. La grande portée de la cour à couvrir, la répartition des appuis très contrainte en raison du bâti existant et les contraintes constructives, qui exigent de trianguler de manière régulière la surface à construire, poussent Buro Happold, bureau d'études chargé du suivi du projet, à recourir à la méthode de relaxation dynamique[33]. Il s'agit d'un modèle permettant de simuler les systèmes de câbles soumis à gravité dont nous avons mentionné que Gaudi les utilisait pour les caténaires de la Sagrada Familia, et que Frei Otto utilisait également pour ses gridshells. Le principe de cette méthode est montré en figure 15 : un réseau de points et de lignes est défini, qui représente un ensemble limité de particules et de ressorts. Parmi ces éléments, certains sont désignés comme fixes dans l'espace : ce sont les points d'ancrage. Des points d'application de poids définis sont ensuite désignés. Enfin, la déformation du système lorsque soumis à la gravité dans cette configuration est simulée : le réseau de

[30] Tomlow, Jos. "Designing and Constructing the Olympic Roof (Munich 1972)." *International Journal of Space Structures* 31, no. 1 (March 2016): p. 62–73.
[31] Françoise Fromonot, Jorn Utzon : *The Sydney OperaHouse*, Phaidon Press, 2002. Cité par William J. Mitchell, "A Tale of Two Cities: Architecture and the Digital Revolution", *Science*, 6 Aout 1999, Vol 285, Issue 5429, p. 839-841.
[32] Mark Burry, Jane Burry et Jordi Faulí, "Sagrada Família Rosassa: Global Computer Aided Dialogue between Designer and Craftsperson (Overcoming Differences in Age, Time and Distance)", dans *Proceedings of the 2001 ACADIA Conference*, p 76-85, 2001.
[33] Collaborative Design Procedures for Architects and Engineers, Oliver Tessmann

points et de lignes se déforme donc, et une géométrie est obtenue, en suivant le même principe que les ficelles déformées par des poids de Gaudi. Cette géométrie peut ensuite être transposée à l'échelle du bâtiment. Transposée par la suite dans plusieurs logiciels et plugins comme RhinoVault ou Kangaroo, cette même méthode est utilisée par exemple par le Block Research Group pour leur structure *Armadillo Vault*, exposée à Venise en 2016 (Figure 14). Les surfaces minimales, obtenues en suivant les mêmes principes mais en figeant le contour complet de la surface comme ancrage, sont aussi particulièrement prisées. EZCT a conçu la scénographie de l'exposition *Architectures Non Standard* en 2003 en y ayant recours, comme TheVeryMany pour la structure *Minima Maxima* ou Sean Ahlquist pour ses *Sensory Playscapes*. Les systèmes de relaxation dynamique servent par ailleurs pour modéliser voûtes, caténaires et surfaces minimales, mais aussi pour le placement des espaces dans le plan. Chaque espace à placer est représenté par un point et une aire autour de ce point, et lié par des lignes aux autres espaces auxquels il doit être connecté. Les différents points bougent dans le plan jusqu'à ce que la distance qui les sépare soit minimale tout en respectant l'aire prévue de chaque espace et les connexions entre eux. Un état d'équilibre est donc recherché, qui permet de satisfaire toutes les contraintes, comme pour les arcs soumis à gravité qui modifient leur géométrie en fonction des poids appliqués. Ces méthodes de recherche de forme s'ancrent dans une tradition constructive, mais une fois ces simulations popularisées, les praticiens y recourent aussi volontiers pour des recherches plus libres. Celles-ci s'éloignent parfois des impératifs physiques, notamment avec la relaxation dynamique, si facile à manipuler pour générer des voûtes. Dans le plugin Kangaroo par exemple, il suffit de modifier certains des paramètres de la simulation, comme la gravité qui peut être aisément fixée à 2, 3 ou 15G, pour obtenir des formes qui n'ont plus grand chose à voir avec la démarche constructive des praticiens qui ont contribué à développer ces algorithmes.

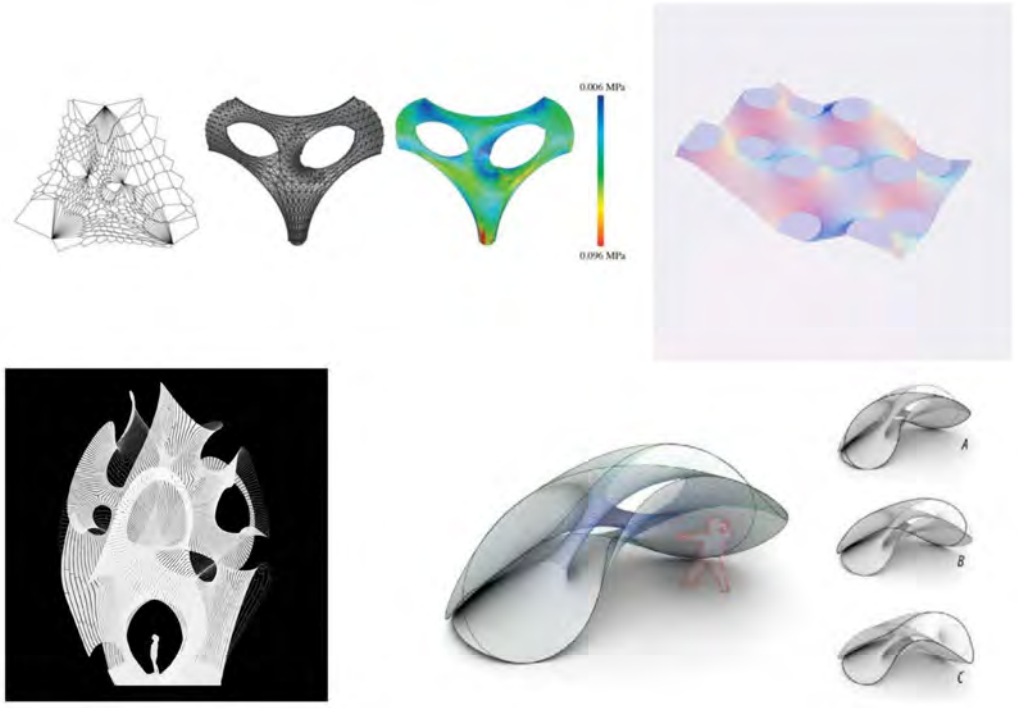



*Figure 14. a. Armadillo Vault, Block Research Group ; b. scénographie de l'exposition Architectures Non Standard, EZCT ; c. Minima Maxima, The Very Many ; d. Sensory Playscapes, Sean Ahlquist*

Le recours aux systèmes de simulations physiques s'inscrit pourtant initialement dans la démarche déjà abordée de rationalisation géométrique dans l'optique d'intégrer les outils computationnels dans les processus de conception usuels de l'industrie de l'architecture. Les difficultés que posent parfois certaines méthodes de recherche de forme, comme la relaxation dynamique qui ne permet pas toujours une couverture par éléments plans et des membres sans torsion, mène à la mise au point, en complément des simulations physiques, d'outils de rationalisation géométrique. L'enjeu se déplace ainsi de la recherche de forme en fonction de paramètres physiques à une recherche de forme en fonction de paramètres de fabrication. La courbure des surfaces et leur division en maillage deviennent des paramètres prédominants, pour garantir que les objets architecturaux conçus puissent ensuite être fabriqués. MeshEdit, LunchBox, Weaverbird et de nombreux autres outils de gestion des maillages apparaissent donc dans le sillage des outils de simulation physique. Le discours qui accompagne le recours à ces méthodes est donc résolument orienté vers la constructibilité. L'équilibre structurel, l'efficacité que ces simulations offrent amène le discours vers des notions d'optimisation structurelle et d'économie de matériaux, voire vers la faible empreinte écologique qui découle de ce dernier point. Les systèmes de simulation physique sont présentés dans ce sens presque comme la logique faite forme. Parmi les différentes typologies à l'étude ici, celle-ci est donc particulièrement influencée par la nature des relations entre architectes et ingénieurs dans les traditions constructives.

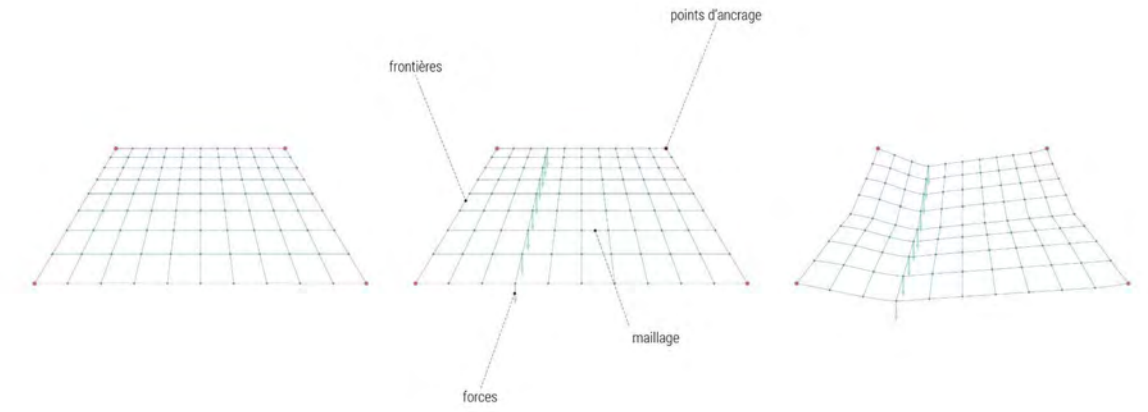



*Figure 15. Règles typologiques d'un système de relaxation dynamique*

Le réseau propre aux simulations physiques se développe, comme les autres, à partir des travaux des générations précédentes (figure 16). Les travaux de Per Galle sur l'optimisation du placement des espaces dans le plan, décrits dans le chapitre III montrent par exemple des prémisses de structures algorithmiques similaires à des systèmes de relaxation dynamique. Mais cette typologie prend aussi racine dans les pratiques de plusieurs groupes institutionnels un peu en dehors du champ computationnel. C'est le cas par exemple des travaux menés par Frei Otto et ses équipes à Stuttgart. Ce réseau fait aussi apparaître, plus que les autres, le pôle allemand et son importance. En effet, on y observe des praticiens comme Achim Menges, formés dans un premier temps dans d'autres pôles du réseau avant de revenir exercer en Allemagne. Ceux-ci, lors de leur période de séjour ailleurs contribuent largement, on l'a vu, au développement d'autres typologies, en particulier les systèmes de croissance. Mais on observe aussi dans le réseau des simulations physiques les équipes comme celles de Frei Otto ou les BET locaux. Ces derniers, comme B+G, participent également au développement d'autres typologies, en particulier les algorithmes génétiques. Mais le travail sur les simulations physiques est le seul à présenter ce double héritage du pôle allemand, ainsi que les travaux plus récents des praticiens revenus exercer notamment en Hesse, c'est-à-dire le pôle allemand dans sa globalité, et non seulement des parties. Le réseau des simulations physiques met aussi en évidence, plus que celui des systèmes de croissance ou celui des systèmes multi-agents, les nouveaux profils de cette génération notamment : agences et BET. Qu'il s'agisse de Schlaich, de Happold ou de Foster, les premiers praticiens à utiliser cette typologie pour des projets ne sont alors pas vraiment partie prenante du champ, mais vont s'y intégrer par le biais de leur travail sur ces algorithmes. Ce sont ensuite eux qui s'investissent dans Smart Geometry, forts de leur expérience sur ces précédents

projets. L'IASS, Smart Geometry et AAG montrent par ailleurs que le réseau des simulations physiques est aussi le lieu où apparaissent le plus d'associations. Ces nouveaux venus et nouvelles associations, les plus investis dans les questions d'industrialisation et de construction, sont très présents dans le réseau de cette typologie d'algorithmes alors même qu'ils sont absents des autres. Ceci découle du lien évident entre les applications de cette typologie et l'ancrage de ces praticiens dans l'industrie de l'architecture.

*Figure 16. Réseau de développement des systèmes de simulation physique*

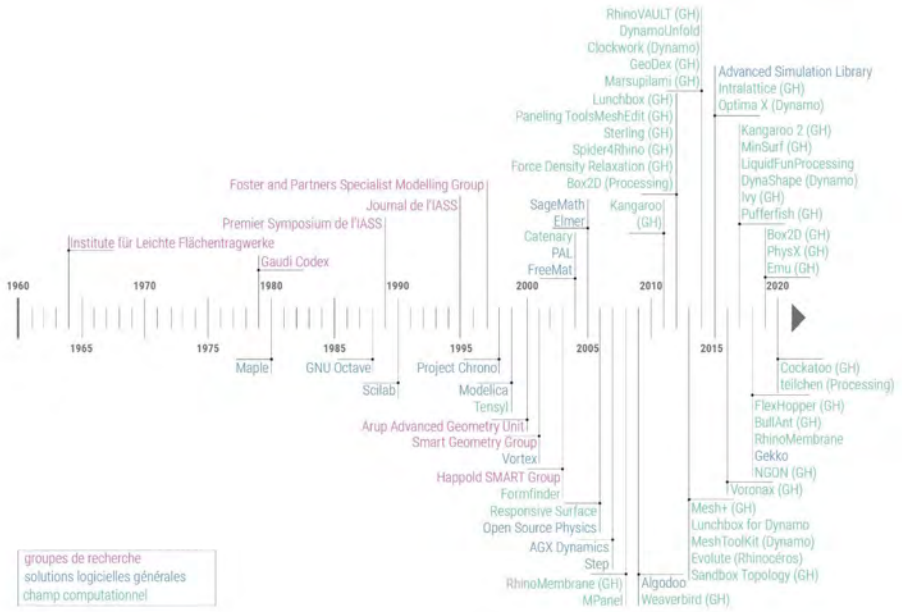



*Figure 17. Chronologie de développement des systèmes de simulation physique*[34]

### *5.1.4 Algorithmes génétiques*

[34] Il faut noter concernant la chronologie des outils de cette typologie que l'environnement de programmation comporte aussi beaucoup de mono-composants programmés par des développeurs extérieurs à Autodesk pour traiter des problèmes de maillage. Bien qu'ils soient disponibles au téléchargement pour d'autres utilisateurs, ils ne sont pas intégrés à la chronologie car il s'agit de solutions personnalisées mineures peu usitées au-delà de leurs développeurs. Il faut également signaler que beaucoup de ces algorithmes sont programmés dans des environnements généralistes.

Les algorithmes génétiques sont d'abord développés par des biologistes, généticiens et biophysiciens dans les années 1950 et 1960. Nils Aall Barricelli[35], Alex Fraser[36] ou John Holland[37] peuvent être cités comme pionniers. Ces simulations de la sélection naturelle deviennent ensuite rapidement un champ d'étude à part entière, point de rencontre de la biologie et de l'informatique. Ces algorithmes sont tous basés sur le même principe de sélection darwinienne[38]. Des populations d'individus sont générées et évaluées, et les caractéristiques des individus les plus performants sont retenues pour développer une nouvelle génération, un processus répété jusqu'à l'obtention d'une série d'individus aussi performants que possible. La typologie des règles pour les algorithmes génétiques comprend a) la sélection de la méthode de calcul, b) la définition du pool génétique (caractéristiques variant chez chaque individu), c) la définition de la fonction d'aptitude (les critères sur lesquels chaque individu est évalué), d) la sélection d'un individu parmi la dernière génération la plus performante (Figure 18). En complément de la fonction d'aptitude, il est également possible d'appliquer un coefficient de pénalisation ou de revalorisation de certains individus pour les différencier. Cela permet de prendre en compte des caractéristiques difficilement représentables par des fonctions mathématiques et ne pouvant donc pas vraiment être intégrés à la fonction d'aptitude. Par exemple, pour disqualifier un objet au maillage mal défini dans le cadre d'un algorithme génétique cherchant à maximiser l'apport solaire sur la surface de cet objet, il est possible de lui attribuer un très grand malus. Sa note devient ainsi automatiquement très mauvaise, le classant parmi les individus les moins performants. Autre exemple, pour prendre en compte le goût de l'utilisateur, il est possible d'attribuer à l'individu dont il a indiqué qu'ils lui conviennent bien un très grand bonus. Cet individu voit donc sa note augmenter fortement, et il sera donc classé parmi les individus les plus performants. Les différents composants de la fonction d'aptitude peuvent eux aussi se voir accoler des coefficients. Ceci permet de hiérarchiser les différents critères entre eux en fonction de leur importance. Le principe de l'optimisation multi-critères étant en effet de trouver une solution satisfaisante sur tous les plans, une négociation doit se faire entre les différents critères d'évaluation, négociation qui peut donc être ajustée via ces coefficients. Les différentes méthodes de calcul grâce auxquelles un algorithme génétique peut tourner varient dans la méthode de sélection des individus, qui peut soit se baser uniquement sur le score d'aptitude, soit y intégrer une part de sélection au hasard. Ceci a pour objectif de chercher dans tout l'espace des solutions plutôt que de risquer que l'évolution des individus concentre la recherche dans une partie de cet espace de recherche. Par ailleurs, la méthode de calcul de base des algorithmes génétique produit

[35] Nils Aall Barricelli, "Esempi numerici di processi di evoluzione", *Methodos* 1954, p. 45–68, 1954.
[36] Alex Fraser, "Simulation of Genetic Systems by Automatic Digital Computers I. Introduction", *Australian Journal of Biological Sciences*, Vol. 10, num. 4, p. 484, 1957.
[37] John Holland, *Adaptation in Natural and Artificial System*s, The MIT Press, 1975.
[38] Calixto, V. et Celani, G., « A literature review for space planning optimization using an evolutionary algorithm approach : 1992-2014 », dans *Project Information for Interaction :Proceedings of the 19th SIGRADI Conference*, Universidade Federal de Santa Catarina, 2015.

une génération après l'autre, de manière linéaire. Il est cependant également possible d'avoir recours à une méthode dite parallèle[39] . Plusieurs populations évoluent alors en parallèle. L'un des enjeux majeurs du recours à un algorithme génétique étant sa vitesse de computation, la formulation de la fonction d'aptitude est extrêmement importante. Mais la méthode de calcul choisie peut également aider à réduire le temps pris par l'algorithme pour converger, c'est-à-dire pour parvenir à un stade ou tous les individus d'une génération ont des performances équivalentes. Autre méthode, celle dite de l'ingénierie génétique : il s'agit de remplacer certaines séquences, ou paramètres, par d'autres identifiées comme efficaces au cours des premières évaluations[40]. L'adéquation des individus augmente donc plus vite. Ceci passe par une modification des opérateurs génétiques. Dans un algorithme génétique classique, ceux-ci sont au nombre de deux. La mutation fait varier certains paramètres au hasard. L'enjambement, ou croisement, croise les caractéristiques de deux individus pour en produire un troisième. Les autres opérateurs disponibles comprennent aussi par exemple l'extinction : pour éviter le problème de stagnation dans un minimum ou maximum local, une caractéristique qui pose problème est supprimée[41]. En complément de sa structure de base, la typologie des algorithmes génétiques comprend donc un certain nombre de possibilités complémentaires, visant à améliorer le processus d'optimisation.

Après les biologistes, ces outils, dit évolutionnaires, sont ensuite adoptés par les ingénieurs pour résoudre des problèmes d'optimisation complexes. Ils sont mis en œuvre pour une grande une variété d'applications : cryptage de données, conception de pièces de moteur ou encore classement des joueurs. Dans de nombreux domaines, les algorithmes génétiques deviennent donc des outils d'optimisation populaires, en raison de leur capacité à prendre en compte de multiples critères et à introduire des mutations afin d'éviter les minima ou maxima locaux, améliorant ainsi leurs performances. Les algorithmes génétiques sont aussi popularisés auprès du grand public, dont les architectes, notamment à travers le travail de Richard Dawkins. En 1986, A la parution de son livre *The Blind Watchmaker*[42], qui porte sur certains aspects de la théorie de l'évolution, il y intègre la description d'un algorithme génétique, BioMorphs. Celui-ci a pour objectif de permettre d'explorer un petit espace de solutions et de faire comprendre au lecteur comment elles se positionnent par rapport à un objectif donné, ainsi que comment la méthode évolutionnaire permet de rechercher un optimum.

[39] Calixto, V. et Celani, G., « A literature review for space planning optimization using an evolutionary algorithm approach : 1992-2014 », dans *Project Information for Interaction :Proceedings of the 19th SIGRADI Conference*, Universidade Federal de Santa Catarina, 2015.
[40] Calixto, V. et Celani, G., « A literature review for space planning optimization using an evolutionary algorithm approach : 1992-2014 », dans *Project Information for Interaction :Proceedings of the 19th SIGRADI Conference*, Universidade Federal de Santa Catarina, 2015.
[41] Gan Zhen Ye et Dae-Ki Kang, "Extended Evolutionary Algorithms with Stagnation-Based Extinction Protocol", *Applied Sciences* 2021, 11(8), 3461.
[42] Richard Dawkins, *The Blind Watchmaker*. New York: W. W. Norton & Company, 1986.

D’abord simple description, BioMorphs est ensuite vendu par l’éditeur de Dawkins sur disquette avec le livre, permettant aux lecteurs de manipuler le programme. L’utilisateur est confronté à plusieurs BioMorphs, petites compositions morphologiques, et doit choisir sa favorite. Le programme réajuste donc les paramètres de dessin des différentes formes proposées pour se rapprocher de ce choix. Au fur et à mesure, le choix de l’utilisateur oriente donc la morphologie des BioMorphs. Ludiques, les petits monstres que permet de créer en un clin d'œil l’algorithme rencontrent un grand succès et contribuent largement à faire sortir les algorithmes génétiques du champ de la biologie et de leur usage initial de modélisation de l’évolution.

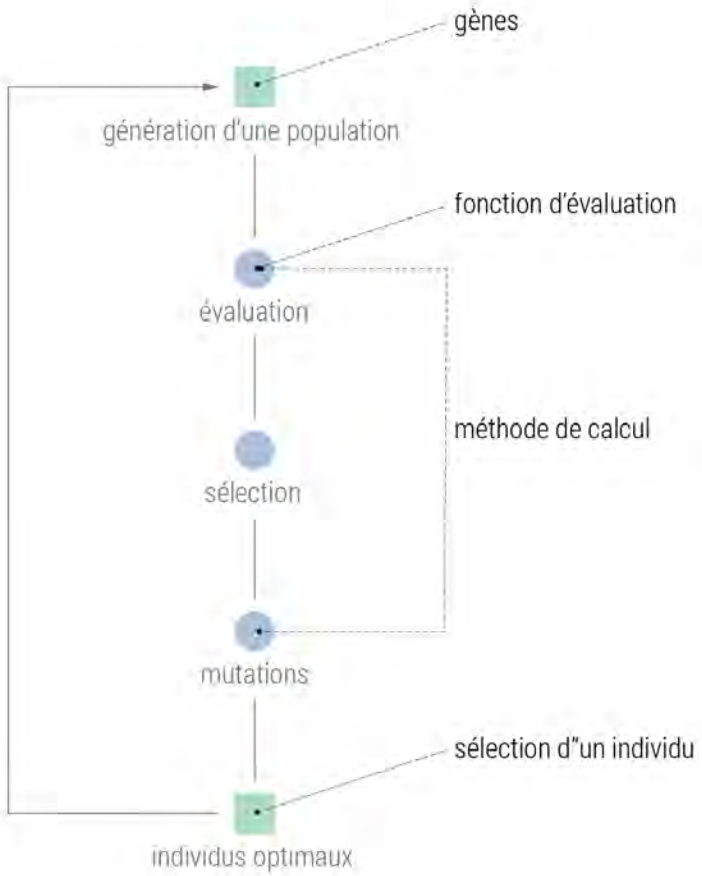



*Figure 18. Règles typologiques d’un algorithme génétique*

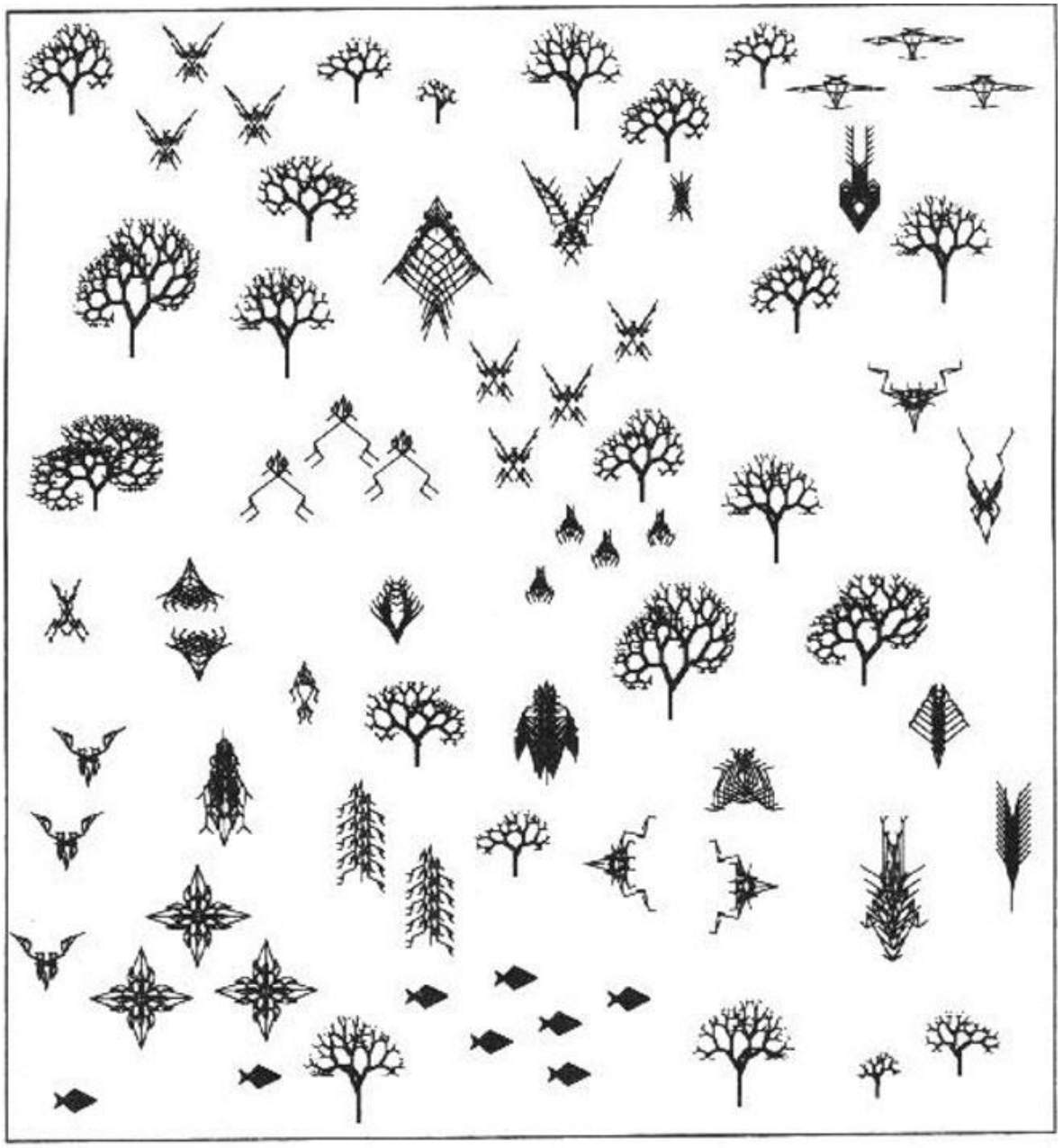


*Figure 19. Biomorphs, The Blind Watchmaker, Richard Dawkins*

A la fin des années 1980, quelques logiciels permettant d'utiliser des algorithmes génétiques pour la résolution de problèmes divers sont publiés, comme le logiciel Evolver. Commercialisé par Axcelis, celui-ci permet d'utiliser des algorithmes génétiques pour évaluer des risques financiers ou optimiser un planning . A partir des années 1990, la plupart des langages incluent des fonctions de programmation pour mettre en œuvre des algorithmes génétiques. C'est le cas de Matlab, de Fortran,

de C++ ou de Python, qui acquièrent tous au fil des années qui suivent des bibliothèques dédiées. Par la suite, la majeure partie de la programmation d'algorithmes génétiques s'effectue dans ces langages de programmations généralistes, à l'aide de ces bibliothèques. Les années 2000 voient un bref pic de développement de logiciels spécifiquement pour le champ de la biologie, où investit notamment Autodesk. Les quelques logiciels qui survivent restent par ailleurs cantonnés à des applications spécialisées. Leur capacité à explorer un vaste espace de conception rend les algorithmes génétiques prisés non seulement par les ingénieurs, mais aussi par les architectes qui, à cette même période, se les approprient de plus en plus, aux côtés des systèmes de croissance également populaires pour leur potentiel formel. Historiquement cependant, l'exploration de ces derniers est d'abord prédominante parmi les expérimentations architecturales, avant que les algorithmes génétiques prennent ensuite progressivement plus d'importance dans le champ. Cette hégémonie s'accompagne du développement de divers outils. Quelques outils confidentiels, en usage seulement dans le cercle autour de leurs développeurs, apparaissent d'abord, comme GEN-lite ou GENR8. Puis Grasshopper intègre Galapagos et Octopus, les deux plugins principaux du logiciel pour avoir recours à des algorithmes génétiques. Dynamo propose également un module très simple d'utilisation, mais aussi des outils de conception de colonnes de circulation ou autres fonctions spécifiques qui intègrent dans leur flux de travail cette typologie.

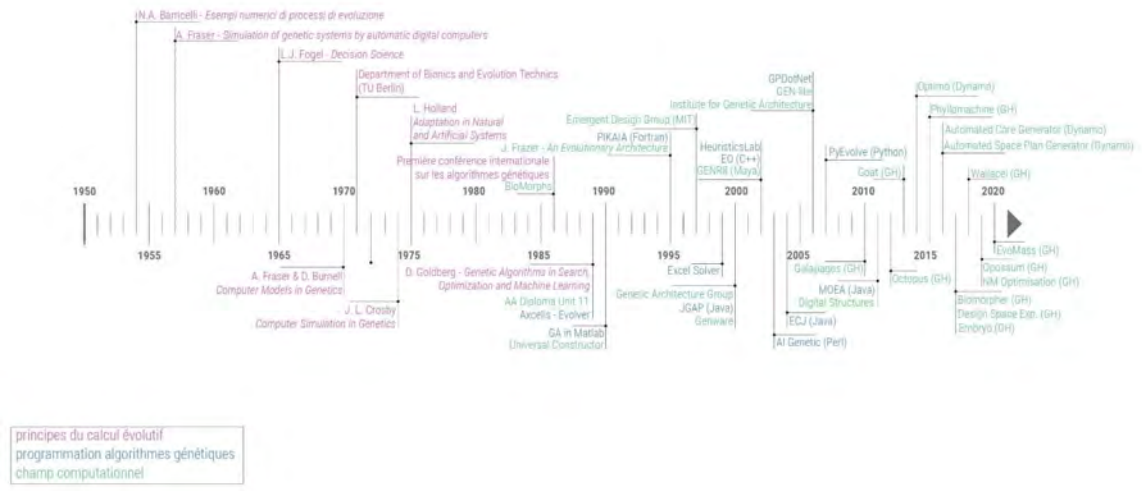



*Figure 20. Chronologie de développement des algorithmes génétiques*

Les praticiens du champ computationnel ont recours aux algorithmes génétiques dans des cadres assez divers. La typologie est assez flexible dans le sens ou il faut simplement pouvoir formuler une fonction d'aptitude et indiquer des paramètres comme liste de gènes. Les domaines d'application ne

dépendent donc que de la réussite de cette explicitation. Néanmoins, dans la mesure où il s'agit d'un outil d'optimisation, cela requiert des critères d'évaluation. Les algorithmes génétiques vont donc main dans la main avec de nombreuses méthodes d'évaluation des caractéristiques d'un espace ou d'un bâtiment, du respect des connections entre les pièces d'un même appartement à sa résistance à un cas de charge ou à son impact environnemental[43]. Par ailleurs, il sont fréquemment utilisés pour de l'optimisation structurale, thermique, lumineuse, ou les trois ensemble puisque l'un des avantages est la combinaison de critères. Chez John Gero et son équipe, parmi les premiers à s'intéresser aux applications de ces mécanismes d'optimisation à la conception architecturale, ils sont utilisés pour de la génération de plans. Pour ce faire, une première étape du flux du travail génère des populations de plans divers. Ceux-ci sont ensuite évalués en fonction de leur adéquation à un set de critères. Le premier est la continuité entre les différents espaces du plan, le second est l'absence de superposition entre deux espaces différents, et le troisième est la relation entre les différents espaces. Par exemple, pour trois espaces, "*A, B et C, A devrait être adjacent à B, B adjacent à C, et A n'être pas adjacent à C*"[44]. EZCT, dont nous avions vu le recours aux algorithmes génétiques pour le concours du pavillon Seroussi, ou ils y avaient recours pour optimiser la répartition de la lumière, n'en est alors pas à son premier coup d'essai. En 2004, le projet *Studies on Optimization : Computational Chair Design* les voit fabriquer une chaise en cubes de bois assemblés. Cette répartition des cubes est le résultat d'un algorithme génétique. Les cubes forment initialement un volume de chaise minimal, fait d'un bloc cubique pour l'assise et un dossier rectangulaire. Pour générer des populations de chaises, l'algorithme repose sur la même logique que le pavillon Seroussi. Il s'agit de tailler dans le volume en supprimant certains des cubes. L'emplacement et l'ampleur des suppressions sont déterminées par une évaluation. Dans le cas des chaises, l'évaluation est assez simple : il s'agit de trouver un équilibre entre la minimisation du nombre de cubes et la capacité de reprise des charges. Autrement dit, la chaise doit avoir le moins de matière possible tout en supportant le poids de quelqu'un s'asseyant dessus[45]. Autre exemple, le travail du groupe Blackbox Studio chez Skidmore, Owings & Merrill. En 2008, le groupe publie dans le papier *Architectural Genomics* un récapitulatif de ses expérimentations avec des algorithmes génétiques[46]. Celles-ci sont menées notamment sur les conseils d'Una May O'Reilly, une informaticienne spécialiste des algorithmes évolutionnaires qui avait déjà assisté le Emergent Design Group dans ses travaux sur le sujet. L'essentiel des recherches du Blackbox Studio porte sur l'utilisation des outils computationnels pour la conception d'enveloppes, notamment pour des tours.

[43] La question de l'évaluation sera traitée en détail au chapitre VII.
[44] "*A, B and C, where A should be adjacent to B, B adjacent to C and A should not be adjacent to C*." Damski, J. C. and Gero, J. S., "An evolutionary approach to generating constraint-based space layout topologies", dans R. Junge (éditeur), *CAAD Futures 1997*, Kluwer, Dordrecht. pp. 855-864, 1997.
[45] EZCT Architecture and Design Research, "Studies on Optimization: Computational Chair Design using Genetic Algorithms (with Hatem Hamda and Marc Schoenhauer)", 2004. http://transnatural.org/wp-content/uploads/2011/09/EZCT_Booklet-Screen.pdf, consulté le 02 Septembre 2021.
[46] Besserud, K and Cotten, J., "Architectural Genomics" Presentation. ACADIA. Minneapolis, USA, 2008.

Nous avions mentionné plus haut que la plupart des projets pour lesquels Skidmore, Owings & Merrill recourent à des algorithmes sont des tours au cours des années 2000, et c'est également pour ces programmes qu'ils expérimentent avec les algorithmes génétiques. Celui présenté dans l'article a pour objectif de maximiser les radiations solaires incidentes sur l'enveloppe. Les gènes qui varient pour obtenir différents individus sont les paramètres de définition de la géométrie de la tour : cinq anneaux à différentes hauteurs, de rayon et d'orientation variables. Pour l'évaluation, l'analyse solaire est ensuite faite dans Ecotect, en prenant en compte les bâtiments environnants également.

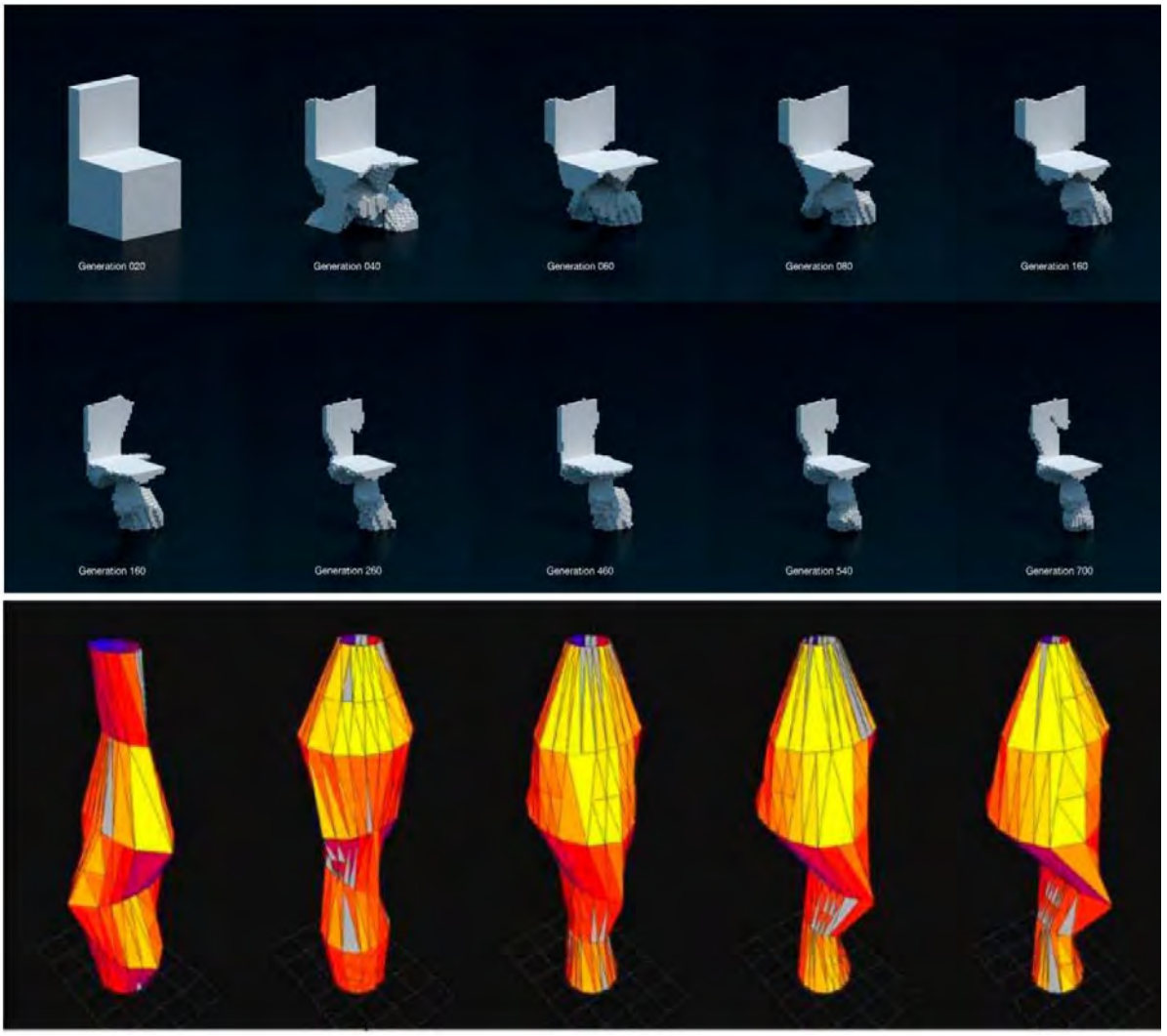



*Figure 21. a. Studies on Optimization: Computational Chair Design, EZCT ; b. Recherches pour une tour, Skidmore, Owings & Merrill*

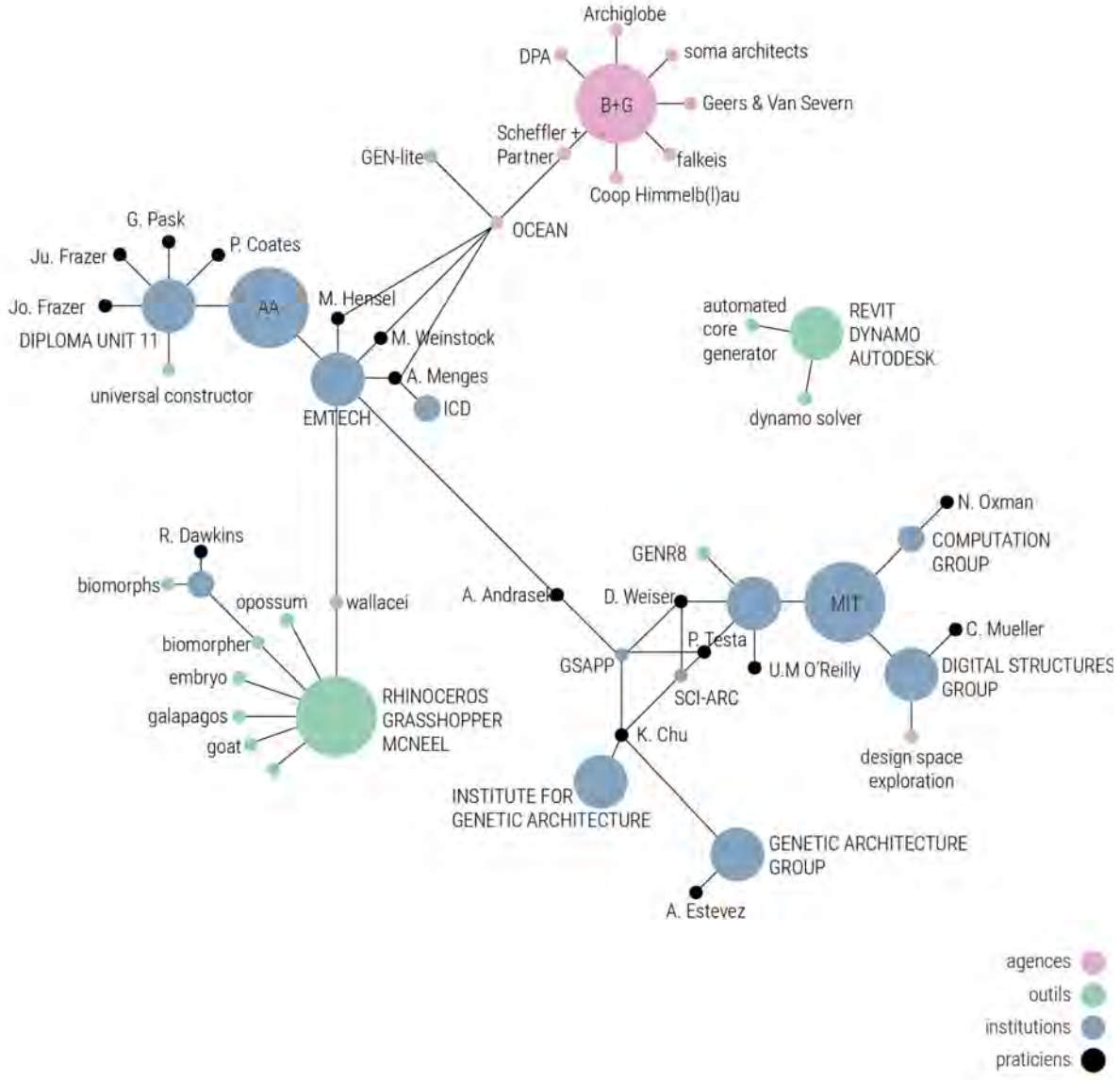



*Figure 22. Réseau de développement des algorithmes génétiques*

Les algorithmes génétiques apparaissent aussi rapidement au sein de l'unité de John Frazer à la AA, au M.I.T., chez Bollinger + Grohmann ou chez Autodesk. Ils sont prisés pour leurs performances d'optimisation, pour la diversité de solutions équivalentes parmi lesquelles ils permettent de choisir, ou de faire choisir à un utilisateur. Avec les systèmes de croissance, ils font de plus partie des typologies les plus en vue au moment où les techniques de morphogenèse sont les plus en vogue dans le champ. Leur fonctionnement calqué sur la théorie de l'évolution darwinienne soutient un discours d'explications des mécanismes d'apparitions des formes naturelles, sur lesquelles la conception architecturale peut donc désormais se baser. Par ailleurs, avec les systèmes de simulation physique, les algorithmes génétiques sont l'autre pan typologique du recours à l'optimisation, et les explications

fournies à leur sujet s'alignent également sur cette rhétorique de l'efficacité. Optimisation et maîtrise de la nature se rejoignent donc dans l'explication donnée par les praticiens du potentiel des algorithmes génétiques pour la conception architecturale[47]. Les algorithmes génétiques apparaissent en premier lieu à l'AA, à la fois entre 1989 et 1996 sous la direction de John Frazer et au début des années 2000 avec les travaux de Michael Hensel, Michael Weinstock et Achim Menges. Il comprend également les premières applications pour des projets construits par Bollinger + Grohmann, en association avec quelques cabinets d'architecture de premier plan à partir de 2007, ainsi que de grands pôles de logiciels tels que McNeel ou Autodesk. Par rapport à d'autres réseaux, comme celui des systèmes multi-agents, constitué d'universitaires ou de petits agences et de logiciels libres, ce réseau implique un plus grand nombre d'acteurs de l'industrie AEC ainsi que de l'industrie du logiciel. Cette plus grande variété d'acteurs dans les pôles implique une plus grande diversité dans les utilisateurs et dans les usages, comme vu plus haut. En outre, alors que le réseau des systèmes multi-agents comporte peu d'outils voyageant d'un pôle à l'autre, le système d'outils évolutifs comporte peu de pôles générant chacun plusieurs outils.

### *5.1.5 Typologies d'algorithmes - en conclusion*

Les algorithmes génétiques nécessitent par ailleurs qu'on leur associe une méthode de recherche de forme puisque ce n'est pas compris dans leur structure. Ils sont donc souvent associés à une autre typologie pour créer les individus à évaluer, quand ce n'est pas une simple géométrie paramétrique. Le programme GENR8 développé par l'Emergent Design Group en est un exemple intéressant. Plutôt que de se servir d'un algorithme pour rechercher directement une forme optimale parmi des variations géométriques, l'algorithme génétique fait évoluer les règles syntaxiques d'un L-système, faisant converger ce second algorithme vers une paramétrisation optimale[48]. Il s'agit d'une combinaison entre systèmes de croissance et algorithmes génétiques prisée par Paul Coates également, qui s'en était servi dans le cadre de plusieurs projets avec ses étudiants à l'UEL. Combiner les typologies se fait souvent, comme les exemples donnés plus haut de combinaison de deux systèmes multi-agents chez supermanoeuvre ou d'un système multi-agent avec un automate cellulaire chez biothing le montrent. Au sein de l'unité 11 des Frazer, la combinaison de plusieurs systèmes de croissance, par exemple de deux automates cellulaires, l'un réagissant aux états du précédent, est également un dispositif auquel les étudiants recourent régulièrement. Sur le même principe que Genware ou que les bibliothèques internes de kokkugia, les membres de l'unité ont accès à une bibliothèque d'algorithmes, relevant tous

[47] Plus de détails sur la question du discours sont donnés au chapitre VII.
[48] Hemberg M., O'Reilly UM., Menges A., Jonas K., Gonçalves M..C., Fuchs S.R. (2008) Genr8: Architects' Experience with an Emergent Design Tool. In: Romero J., Machado P. (eds) *The Art of Artificial Evolution*. Natural Computing Series. Springer, Berlin, Heidelberg

soit des systèmes de croissance soit des algorithmes génétiques - que Frazer rassemble sous le nom de méthodes évolutionnaires. Leurs projets sont donc le fruit de programmes combinant ces méthodes de génération. Autre exemple, kokkugia propose dans son projet Mars la combinaison entre une simulation physique permettant d'obtenir une surface minimale, qui fait office de coque. Celle-ci est ensuite ponctuellement transformée, ajourée par les déplacements d'un système d'agents[49]. Entre utilisations telles quelles et combinaisons, les typologies couvrent la majeure partie des pratiques du champ à partir de leur cristallisation dans les années 1990, et pendant le fort développement du champ au cours des années 2000.

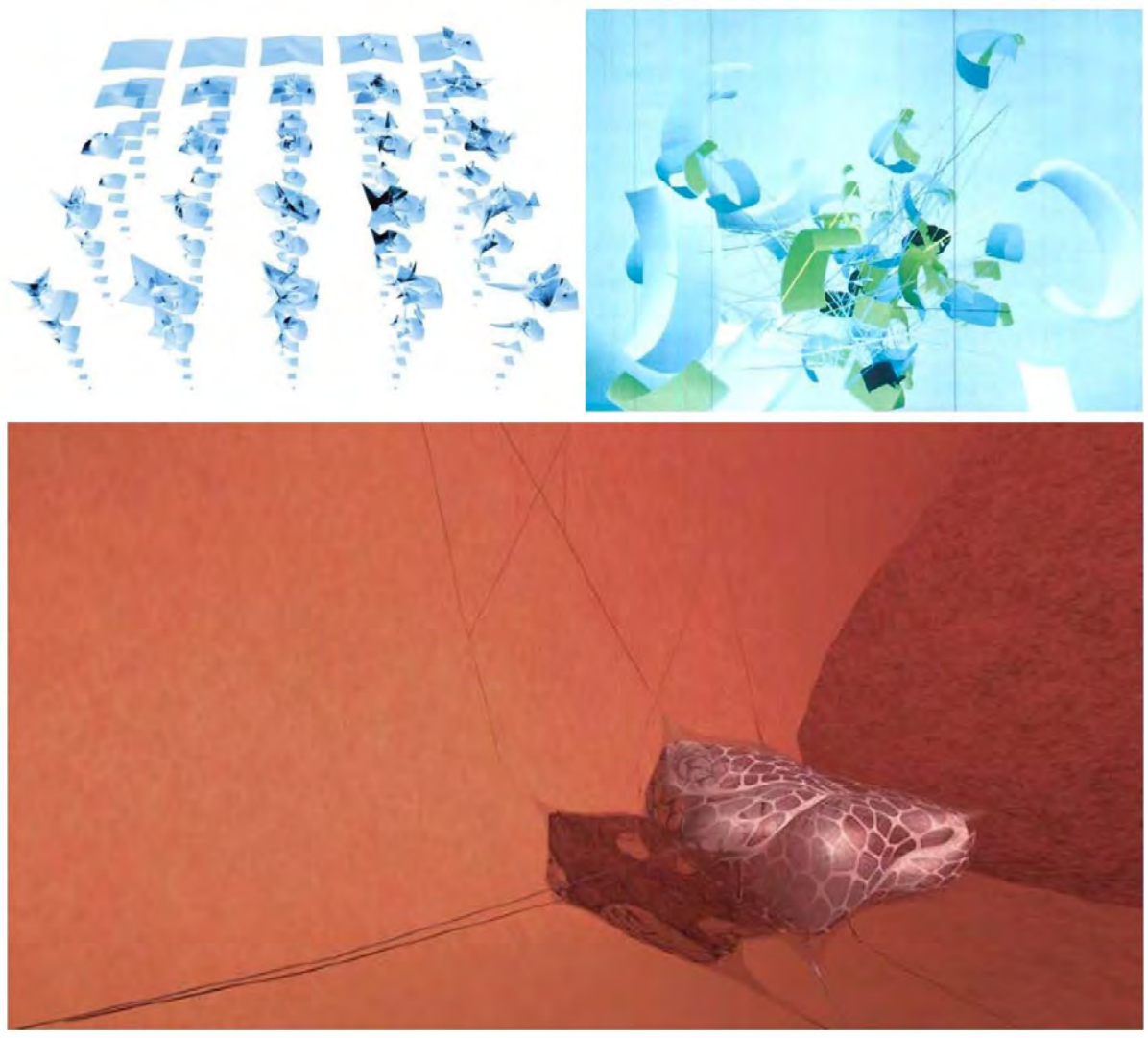


*Figure 23. a. GENR8, Emergent Design Group ; b. Evolutionary Architecture, John Frazer ; c. Projet MARS, kokkugia*

[49] https://www.kokkugia.com/MARS, consulté le 21 Aout 2021.

Observer l'apparition des typologies met en évidence deux origines qui s'entremêlent. D'une part, les prémisses des typologies dans les pratiques des générations précédentes, qui forgent la manière d'y recourir par la suite. D'autre part, l'appropriation à partir de typologies algorithmiques d'autres domaines, comme la biologie ou l'animation, des emprunts qui viennent enrichir tant l'arsenal d'outils à disposition des praticiens que leur pratique architecturale et leur discours. L'apparition des typologies dans le champ computationnel prend par ailleurs racine dans l'histoire de l'informatique telle qu'elle s'est développée en dehors du champ de l'IA de ses débuts, ou du champ de l'IA tel qu'il existe ensuite, comme domaine de recherche spécialisé. Nous avons vu au chapitre III les débuts de l'infographie, les systèmes multi-agents dévoilent ici la suite de l'impact sur le champ computationnel en architecture de l'industrie de l'animation. Les systèmes de croissance sont le fruit d'expérimentations des mathématiciens avec l'informatique naissante. La typologie donne aussi à voir, avec les algorithmes génétiques, les expérimentations des biologistes, alors que le champ de l'informatique fournit peu à peu des ordinateurs à d'autres domaines. Enfin, les simulations physiques, si elles servent ailleurs pour simuler vent et soleil, sont, peut-être un peu plus que les autres, l'espace de développement de structures algorithmiques propres à l'architecture et à la construction. Le devenir commun de l'informatique partout qu'on aperçoit entre les lignes du développement de ces typologies, ancrant plus fortement que jamais le champ computationnel, malgré une certaine confidentialité dans le champ architectural parfois, dans une histoire plus large. L'apparition des typologies dans le champ computationnel symbolise à ce titre le devenir commun de l'ordinateur dans les pratiques de tous les domaines.

Les typologies sont aussi des outils techniques qui donnent à voir la construction d'un discours pour accompagner les pratiques du champ computationnel. Chacune d'entre elles s'articule avec un aspect particulier du rôle des pratiques computationnelles dans la conception architecturale. En complément des possibilités techniques, c'est aussi ceci qui oriente les praticiens vers le recours à une typologie plutôt qu'une autre : programmer la matière, viser une efficacité structurale, optimiser le projet, trouver les formes les plus naturelles. Mais des notions clés affleurent cependant autour de toutes les typologies : constructibilité, efficience, maîtrise de la nature. Dans l'ensemble c'est un discours de maîtrise, de contrôle, de rationalité, beaucoup plus qu'une reconnaissance de la dimension expérimentale, prospective qu'on observe pourtant dans les pratiques. Le discours qui accompagne le recours aux typologies est lui aussi le signe d'un certain écart entre pratiques et discours. Cette divergence s'observe notamment en comparant les entretiens aux prises de paroles en conférences et aux textes publiés. En entretien, les praticiens sont beaucoup plus ancrés dans la pratique, ils citent parfois quelques lectures mais peu. Leur pratique est décrite comme des architectes sur forme et

espace, avec des algorithmes mobilisés comme n'importe quels outils. Ces algorithmes sont par ailleurs décrits de manière simple et technique, sans tout l'appareil théorique qui leur est prêté ailleurs. Certains points du discours général sur le recours aux outils algorithmiques pour la conception architecturale, formulés par les praticiens des générations précédentes. Les machines à architecture de Nicholas Negroponte, la logique de l'architecture de William Mitchell, le non-standard de Bernard Cache sont toujours présents dans le discours[50]. La nature comme source d'inspiration et de méthode apparaît cependant très fortement en complément de ces points, avec beaucoup d'emprunts au champ de la biologie dans le vocabulaire. Enfin les typologies sont présentées comme des vecteurs pour y parvenir. De plus, le discours est accompagné par un grand enthousiasme et une conviction que la pratique de l'architecture est révolutionnée par le recours aux outils computationnels. Ce développement du discours coïncide avec la mise en lumière des pratiques computationnelles dans les années 2000, dans des expositions et des publications. Le discours des praticiens se construit pour accompagner la reconnaissance de leurs pratiques comme un champ à part entière, qui nécessite donc un discours théorique pour accompagner les pratiques.

Les typologies permettent par ailleurs de comprendre la structuration du champ computationnel, et ses méthodes de transmission du savoir-faire. La filiation entre générations qui s'établit dans le champ passe par ces typologies, comme le montrent les réseaux. Il s'agit d'une filiation institutionnelle mais aussi pratique. Ces typologies représentent des outils particulièrement efficaces pour transmettre le savoir-faire accumulé par les praticiens de ces premières générations du champ. Avec leurs règles fixes, ce sont des outils technique particulièrement utiles, parce qu'elles condensent la variabilité en quelques paramètres qu'un novice peut apprendre à manipuler assez vite. Leurs structures algorithmiques sont aisées à saisir, sans rester figé dans un format d'outil donné. Les typologies peuvent en effet être mise en œuvre indifféremment en programmant intégralement un algorithme, en utilisant un algorithme programmé par quelqu'un d'autre, voire en passant par un logiciel. Les typologies bénéficient de niveaux de complexité différents. Il est possible par exemple de modifier les paramètres d'un comportement prédéfini d'un système multi-agents pour comprendre la notion d'interactions des particules, de comportement émergent. Puis ensuite, une fois cette étape maîtrisée, d'aller plus dans le détail pour modifier l'écriture du comportement plutôt que simplement ces paramètres. C'est le même principe pour les algorithmes génétiques, pour lesquels on peut modifier la fonction d'aptitude et les gènes avant de s'intéresser dans un second temps aux opérateurs génétiques et à la méthode de calcul. Les typologies permettent donc de se lancer facilement dans une expérimentation procédurale, et d'augmenter la maîtrise de la programmation au fur et à mesure, en voyageant dans les formats d'outils tout en conservant la structure de la typologie en point de repère.

[50] Voir Chapitre III sur les textes fondateurs du champ.

Or c'est précisément cette expérimentation que les praticiens expérimentés cherchent à déclencher pour permettre l'apprentissage du tacite nécessaire, le savoir spécifique au champ. Mais la courbe d'apprentissage que permet ce voyage entre les formats, clé de l'apprentissage des pratiques computationnelles, passe aussi par l'interface offerte par les différents formats d'outils.

## 5.2 Formats d'outils

### *5.2.1 Algorithmes sur mesure : des constructions affinées*

L'apprentissage du savoir-faire du champ computationnel passe par les typologies, mais aussi par les formats d'outils, qui varient. Les typologies désignent les structures algorithmiques mises en œuvre au cours du processus de conception. Les formats d'outils, eux, désignent les différents outils de programmation, qui diffèrent dans la manière d'accéder aux différentes fonctions et dans la complexité des fonctionnalités. Le premier format à disposition des praticiens du champ computationnel sont les algorithmes sur-mesure. Ceux-ci sont écrits entièrement par les praticiens, et les programmes exécutés sont ensuite composés d'un assemblage de ces algorithmes[51]. Les algorithmes sont écrits au sein d'nu environnement de programmation, une interface qui permet au praticien de saisir les mots qui composent le code puis de l'exécuter. Au sein du champ computationnel, ces environnements de programmation sont systématiquement associés à une plateforme de visualisation des géométries qui sont usuellement le résultat de l'exécution. Ces plateformes sont généralement des logiciels de modélisation, dont le code mobilise les capacités de dessin et d'affichage au moment de l'exécution. L'environnement de programmation et la plateforme de visualisation sont donc les éléments principaux du set-up des praticiens, qui régit leurs possibilités de programmation. Ce qu'il est possible d'implémenter dans un algorithme sur-mesure dépend donc à la fois du niveau du langage de programmation employé et du kernel du modeleur[52]. Ceci est fonction des choix de développement fait par les développeurs des deux éléments, qui dépendent à leur tour des utilisateurs et des applications ciblées. L'une des caractéristiques qui déterminent les possibilités offertes par ces algorithmes sur-mesure est le paradigme de programmation dans lequel s'inscrit le code. Le paradigme de programmation le plus répandu au sein du champ computationnel est la programmation orientée objet. Comme son nom l'indique, celle-ci est une logique de programmation construite autour d'objets quels qu'ils soient. Il s'agit donc d'exprimer par le code les propriétés de ces objets et les circonstances dans lesquelles elles se modifient[53]. Robert Aish, développeur chez Bentley Systems et Autodesk dont nous avons déjà brièvement parlé et dont nous allons voir qu'il a

[51] Certains pouvant être réutilisés dans d'autres programmes évidemment.
[52] Voir Chapitre II et III.
[53] T. Elrad, R. E. Filman, and A. Bader. Aspect–Oriented Programming. Communications of the ACM, 44(10):29–32, October 2001.

joué un rôle clé dans le développement des outils du champ computationnel, présente la programmation orientée objet comme “basée sur une métaphore entre les objets du monde réel et les objets computationnels”[54]. Ceci est supposé permettre d’organiser un programme de manière plus proche des structures mentales de ceux qui programment, puisque n’importe quel objet du monde réel peut trouver un équivalent computationnel. Le travail de programmation consiste d’abord et avant tout à construire le modèle de représentation du phénomène que l’on cherche à modéliser[55]. Ce modèle peut ensuite être directement exprimé dans le programme, sans s’encombrer des couches logiques propres à la programmation dans d’autres paradigmes. La programmation orientée objet comporte donc des avantages techniques , mais aussi théoriques. Le discours tenu par les praticiens du champ qui montrent qu’ils trouvent un sens philosophique à ces structures de programmation, promouvant d’autant plus les outils associés. Ce paradigme, en plus d’avoir été une révolution dans le monde de la programmation, est aussi indissociable des algorithmes sur-mesure développés par les praticiens du champ computationnel, dont les modeleurs et les environnements de programmation sont construits sur des langages orientées objets.

Maya, nous l’avons vu plus tôt, est un logiciel de création d’images de synthèse, commercialisé à partir de 1998 par Alias, dont c’est le produit phare après la première série de rachats des éditeurs de logiciels des débuts de l’infographie[56]. A la suite du rachat d’Alias par Autodesk en 2005, Maya devient Autodesk Maya, mais peu de choses changent dans les fonctionnalités du logiciel et dans ses usages. Le logiciel est particulièrement prisé pour les algorithmes de systèmes multi-agents et de simulations physiques qui permettent d’animer de l’eau, du feu ou encore des tissus[57]. Mais il est également assez prisé des praticiens du champ computationnel en architecture, comme Greg Lynn, Mark Foster Gage ou Zaha Hadid[58]. L’une des raisons de cette affection est l’existence de MEL, langage de programmation associé à Maya. Une partie de Maya est programmé en C++, mais le reste est programmé en MEL, notamment son interface. Ceci permet à Maya d’offrir un module de mémorisation des commandes aisément transformable en script. Lorsque l’utilisateur exécute une série de commandes, il peut ensuite accéder aux lignes de code MEL correspondantes, et enregistrer ce script en tant que fonction. Ce faisant, un bouton peut être créé sur l’interface, et l’utilisateur peut accéder à cet enchaînement de commandes directement la fois suivante. MEL sert donc en premier lieu à éviter les répétitions au cours du processus de modélisation, sur le même principe que les macro

[54] “*Object oriented software is based on a metaphor between real-world objects and computational objects*.” Robert Aish, “First Build Your Tools”, dans Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.
[55] Robert Aish, “First Build Your Tools”, dans Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.
[56] Voir Chapitre III.
[57] Par exemple, c’est Maya qui a servi à animer le tapis volant dans le film d’animation de Disney *Aladdin*.
[58] Voir Chapitre III.

dans Excel ou les actions dans Photoshop. Ses utilisations principales sont la customisation d'interface et l'automatisation de commandes de modélisation. Mais en plus d'enregistrer facilement des raccourcis vers des suites de commandes manuelles, MEL permet bien sûr aussi de programmer des algorithmes plus poussés. Les praticiens du champ computationnel s'en servent donc pour développer des géométries paramétriques complexes, mais aussi des systèmes de relaxation dynamiques, des systèmes multi-agents, des L-systèmes et des algorithmes génétiques. GENR8, par exemple, dont nous avons parlé plus haut, est conçu dans Maya avec MEL. MEL reste néanmoins assez limité puisqu'il est initialement conçu comme un journal des commandes. Il ne dispose donc pas des caractéristiques des langages de programmation orientée objet[59]. MEL traduit simplement des instructions depuis le script vers l'API de Maya. Ce genre de langage est parfois appelé *méta-programmation* : c'est un langage qui transmet des instructions à un autre langage. En 2007, Autodesk fait le choix d'ajouter une module de programmation Python dans la version 8.5 de Maya, ce qui étend considérablement les possibilités de programmation au sein du logiciel. L'entreprise soigne aussi particulièrement les outils de modélisation des fluides disponibles dans le logiciel. Ce sont ces derniers qui font la popularité du logiciel pour flammes, vents et autres animations, et ceux avec lesquels les architectes jouent pour faire émerger des formes inattendues. Si Maya a été utilisé un peu partout dans le champ computationnel au cours de la dizaine d'années qui ont suivi sa sortie, il est en particulier le logiciel de prédilection chez Zaha Hadid Architects. Maya et MEL y sont utilisés pour de nombreux projets, notamment tours et schémas directeurs urbains, mais aussi en cours avec les étudiants des architectes de l'agence[60]. Celle-ci fait ainsi partie du petit groupe d'agences du champ computationnel qui ont encore aujourd'hui recours à Maya et MEL pour programmer leurs algorithmes, malgré une volte technique au milieu des années 2000 qui voit presque l'intégralité du champ se tourner vers Rhinocéros et Python.

Rhinocéros est un modeleur 3D généraliste, là ou Maya est orienté vers l'animation et ou la plupart des autres modeleurs en usage dans le champ computationnel sont orientés vers un domaine en particulier et les habitudes de modélisation qui lui sont associées[61]. Son kernel permettant de modéliser maillages et courbes avec une grande efficacité[62], les très nombreux formats qu'il supporte et son prix plutôt bas en comparaison d'autres modeleurs en font un logiciel très utilisé. Comme

---

[59] "*As a language, MEL is descended from UNIX shell scripting. This means MEL is strongly based on executing commands to accomplish things (like executing commands in a UNIX shell), rather than manipulating data structures, calling functions, or using object oriented methods as in other languages*." Jon Macey, "A technical Introduction to Maya", présentation au National Centre for Computer Animation, Bournemouth University, 2020.

[60] Entretien A. Rossi.

[61] "*In many respects Rhino is one of the few CAD tools that isn't 'flavoured' for any one vertical market or workflow*". Greg Corke, "Rhino 3D", 25 Juillet 2008, https://develop3d.com/product-design/rhino-4/, consulté le 02 Septembre 2021.

[62] Voir Chapitre III.

Maya, Rhinocéros possède son propre langage de script, RhinoScript. Celui-ci est basé sur Visual Basic, langage et environnement de programmation développé par Microsoft. RhinoScript fait appel à des commandes de Rhinocéros, de la même manière que MEL permet à l'utilisateur d'écrire des scripts à partir des commandes de Maya. La première version commercialisée de Rhinocéros date de 1998. En 2007, la sortie de Rhinocéros 4, cinq ans après la dernière mise à jour du logiciel, change cependant la donne pour McNeel et son modeleur. Cette nouvelle version de Rhinocéros intègre en effet un très grand nombre d'améliorations dans la plupart des domaines de fonction du logiciel[63]. Les graphismes tirent parti de tout le potentiel des cartes graphiques les plus récentes, la modélisation des solides intègre de nouvelles opérations booléennes, de nouvelles commandes d'édition des objets géométriques sont ajoutées. Rhinocéros 4 intègre également un éditeur de macros et des commandes d'animations qui, sans être aussi spécialisées, concurrencent l'avantage de Maya sur la représentation des objets modélisés. Deux nouveautés non négligeables sont aussi ajoutées. D'une part la "*technologie de déformation universelle*", ou Universal Deformation Technology, qui permet de déformer les objets en les manipulant à la souris de manière très intuitive. D'autre part, la beta de Grasshopper est également disponible, avant la publication de la version 1 finale en 2008. Avec cet ensemble d'ajouts et d'améliorations combiné à ses NURBS, Rhinocéros passe d'un modeleur 3D standard à un modeleur 3D extrêmement performant, tout en étant assez facile à prendre en main. Entre 2010 et 2011, McNeel intègre à Rhinocéros RhinoPython, qui permet d'écrire des scripts pour Rhinocéros en Python dans l'environnement du logiciel. Python offre plusieurs avantages par rapport à MEL et RhinoScript. C'est un langage de programmation orienté objet de syntaxe et de structure assez simple comparé aux autres langages de même puissance. Il permet de programmer tout ce que Maya avait comme fonctions intégrées de modélisation de systèmes de particules, mais aussi des simulations de croissance, des simulation physiques ou des algorithmes génétiques. Python est par ailleurs un langage très populaire en dehors de la communauté de Rhinocéros, là ou MEL est cantonné à Maya et à l'industrie de l'animation essentiellement. Il y a donc beaucoup de ressources mises à disposition par divers utilisateurs : tutoriels, scripts, bibliothèques pour de nombreuses applications sont faciles à trouver.

Les projets documentés et les entretiens menés indiquent tous un changement généralisé entre 2006 et 2007. Les algorithmes étaient en majeure partie écrits en MEL et visualisés dans Maya jusqu'alors, et deviennent pour la plupart des programmes Python visualisés dans Rhinocéros. Python fait beaucoup pour le passage à Rhinocéros comme modeleur principal du champ. Le langage est pointé par tous les

[63] "*So with Rhino 4, McNeel's development team have left no stone unturned, with enhancements covering modelling, editing, user-interface, display technology, rendering animation, 2D, mesh and analysis and large model capability.*" Greg Corke, "Rhino 3D", 25 Juillet 2008, https://develop3d.com/product-design/rhino-4/, consulté le 02 Septembre 2021.

praticiens interrogés comme la raison du changement d'un setup à l'autre. Ils sont unanimes sur les capacités bien plus grandes de Python comme langage de programmation, auquel ils sont encore fidèles quinze ans plus tard[64]. Quelques utilisateurs de Maya perdurent, mais même ZHA qui utilise toujours Maya a recours à Rhinocéros, Grasshopper et Python pour certaines parties de ses flux de travail. Mais Python arrive pourtant plus tard dans Rhinocéros que dans Maya, auquel Autodesk a intégré le langage en 2007, soit au moins trois ans avant que McNeel ne publie RhinoPython. Comment expliquer cette volte-face massive vers Rhinocéros et Python quand l'explication donnée par les praticiens ne concorde pas chronologiquement ? Il est possible que McNeel ait mis à disposition des praticiens du champ computationnel le module RhinoPython plus tôt que 2010, en leur donnant accès à des versions beta à essayer. Python aurait ainsi pu être disponible sans doute pas plus tôt que dans Maya, mais au moins en même temps. Autre possibilité, les praticiens du champ computationnel ont pu avoir recours à Python dans un autre environnement et intégrer Rhinocéros comme une partie du flux de travail tel qu'organisé depuis cet autre environnement de programmation. Les projets documentés montrent en effet bien Rhinocéros et Python présents ensemble dans les flux de travail dès 2006. Il est donc envisageable, parce que Python a beaucoup d'avantages comme langage, et Rhinocéros beaucoup d'avantages comme modeleur, que les praticiens se soient tournés indépendamment vers l'un et l'autre comme solution favorite. McNeel aurait développé RhinoPython dans un second temps, en réponse à cette double affection observée chez ses utilisateurs, contribuant à asseoir le bloc Rhinocéros/Python tel que décrit par les praticiens et observé dans les programmes[65]. La proximité de McNeel avec les praticiens du champ computationnel et leur écoute de leurs utilisateurs pour le développement de leur logiciel au fil du temps rend crédible cette hypothèse. Le développement de Rhinocéros dans les années 2000 s'accompagne en effet de partenariats avec Skylar Tibbits ou avec Hanno Stehling et de forums d'échange pour et avec les utilisateurs autour des pratiques computationnelles en architecture[66]. Le fait que McNeel soigne particulièrement ses relations avec ses utilisateurs, alors que Maya change plusieurs fois de mains à cette époque et a donc un avenir plus incertain[67], mais aussi la politique générale des deux entreprises [68] sont d'autres raisons possibles du changement. Enfin, dernier élément ayant peut-être joué en faveur de Python, le changement se fait à une période ou la maîtrise de la programmation des praticiens est plus forte. Les quelques années de pratiques qui ont précédé leur permettent ainsi d'évoluer vers des

[64] Entretiens Fabio Gramazio et Jose Sanchez.
[65] Ces deux étapes de développement auraient ensuite été reconstruites comme une seule dans la mémoire des praticiens.
[66] Forum RhinoScript 2008 puis forum McNeel - voir chapitre VI.
[67] Voir chapitre III.
[68] McNeel aurait une image plus ouverte, moins agressive, mieux alignée avec la posture d'une partie des praticiens. Voir chapitre VI.

algorithmes sur-mesure plus complexes et donc vers des langages de programmation plus élaborés que MEL, comme Python.

La question de la maîtrise de programmation poussent aussi les praticiens Processing. Il s'agit d'un environnement de programmation développé en 2001 par Ben Fry et Casey Reas. Le premier est un spécialiste de la visualisation des données alors en doctorat au sein du Aesthetics and Computation Group du MIT. Le second est un artiste multimédia qui termine son master au sein du même groupe. Les deux sont encadrés par John Maeda, graphiste directeur du groupe, qui considère la programmation comme un moyen d'expression artistique. Processing est le prolongement du projet *Design by Numbers* de Maeda, qui propose une méthode d'apprentissage de la programmation pour les non-informaticiens, notamment les designers. John Maeda lui-même a été formé par Muriel Cooper, fondatrice en 1975 du Visible Language Workshop au MIT et pionnière du design graphique informatisé. Initiée à l'informatique par Negroponte, invitée par ce dernier à intégrer son groupe au Medialab, elle y devient notamment collègue de Marvin Minsky et Seymour Papert, qu'elle connaissant déjà bien pour avoir travaillé sur la publication de leurs livres à l'époque où elle était la graphiste de MIT Press. Au Medialab, elle propose avec Negroponte *Books without Pages*, d'abord à la NSF qui refuse, puis à la DARPA, qui accepte de financer cette étude de la visualisation des données, en lien avec le projet de Media Room porté par Negroponte[69]. Elle poursuit également ses travaux sur la lecture et le design graphique à l'écran, des explorations regroupées sous le titre *Information Landscapes* et considérées comme très prometteuses au moment de sa mort en 1994[70]. L'environnement de programmation Processing est basé sur Java, et pensé au départ comme une extension de Java spécifique aux artistes et designers. Au fil des années, un grand nombre de développements divers ont été ajoutés à cette extension spécifique, en faisant un langage à part entière. Processing est pensé comme un outil d'exploration du code, qui met l'accent sur le processus de programmation, d'où son nom. Les fichiers sont par ailleurs appelés dans la communauté des *sketchs*, ou croquis, mettant l'accent sur le fait que ses développeurs considèrent Processing comme un sketchbook, un carnet de croquis[71]. Processing s'ancre à la fois dans l'histoire de l'intelligence artificielle des années 1960, au M.I.T., et dans le développement de l'infographie et des technologies de l'animation, donc doublement dans le réseau des origines du champ computationnel. Ses origines font aussi écho à la question de la programmation comme un artisanat, voire comme un art, et à la question d'un savoir-faire propre à ces pratiques et son apprentissage. Muriel Cooper soulignait en

[69] Pour le media room voir chapitre II.
[70] Nolwenn Maudet, "Muriel Cooper ,Informations Landscapes", *Back Office #1 - Faire avec*, Éditions B42, 2017.
[71] Casey Reas et Ben Fry, "A Modern Prometheus, The History of Processing by Casey Reas and Ben Fry", 29 Mai 2018, https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85, consulté le 02 Septembre 2021.

parlant du modèle mis en place au sein du Visible Language Workshop son hybridité entre art et science[72], et le travail de son équipe comme celui de celle de John Maeda reflète cette mise à égalité des deux praticiens - programmeur et artiste -, et des deux savoir-faire.

Connaître la programmation pour mieux comprendre le fonctionnement des objets techniques du monde contemporain, et pour permettre aux artistes de produire eux-mêmes leur travail est le coeur des ambitions de la fondation Processing, fondée en 2012 pour encadrer les activités autour de Processing[73]. L'environnement de programmation est un logiciel libre, et tout le projet est dirigé par des principes open-source. Cette décision est à la fois éthique et en lien avec la question de l'apprentissage de la programmation, comme le formulent Ben Fry et Casey Reas : "*Nous avons appris à coder en grande partie en regardant le code des autres, et nous voulions que cette ouverture soit centrale au projet*"[74]. En complément de Processing, de nombreuses ressources et exemples sont donc disponibles en ligne. Le forum principal de Processing a pris différentes formes au fil des années, mais la fondation maintient toutes les versions accessibles, même celles où plus personne ne contribue, pour que les utilisateurs ne perdent pas l'accès aux conseils et sketchs qui y figurent. Divers utilisateurs ont aussi contribué à créer d'autres ressources indépendamment de Ben Fry, de Casey Reas et de la fondation, comme OpenProcessing, un forum de partage des sketchs, ou Processing Cities, qui mets en relation des utilisateurs de Processing dans une même ville. La structure de processing rend aisée la mise à jour d'objets au fil d'itérations - par exemple le déplacement d'un cercle sur l'écran, d'où l'aisance pour animer des objets dans cet environnement. Ceci rend aussi Processing efficace pour créer des systèmes multi-agents, et l'environnement permet donc d'accomplir de nombreuses choses que Maya permettait aussi, le reléguant un peu plus encore en arrière-plan au sein du champ computationnel. Puisque le langage se base sur Java, il est aussi très facile de transformer les sketchs Processing en sites internets ou en applications, avec des interfaces pour modifier rapidement des variables et visualiser le résultat sans repasser par le code. Les outils comme Flower Power ou l'interface du pavillon Turing s'appuient notamment sur cette capacité. Enfin, autre grand domaine d'utilisation, Arduino est un dérivé de Processing qui permet d'utiliser des microprocesseurs et microcontrôleurs en combinaison avec le langage pour fabriquer des objets électroniques divers, souvent qui interagissent avec leur environnement à partir de capteurs. Comme

---

[72] Janet Abrams, "Muriel Cooper's Visible Wisdom", *ID Magazine* Septembre-Octobre 1994.

[73] Fry et Reas parlent de "*software literacy*", alphabétisation logicielle. Casey Reas et Ben Fry, "A Modern Prometheus, The History of Processing by Casey Reas and Ben Fry", 29 Mai 2018, https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85, consulté le 02 Septembre 2021.

[74] "*We learned coding in large part by looking at others' code, and wanted this openness to be central to the project*". Casey Reas et Ben Fry, "A Modern Prometheus, The History of Processing by Casey Reas and Ben Fry", 29 Mai 2018, https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85, consulté le 02 Septembre 2021.

Processing, l'objectif d'Arduino est de créer des ressources simples et accessibles financièrement à tout le monde, mais cette fois dans le domaine de l'électronique. Les machines qu'Arduino permet de développer sont elles aussi appréciées dans le champ computationnel, pour la fabrication de prototypes et d'outils divers à associer par exemple à un robot six axes, comme des pinces de préhension par exemple.

Si ses origines sont communes à celles du champ, Processing reste néanmoins un environnement un peu à part dans son utilisation, avec seulement une poignée de praticiens habitués de son usage pour la programmation de leurs algorithmes. Sa communauté au sein du champ coïncide essentiellement avec le réseau des systèmes multi-agents, en raison de la facilité de programmation d'animations dans Processing en comparaison avec Rhinocéros, et avec le pôle australien. Les praticiens de celui-ci spécialisés dans le recours aux systèmes multi-agents se sont en effet saisis de Processing à la fois pour leur pratique personnelle, mais aussi pour son modèle d'apprentissage de la programmation, sur lequel ils s'appuient pour leurs cours. La communauté des praticiens qui ont recours à Processing dans le champ computationnel est cependant essentiellement composé de quelques praticiens particulièrement versés dans la maîtrise de la programmation. Ceci peut paraître contradictoire au fait que Processing est pensé pour être d'une grande facilité d'utilisation et permettre aux novices d'apprendre la programmation. Plusieurs points permettent cependant d'expliquer ceci. En premier lieu, les structures algorithmiques que Processing est pensé pour mettre en œuvre, comme les systèmes multi-agents, sont parmi les typologies les plus complexes en usage dans le champ. Les systèmes multi-agents impliquent par exemple de gérer non seulement l'animation des particules, mais aussi la définition du contexte géométrique dans lequel elles évoluent, puis l'exploitation du comportement du nuage de particules pour en tirer une proposition architecturale. Comparés aux L-systèmes qui demandent simplement d'ajuster quelques règles syntaxiques puis peuvent être directement transposés comme des poteaux sans changements de la géométrie, cela demande une plus grande maîtrise et un plus long travail de programmation. Ensuite, l'usage de Processing dans le champ computationnel se fait souvent dans le cadre de partenariat entre des praticiens avec une grande maîtrise de la programmation et des praticiens moins versés ou des étudiants novices. L'efficacité pour créer des interfaces rapidement de Processing joue donc, par la facilité de manipulation et de visualisation de l'effet de modification des paramètres qu'elle autorise. Enfin, l'apprentissage dans le champ computationnel passe beaucoup plus souvent par Grasshopper qui devient au fil des années qui suivent sa sortie le plus utilisé des environnements, comme nous allons le voir ensuite. Ceci s'explique par le fait que Grasshopper est une extension de Rhinocéros et se connecte donc au couple Rhinocéros/Python, lui-même très apprécié. Ceci s'explique aussi par le fait que nombre d'enseignants du champ computationnel ne passent que peu de temps à établir une

pédagogie de la programmation et renvoient simplement leurs étudiants automatiquement vers Grasshopper, contrairement aux utilisateurs de Processing qui en ont saisi les ambitions et qui le mobilisent pour transmettre leur savoir-faire autour de la programmation plus sciemment.

Si l'association Rhinocéros/Python domine à partir de la fin des années 2000, Processing a son cercle d'utilisateurs, et dans une moindre mesure l'association Maya/MEL aussi. En complément de ces couples modeleur-langage principaux, quelques autres solutions sont aussi employées. Form*Z n'a pas totalement disparu, et Digital Project maintient également un cercle d'utilisateurs, et tous les modeleurs en usage peuvent être associés avec des algorithmes, généralement en Python ou en C++. Autodesk offre également des possibilités de programmation dans AutoCAD avec les langages AutoLISP et DesignScript[75]. Mais ces alternatives sont moins utilisées que les premières. Comme l'arrivée successive de modules de programmation Python dans Maya, Rhinocéros et Processing[76] l'indique, la grande majorité des praticiens programme désormais en Python. Au fil des années 2010, le langage connaît d'ailleurs une nouvelle vague de popularité non seulement dans le champ computationnel mais aussi au-delà. Ceci va de pair, nous l'avons dit, avec un apprentissage et une maîtrise poussée de la programmation. Le recours aux algorithmes sur-mesure s'ancre aussi fortement dans l'idée de fabriquer ses propres outils[77]. Leur statut les place à la lisière entre résultat d'une activité de programmation et outil à utiliser pour obtenir un résultat. La maîtrise de la programmation qu'ils impliquent permet aussi aux praticiens d'avoir les compétences pour ensuite les transformer en outils. Enfin, nombre des praticiens qui recourent à ces environnements de programmation partagent l'avis de Ben Fry et Casey Reas : "*au lieu d'apprendre (...) à utiliser des logiciels, il est tout aussi important d'apprendre à créer des logiciels*"[78]. Les algorithmes sur-mesure impliquent parfois de recourir à des setups avec de nombreux éléments différents en fonction des besoins, comme nous l'avons vu avec le concours du pavillon Seroussi. Ils sont un moyen de créer à partir des typologies, mais aussi d'automatiser un flux de travail en permettant de lier les différents logiciels. Ceux-ci sont choisis en fonction de ce qu'ils permettent le mieux de faire : le kernel de Rhinocéros, l'animation pour Maya, les interfaces pour Processing ou les dessins d'exécution pour AutoCAD. Les praticiens jonglent donc d'un logiciel à l'autre au fil des étapes, même quand ces derniers sont supposés offrir les mêmes fonctionnalités, et même si le choix se fait aussi en fonction de ce que les praticiens ont le plus l'habitude de manipuler. Les environnements de programmation les plus usités dans le champ le

[75] Voir chapitre VI.
[76] Processing se dote à son tour d'un module de programmation en Python en 2010.
[77] Voir chapitre III.
[78] "*Instead of teaching students how to use software, we thought it was just as important to teach them how to create software*". Casey Reas et Ben Fry, "A Modern Prometheus, The History of Processing by Casey Reas and Ben Fry", 29 Mai 2018, https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85, consulté le 02 Septembre 2021.

sont aussi parce que les langages qu'ils permettent d'utiliser, ou dont ils sont dérivés sont des "langages-colles"[79], des langages qui permettent de programmer des passerelles entre deux logiciels ou deux langages. C'est le cas de Python, mais aussi de Java ou de VB. Cette caractéristique est extrêmement utile au vu des pratiques d'interopérabilité qui caractérisent le champ dans les années 2000, un rôle essentiel pour les algorithmes sur-mesure alors produits par les praticiens. Ceci ajoute néanmoins encore un ensemble de connaissances à maîtriser pour pouvoir utiliser des algorithmes sur-mesure. Il faut tout gérer, de l'interface à l'affichage du résultat en passant par la transformation des données elle-même. Saisir l'ensemble des instructions de programmation nécessite de connaître la syntaxe du langage, les possibilités des différents environnements, les options offertes par d'autres logiciels, les possibilités de connexions entre eux. Les algorithmes sur-mesure offrent donc une très grande liberté dans la programmation des projets, mais au prix d'un travail de construction du programme conséquent.

### *5.2.2 Bibliothèques : un raccourci vers des fonctions sur-mesure*

Lorsque les algorithmes sur mesure s'allongent, le travail de programmation peut être très long. Pour gagner du temps ont été développées ce qu'on appelle des bibliothèques. Celles-ci sont des collections de blocs de code prédéfinis, développés par des programmeurs comme des raccourcis vers des fonctions déjà écrites. Plutôt que de réécrire systématiquement à partir de rien, les bibliothèques permettent de s'appuyer sur du travail effectué en amont lorsque l'algorithme en cours d'écriture recours à des fonctions déjà développées avant. Les bibliothèques sont donc des répertoires de fonctions auxquelles faire appel depuis l'environnement de programmation dans lequel on code. Elles sont donc associées à des langages - Python, C++, C#, MEL ou Processing - dans lesquels les fonctions sont écrites puis résumées en un mot, qui lorsque écrit dans le code fait appel à cette fonction directement, sans avoir besoin de la réécrire en entier. En plus de fournir une économie de temps non négligeable lorsqu'il s'agit de coder des algorithmes longs, les bibliothèques constituent donc une extension des possibilités de codes dans un langage donné, et constituent donc un important soutien aux pratiques de programmation. En effet, un programmeur peut faire appel à des bibliothèques codées par lui-même, mais aussi à celles d'un autre programmeur. Les bibliothèques de code peuvent être comparées à des bibliothèques physiques : des collections de ressources collective à utiliser. Il existe des bibliothèques publiques, bibliothèques personnalisées et personnelles, ou bibliothèques intermédiaires partagées seulement par quelques praticiens. Les structures ou sont partagées les bibliothèques diffèrent, et donnent donc accès à ces collections de fonctions à des utilisateurs différents.

[79] "*glue language*".

Nombre de bibliothèques sont en usage dans le champ computationnel. Parmi celles-ci se trouvent des bibliothèques de programmation généralistes, et des bibliothèques spécifiques au champ. Parmi les généralistes se trouvent une grande diversité d'outils. Processing offre par exemple des bibliothèques pour gérer divers formats non supportés par défaut, comme les fichiers pdf ou les images svg[80]. D'autres bibliothèques permettent de paramétrer un affichage sur des écrans multiples plutôt que sur l'unique console d'affichage dont disposent usuellement les utilisateurs[81]. Toujours dans l'idée de personnaliser l'expérience utilisateur, certaines bibliothèques, comme interfascia[82], permettent d'ajouter à l'affichage observé lorsque le script est en cours d'exécution des éléments d'interfaces supplémentaires, comme des boutons, des curseurs ou des commentaires à destination de l'utilisateur. Processing compte également un grand nombre de bibliothèques augmentant ses capacités de description géométriques, comme geomerative, orientée vers la conception typographique; ou igeo ou lunar, des bibliothèques de traitement des maillages[83]. Python s'adresse à une communauté d'utilisateurs légèrement différente, et propose donc d'autres types de bibliothèques. NumPy est un ensemble de fonctions informatiques diverses utiles à la pratique scientifique, grâce auxquelles les utilisateurs peuvent intégrer à leur programme de l'analyse ,mathématique ou statistique, du traitement des signaux ou encore du traitement d'images[84]. La popularité de Python dans le domaine scientifique est grande, et de nombreuses autres bibliothèques permettent d'autres formes de traitement des données, comme matplotlib[85]. Autres exemples au sein de Python, PyBrain permet de programmer des réseaux neuronaux dans Python, et Dash est un journal d'expériences[86]. Processing et Python offrent également des bibliothèques spécialisées pour le design computationnel. L'une des bibliothèques les plus prisées de Python est RhinoScript, qui donne accès aux fonctions du modeleur Rhinocéros à partir de l'interface de programmation de Python, et crée ainsi une passerelle pour les utilisateurs les plus friands du couple Rhinocéros / Python. Toxiclibs donne accès à des courbes B-splines pour étendre les capacités de modélisation dans Processing[87]. Processing Combinatorics mets à disposition des règles de combinaison de modules entre elles, dans le prolongement de l'axe de recherche que nous avions vu au Chapitre II, dont l'objectif est de décomposer en ensemble de modules géométriques régis par des règles d'association[88]. Culebra met à disposition des utilisateurs

[80] https://processing.org/reference/libraries/svg/index.html, consulté le 02 Septembre 2021.
[81] https://github.com/shiffman/Most-Pixels-Ever-Processing, consulté le 02 Septembre 2021.
[82] https://github.com/brendanberg/interfascia, consulté le 02 Septembre 2021.
[83] http://www.ricardmarxer.com/geomerative/, http://igeo.jp/, https://github.com/boydhont/Lunar-Processing, consultés le 02 Septembre 2021.
[84] https://numpy.org/, consulté le 02 Septembre 2021.
[85] https://matplotlib.org/, consulté le 02 Septembre 2021.
[86] http://pybrain.org/, https://dash.plotly.com/, consulté le 02 Septembre 2021.
[87] http://toxiclibs.org/, consulté le 02 Septembre 2021.
[88] https://github.com/fjenett/combinatorics, consulté le 02 Septembre 2021.

une bibliothèque de comportements pour les particules de systèmes multi-agents[89]. Box2D propose les fonctions nécessaires à la programmation d'une simulation physique, par exemple une relaxation dynamique[90]. Les bibliothèques permettent donc notamment d'avoir recours à des typologies algorithmiques, mais aussi à des ensembles de fonctions développées par les praticiens au fil des années, qui y sont stockées pour les retrouver facilement et les réemployer par la suite.

L'agence kokkugia offre un exemple particulièrement significatif de la constitution de ces bibliothèques personnelles, car celles-ci sont au cœur de la pratique commune de Roland Snooks et Robert Stuart-Smith. kokkugia est spécialisée dans la conception architecturale à l'aide de systèmes multi-agents, ses bibliothèques sont donc des collections de fonctions énonçant des règles de comportement pour leurs agents. Chaque élément de la bibliothèque est un programme définissant le comportement d'un nuage de particules, que les praticiens peuvent réintégrer à un programme pour un projet donné. Ceci implique de garder en tête "l'effet" de modélisation que ce comportement crée pour savoir quand employer lequel. Ces "effets", du mot choisi par les praticiens au cours des entretiens pour les décrire, désignent en premier lieu les ornements qui parcourent l'enveloppe des bâtiments imaginés par l'agence[91]. Ce choix d'ornement est décrit par Roland Snooks comme une intention architecturale majeure dans la pratique de l'agence, certains de ces effets étant adéquats pour certains projets mais non d'autres. Les deux praticiens citent notamment Michael Hansmeyer et son projet Digital Grotesque comme référence, alors même que ce dernier s'inscrit dans des pratiques computationnelles très différentes, et n'a justement de commun avec kokkugia que cette attention à l'ornementation[92]. Ces effets localisés, de petite échelle, et leur relation avec la structure plus monolithique des bâtiments sont typiques de l'approche de kokkugia, qui met toujours en balance cet aspect "fibreux" et la masse des projets proposés (figure 24). Roland Snooks et Robert Stuart-Smith tissent avec ces notions une continuité entre algorithmes et approches architecturales et entre différents projets via leurs bibliothèques . Le projet du Babiy Yar Memorial exploite particulièrement ce jeu esthétique. Le point de départ du travail est un "monument inversé"; au sens d'une inversion matérielle des statues classiques : plutôt qu'un socle supportant une œuvre en bronze, la structure se retourne sur elle-même[93]. Le bâtiment est donc composé d'une enveloppe sobre blanche éventrée par une partie ornementale en bronze beaucoup plus complexe, générée par une des bibliothèques d'agents de kokkugia. Le terminal de l'aéroport de Riga tire lui partie de ces ornements pour instaurer

[89] http://culebra.technology/culebra-1/, consulté le 02 Septembre 2021.
[90] https://github.com/shiffman/Box2D-for-Processing, consulté le 02 Septembre 2021.
[91] Entretien Roland Snooks.
[92] Roland Snooks, "Behavioral Formation: Multi-Agent Algorithmic Design Strategies". Thèse de Doctorat, RMIT University, 2014.
[93] Roland Snooks, "Behavioral Formation: Multi-Agent Algorithmic Design Strategies". Thèse de Doctorat, RMIT University, 2014.

une liaison entre les différents espaces. Le projet se concentre notamment sur les espaces d'attente des voyageurs souvent négligés au profit de l'axe central du terminal, par un plafond ouvragé, ornementé par les traces du parcours d'un nuage d'agents. Les deux projets montrent un recours à un dispositif algorithmique engrainé dans des intentions architecturales nettes, qui mènent au choix d'un système d'ornements à un endroit précisément défini, et à la sélection d'une liste de comportements d'agents tout aussi précis.

*Figure 24 - a. Babi Yar Memorial, kokkugia, b. et c. Terminal Aéroport Riga, kokkugia ; d. National Art Museum of China, kokkugia*

L'explication donnée du développement de ces outils algorithmiques dans le doctorat de Roland Snooks est une recherche chronologique qui mène l'agence d'abord à la programmation d'une séries de comportements différents pour les agents, puis à l'élaboration d'une série de stratégies pour combiner ces comportements entre eux. L'agence se repose sur vingt-quatre comportements pour ses systèmes multi-agents; relevant de trois catégories différentes. Les premiers régissent l'orientation et le déplacement basique des particules dans l'espace, dans le même esprit que les comportements élaborés par Craig Reynolds pour ses boids. Les seconds sont conçus pour que les agents s'amalgament entre eux le long de lignes directrices et forment ce que Snooks et Stuart-Smith appellent des brins[94]. Les brins peuvent ensuite se combiner entre eux en des structures plus vastes, que l'agence nomme des corps. Enfin, dans certains projets, les praticiens de kokkugia privilégient la définition d'une géométrie générale pour le projet en amont du recours aux agents. Ceux-ci sont

[94] "*strands*".

ensuite utilisés le long d'une surface, comme dans le cas du terminal de l'aéroport de Riga. Pour parvenir à ceci, la dernière catégorie de comportements des agents de kokkugia contraint les particules à suivre une géométrie guide pendant leurs déplacements. Ces vingt-quatre comportements sont sélectionnés et combinés dans ce que l'agence nomme des stratégies, qui forment en fait les bibliothèques sur lesquelles la pratique quotidienne se repose. En 2014, Roland Snooks en identifie huit dans sa thèse de doctorat : *agency of structure, manifold swarms, agent bodies, behavioral composites, polyscalar tectonics, fibrous assemblages, strange feedback, swarm urbanism*[95]. L'une de ces stratégies porte sur la combinaison à plusieurs échelles des systèmes multi-agents. Un premier système peut par exemple permettre de donner forme à la géométrie d'ensemble du projet, puis un second faire le travail d'ornementation prisé de kokkugia. C'est ainsi que fonctionnent les projets de la Cliff House ou de National Art Museum of China. L'effet obtenu avec un groupe de comportements donnés identique n'est pas le même en fonction de l'échelle et de l'objet auquel il est appliqué, une caractéristique que l'agence exploite dans ses algorithmes. Les différents nuages de particules sont de plus programmés pour réagir au comportement des nuages à l'œuvre à d'autres échelles, une interaction également jugée intéressante. Les comportements des agents relatifs aux brins et corps sont le résultat du dev de cette approche, qui permet de multiplier les systèmes d'agents au sein d'un même projet. Autres exemples de stratégies développées par l'agence, celles qui permettent de passer des effets de surface à la conception d'un système structurel. Les premières bibliothèques de kokkugia sont des ensembles de comportements qui ne permettent pas aux agents de générer des formes, mais seulement d'intervenir sur une forme préexistante. Les agents génèrent simplement des motifs sous forme de brins, contraints par des comportements qui les font coller à des surfaces prédéfinies, pour générer des ornements. Mais ces brins deviennent ensuite un moyen pour kokkugia de générer une structure principale sous forme de treillis, comme dans le cas du projet pour la Aalto University, pour lesquels un système d'évaluation structurelle est intégré à l'algorithme. A terme, les effets ne sont donc plus pour l'agence uniquement des motifs créés sur une surface grâce aux agents. Ils sont aussi utilisables pour concevoir la forme générale du bâtiment, que ce soit sa structure ou même la hiérarchisation des espaces de son programme, des nodules dont le système d'agents détermine le placement, comme pour le projet du Yeosu Pavilion. Cette stratégie est décrite par l'agence comme l'émulation des essaims de fourmis lorsqu'ils doivent former un pont, et que les animaux s'agrègent en se marchant les uns sur les autres jusqu'à y parvenir. Les différentes bibliothèques de l'agence sont utilisées régulièrement par Roland Snooks et Robert Stuart-Smith, et les effets qu'elles permettent d'obtenir peuvent donc être observés dans plusieurs projets. Ces effets sont même mobilisés par les deux praticiens dans leurs activités séparées, chez Robert Stuart-Smith dans les projets de son agence

[95] agentivité des structures, essaims multiples, composites comportementaux, agents-corps, tectoniques polyscalaires, assemblages fibreux, rétroactions étranges, urbanisme en essaim.

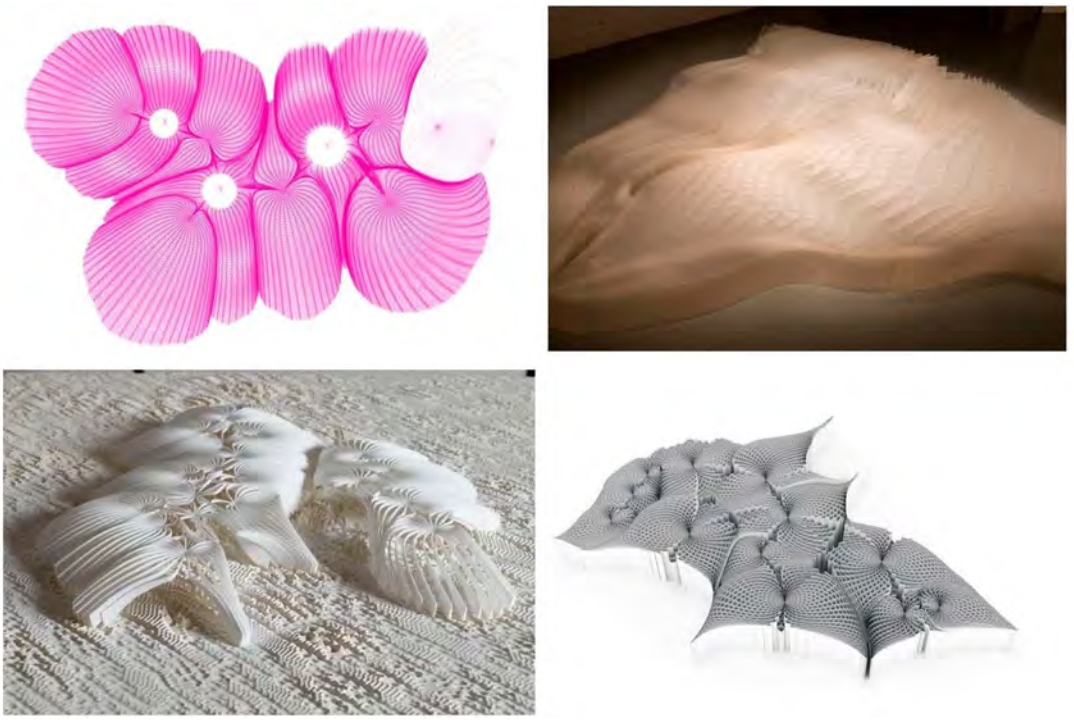
comme la bibliothèque d'Helsinki, chez Roland Snooks dans les projets Fold Pavilion ou Kazakhstan Symbol (figure 25).

*Figure 25. a. Salle d'escalade, Robert Stuart-Smith ; b. Fold Pavilion, Roland Snooks ; c.Kazakhstan Symbo, Roland Snooks ; d. Bibliothèque d'Helsinki. Robert Stuart-Smith*

Autre bibliothèque du même type, le projet Genware mené par Alisa Andrasek entre 2001 et 2006. Le nom du projet provient d'une métaphore d'ingénierie génétique. Les bibliothèques de genware sont ainsi à combiner comme les gènes permettant à une projet de s'exprimer dans une itération donnée[96]. Bien qu'il ne soit aujourd'hui pas disponible publiquement, Genware est donc une plateforme de partage de programmes de design computationnel qui a vocation à regrouper les algorithmes écrits et utilisés par biothing, l'agence d'Alisa Andrasek. Cette plateforme sert avant tout à la pratique interne de l'agence, pour une raison en particulier : bien que Alisa Andrasek en soit la fondatrice et la seule à avoir été membre de l'agence pendant toute l'existence de cette dernière depuis, ce sont les autres membres qui ont écrit les différents programmes. Ezio Blasetti, Tobias Schwinn, Kyle Steinfeld, Iain

[96] "*The genware project was conceived as a sharing collaborative platform. Incorporating both aspects of genetic engineering and software design, it allows a designer to work at the scale of information. Not unlike a genetic engineer, the designer writes and manipulates computer scripts and code sequences in the generation of abstract forms of digital intelligence. This intelligence is then channelled into any number of potential material sites and scales. Like a virus it circulates through a number of disciplinary contexts such as architecture, product design and fashion. In each case, abstract geometric transformations are linked to specific material and fabrication constraints as well as scales of production, allowing for a synthesis of design intuition, algorithmic programming and parametric limits as the very foundation of the design process."* Ednie-Brown, P. (2006), All-over, over-all: biothing and emergent composition. Archit Design, 76: 72-81.

Maxwell ou encore Jose Sanchez sont à l'origine des différentes bibliothèques qui composent Genware, programmées à l'occasion de stages au sein de l'agence, de workshops ou de cours en tant que salarié de l'agence ou dans le cadre de collaborations avec Alisa Andrasek, voire de simples coups de main sur un projet donné. Ceci a pour conséquence une grande attention à l'interface au sein des différents éléments de Genware, qui doivent pouvoir être facilement utilisés par d'autres que leurs programmeurs. Les différentes bibliothèques sont par ailleurs identifiées par fonctions plutôt que par comportements ou effet : *curtain*, *partition*, *shelves*, *highrise structure* ou *surface accessory* en sont les intitulés[97]. Les différentes typologies explorées par Genware - principalement des systèmes de croissance et des systèmes multi-agents - y sont instanciés dans ces objets architectoniques. Les projets Bifid, Orbita et Mesonic Fabrics font ainsi tous appel au même module déjà utilisé dans le projet du Pavillon Seroussi (figure 26). Plutôt que des variations au sein d'une même structure algorithmique, comme le fait kokkugia, Genware rassemble différentes structures typologiques que biothing combine ensuite dans diverses combinaisons. Il ne s'agit donc pas d'une bibliothèque qui se construit autour du développement de stratégies autour d'une famille d'agents, mais simple bibliothèque de techniques algorithmiques diverses explorées au fur et à mesure des projets, une différence de profondeur qui s'explique sans doute elle aussi par le fort roulement des développeurs de ces algorithmes.



[97] mur-rideau, cloison, étagère, structure de gratte-ciel, accessoire de surface. Alisa Andrasek, biothing, HYX, 2009.

*Figure 26. Projets de l'agence biothing utilisant la bibliothèque Genware : Bifid, Orbita, Mesonic fabrics et le pavillon Seroussi*

Si les bibliothèques de kokkugia et biothing sont classifiées dans un groupe d'éléments de programmation formalisé, certaines agences accumulent aussi les bibliothèques , mais sans forcément en faire un ensemble hiérarchisé et trié. C'est parfois parce que ces bibliothèques sont simplement des aides fonctionnelles pour certains points du flux du travail, ou pour des projets très différents, comme chez designtoproduction. Le bureau d'études, fondé en 2007 par Fabian Scheurer et Arnold Waltz avec Hanno Stehling après avoir évolué dans diverses institutions et agences du champ, développe rapidement une expertise pour la construction de formes complexes. Leur travail porte essentiellement sur la rationalisation de la géométrie des projets et sur la production d'instructions pour des machines à commande numérique. Les bibliothèques du BET vont d'outils géométriques basique, comme la détection de conflits géométriques entre deux éléments dans Rhinocéros, une fonctionnalité qui n'existe pas dans le modeleur par défaut et qui entraîne souvent des erreurs dans les modélisations fournies à l'agence par les architectes. designtoproduction s'est très vite spécialisé dans la production d'architectures en bois, et possède donc parmi ses bibliothèques un ensemble d'algorithmes destinés à la gestion des processus de fabrication associés.  L'une d'entre elle vise à pouvoir lire le format BTL, utilisé par toutes les machines-outils du secteur de la menuiserie, dans tous les environnements de programmation en usage dans le champ. Une incursion dans le domaine du béton dans le cadre du projet de recherche *tailorcrete*, mené avec l'essentiel du pôle danois, est aussi l'occasion pour designtoproduction de concevoir une bibliothèque d'outils permettant la création de coffrages bois sur-mesure à partir de la géométrie de la forme en béton visée. Enfin designtoproduction a également participé à un projet de recherche portant sur la visualisation de données de fabrication pour des couvertures en bois composées d'un très grand nombre d'éléments, et sur le partage d'informations au sujet de ces éléments entre les différents corps de métiers impliqués dans la production : ingénieurs, architectes, pré-fabricants et ouvriers sur le chantier. Ceci a également donné lieu à l'écriture d'un ensemble de bibliothèques , destinées à être utilisées par la suite par l'agence au fil de ses interactions avec les différents acteurs de ses projets. Les bibliothèques de designtoproduction sont donc très fonctionnelles et ancrées dans des enjeux pratiques de production.

Ces bibliothèques personnelles font l'essentiel du recours à ce format, mais certains groupes de praticiens choisissent par ailleurs de les rendre accessibles à d'autres, notamment en classe. supermanoeuvre dispose par exemple d'un ensemble de bibliothèques comme kokkugia et biothing. Les trois agences capitalisent sur ces formats pour les partager dans le cadre d'ateliers et de classes avec leurs étudiants, leur fournissant des raccourcis pour l'écriture de leurs propres algorithmes.

Au-delà du partage au sein d'un groupe fermé d'étudiants, Jose Sanchez, ancien collègue et élève de Andrasek à la AA et à biothing a créé le Plethora Project en 2011 justement pour dépasser ce cadre fermé et rendre ces outils accessibles à plus de praticiens. Jose Sanchez s'inscrit dans une lignée et un groupe de praticiens qui favorise la plateforme Processing. Plethora Project propose justement des tutoriels pour apprendre à programmer dans cet environnement. Ce groupe de praticiens est aussi celui où le recours aux systèmes multi-agents est plus développé. En complément des tutoriels, Plethora Project propose donc une bibliothèque de comportements d'agents. Contrairement aux bibliothèques de kokkugia, biothing ou supermanoeuvre, qui circulent en interne, celle de Plethora Project est publique. Autre bibliothèque publique, la bibliothèque COMPAS développée par le Block Research Group à l'ETH Zurich. Bibliothèque conçue pour Python, elle rassemble les différents outils de travail développés par le groupe au fil des années. COMPAS propose des passerelles entre Python, certains logiciels de CAO et de modélisation 3D, ainsi que quelques autres outils utiles à la conception de structures, comme le logiciel de simulation aux éléments finis Abaqus. Les différents ensembles de fonctions que COMPAS regroupe permettent de concevoir des voûtes, de faire de la statique graphique et d'aider à la fabrication, puisque la bibliothèque inclut également un slicer pour l'impression 3D et des fonctions de contrôle robotique. La diversité des bibliothèques disponibles le montre : les bibliothèques sont un format qui occupe une place importante dans les pratiques du champ computationnel. L'intermédiaire qu'elles constituent entre algorithmes sur-mesure mais longs à coder et automatismes trop développés en fait des outils puissants. Le recours à ces bibliothèques est caractéristique de la génération dont nous avons étudié le travail au chapitre précédent. Ce format apparaît donc comme un des plus efficaces pour implémenter la traductibilité sur laquelle repose les pratiques computationnelles et permettre la concentration sur les enjeux architecturaux du projet : des caractéristiques qui sont justement d'autres traits de cette génération.

### *5.2.3 Plug-ins : rendre les fonctions sur-mesure accessibles*

De la même manière que les bibliothèques sont des extensions des langages de programmation les dotant de plus de fonctions, les plug-ins sont des ajouts de fonctionnalités à un logiciel. Par exemple, les plug-ins evolute ajoutent au logiciel Rhinocéros des fonctionnalités de manipulations de maillage, par exemple de subdivision des surfaces. Une fois le plugin installé. Le recours à ces fonctionnalités se fait exactement comme n'importe quelle autre fonctionnalité du logiciel. La logique de travail et de manipulation des modèles 3D est la même. L'installation du plug-in crée des boutons supplémentaires sur l'interface du logiciel, qui permettent de faire appel à ces fonctionnalités additionnelles. Un plugin peut donc être compris comme une bibliothèque dans un logiciel plutôt que dans un environnement de programmation, c'est-à-dire une bibliothèque avec une interface logicielle. De la même manière que

les bibliothèques doivent être écrites dans le langage et donc dans la logique du langage de programmation qu'elles étendent, les plug-ins doivent adopter l'environnement que constitue le logiciel. Pour evolute par exemple, cela implique de s'appuyer sur le kernel de Rhinocéros et ses NURBS. Les plugins permettent notamment de rendre les fonctionnalités sur-mesure développées par des programmeurs accessibles non seulement à d'autres programmeurs, mais aussi à des utilisateurs de logiciels. Comme les bibliothèques, les plugins sont des ressources construites autour de la notion de bac à sable, ou de boîte à outils. Ce sont des catalogues d'extensions qui permettent de se construire un environnement logiciel sur-mesure, adapté à la pratique de chacun. Le recours aux plug-ins dans le champ computationnel s'est développé autour de plusieurs environnements logiciels clés, associés à des modeleurs et pensés comme des espaces d'accueil de collections de plug-ins. Ces environnements sont par ailleurs difficiles à dissocier du recours à la programmation visuelle, sur laquelle ils s'appuient largement.

La première des investigations dans ce domaine est pilotée par Bentley Systems. Il s'agit de Generative Components, un logiciel de modélisation développé par l'éditeur spécifiquement pour l'industrie de l'architecture. Generative Components associe, comme les autres plateformes du champ computationnel, un modeleur et un environnement de programmation. Ce dernier possède cependant deux versions : une interface d'écriture de programmes classiques et une interface de programmation visuelle. Cette dernière permet d'établir ce que les concepteurs de Generative Components nomment un modèle symbolique : une carte des relations entre les différents éléments du modèle. Cette triple expérience utilisateur est pensée pour permettre à ces derniers de choisir ce qui leur est le plus utile, mais offre aussi un système de niveaux d'apprentissage. Le modeleur de Generative Components repose sur une modélisation des courbes et des surfaces par des courbes de Bézier, comme Rhinocéros [98], et dispose donc des mêmes capacités de représentation de formes complexes. Generative Components est aussi proposé seul, sans ce modeleur, pour être combiné avec le modeleur Microstation. Bentley Systems souhaite populariser à travers Generative Components la modélisation "solide", ou paramétrique, déjà beaucoup utilisée dans l'aéronautique ou l'automobile, dans l'industrie de l'architecture. Cette méthode permet de coller au plus près à la réalité physique des objets, et est donc particulièrement populaire dans les industries de manufacture. Au moment ou Bentley Systems se lance dans le développement de Generative Components, les géométries de plus en plus complexes que promeut la popularisation des premiers modeleurs dans le champ de l'architecture rends cette précision de plus en plus nécessaire pour parvenir à fabriquer les composants du bâtiment pour sa construction. Generative Components s'ancre donc aussi dans la notion de

[98] Il n'utilise néanmoins pas de NURBS. Pour plus de détails sur les kernels voir chapitre III.

non-standard telle que développé les années précédentes au sein du champ computationnel[99]. Le logiciel se concentre sur la relation entre les différents composants géométriques du modèle, et la paramétrisation de ces relations[100], ce que montrent bien les travaux étudiants réalisés aux cours d'ateliers pendant les années de développement du logiciel. Ces travaux dévoilent la logique de placage d'éléments sur géométrie guide, avec une paramétrisation en fonction de divers critères, ce qui rend possibles des variations géométriques d'un élément à l'autre. Generative Components se concentre essentiellement sur ces géométries paramétriques, même si les versions les plus tardives du logiciel proposent des systèmes de croissance ou des simulations physiques. Robert Aish, le développeur principal de Generative Components, parle ainsi à propos du logiciel de "bibliothèque de primitives géométriques"[101].

Robert Aish est un designer industriel formé au Royal College of Art de Londres, où il a notamment étudié avec Bruce Archer[102], et à l'université d'Essex, dont il est titulaire d'un doctorat en Human Computer Interaction. Il découvre très jeune Sketchpad[103], dès la fin des années 1960. Il est aussi impressionné par les possibilités offertes par ce programme précurseur des logiciels CAD que séduit par les idées de Nicholas Negroponte, dont il découvre les machines à architecture en collaboration étroite avec un architecte humain peu après. C'est cette inspiration qui le pousse à entreprendre un doctorat dans le champ des interactions entre humain et machine, puis qui le mène à exercer comme développeur pour l'industrie de l'architecture. Il est notamment crédité de l'invention du terme building modelling et d'une contribution majeure au développement des systèmes BIM contemporains. A cheval entre le monde informatique et le monde du design depuis ses débuts, il exerce tour à tour au chantier naval de Gdansk, chez Bentley Systems, au sein des bureaux d'études YRM et Arup, et chez Autodesk. Il a pour mission à chacun de ces postes de développer des logiciels aidant à la conception les praticiens qui l'entourent. Chez Bentley Systems, ce logiciel, c'est Generative Components, où il a la possibilité de mettre en œuvre l' expérience gagnée au cours de ses précédents travaux. Ses ambitions pour Generative Components reposent sur deux points principaux. En premier lieu, construire des passerelles entre le savoir-faire des praticiens en architecture et celui des programmeurs[104]. Ceci passe par une aide à l'apprentissage des pratiques computationnelles de

[99] Voir Chapitre III.
[100] Robert Woodbury, Robert Aish et Axel Kilian, "Some Patterns for Parametric Modeling", dans *Expanding Bodies : Art, Cities, Environment. Proceedings of the 2007 ACADIA Conference*, 2007.
[101] "*library of geometric primitives*" Kristina Shea, Robert Aish et Marina Gourtovaia, "Towards integrated performance-driven generative design tools", *Automation in Construction*, Volume 14, Issue 2, p. 253-264, 2005.
[102] Voir Chapitre II.
[103] Voir Chapitre III.
[104] "*Some architects can program computers, some programmers are architects—but having the one skill shouldn't mean having to have the other, says CAD pioneer Robert Aish, Bentley Systems' director of research*". Deborah Snoonian Glenn, "Innovation: Technology Briefs. GenerativeComponents Software.

design pour les premiers[105], jusqu'à leur donner la capacité de faire ses propres outils[106]. En créant des espaces de rencontre pour ces savoir-faire, l'objectif de Robert Aish est de donner la possibilité aux designers de devenir "*computationnellement expressifs*"[107]. En d'autres termes, l'objectif premier de Generative Components pour Aish est de créer une plateforme logicielle qui permette de transmettre le savoir-faire propre au champ computationnel en architecture. Deuxièmement, Generative Components doit permettre de diffuser ce savoir-faire en agence. Les praticiens qui président à son développement évoluent en agence, et c'est dans ses pratiques que la plateforme s'ancre, bien plus que dans le champ computationnel académique.La filiation que décrivent Robert Aish et les fondateurs de Smart Geometry pour Generative Components intègre certes les *design patterns* de Christopher Alexander, mais leur récit se concentre surtout sur les systèmes CAD observés dans le cadre de leur pratique en agence, et les possibilités d'amélioration qu'ils distinguent alors déjà. Generative Components s'ancre à ce titre aussi dans la vision de Robert Aish pour le BIM, qu'il développe plus avant dans ses projets suivants.

L'une des particularités les plus importantes de Generative Components est la méthode de développement employée par Bentley Systems et Robert Aish. Celle-ci est peu habituelle pour l'éditeur de logicielle, parce qu'elle implique de nombreux échanges avec une communauté de praticiens et la prise en compte de leurs retours, voire de leurs contributions très tôt au cours du processus de programmation du logiciel[108]. Cette communauté, c'est celle de Smart Geometry, fondée en 2001 par Hugh Whitehead de chez Foster and Partners, Lars Hesselgren de chez KPF et Jay Parish de chez Arup[109]. Les trois fondateurs de Smart Geometry connaissent bien Robert Aish : ils ont

---

GenerativeComponents Software gives "bending the rules" a whole new meaning. 01 Octobre 2003. https://www.architecturalrecord.com/articles/12260-generativecomponents-software, consulté le 02 Septembre 2021.

[105] "*With this in mind we might summarise the mission of SG: to encourage the development of those cognitive and creative skills that matched the geometric and computational possibilities of a new generation of design software*." Robert Aish, "First Build Your Tools", dans Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.

[106] "*If infrastructure design is considered a craft, then GC follows the tradition of craftspeople crafting their own tools for their specific crafts to meet their very individual requirements, perhaps due to ergonomic considerations and more frequently because of individual advancement of their craft that cannot be reflected by the standard tools generally available for their craft. In computational design crafting one's own tools is then the creation of tools that generate the desired designs which otherwise would not be possible to generate with conventional computer aided design tools*". Robert Aish, "First Build Your Tools", dans Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.

[107] "*computationally expressive"*. Robert Aish, "First Build Your Tools", dans Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.

[108] Robert Aish et Robert Woodbury, "Multi-Level Interaction in Parametric Design", Conference: Smart Graphics, 5th International Symposium, SG 2005, Frauenwörth Cloister, Germany, August 22-24, 2005, Proceedings.

[109] Voir chapitre IV.

travaillé avec lui chez YRM dans les années 1980. YRM est alors un gros bureau d'études anglais, qui décide de se doter d'une section d'experts en modélisation numérique, pour aider à la conception des projets que l'agence gère. C'est à cette occasion que Whitehead, Hesselgren et Parrish, qui exploraient déjà le domaine du CAD dans diverses structures depuis la fin de leurs études quelques années plus tôt, sont embauchés. Ils rencontrent peu de temps après Robert Aish à une conférence et l'invitent à les rejoindre chez YRM pour développer un système de modélisation par association - c'est-à-dire un système qui fonctionne sur les mêmes principes de mise en relation des objets géométriques que le futur Generative Components[110]. Il s'agit alors pour les quatre praticiens d'une période et d'un espace de test des nouveaux systèmes CAD. Ceci intervient cependant dans un cadre professionnel dans lequel ils doivent tenter de convaincre que le CAD est intéressant pour la conception architecturale, une idée auquelle peu de monde accorde de crédit.En raison de ces relations professionnelles de longue date, Robert Aish est inclu dans les discussions de Smart Geometry dès le début, au point d'en être parfois cité comme un des fondateurs. Il est alors chez Bentley Systems, et les années 2001 à 2003 sont l'occasion de travailler sur *CustomObjects*, version de travail de Generative Components. Il propose alors à ses trois complices de modéliser avec ce nouveau système des projets sur lesquels ils ont travaillé précédemment, pour le tester - la Waterloo station et un concept de stade étudiés chez YRM quinze ans plus tôt, et le Pinnacle, un gratte-ciel londonien sur lequel KPF travaille. Quelques autres praticiens sont ensuite invités à faire le même exercice[111]. Ce travail de test sur des projets d'agence devient à partir de là une des bases du développement de Generative Components. En 2003, Smart Geometry organise sa première conférence, couplée à un atelier. Whitehead, Hesselgren et Parrish sollicitent alors Aish pour lui proposer de tester Generative Components, toujours en cours de développement au cours de cet atelier, avec les participants à ce petit évènement. Finalement, l'ensemble de l'évènement est sponsorisé par Bentley Systems, et marque le début d'allers-retours constants entre la communauté de Smart Geometry et l'éditeur pour le développement de Generative Components. En 2007, Generative Components est finalement officiellement commercialisé par Bentley Systems[112], et Robert Aish quitte son poste chez l'éditeur dans la foulée pour rejoindre Autodesk. Generative Components bénéficie néanmoins au cours des années qui précèdent d'un système unique de développement, porté par Aish qui a continué après son départ à participer à Smart Geometry et à établir des liens entre Bentley et ses usagers via le groupe. Jusqu'en 2013 au moins, Smart Geometry a continué à organiser des ateliers annuels de travail sur quatre versions successives

[110] Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.
[111] Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.
[112] Même si de nombreuses institutions du champ computationnel proposent entre temps des formations à Generative Components au même titre que Maya ou Rhinocéros, et même si il est en usage dans de nombreuses agences envoyant des membres à Smart Geometry.

de Generative Components[113], en partenariat avec Bentley Systems qui sponsorise toujours la conférence[114].

Generative Components est donc indissociable des ambitions et des activités du noyau dur de praticiens de Smart Geometry, dont nous avons vous au chapitre précédent qu'il s'agit d'une partie spécifique du réseau. L'usage du logiciel se concentre autour d'une communauté londonienne de praticiens, d'agences d'architecture et de bureaux d'études techniques. Foster & Partners, KPF, Arup mais aussi Grimshaw, Aedas ou Morphosis font partie de ses premiers utilisateurs et cercle de testeurs. Mais malgré la longue liste d'agences renommées et de praticiens expérimentés qui s'investissent dans son développement, Generative Components n'a qu'un succès mitigé en dehors de ce réseau. Si un certain nombre d'agences américaines et anglaises y ont recours, ce sont souvent des agences de grande taille qui peuvent se permettre l'entretien d'un petit groupe de spécialistes. L'utilisation de Generative Components est souvent cantonnée à ce groupe de spécialistes, sans passer entre les mains de la majorité des praticiens employés par l'agence[115]. Ces spécialistes font de plus déjà partie de Smart Geometry et du champ computationnel. Generative Components est donc enfermé au sein d'un réseau autonome, qui travaille sur la majeure partie des projets de la période qui impliquent de par leur envergure un recours à des outils numériques de pointe[116]. Le logiciel fait des aller-retours entre les praticiens du champ participant à Smart Geometry et leurs agences ou ils l'importent, sans sortir de cette trajectoire. Ceci est aussi une conséquence du paysage du champ computationnel, qui reste à cette période un microcosme même si le nombre de praticiens augmente. L'histoire de Generative Components, indissociable de celle de Smart Geometry, l'ancre dans un usage en agence. Mais le triple principe de programmation, programmation visuelle et modélisation sur lequel il repose en fait un outil trop complexe pour des architectes de formation classique, l' empêchant de se répandre comme espéré[117]. Le départ de Robert Aish, dont la vision à la fois du potentiel de la computation pour l'architecture et des enjeux de l'adoption par la profession de ces outils avait porté le développement de Generative Components, joue certainement aussi sur les ralentissements que le logiciel voit ensuite dans son utilisation. Enfin, la lenteur de l'industrie de l'architecture et de la construction à adopter de nouvelles technologies de production, alors que c'est

[113] Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.
[114] Même s'ils ne sont plus les seuls.
[115] Sauf par exemple chez Arup mais il s'agit d'un bureau d'études techniques, ou les pratiques informatiques sont déjà beaucoup plus répandues.
[116] On retrouve la circonscription de ces outils à des projets d'exception, qu'on a abordée au chapitre précédent.
[117] Au chapitre VI, on verra les problèmes que pose ce format plus en détail.

pourtant ce monde que vise Bentley Systems avec Generative Components, n'aide pas non plus au décollage du logiciel[118]. Peut-être ses promoteurs sont-ils arrivés encore une fois trop tôt.

Un autre élément joue également sur la trajectoire de Generative Components au sein du champ computationnel : l'arrivée de Grasshopper, concurrent direct du logiciel, change la donne. Le succès de Grasshopper condamne en effet Generative Components à ne jamais vraiment décoller. Grasshopper est un environnement de programmation visuelle associé au modeleur Rhinocéros, commercialisé par McNeel. En 2006, McNeel recrute David Rutten, un architecte et urbaniste diplômé de la TU Delft. Celui-ci s'est formé en autodidacte à la programmation et se tourne immédiatement vers une carrière de développeur pour l'industrie de l'architecture après ses études. Pour McNeel, il écrit notamment sur le guide RhinoScript101, aide à la prise en main des outils de programmation disponibles au sein de Rhinocéros - d'abord RhinoScript, puis Python dans les versions ultérieures du guide. Le guide est initialement rédigé à l'occasion d'un atelier à la Angewandte de Vienne, ou les étudiants doivent se former à la programmation en seulement quelques jours[119]. Cette forte contrainte donne donc une expérience de départ particulièrement importante à David Rutten et à McNeel sur les enjeux de la transmission du savoir-faire de la programmation aux novices architectes. Rutten développe ensuite ce qui s'appelle le Monkey Script Editor, l'interface qui deviendra à terme celle qui sert actuellement pour programmer des scripts personnalisés. Une première version de Grasshopper sous son nom de travail Explicit History sort en 2007, puis sous son nom actuel en 2008. Une version pour Rhinocéros Mac est publiée en 2016, et Grasshopper, qui devait jusqu'alors être téléchargé séparément, est inclus dans Rhinocéros par défaut en 2018, signe de son succès et du rôle clé qu'il joue désormais chez les utilisateurs de McNeel. Grasshopper est écrit dans un langage de programmation spécifique, mais les composants manipulés par les utilisateurs peuvent aussi être programmés en RhinoCommon, un langage qui donne accès à RhinoScript et à Python dans toutes les versions de Rhinocéros. Pensé comme un environnement de programmation complémentaire à celui qui permet de programmer des scripts directement, Grasshopper prend la forme d'une initiation à la programmation orientée objet pour les utilisateurs novices.

Mais avant ça, Grasshopper est tout simplement un environnement de programmation visuelle, tout comme Generative Components dont il s'inspire très probablement à ce titre. Robert McNeel indique

---

[118] Generative Components survit quand même très bien maintenant dans un autre domaine que le champ computationnel, celui des infrastructures.

[119] *"This booklet was originally written as a workshop handout for the students at the Architecture faculty of the Universität für Angewandte Kunst in Vienna. The aim of the workshop was to teach them how to program Rhino in no more than four days and, counter all my expectations, they did. Most of them had never programmed before so I had to make sure the text was suitable for absolute beginners. I did not expect at the time that this proved to be the most successful aspect of the primer."* David Rutten, RhinoScript Primer 101.

dans un entretien avec le magazine AEC s'être peu documenté sur Generative Components avant de lancer le développement de Grasshopper[120]. Mais malgré ces déclarations, Grasshopper présente certaines similitudes avec son concurrent qui laissent penser que McNeel s'est tout de même inspiré de Generative Components et de ce qui fait alors son succès. Les principes de programmation visuelle proposés par Grasshopper n'existent à ce moment que chez Generative Components dans le champ, mais ce depuis 5 ans. Des similitudes existent aussi dans le modèle de développement des deux logiciels, or nous avons souligné combien Bentley a adopté un système peu commun au moment du développement de Generative Components quelques années plus tôt. Comme Generative Components avant lui, Grasshopper est d'abord annoncé comme gratuit pour les utilisateurs de Rhinocéros tant que le développement n'en est pas terminé[121]. Autre trace d'un développement similaire, le revendeur principal de Rhinocéros à Londres, Simply Rhino, organise à partir de 2005 la conférence biannuelle Shape To Fabrication[122]. Celle-ci réunit un ensemble de praticiens, en grande partie issus du champ computationnel, pour présenter et discuter les nouveaux développements du logiciel et de ses plug-ins. A partir de sa seconde édition, en 2007, Grasshopper, qui vient de sortir, est bien sûr au cœur de l'événement. Il est aussi l'un des éléments les plus discutés au cours des conférences suivantes. Les programmeurs de chez McNeel, notamment David Rutten, y sont présents pour échanger avec les utilisateurs et réfléchir aux développements prochains. L'événement fonctionne donc sur un modèle très proche des conférences de Smart Geometry.  Il accueille des présentations des développeurs de chez McNeel et de chez les quelques entreprises qui proposent des plug-ins pour le modeleur, et occasionnellement des conférences de chercheurs renommés du champ computationnel, comme Rivka Oxman en 2007. Mais surtout, l'événement accueille des praticiens du monde de l'architecture et de la construction, avec quelques agences et bureaux d'études qui reviennent tous les ans ou presque présenter des cas d'études de projets modélisés avec Rhinocéros et Grasshopper et offrir à McNeel un

---

[120] *Bob McNeel admitted that the company does not know much about GC, "except that people tell us that it is harder to learn and use than Grasshopper. Since Grasshopper is very flexible, users can set up most any kind of relationship they like, so I guess you could say some of those relationships are parametric. But if the user wants to organise their generative model more like a script, it is more script-like. We are trying not to limit anyone's shape generation process by forcing them to think about it in a certain way. In most cases, Grasshopper is instantly interactive when you change an input (geometry or parameter) or when you change the definition."*
Martyn Day, "Rhino Grasshopper", AEC Magazine, 2 Juin 2009, https://aecmag.com/news/rhino-grasshopper/, consulté le 02 Septembre 2021.

[121] Grasshopper reste finalement gratuit jusqu'à être intégré dans Rhinocéros 6 en 2018. Il devient alors payant dans le sens ou il faut acheter une licence Rhinocéros pour pouvoir y accéder, même si la licence Rhinocéros n'augmente pas de prix pour autant et même si les versions antérieures de Grasshopper peuvent toujours être téléchargées gratuitement.

[122] "*Rhino is particularly popular with expressive London-based architects, such as Zaha Hadid, Buro Happold, HOK Sport and Foster + Partners. Fostering Grasshopper's usage in London is SimplyRhino, the largest Rhino reseller. The company runs the annual Shape to Fabrication event which focuses on the use of Rhino and Grasshopper in modelling forms and shapes, through to complex engineering analysis and final manufacturing. McNeel programmers and even Bob McNeel usually make an appearance and are very accessible*." Martyn Day, "Rhino Grasshopper", AEC Magazine, 2 Juin 2009, https://aecmag.com/news/rhino-grasshopper/, consulté le 02 Septembre 2021.

retour utilisateur. L'Advanced Geometry Group d'Arup, Adams Kara Taylor, Aedas, le Smart Group de Buro Happold, Populous, Heatherwick ou ZHA sont ainsi des habitués.... comme ils l'étaient de Smart Geometry pour Generative Components.

Les deux environnements partagent les qualités apportées par le recours à la programmation visuelle. Moins d'opérations sont nécessaires pour construire un programme, un certain nombre d'instructions étant déjà formulées en amont. Par exemple, il n'est pas nécessaire de déclarer ses variables. La mise en diagramme d'étapes logiques que demande la programmation visuelle est également une aide. Le diagramme fonctionne à la manière d'une carte mentale, ce qui aide de nombreux utilisateurs à mieux visualiser la structure de l'algorithme. Enfin, la programmation visuelle permet une évaluation progressive du programme. Plutôt que de l'écrire en entier et de l'exécuter pour voir s'il fonctionne, l'utilisateur peut vérifier à chaque étape que celle-ci donne les résultats attendus[123]. Ceci fait de la programmation visuelle une méthode d'apprentissage de la programmation particulièrement appréciée des novices, comme le souligne des études consacrées à ces techniques de programmation telles qu'utilisées non seulement dans le champ computationnel mais aussi ailleurs[124]. Mais Grasshopper gagne en popularité face à son concurrent notamment en raison de son interface. Celle-ci permet un aller-retour entre quatre niveaux de manipulation des algorithmes : la modélisation, la programmation visuelle, le script et la programmation. Ce système en couches est choisi par David Rutten, qui le considère comme le meilleur compromis pour satisfaire un grand nombre d'utilisateurs avec des niveaux de programmation disparates[125]. Si Rutten est plus un exécutant pour McNeel qu'un praticien mu par des ambitions pédagogiques, comme l'est Robert Aish, il est néanmoins extrêmement attentif à l'interface de Grasshopper et à la courbe d'apprentissage qu'elle offre aux utilisateurs. Generative Components offre d'ailleurs dans ses versions ultérieures une interface qui ressemble beaucoup plus à celle de Grasshopper que son interface initiale. Couplée avec Rhinocéros, qui permet de modeler des surfaces complexes de manière très intuitive, Grasshopper permet de se prendre au jeu de la

[123] Green,T. R. G., Petre, M., "Usability Analysis of Visual Programming Environments: a Cognitive Dimensions Framework", *Journal of Visual Languages and Computing*, 1996, 7(2), 131-174. Cité par Celani, Gabriela, et Carlos Eduardo Verzola Vaz. "CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison from a Pedagogical Point of View." International Journal of Architectural Computing 10, no. 1 (March 2012): 121–37.

[124] James Novak et Jennifer Loy, "Recoding Product Design Education: Visual Coding for Human Machine Interfaces", *KnE Engineering* 2(2):227, 2017. Celani, Gabriela, and Carlos Eduardo Verzola Vaz. "CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison from a Pedagogical Point of View." International Journal of Architectural Computing 10, no. 1 (March 2012): 121–37.

[125] David Rutten, "Discoverability", 25 Mai 2012, https://ieatbugsforbreakfast.wordpress.com/2012/05/25/discoverability/#more-431, consulté le 02 Septembre 2021.

programmation orientée objet, avec une grande facilité d'utilisation pour des novices[126]. Ceci le rends populaire particulièrement auprès des étudiants[127], et pousse d'autant les praticiens du champ computationnel qui enseignent à se tourner vers cet environnement.

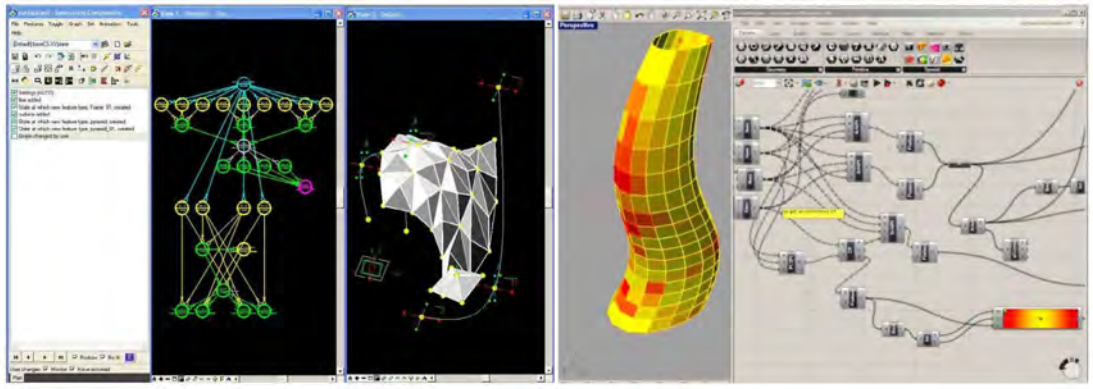


*Figure 27. Interface des logiciels Generative Components et Grasshopper*

Grasshopper est aussi plus didactique que Generative Components en raison de la communauté qui existe autour du logiciel. En complément du groupe de praticiens de Shape to Fabrication, celle-ci est composée de tous les utilisateurs qui souhaitent y participer. Les forums successifs offerts par McNeel sont un lieu d'échange pour les utilisateurs, et les développeurs de chez McNeel y sont très actifs, répondant très rapidement aux requêtes des utilisateurs. Non seulement les utilisateurs novices ont accès beaucoup plus facilement à des fichiers d'exemples ou à de l'aide sur leur travail de programmation en cours, mais la diversité des projets exécutés sur Grasshopper tels que montrés par le forum est bien plus vaste que chez Generative Components. Les projets construits par des agences

---

[126] Voir par exemple cette critique dans AEC Magazine "*Grasshopper is being used and talked about by the same people that had advanced geometry needs and bought into Bentley Systems' Generative Components (GC). However, GC is script-based and requires training. It is also based on MicroStation, which has a parametric modeller, while Grasshopper uses a very visual plug and play interface to automate the scripting and is based on Rhino, which is a non-parametric surface modeller. (...) With Bentley already there and Autodesk in hot pursuit I was pretty amazed to see an early demonstration of Grasshopper in Rhino last year. The output appeared to be similar to GC but the way the scripts are generated was amazingly "plug and play". You don't need to know how to program to create a parametric model using Grasshopper as there is a new window for creating what amounts to a bank of geometry "effects peddles", as you would with a guitar. You take the input, which may be a line and you literally plug it into an effect, which can be any kind of mathematical transform, and which can subsequently be plugged into other mathematical effects peddles, all driving the creation of a script (in the background) that drives Rhins geometry engine. Its all drag and drop and plug and play.*" Martyn Day, "McNeel", AEC Magazine, 01 Juillet 2008, https://aecmag.com/features/mcneel-flying-circus/, consulté le 02 Septembre 2021.

[127] Notamment en comparaison de la programmation classique, voir par exemple James Novak et Jennifer Loy, "Recoding Product Design Education: Visual Coding for Human Machine Interfaces", *KnE Engineering* 2(2):227, 2017.

ne sont qu'une infime partie de ce que le forum met en valeur : un grand nombre de projets de papier et de projets d'étudiants sont aussi présents. Plusieurs observateurs associent l'environnement de Grasshopper à ce que Eric Raymond, célèbre programmeur américain inventeur du terme open-source, nomme un "bazar", par opposition à une "cathédrale". La seconde fait référence aux environnements logiciels classiques, de vastes applications développées par un petit groupe dans un environnement fermé. Le premier désigne les environnements logiciels qui émergent grâce aux contributions d'une communauté plus vaste, ouverte[128]. Grasshopper s'appuie sur un certain nombre de principes open-source qui font son succès : la communauté qui l'entoure et leur participation au développement, la gratuité du téléchargement, la possibilité de le modifier en fonction de ses besoins. McNeel est en effet, nous l'avons vu, très à l'écoute de ses utilisateurs et de leurs besoins, et très réactif à leurs sollicitations. Mais Grasshopper n'est pas vraiment gratuit : il faut avoir acheté Rhinocéros pour pouvoir s'en servir[129] . Grasshopper est par ailleurs effectivement ouvert et n'importe qui peut accéder à son API ou ajouter des fonctionnalités. Mais Maya aussi est personnalisable, l'API d'AutoCAD aussi est accessible aux développeurs. La communauté de Grasshopper est sans conteste une de ses forces, mais McNeel a aussi su capitaliser sur une approche marketing autour de la notion d'open-source qui joue en sa faveur. Cette image de gentils éditeurs dans une industrie très compétitive, que soulignent de nombreux échanges du forum[130] comme des articles de magazines particulièrement flatteurs[131], joue certainement aussi dans le succès de Grasshopper au sein du champ computationnel.

L'écosystème que constitue Grasshopper est particulièrement visible à travers les plug-ins auxquels il donne accès. Generative Components est un précurseur de Grasshopper, dont les ambitions et le mode de développement saisissent déjà des enjeux critiques à la genèse d'un nouvel environnement de programmation pour le champ computationnel. Mais Grasshopper, qui reprend ces ambitions et ce mode de développement, les combine avec un système de plug-ins ouvert qui favorise à la fois la prise en main par des novices et les échanges au sein d'une communauté très large. Celle-ci capitalise sur le fait que McNeel et Grasshopper ont profité de la vague internet et du développement des espaces de

---

[128] "*The self-contained design environments set up to support these monolithic processes belong to a software category that Eric Raymond calls 'cathedrals' – large applications crafted by a highly talented group working together in isolation.3 Raymond contrasts the cathedral with the 'bazaar' – a marketplace in which the collective action of individuals contributes to the larger community. For architects, the dichotomy exposes the fact that for many decades design environments have almost exclusively consisted of cathedrals. Breaking the norm, Robert McNeel & Associates' Grasshopper® is a bustling bazaar-type environment*." Davis, D. and Peters, B. (2013), Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins. Archit Design, 83: 124-131

[129] Même si Rhinocéros est moins cher que les autres.

[130] Voir par exemple l'échange "Love for McNeel" sur le forum Rhinocéros, démarré en Aout 2019, https://discourse.mcneel.com/t/love-for-mcneel/87678/48, consulté le 02 Septembre 2021.

[131] Martyn Day, "McNeel", AEC Magazine, 01 Juillet 2008, https://aecmag.com/features/mcneel-flying-circus/, consulté le 02 Septembre 2021.

partage en ligne[132], ce que Generative Components n'a pas vraiment su faire. Mais c'est aussi Grasshopper qui donne la pleine mesure du format des plug-ins, plus que Generative Components. L'environnement McNeel est un portail vers les outils du champ, sous la forme de plug-ins. Ceci structure d'une nouvelle manière l'accès aux typologies. Pour chacune d'entre elles, différents plug-ins existent, donnant accès à des algorithmes pré structurés aux utilisateurs, qui n'ont plus qu'à en faire varier les paramètres principaux. Grasshopper peut par exemple être étendu avec Galapagos, un plug-in qui permet d'utiliser des algorithmes génétiques, développé par David Rutten en 2010[133]. Kangaroo, développé en 2011 par Daniel Piker, membre du Specialist Modelling Group de Foster + Partners, donne accès à un moteur physique permettant d'exécuter de nombreuses simulations, y compris des modèles de relaxation dynamique[134]. Ces deux plug-ins sont si usités par la communauté Grasshopper qu'ils sont intégrés par défaut dans Rhinocéros 6 également. Autre plugin du Specialist Modelling Group, Pachyderm est développé par Arthur van der Harten, acousticien qui rend par ce biais ses connaissances de spécialiste accessibles non seulement au reste de l'agence, mais aussi à tout utilisateur intéressé[135]. Weaverbird, développé en 2009 par l'architecte Giulio Piacentino, lui aussi recruté par McNeel pour travailler sur l'écosystème Grasshopper, mets à disposition des outils de rationalisation des maillage basé sur les algorithmes principaux disponibles : subdivision Catmull-Clark, triangulation de Delaunay ou de Sierpinski[136]. Karamba, module d'analyse structurelle par la méthode des éléments finis, est développé par Clemens Preisinger, un des ingénieurs de l'équipe viennoise de Bollinger + Grohmann[137]. Le plug-in repose sur un programme d'analyse aux éléments finis écrit pour pouvoir être combiné avec un algorithme génétique. Le programme devant pouvoir évaluer un grand nombre d'options, l'évaluation structurelle doit donc être particulièrement rapide, ce qui n'est pas le cas avec d'autres modules d'analyse aux éléments finis. Le programme obtenu ayant été jugé par les équipes de Bollinger + Grohmann comme particulièrement intéressant à utiliser au-delà du projet de recherche dans le cadre duquel il avait été écrit, Preisinger choisit d'en faire un plug-in Grasshopper, et ainsi de tirer parti de l'interface déjà existante de l'environnement. Dans l'ensemble, les développeurs des plug-ins sont poussés par deux motivations. Certains d'entre eux, comme Arthur van der Harten, souhaitent mettre leurs connaissances à disposition d'autres praticiens, et sont donc mus par des ambitions pédagogiques. D'autres, comme Clemens Preisinger ou Giulio Piacentino, sont confrontés à des difficultés pratiques auxquelles la programmation d'un

[132] Martyn Day, "McNeel", AEC Magazine, 01 Juillet 2008, https://aecmag.com/features/mcneel-flying-circus/, consulté le 02 Septembre 2021.
[133] Rutten, D. (2013), Galapagos: On the Logic and Limitations of Generic Solvers. Archit Design, 83: 132-135.
[134] Piker, D. (2013), Kangaroo: Form Finding with Computational Physics. Archit Design, 83: 136-137.
[135] van der Harten, A. (2013), Pachyderm Acoustical Simulation: Towards Open-Source Sound Analysis. Archit Design, 83: 138-139.
[136] Voir chapitre III pour une explication plus détaillée de ces algorithmes. Pour Weaverbird, Piacentino, G. (2013), Weaverbird: Topological Mesh Editing for Architects. Archit Design, 83: 140-141.
[137] Preisinger, C. (2013), Linking Structure and Parametric Geometry. Archit Design, 83: 110-113.article Karamba

nouvel outil leur permet de répondre, et partagent ensuite cet outil avec les praticiens qui rencontrent les mêmes difficultés.

D'autres plug-ins sont spécifiquement conçus pour servir de passerelle entre Grasshopper et un autre environnement. C'est le cas de Firefly, qui transmet et reçoit des données de la plateforme Arduino, dont nous avons vu qu'elle sert à produire des objets électroniques open-source. C'est aussi le cas de GECO, qui fait le lien entre Grasshopper et le logiciel d'analyse environnemental Ecotect. Grasshopper a donc aussi pour qualité de permettre aux praticiens de se construire un écosystème de programmation en liant plusieurs logiciels différents[138]. Nous avons vu que Python et les autres "langages-colle" sont prisés des praticiens du champ computationnel justement parce qu'ils permettent de faire le lien entre différents outils. C'est aussi le cas de Grasshopper, un atout d'interopérabilité supplémentaire qui explique sa popularité. L'API de Grasshopper permet en effet un échange d'information autour de géométries primitives très simples, plutôt qu'autour d'attributs - c'est-à-dire de propriétés des objets - comme le font Generative Components ou d'autres environnements de programmation[139]. Ceci pose parfois des limites en termes de modélisation, mais rend l'échange entre programmes largement plus facile. Daniel Davis et Brady Peters résument ceci en l'associant à aux principes de programmation de Douc McIlroy, l'un des inventeurs du système UNIX : il faut "écrire des programmes qui font une chose et qui le font bien, écrire des programmes pour qu'ils puissent fonctionner ensemble"[140]. Dominik Holzer, professeur associé à Melbourne, résume ceci par l'idée que Grasshopper est une écologie d'outils[141]. Rhinocéros, en association avec Python et Grasshopper, est en effet un environnement particulièrement riche et complet, à la fois en termes de modalités d'utilisation, de capacités de modélisation, et d'offre, avec sa liste de plug-ins sans cesse en augmentation. La genèse des environnements d'accueil des plug-ins en usage dans le champ montre qu'ils s'appuient sur les trois piliers formulés par Ben Fry et Casey Reas : langage, environnement, communauté. Ces plugins sont une extension de la programmation dans un

[138] Ceci est désigné dans le champ par la notion de "*design ecosystem*". Davis, D. and Peters, B. (2013), Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins. Archit Design, 83: 124-131.

[139] "*The Grasshopper application programming interface (API) developed by David Rutten, which formalises the exchange of data around simple collections of basic geometric primitives. This 'geometric content-based' data exchange is in opposition to BIM's 'assigned attribute-based' data structures, and is a simplification that enables plug-ins to easily work together*". Davis, D. and Peters, B. (2013), Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins. Archit Design, 83: 124-131.

[140] "*The focus of each plug-in on a particular niche problem follows what Doug McIlroy has termed the Unix philosophy of programming: to 'write programs that do one thing and do it well. Write programs to work together.' It is the 'working together' that distinguishes the Grasshopper environment, for all the plug-ins within it can freely exchange data with one another*". Davis, D. and Peters, B. (2013), Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins. Archit Design, 83: 124-131.

[141] Dominik Holzer, "Design exploration supported by digital tool ecologies", September 2016, Automation in Construction 72

environnement logiciel plus facile d'utilisation. Le principe des plugins est d'une certaine manière le passage de l'ambition de rendre les fonctions sur-mesure accessibles à l'ambition de rendre les fonctions prêtes à l'emploi. Le profil des utilisateurs qui y recourent évolue donc. Les plug-ins constituent donc une autre stratégie d'initiation à la programmation, comme la réflexion autour des courbes d'apprentissages intégrée aux environnements principaux d'accueil des plug-in le montre.

L'acquisition du savoir-faire du champ computationnel en architecture est donc conditionné notamment au format des outils de programmation employés. Ces formats diffèrent dans la relation qu'ils permettent entre accessibilité des fonctions logicielles et transmission du savoir-faire. Cet équilibre est aussi affecté par le positionnement des différents outils entre programmation généralistes et programmation spécialisée. Plus que les autres néanmoins, les bibliothèques apparaissent comme un intermédiaire équilibré entre l'algorithme et le logiciel. Le logiciel se place à un extrême trop simplifié, qui tend à produire toujours les mêmes architectures. Les algorithmes se placent à un niveau trop granulaire, car ils ont besoin d'être recomposés à chaque projet. Des algorithmes aux scripts, des scripts aux bibliothèques et des bibliothèques aux plugins, la transmission des savoirs du champ computationnels se fait donc par le biais d'outils plus ou moins strictement définis au préalable. Dans le cas des scripts et des bibliothèques , l'intermédiaire qu'elles offrent entre devoir partir de zéro ou ne pas avoir assez de marge d'adaptation est précisément la qualité qui leur permet de devenir des vecteurs de transmission puissants.

Faire ses propres outils est toujours une idée porteuse dans le champ computationnel. Les formats que nous venons d'examiner sont un des résultats du développement des outils en fonction de l'accessibilité de cette possibilité pour des praticiens qui sont initialement formés non pas à la programmation mais à la conception. Ils interrogent donc la place de l'utilisateur face à son outil. Les formats représentent à ce titre des marches d'apprentissage, des niveaux de progression, car ils nécessitent différents niveaux de compétence. Leur accessibilité à travers des boîtes à outils qu'ils représentent permet de dire que même les plugins peuvent être envisagés dans certains environnements comme demandant d'assembler des groupes de fonctions spécifiques pour chaque projet. Ainsi, chacun des formats, même les plus simples, permettrait de construire son propre code, et donc son propre outil. La programmation visuelle à laquelle les plug-ins sont souvent associés fournit un second intermédiaire intéressant entre logiciels complets et la liberté parfois trop grande de la programmation, comme nous allons le voir maintenant. En effet ce qui caractérise les différents outils, ce n'est pas seulement leur format, c'est aussi leur interface : les deux s'entrelacent dans la description de la relation des praticiens à leurs outils.

## 5.3 Une question d'interfaces

### *5.3.1 Familles d'interfaces*

Nous avons vu non seulement les diverses typologies auxquelles se sont intéressés au fil des années les praticiens du champ computationnel, mais aussi les différents supports de leur mobilisation. Les premiers à apparaître sont les programmes écrits directement dans des langages de programmation divers, d'ALGOL et de FORTRAN des débuts à Python, à C# et à Java à des périodes plus récentes. Ceux-ci vont ensuite évoluer vers les bibliothèques, ces packages faisant office d'intermédiaire entre l'écriture d'un programme sur mesure et la formalisation d'un logiciel ou d'un plug-in à part entière. Enfin, les plug-ins, sortes de mini-logiciels, extensions de logiciels déjà existants leur ajoutant de nouvelles fonctions, et les logiciels à part entière, programmes formalisés auxquels s'ajoute une interface utilisateur. Cette dernière prendra une forme différente des interfaces qu'offrent les autres supports, notamment parce qu'elle n'a pas le même objectif : comme son nom l'indique, l'interface utilisateur s'adresse à un utilisateur qui se différencie du programmeur par ses connaissances. La démocratisation à laquelle se confronte le champ computationnel demande précisément de s'adresser à des utilisateurs qui n'ont pas le savoir-faire pointu des praticiens jusqu'ici impliqués : les nouveaux outils, conçus comme les vecteurs de cette popularisation, doivent le prendre en compte. Et pour comprendre les évolutions récentes du paysage computationnel en architecture, il est donc nécessaire d'analyser les interfaces des différents outils à disposition, élément clé de leur manipulation. L'interface est le dispositif qui permet à l'utilisateur d'échanger des informations avec la machine, or c'est cette communication qui est la clé de la conception algorithmique en architecture[142]. Les interfaces informatiques peuvent être de deux types : physiques ou logicielles. Les interfaces physiques sont les objets matériels manipulés par l'utilisateur : la souris, l'écran, le clavier pour nos ordinateurs contemporains, mais aussi les casques de réalité virtuelle ou les objets plus anciens comme le stylet ou les cubes qu'utilisaient les premiers praticiens du champ computationnel[143]. Les interfaces logicielles sont les éléments graphiques qui apparaissent sur l'écran, et que l'utilisateur peut manipuler à l'aide des interfaces physiques. C'est à ces interfaces logicielles qu'on va s'intéresser ici, car ce sont elles qui se sont modifiées significativement au fil du développement du champ computationnel, les interfaces physiques restant presque toujours l'écran, le clavier et la souris qu'on connaît, à la suite de leur développement dans les années 1970.

[142] Voir chapitre I.
[143] Voir chapitre II.

Examinons d'abord les interfaces utilisées pour manipuler directement les langages de programmation qu'utilisent les praticiens du champ computationnel, comme Python, C#, MEL ou AutoLISP (figure 28). Ce sont usuellement ces interfaces auxquelles ils sont confrontés lorsqu'on dit qu'ils programment, qu'ils codent ou qu'ils scriptent[144]. Ces différents langages de programmation sont associés à un compilateur qui permet de les manipuler. Les compilateurs sont dotés d'interfaces dites en ligne de commande, ou l'utilisateur saisit des instructions sous forme de texte. Le texte saisi n'est cependant pas du langage naturel, puisqu'il faut se conformer aux particularités du langage de programmation utilisé. Les langages de programmation peuvent être de différents niveaux d'abstraction : ceci désigne la distance entre les instructions comprises par le processeur de la machine et la formulation des instructions dans le langage de programmation. Plus le langage de programmation se rapproche du langage naturel[145] et plus on considère qu'il est d'un haut niveau d'abstraction, car il s'éloigne de plus en plus des instructions telles qu'elles sont exécutées par le processeur. Les langages de programmation en usage dans le champ computationnel sont presque toujours de haut niveau[146], c'est-à-dire que certains mots du langage naturel sont compris directement comme des instructions. Des fonctions sont pré-associées à ces mots : par exemple, pour faire tracer à l'ordinateur un cercle dans Processing, il faut écrire *circle (x, y, r)*, ou x et y sont les coordonnées du centre et r le rayon. Il n'est pas nécessaire de rédiger l'ensemble des instructions correspondant à une méthode de tracé d'un cercle, ni aucune des instructions d'affichage de ce cercle. Les bibliothèques sont donc justement des ensembles de fonctions ajoutées, qui permettent d'augmenter le niveau du langage utilisé. Elles regroupent souvent des mots spécialisés en fonction de la discipline pour laquelle est développé le langage. Par exemple, dans le cas de l'architecture ou de l'animation, un vocabulaire spécifique peut exister, qui désigne des opérations géométriques sur les volumes : *join* (joindre des éléments), *split* (couper un élement) ou *scale* (mettre à l'échelle) par exemple. En plus des fonctions, ces langages de programmation disposent d'une syntaxe pour reconnaître les commandes et la manière de les exécuter. Toujours dans Processing par exemple, un fichier comporte usuellement plusieurs fonctions composées par l'utilisateur pour construire son programme, qui doivent chacune être nommées et délimitées par une accolade ouverte et une accolade fermées, signes qui permettent au compilateur de lire la structure du programme. Il faut également comprendre les signaux de l'interface, qui indiquent à l'aide d'une autre série d'éléments de syntaxe que le programme est prêt à exécuter les instructions de programmation, que le calcul est en cours ou encore que les résultats de l'exécution peuvent être consultés. Le compilateur sert à traduire le programme écrit en langage de haut niveau en langage de bas niveau. Certains logiciels associent directement un langage et un compilateur, quand d'autres permettent d'écrire en plusieurs langages différents en

[144] Argot dérivé du mot script.
[145] Le langage naturel est celui que nous utilisons pour parler ou écrire.
[146] Seul C fait exception, car il peut être considéré comme un langage de bas ou de haut niveau.

fonction des besoins - ce qui est utile notamment quand le programme en cours d'écriture fait appel à plusieurs langages, comme c'est le cas pour de nombreux plugins Grasshopper mentionnés jusqu'ici. Les langages sont usuellement structurés pour des usages spécifiques, ce qui explique le type de compilateur auquel il est fait appel et leur apparence, les différentes fonctions qui y sont intégrées et de manière générale les particularités de chacun : on retrouve la versatilité de ces outils tels que nous l'avons vue au chapitre 4. Les interfaces en ligne de commande vont des plus minimalistes, ou il s'agit simplement d'entrer le texte et de lancer l'exécution, qui fournissent très peu d'informations complémentaires sur ce qui a été saisi aux plus guidantes, qui offrent une structure de fenêtre en différentes parties, des codes couleur pour repérer syntaxe et fonction et facilitent la visualisation des résultats une fois le programme exécuté. Processing, par exemple, dont on a vu qu'il s'appuie sur le langage Java et qu'il est essentiellement utilisé pour faire de l'animation visuelle[147] et qu'il possède un grand nombre de fonctions et de bibliothèques facilitant ce travail, se présente comme montré sur la figure 28. À l'ouverture, l'utilisateur est face à une interface en quatre parties. En premier, les onglets textuels, qui permettent comme dans la plupart des autres logiciels d'ouvrir un fichier existant, d'ouvrir le menu d'aide ou autres fonctions classiques. En second, deux icônes, play et stop, qui permettent de lancer l'exécution ou de l'interrompre. En troisième, un bloc d'écriture de texte vide. En quatrième, un petit espace de texte noire, réservé aux informations qui s'affichent pendant que le code est en cours d'exécution, en particulier les erreurs d'exécution, qui permettent d'orienter l'utilisateur vers les parties du fichier qui posent problème en cas de difficulté. Le texte saisi dans le bloc de texte apparaît coloré selon un code qui permet de voir ce qui est identifié : bleu pour les fonctions, rose pour les variables. Lorsqu'on lance l'exécution, une seconde fenêtre s'ouvre pour visualiser le résultat de l'exécution, ceci étant fait de la sorte puisque Processing sert essentiellement à des pratiques artistiques, du design génératif ou de l'animation.

[147] Voir chapitre III.

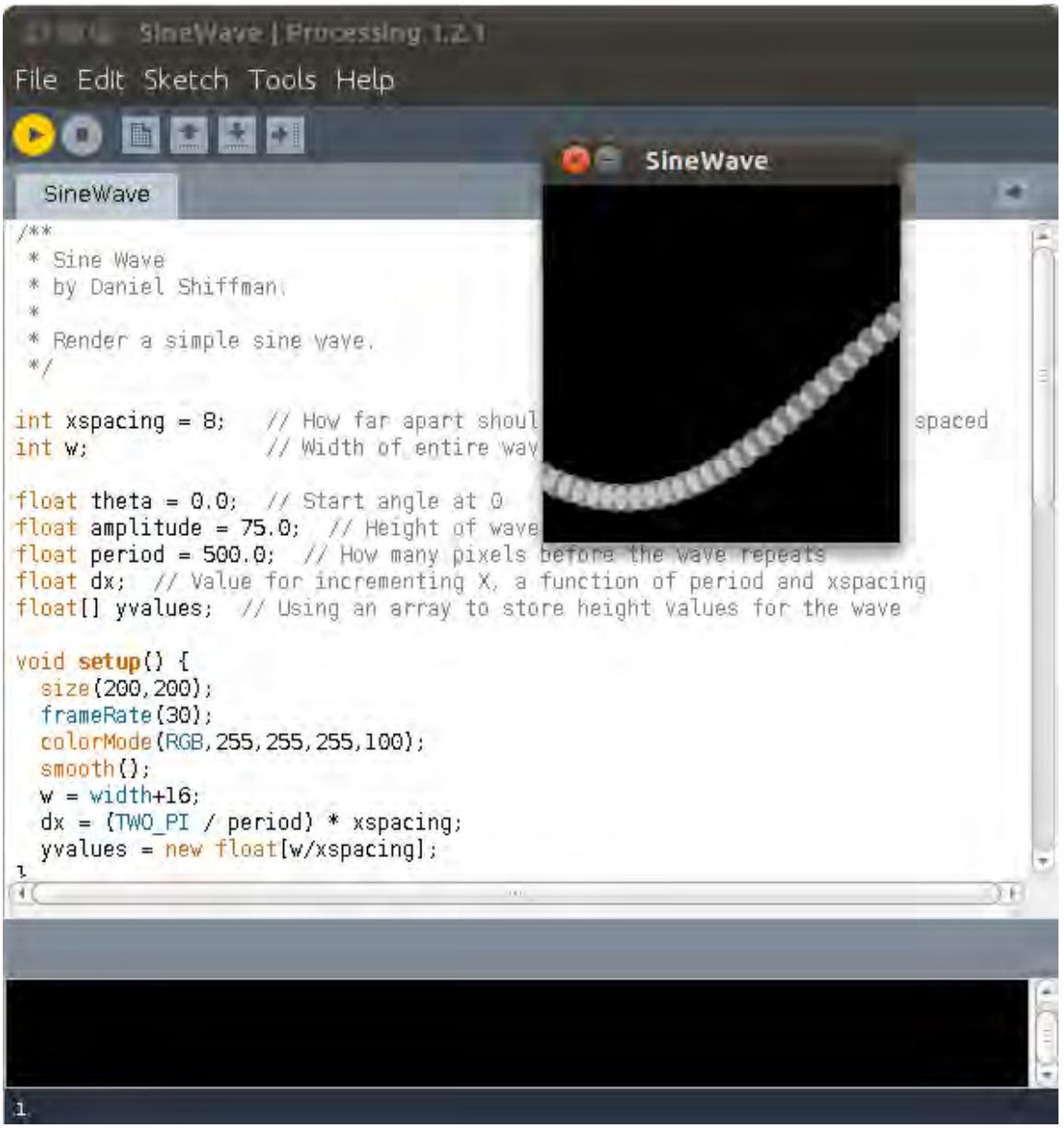



*Figure 28. Interface du logiciel Processing*

Cette double fenêtre de code et d'affichage est un dispositif prisé dans le champ computationnel car il permet des allers-retours entre la visualisation de la structure de l'algorithme et celle de son résultat géométrique. On la retrouve donc fréquemment, y compris avec d'autres types d'interfaces de programmation. C'est le cas par exemple des interfaces de programmation visuelle, ou programmation graphique, dans lesquelles on assemble des éléments graphiques pour construire le programme, plutôt que de manipuler des éléments textuels : Generative Components, Grasshopper, Dynamo. Les fonctions, plutôt que d'être représentées par des mots, sont généralement symbolisées par des blocs,

qu'on peut associer entre eux de différentes manières en fonction des logiciels. Le programme ainsi construit est ensuite exécuté de manière similaire à l'exécution d'un programme écrit avec une interface en lignes de commande : en étant d'abord traduit en langage de plus bas niveau, puis calculé par le processeur. Si on reprend l'exemple du cercle, pour parvenir à le tracer dans Rhinocéros à partir de Grasshopper, l'utilisateur a le choix entre plusieurs composants : *Circle 3Pt*, *Circle TanTan*, *Circle TanTanTan*, *Circle CNR*, ou *Circle*. Chacun correspond à une méthode de tracé différente du cercle, plus ou moins adéquate en fonction du programme : construire le cercle à partir de trois points par lesquels il passe, à partir de deux ou trois courbes dont il est tangent, à partir de son centre, de sa normale et de son rayon. La dernière, comme dans le cas de Processing, nécessite d'indiquer les coordonnées du centre et le rayon du cercle pour le faire tracer. Cependant, comme on peut le voir sur la figure 29, au lieu d'indiquer numériquement ces données, elles sont indiquées par d'autres boites : une boite *Point*, qui représente la fonction de construction d'un point et trois *sliders*, servant à indiquer des valeurs numériques. Deux d'entre eux sont connectés par un câble à la boite *Point*, elle-même connectée à la boîte Circle avec le troisième *slider*. En fonction des connexions effectuées grâce aux câbles, le logiciel assemble et hiérarchise les fonctions de manière différente pour créer le programme, qu'il exécute en temps réel. L'exemple du cercle peut sembler fastidieux en comparaison de la simplicité de la fonction circle de Processing, mais les boîtes manipulées peuvent cependant aussi être des commandes de beaucoup plus haut niveau : un programme entier écrit dans Processing peut devenir une simple boîte dans Grasshopper. C'est le cas par exemple avec les règles de déplacement des nuages de particules dans le cas des systèmes multi-agents : un comportement[148], qui nécessite plusieurs fonctions écrites les unes à la suite des autres dans Processing, peut devenir une seule boîte dans Grasshopper. Le principe des bibliothèques, ou extensions, est néanmoins conservé, même si elles changent de nom: ce sont désormais des plugins. Les outils de programmation visuelle ont une interface qui se classe déjà dans la catégorie des interfaces graphiques, mais on y visualise néanmoins encore la structure du programme en cours d'écriture. Initialement dérivé des flowcharts, ces cartes algorithmes conçues pour en représenter la structure visuellement, ces interfaces sont maintenant très appréciées pour l'apprentissage de la programmation, car elles sont à mi-chemin entre un logiciel avec une interface utilisateur usuelle et facile à manipuler, et la familiarisation avec ce qu'est qu'une structure informatique, puisqu'on y enchaîne encore des séries d'instructions. Nombre des logiciels de programmation graphique sont d'ailleurs conçus pour les enfants, comme Scratch et ScratchJR, conçus par le MIT pour les enfants à partir de cinq ans. L'ancien nom de Grasshopper, appelé à sa création Explicit History, pointe d'ailleurs vers cette volonté de visualiser les différentes étapes de la réalisation d'une géométrie, donc de voir l'algorithme qui permet de la construire.

[148] Pour plus de détail se référer au chapitre IV.

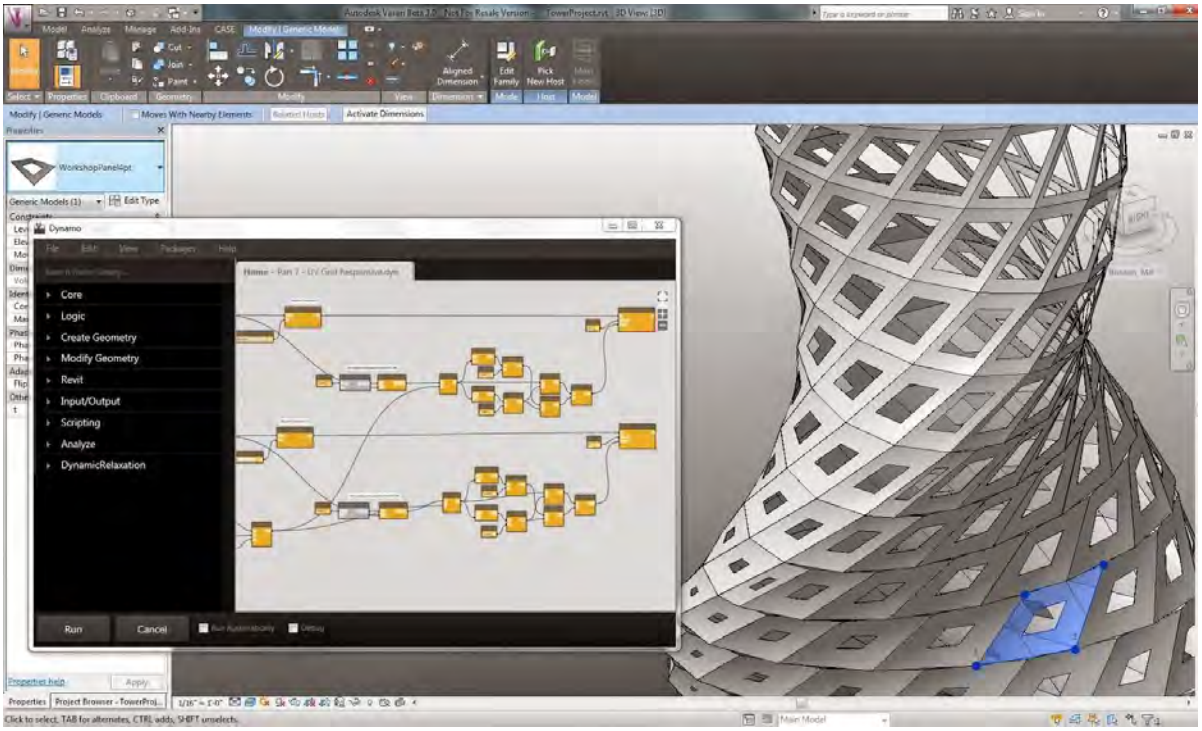



*Figure 29. Interface du logiciel Revit et de son plug-in Dynamo*

Dans le cas des interfaces graphiques, il ne s'agit plus d'écrire un programme, mais simplement d'en utiliser. Les fonctions mobilisées ne sont plus représentées par des mots ou des blocs qu'il faut assembler pour construire le programme qui est ensuite exécuté pour obtenir le résultat. Cette fois-ci, des pictogrammes font appel à des programmes déjà écrits, qui sont alors exécutés pour obtenir le résultat souhaité. Par exemple, l'interface de Paint propose une série de commandes pour dessiner en suivant le mouvement de la souris : on peut régler le type de trait, son épaisseur ou sa couleur, dessiner à partir de formes prédéfinies ou encore insérer du texte. Les modeleurs 3D comme Rhinocéros, Revit ou Form*Z ont tous des interfaces qui relèvent de cette catégorie. C'est aussi le cas des logiciels d'évaluation structurelle ou d'optimisation topologique, comme par exemple Altair Inspire, dans lequel les différents pictogrammes permettent d'indiquer sur un modèle 3D existant des supports et des charges, avant d'exécuter un programme qui supprimera la matière surnuméraire en fonction de ces indications. Si on reprend l'exemple du cercle, dans Rhinocéros, on trouve de nouveau plusieurs pictogrammes, qui correspondent à la même série de méthodes trouvées dans Grasshopper à partir de points ou à partir de courbes. La commande *Circle* permet toujours de tracer un cercle à partir d'un point central et d'un rayon. Mais plutôt que d'indiquer les variables nécessaires par du texte ou par d'autres blocs, il faut simplement cliquer une première fois sur l'écran, dans l'espace de travail, pour indiquer l'emplacement du centre, et une deuxième fois pour indiquer le rayon.

Rhinocéros possède par ailleurs une gumball - un ensemble de pictogrammes associés à chaque objet géométrique dans l'espace de travail, qui permet de les déformer en quelques clics. Si dans Processing il faut donc modifier la commande écrite, et dans Grasshopper l'assemblage des blocs, ici pour modifier le cercle il faut se servir de ce dispositif, qui permet de le transformer en cliquant et glissant avec la souris en ellipse ou en courbe indéfinie Certains praticiens du champ computationnel choisissent dans certains cas d'associer les programmes écrits à des interfaces graphiques : c'est le cas par exemple de l'agence ZHA pour son travail sur la conception de l'espace à partir des comportements des gens, ou du Bloc Research Group pour certains modules de COMPAS. L'objectif de ces interfaces là est de simplifier autant que possible le recours aux fonctions mathématiques et ceux sont celles qui sont utilisées par tous les novices en informatique, celles dont tout le monde à l'habitude de manipuler avec son traitement de texte, sa boîte mail ou son smartphone.

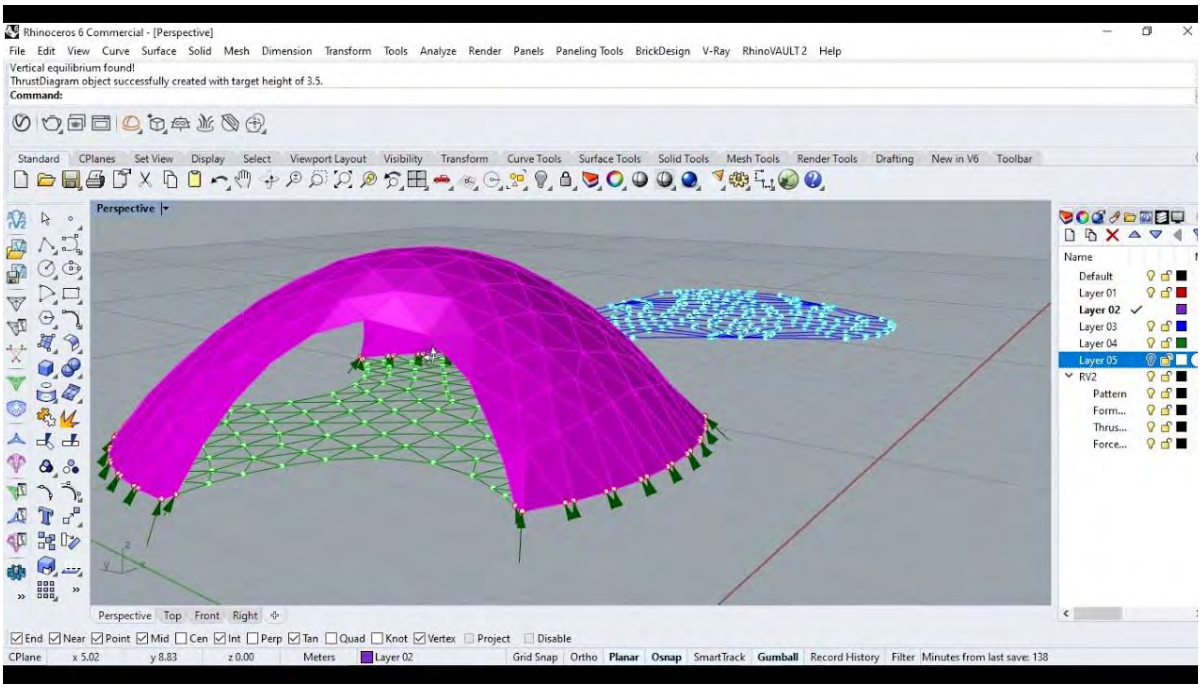



*Figure 30. Interface du logiciel Rhinocéros*

La combinaison de ces différentes catégories d'interface est fréquente dans les ordinateurs contemporains, et des interfaces mixtes existent aussi dans le champ computationnel. C'est bien sûr le cas quand Rhinocéros et Grasshopper ou Revit et Dynamo sont utilisés ensemble, puisque les premiers possèdent des interfaces graphiques et les seconds des interfaces de programmation visuelle. Autre combinaison, celle de Maya et MEL ou de Rhinocéros et Python, qui combinent ensemble une interface graphique et une interface en ligne de commande. C'est aussi le cas pour Flowerpower, dont

nous avions vu l'utilisation au chapitre 4 pour certains travaux de l'agence biothing, qui combine deux interfaces graphiques, celle de Rhinocéros et celle de Flowerpower, mais aussi l'interface de Python dans Rhinocéros, ou le programme écrit est disponible. Il n'est pas nécessaire d'avoir recours à cette dernière pour faire fonctionner Flowerpower et générer des formes, mais l'ajustement du programme peut être nécessaire et y est accompli. Autre exemple encore, Galapagos ou Octopus combine elles aussi trois niveaux d'interface : Rhinoceros, Grasshopper, et une deuxième interface graphique, qui permet d'accéder aux résultats de l'algorithme génétique, d'afficher les performances des solutions et de naviguer entre les générations d'individus. Combiner différentes interfaces peut être un moyen de simplifier le recours au programme en fonction des différents profils d'utilisateurs : c'est le cas pour Weaverbird, qui possède une interface de programmation visuelle, puisque ses fonctions apparaissent comme des composants de Grasshopper, mais aussi une interface graphique, puisque ses fonctions apparaissent également en pictogrammes dans Rhinocéros. Dans le champ computationnel comme ailleurs, ce sont ces interfaces qui permettent à des utilisateurs non formés ou en cours de formation d'accéder à des modèles de calcul informatique complexes, sans forcément en saisir les détails ou le fonctionnement.

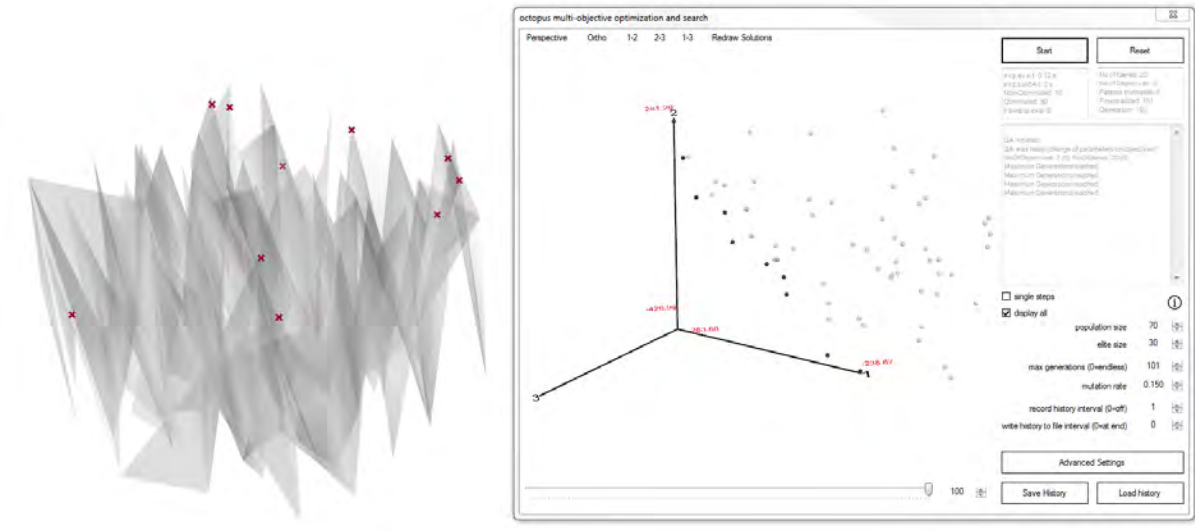



*Figure 31. Interface du plug-in Octopus*

Pour comprendre comment sont pensées les interfaces, et l'analyse proposée ci-après de celles du champ computationnel, quelques notions d'*UX design* peuvent être utiles. L'UX design, ou *User Experience desig*n, est un terme anglais qui désigne la partie de la conception des interfaces qui s'intéresse à l'expérience de l'utilisateur (*UI* en anglais). C'est l'ensemble des processus qui permettent d'évaluer la facilité, le confort et l'attractivité de l'expérience d'utilisation d'une interface

informatique, mais aussi d'autres produits ou objets du quotidien. L'objectif de la conception de l'expérience utilisateur est généralement de trouver comment donner à cet utilisateur une expérience agréable, qui lui permette de réaliser simplement son objectif, et d'apprécier le temps passé à l'accomplir. Pour rendre facilement compréhensible une interface à son utilisateur, le concepteur peut intégrer des indices, qui vont lui faire comprendre comment la manipuler. C'est ce qu'on appelle des *nudges*, ou en français des "coups de coude". Les *nudges* vont des dispositifs les plus simples, comme surligner en couleur le bouton qui permet de passer à la page suivante sur un formulaire web ou faire clignoter une flèche qui indique la prochaine étape, aux plus subtils : par exemple, les tuyaux du jeu vidéo Mario en sont car leur forme incite les joueurs à diriger le personnage vers l'intérieur, et ainsi à accéder à un autre niveau. Ces *nudges* peuvent cependant aussi être utilisés non pour guider l'utilisateur et lui faciliter l'expérience, mais pour le pousser à faire ce qu'on attend de lui. Surligner en vert le bouton qui lui fera accepter tous les cookies d'un site web en laissant les options de personnalisation ou de refus grisées par exemple. Par réflexe, l'utilisateur a beaucoup plus de chances de cliquer sur ce bouton vert, même s'il aurait préféré une autre option. Ou dissimuler le bouton de fermeture d'un écran de publicité en le rendant très petit, pour faire chercher l'utilisateur, qui sera donc exposé plus longtemps au contenu de la publicité, voire qui risque en plus de cliquer sur la publicité et d'ouvrir le lien par erreur alors qu'il essaye de viser la croix pour la fermer. Concevoir une interface, c'est intégrer des dispositifs qui peuvent amener l'utilisateur à apprendre, comme le fait Grasshopper, ou à corriger une erreur, comme Processing et de nombreuses interfaces en ligne de commande cherchent à le faire en le guidant parmi les lignes de code vers l'erreur la plus probable. Mais cela peut aussi être, moins éthiquement, l'amener à se plier à la volonté des concepteurs ou des propriétaires du programme. C'est ce qu'on appelle dans le champ de l'UX le *malevolent design* - le design malveillant[149]. En fonction de sa maîtrise de la programmation et du domaine concerné, et en fonction de la structure de l'interface, l'utilisateur peut donc être plus ou moins à la merci du programmeur.

### *5.3.2 Les interfaces du champ computationnel*

Apprendre à programmer l'architecture, c'est acquérir un savoir-faire qui implique de combiner une connaissance de la programmation à la connaissance de la conception spatiale que les architectes mobilisent usuellement dans leur pratique. Il faut donc savoir utiliser un ordinateur, c'est-à-dire, comme nous l'avons vu au chapitre II, être capable de communiquer des informations à cette machine, pour qu'elle exécute ensuite une série d'instructions et produise le résultat attendu. Dans nos

[149] Colin M. Gray, Yubo Kou, Bryan Battles, Joseph Hoggatt, and Austin L. Toombs, "The Dark (Patterns) Side of UX Design", dans *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. Association for Computing Machinery*, New York, NY, USA, Paper 534, p. 1–14, 2018.

ordinateurs contemporains, c'est l'interface qui permet cette communication. Toutes les familles d'interfaces que nous venons de voir sont en usage dans le champ computationnel, et permettent différentes modalités de communication avec la machine. Elles ne sont cependant pas toutes utilisées au même moment ou pour les mêmes raisons, ni pour les mêmes typologies d'algorithmes. Or, puisque ce sont elles qui se font l'intermédiaire entre praticiens et algorithmes, leur influence sur les modalités de pratique est grande. Nous allons donc examiner de plus près comment les différentes familles d'interfaces se répartissent au sein du champ computationnel, la manipulation des typologies d'algorithmes qu'elles permettent, et les possibilités de transmission que la place qu'elles ont chacune prise dans le champ impliquent.Pour cela, nous allons utiliser une notion d'UX design en particulier : l'épaisseur des interfaces. Celle-ci sert à représenter et à mesurer la distance entre l'intention d'un utilisateur et les instructions reçues par l'ordinateur, distance créée par les multiples couches de programmes entre ce que manipule l'utilisateur à l'écran et les opérations effectuées par l'ordinateur [150]. Dans le champ computationnel, elle permet donc de mesurer la distance entre l'intention architecturale initiale et le projet final.

La figure 32 nous donne d'abord deux exemples de setups techniques détaillés pour un programme. Le premier est celui du pavillon Seroussi tel qu'écrit par EZCT Architecture & Design Research, et le second montre l'infrastructure logicielle nécessaire à la génération d'une géométrie selon les mêmes étapes de conception, mais en la programmant sur Grasshopper avec les plug-ins disponibles aujourd'hui. Le premier faisait appel à un grand nombre de logiciels et de sous-programmes depuis un environnement de programmation principal pour parvenir à proposer à l'aide d'un algorithme génétique et d'un modèle de ciel une répartition de la lumière la plus homogène possible dans l'espace. Le second peut s'appuyer sur les plug-ins Ladybug, Octopus et Karamba 3D pour faire de même et intégrer les motifs structurels que le premier proposait pour les vitrages. La représentation proposée cherche à souligner plus particulièrement deux aspects. En premier lieu, l'épaisseur des interfaces, ou plus exactement leur succession. L'épaisseur du setup auquel sont confrontés les utilisateurs est examinée par le biais d'une représentation sous forme de superpositions : chaque setup technique se compose d'un ensemble d'interfaces de communication différentes, qui transmettent chacune des informations relatives à certains des aspects du programme en cours d'exécution. Sur la figure, elles sont superposées en fonction de l'accès par les utilisateurs à ces différentes interfaces : en haut, les interfaces de visualisation, qui permettent d'observer la géométrie du projet, puis les environnements de programmation principaux, qui permettent d'interagir avec le programme dans son ensemble. Viennent ensuite les logiciels et sous-programmes qui permettent d'obtenir les données

[150] Masure, Anthony. 2014. Le design des programmes : des façons de faire du numérique, Ph.D. Thesis, Université Paris-1 Panthéon Sorbonne.

nécessaires à l'exécution du programme principal, par exemple Radiance, qui permet d'obtenir un modèle de ciel pour étudier la répartition de la lumière. Ensuite, les infrastructures informatiques permettant de faire tourner les différents logiciels : par exemple, dans le cas des modeleurs 3D utilisés pour la visualisation, il existe une première couche d'interface utilisateur en haut, qui est celle utilisée, mais il existe également un programme, qui permet de faire tourner le logiciel, programme représenté plus bas dans l'empilement puisque rarement mobilisé[151]. Enfin, les infrastructures de l'ordinateur qui exécutent le calcul à proprement parler se trouvent tout en bas.

En complément de cet empilement d'interfaces que mobilisent les deux setups techniques, les paramètres principaux des deux programmes sont également recensés. Puisqu'il s'agit d'une combinaison entre un algorithme génétique, une évaluation structurelle et une évaluation lumineuse, le programme regroupe trois modèles de calcul, un par typologie, et trois ensemble de paramètres régissant ce calcul. Ces paramètres sont disponibles à différentes profondeurs en fonction du setup. Dans le cas du programme sur-mesure écrit par EZCT, l'algorithme génétique, au cœur du projet, est programmé dans l'environnement principal, et fait appel aux résultats de l'évaluation structurelle et lumineuse, chacune écrite dans un sous-programme dans un autre environnement de programmation. Ces évaluations font elles même appel à des bibliothèques et logiciels extérieurs. Les paramètres sont donc étalés entre ces différentes infrastructures. La majeure partie des paramètres est néanmoins accessible à la troisième couche, dans l'environnement de programmation principal, puis dans les deux environnements secondaires, aux niveaux 4 et 5. Seul le modèle de ciel, fourni par Radiance, et la méthode de calcul pour l'évaluation structurelle, fournie par une bibliothèque C++ générique d'éléments finis, nécessitent de plonger plus bas dans les infrastructures pour les modifier, aux niveaux 6 et 7. Dans le cas du programme prêt à l'emploi reconstitué dans Grasshopper, les paramètres régissant le calcul sont eux aussi presque tous accessibles au niveau 3, dans l'interface de programmation visuelle du logiciel. Le paramètre de sélection finale, lui, se trouve néanmoins un cran plus haut, dans l'interface utilisateur d'Octopus. Les méthodes de calcul des trois typologies et le modèle de ciel se trouvent dans différentes couches, tous plus bas : aux niveaux 5, 6, 7 et 8.

[151] Peu de praticiens vont en cours d'élaboration de leur algorithme modifier le kernel d'un modeleur.

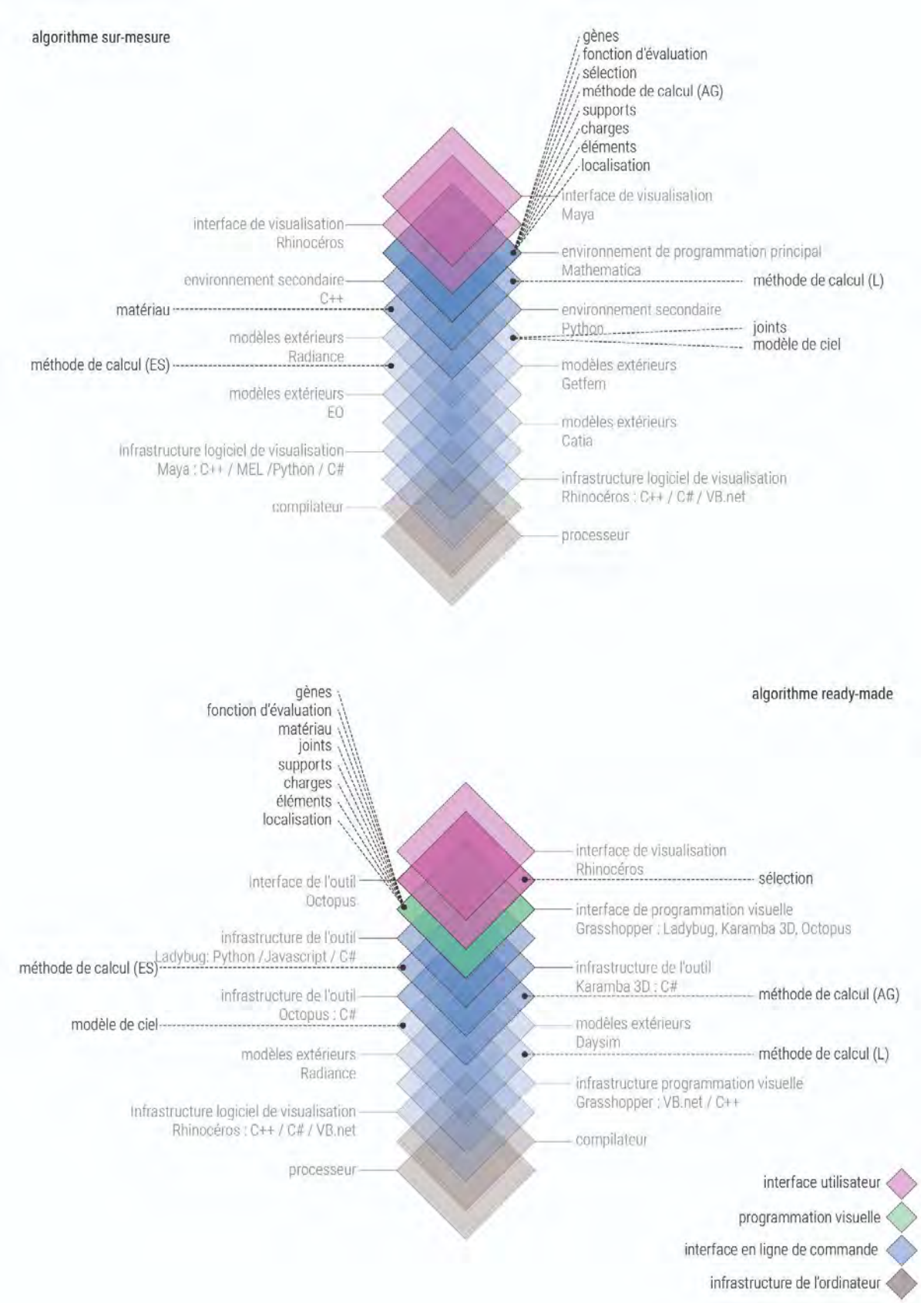

algorithme sur-mesure
gènes
fonction d'évaluation
sélection
méthode de calcul (AG)
supports
charges
éléments
localisation
interface de visualisation
Maya
interface de visualisation
Rhinocéros
environnement de programmation principal
Mathematica
environnement secondaire
C++
méthode de calcul (L)
matériau
environnement secondaire
Python
joints
modèle de ciel
modèles extérieurs
Radiance
méthode de calcul (ES)
modèles extérieurs
Getfem
modèles extérieurs
EO
modèles extérieurs
Catia
infrastructure logiciel de visualisation
Maya : C++ / MEL /Python / C#
infrastructure logiciel de visualisation
Rhinocéros : C++ / C# / VB.net
compilateur
processeur
algorithme ready-made
gènes
fonction d'évaluation
matériau
joints
supports
charges
éléments
localisation
interface de visualisation
Rhinocéros
sélection
Interface de l'outil
Octopus
interface de programmation visuelle
Grasshopper : Ladybug, Karamba 3D, Octopus
infrastructure de l'outil
Ladybug: Python /Javascript / C#
méthode de calcul (ES)
infrastructure de l'outil
Karamba 3D : C#
méthode de calcul (AG)
infrastructure de l'outil
Octopus : C#
modèle de ciel
modèles extérieurs
Daysim
méthode de calcul (L)
modèles extérieurs
Radiance
infrastructure programmation visuelle
Grasshopper : VB.net / C++
Infrastructure logiciel de visualisation
Rhinocéros : C++ / C# / VB.net
compilateur
processeur
interface utilisateur
programmation visuelle
interface en ligne de commande
infrastructure de l'ordinateur

*Figure 32. Structure d'interface pour deux exemples d'algorithmes - prêt à l'emploi et sur-mesure*

L'épaisseur des deux setups techniques est à la fois conséquente et comparable - treize niveaux pour l'algorithme sur-mesure, douze pour celui sur Grasshopper. Il y a cependant une différence dans les types d'interfaces utilisées : les interfaces utilisateur de la version sur-mesure ne sont utilisées que pour la visualisation du résultat, quand l'une d'entre elles est utilisée pour le paramètre de sélection dans la version prête à l'emploi. L'interface principale de ce dernier est celle de programmation visuelle, quand ce sont directement des interfaces en ligne de commande dans la version sur-mesure. Le parcours de l'utilisateur à travers ces différentes couches diffère, et en conséquence l'accès aux paramètres est également modifié. Les paramètres qui deviennent moins accessibles car ils sont placés plus bas dans le setup prêt à l'emploi que dans le setup sur-mesure sont parmi les plus importants, puisque ce sont les méthodes de calcul, qui permettent d'ajuster précisément la manière dont la forme du projet est générée et la manière dont il sera évalué. Valider les gènes et la fonction évaluation plutôt que de les définir en partant de zéro pose le même problème - la position des paramètres au sein du setup joue donc beaucoup sur la maîtrise du programme par l'utilisateur. De la même manière, modifier les règles de comportement des particules d'un système multi-agent, et non simplement les variables qui régissent ces comportements ne permet pas d'avoir le même impact sur le déplacement des particules et sur le projet qui se construit à partir de ce déplacement. Le projet Living Morphologies et son équivalent dans Grasshopper nous donneraient justement une comparaison similaire s'ils étaient représentés en suivant le même principe. Simplement, plutôt que de montrer un set-up permettant de combiner un algorithme génétique, une évaluation structurelle et une évaluation de la lumière, cela montrerait la combinaison d'un système multi-agents, d'une évaluation structurelle et d'une évaluation de la lumière. La version contemporaine sur Grasshopper ferait donc appel à Ladybug et à Karamba 3D, et à Quelea plutôt qu'à Octopus. Les paramètres clés de l'algorithme génétique seraient remplacés par ceux des systèmes multi-agents, mais la comparaison entre le setup sur-mesure et le setup prêt à l'emploi révèlerait la même chute des paramètres clés - la description du comportement des agents - dans les profondeurs.

C'est ce que la figure 33 permet d'examiner, en mettant en parallèle une série d'exemples pour chaque typologie d'algorithmes et pour chaque famille d'interface - plus exactement pour la famille d'interface qui corresponds à la couche principale avec laquelle l'utilisateur interagit dans le setup technique décrit. Le même système de représentation par empilement et le même code couleur que précédemment sont employés, et les points présents à côté de chacun des setups permettent de voir où se trouvent les différents paramètres de la typologie examinée. La figure précédente présentait une structure d'interfaces permettant de visualiser un mélange de plusieurs typologies algorithmiques au

sein d'un même programme, celle-ci permet de voir ce qu'implique le recours à une seule typologie. Ce système de représentation est conçu pour montrer les circulations verticales d'un utilisateur entre les différents niveaux du set-up. Plutôt que d'examiner la circulation horizontale d'un logiciel à l'autre au cours du flux de travail, que nous observerons plus bas, il s'agit d'observer la hiérarchie des éléments d'interface du setup, des plus utilisés à celles qui sont reléguées en arrière-plan. Les plus simples ne comptent que deux couches utilisées : une interface de visualisation et un environnement de programmation au sein duquel tout le travail est accompli. C'est le cas des exemples donnés pour l'interface en ligne de commande dans le cas des simulations physiques et des systèmes multi-agents. D'autres impliquent un environnement secondaire de programmation, ou une infrastructure pour le logiciel de visualisation qui n'est pas la même que l'environnement principal de programmation, par exemple pour les interfaces en ligne de commande dans le cas des systèmes de croissance ou des évaluations structurelles. Cette infrastructure est alors reléguée plus bas dans le diagramme puisqu'il n'est pas nécessaire d'aller y faire des modifications : elle permet simplement d'afficher les résultats de l'exécution du programme. Dans ces cas, l'interface de visualisation n'est rien de plus qu'un moyen d'affichage, d'une 3D du projet, de diagrammes intermédiaires ou encore simplement d'une série de coupes. C'est le cas par exemple pour Trabeculae / Photosynthesis, le projet de supermanoeuvre pris comme exemple d'utilisation d'une interface en ligne de commande pour la programmation d'un système multi-agents, ou les séries de coupes résultant de l'exécution du programme étaient assemblées dans un second temps pour former la 3D finale. Dans d'autres cas, l'interface de visualisation est aussi celle de l'outil, c'est-à-dire qu'elle permet non seulement d'afficher la géométrie du projet, mais également de manipuler certains paramètres de l'algorithme. C'est ce qui se passe dans Cuckatoo, exemple d'utilisation d'une interface utilisateur pour de la simulation physique : l'utilisateur peut indiquer le maillage, les frontières et les points d'ancrage dans Rhinocéros, puis y observer la relaxation du système. Même principe dans le cas du pavillon Turing de biothing, ou l'interface se divise entre la fenêtre de programmation de Processing, et sa fenêtre de visualisation, ou l'utilisateur a accès à une partie des paramètres pour adapter l'équation de réaction-diffusion sur laquelle est basée le projet. Les plus complexes de ces circulations verticales comptent jusqu'à huit niveaux successifs, comme l'exemple montré d'utilisation d'une interface utilisateur pour un algorithme génétique. Le solveur de Revit qui est montré est conçu pour faciliter le plus la tâche à un utilisateur et faire en sorte qu'il n'ait jamais besoin de programmer la moindre instruction, mais simplement d'utiliser des fonctionnalités classiques de logiciel - il n'est donc confronté qu'à une série de fenêtres permettant de sélectionner des options pour les paramètres de l'optimisation. Cette figure montre aussi la variété des set-ups existants, avec cinq configurations de niveaux différents et six configurations de répartition des paramètres. Un point commun cependant entre ces cas de figure : les paramètres sont tous sur la même couche lorsqu'il s'agit d'interfaces en ligne de commande, alors

qu'ils se divisent sur deux ou trois couches différentes lorsque ce sont des interfaces de programmation visuelle ou des interfaces utilisateur. Et comme l'exemple détaillé abordé plus haut le montre, ce sont souvent les paramètres les plus décisifs : programmation du comportement des agents, méthode d'évaluation des générations ou encore méthode de calcul des simulations physiques[152].

Les différents cas de figure présentés donnent à voir une variété d'épaisseurs des set-ups, quelque soit la famille d'interfaces concernées. Aux interfaces en ligne de commande correspondent cependant des setups globalement peu épais, car elles permettent de construire le programme dans un environnement de programmation principal et flexible, qui offre une grande variété de possibilités : on peut coder un algorithme génétique, un L-système ou un système multi-agent aussi bien en Python qu'en C++. Les interfaces de programmation visuelle, elles, sont systématiquement dotées d'un plus grand nombre de couches, car elles combinent toujours au moins une interface de visualisation, l'interface de programmation visuelle dans laquelle l'utilisateur évolue, et une interface en ligne de commande qui permet à l'interface de programmation visuelle de fonctionner. Les interfaces utilisateurs, enfin, se partagent entre des systèmes combinant une interface de visualisation-utilisation et un environnement de programmation principal la régissant, et des systèmes combinant de multiples logiciels, dont l'accès est rassemblé à un endroit, et leur épaisseur varie en fonction du dispositif mis en place. Comme on le voit sur la figure 34, dans les cas de figures ou elles cumulent le plus de niveaux, elles deviennent les plus épaisses des trois familles. Dans les outils du champ computationnel, c'est souvent ce qui se produit : plus on évolue dans un set-up qui favorise une interface utilisateur ou une interface en programmation visuelle, plus son épaisseur est grande, et moins l'utilisateur a la main sur l'ensemble des paramètres qui régissent la typologie à laquelle il cherche à avoir recours. Alors que les algorithmes personnalisés qu'offre la programmation via les interfaces en ligne de commande permettent un accès aux règles dans les couches supérieures et sont composés d'un petit nombre de couches, les outils prêts à l'emploi sont composés de nombreuses couches et l'accès à la programmation de règles spécifiques est situé dans les couches inférieures, plus difficiles d'accès pour les utilisateurs ayant peu de compétences en programmation.

[152] Florian Chéraud, "Beyond Design Freedom Providing a Set-Up For Material Modelling within Kangaroo Physics", September 2020, Conference: eCAADe 2020 At: BerlinVolume: 1.

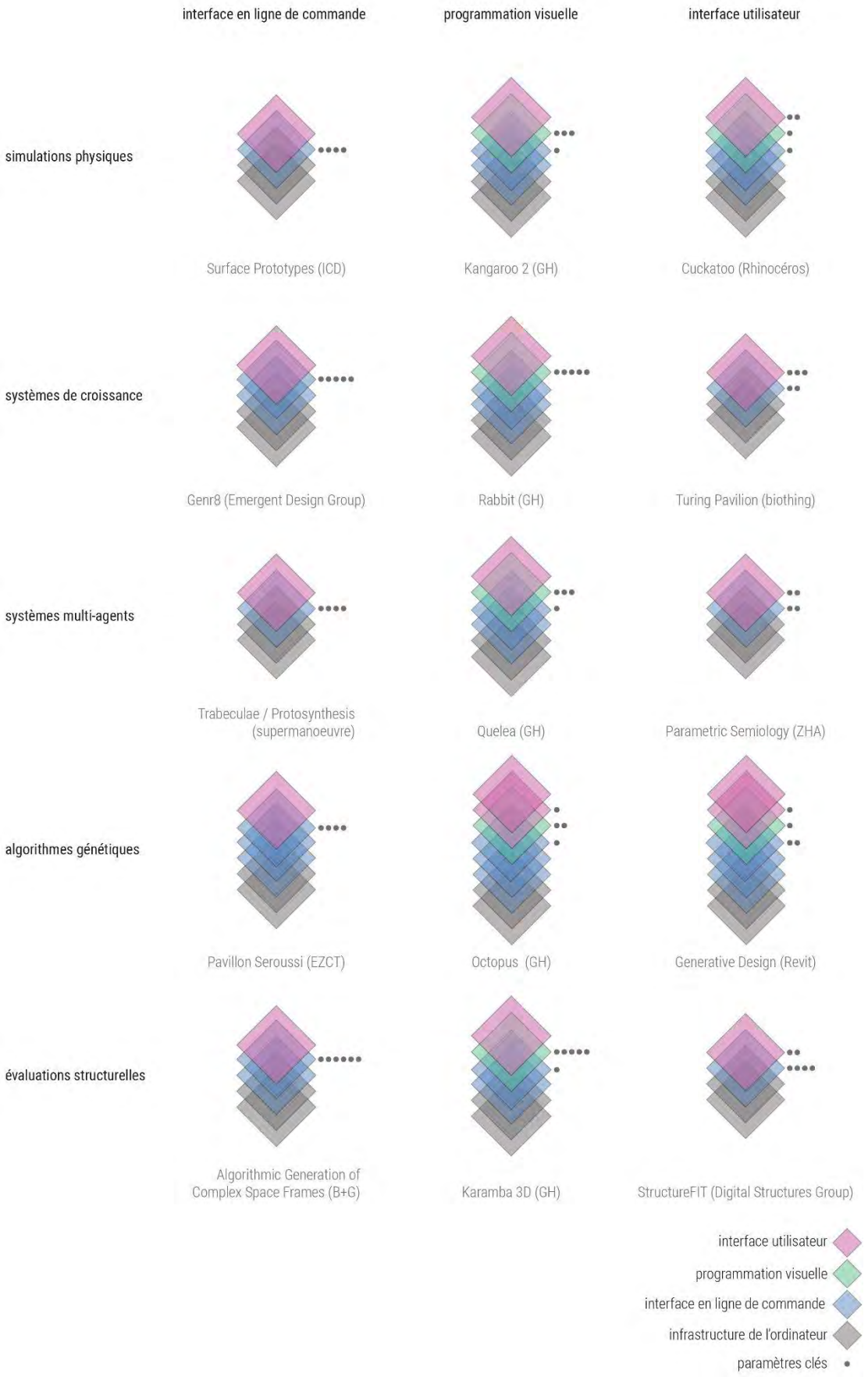

interface en ligne de commande
programmation visuelle
interface utilisateur
simulations physiques
Surface Prototypes (ICD)
Kangaroo 2 (GH)
Cuckatoo (Rhinocéros)
systèmes de croissance
Genr8 (Emergent Design Group)
Rabbit (GH)
Turing Pavilion (biothing)
systèmes multi-agents
Trabeculae / Protosynthesis (supermanoeuvre)
Quelea (GH)
Parametric Semiology (ZHA)
algorithmes génétiques
Pavillon Seroussi (EZCT)
Octopus (GH)
Generative Design (Revit)
évaluations structurelles
Algorithmic Generation of Complex Space Frames (B+G)
Karamba 3D (GH)
StructureFIT (Digital Structures Group)
interface utilisateur
programmation visuelle
interface en ligne de commande
infrastructure de l'ordinateur
paramètres clés

*Figure 33. Exemples de structures d'interface par type d'interface et par typologie*

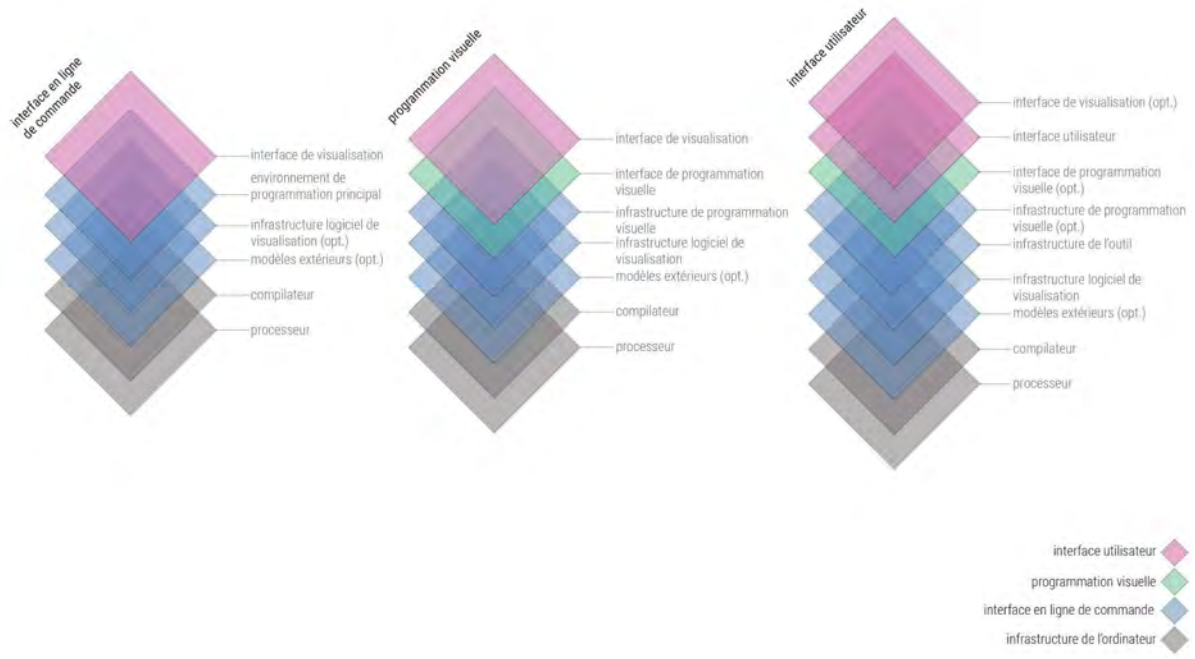



*Figure 34 - Structures d'interfaces types*

Cette représentation en couches d'interfaces a néanmoins une limite : si elle permet d'examiner la succession des interfaces devant lesquelles l'utilisateur est mis au fur et à mesure qu'il souhaite intervenir sur les différents paramètres de l'algorithme, elle fausse néanmoins légèrement la vision du flux de travail tel que accompli au cours du temps pendant la réalisation des projets. Les figures 35 et 36 permettent d'observer deux flux de travail tels que représentés par les auteurs de chaque projet, le pavillon Seroussi et le projet Associative Components Structure. Ces diagrammes donnent à voir la succession des étapes temporelles et donc la succession des interfaces auxquelles l'utilisateur sera forcément confronté au cours de l'exécution du programme, c'est-à-dire les circulations horizontales plutôt que verticales au sein du setup. Dans le cas du Pavillon Seroussi, le diagramme proposé par EZCT montre les multiples interfaces que le setup imposait alors pour mettre en place toutes les étapes de génération de la géométrie du projet. Ce sont cependant des praticiens différents qui y accèdent. Les trois environnements de programmation vus dans le diagramme en couches de l'interface correspondent au travail de trois praticiens distincts, chacun spécialisé dans un champ (analyse structurelle, analyse lumineuse et conception architecturale) et chacun maîtrisant suffisamment la programmation pour être à même d'écrire un programme traitant d'un de ces aspects.

La représentation proposée par l'agence met aussi l'accent sur la diversité des outils plutôt que sur la fréquence de recours au cours du flux de travail et met de ce fait les modèles extérieurs sur le même plan que les environnements de programmation. Du fait de cette absence de hiérarchisation, les circulations horizontales qui ont lieu au cours de l'écriture puis dans une moindre mesure lors de l'utilisation du programme sont bien visibles. Ce qui apparaît notamment est la méthode de travail qui consiste à faire appel à d'autres éléments depuis un environnement principal ; ces sauts d'un environnement à l'autre sont justement ce qui disparaît de l'expérience utilisateur dans l'équivalent sur Grasshopper décrit plus haut. L'utilisateur reste dans un seul environnement, avec une à deux interfaces - dans ce cas-ci deux, l'interface utilisateur de Rhinocéros pour la visualisation et l'interface de programmation visuelle de Grasshopper pour la manipulation du programme[153]. Les paramètres dont il a été vu plus haut qu'ils sont désormais dans des couches plus profondes ne sont donc plus accessibles dans cet environnement unique. Ce sont justement des paramètres automatisés par la programmation en amont de ce programme prêt à l'emploi, en préparation de son utilisation par un utilisateur tiers. Il n'y a plus besoin d'y accéder pour faire fonctionner le programme, quand bien même ce sont potentiellement des paramètres qui influent beaucoup sur le résultat. La figure 36 montre un flux de travail similaire, avec l'utilisation successive de plusieurs logiciels et une circulation horizontale au sein du setup. Il s'agit du flux de travail du projet Associative Component Structure, réalisé en 2005 par Giannis Douridas et Mattia Gambardella lors de leur passage au sein du master de la AA Emergent Technologies and Design, sous la direction de Achim Menges et Michael Hensel[154]. L'objectif est de générer une surface au hasard, puis de la décomposer en pans quadrangulaires. Ces pans sont ensuite transformés en un ensemble de membranes à simple et double courbure, maintenues ensemble par une série de cadres en bois. La modélisation se fait donc dans un objectif de fabrication, et intègre des mesures réalisées sur des prototypes physiques qui permettent d'ajuster la génération des membranes à des variables liées au matériau sélectionné. Le flux comme le projet sont beaucoup plus simples que dans le pavillon Seroussi, mais l'exemple permet de voir, même pour un travail de géométrie aussi simple, les sauts d'un environnement à l'autre : plusieurs modules assemblés au sein de Generative Components, visualisation dans une seconde interface, import dans un logiciel CAD, extraction des coordonnées et enfin fabrication. Ces sauts et le travail de gestion de l'interopérabilité qu'ils imposent aux praticiens sont fréquents dans les pratiques du champ computationnel avant l'apparition des interfaces de programmation visuelle de Grasshopper et Dynamo, qui vont changer les pratiques d'utilisation. La simplification de l'expérience utilisateur qu'elles permettent entraîne une augmentation du nombre d'utilisateurs, et l'apprentissage de la

[153] Dans certains cas une seule puisque la visualisation et la manipulation se font dans une interface utilisateur commune.

[154] Hensel, M., Menges, A.: 2006, Material and Digital Design Synthesis: Integrating Material Self-Organisation, Digital Morphogenesis, Associative Parametric Modelling and Computer-Aided Manufacturing, Architectural Design, Vol. 76 No. 2, pp. 88-97

conception procédurale qu'elles facilitent va contribuer à l'extension du champ computationnel, mais la nouvelle répartition des paramètres qu'elles promeuvent modifie en profondeur les pratiques.

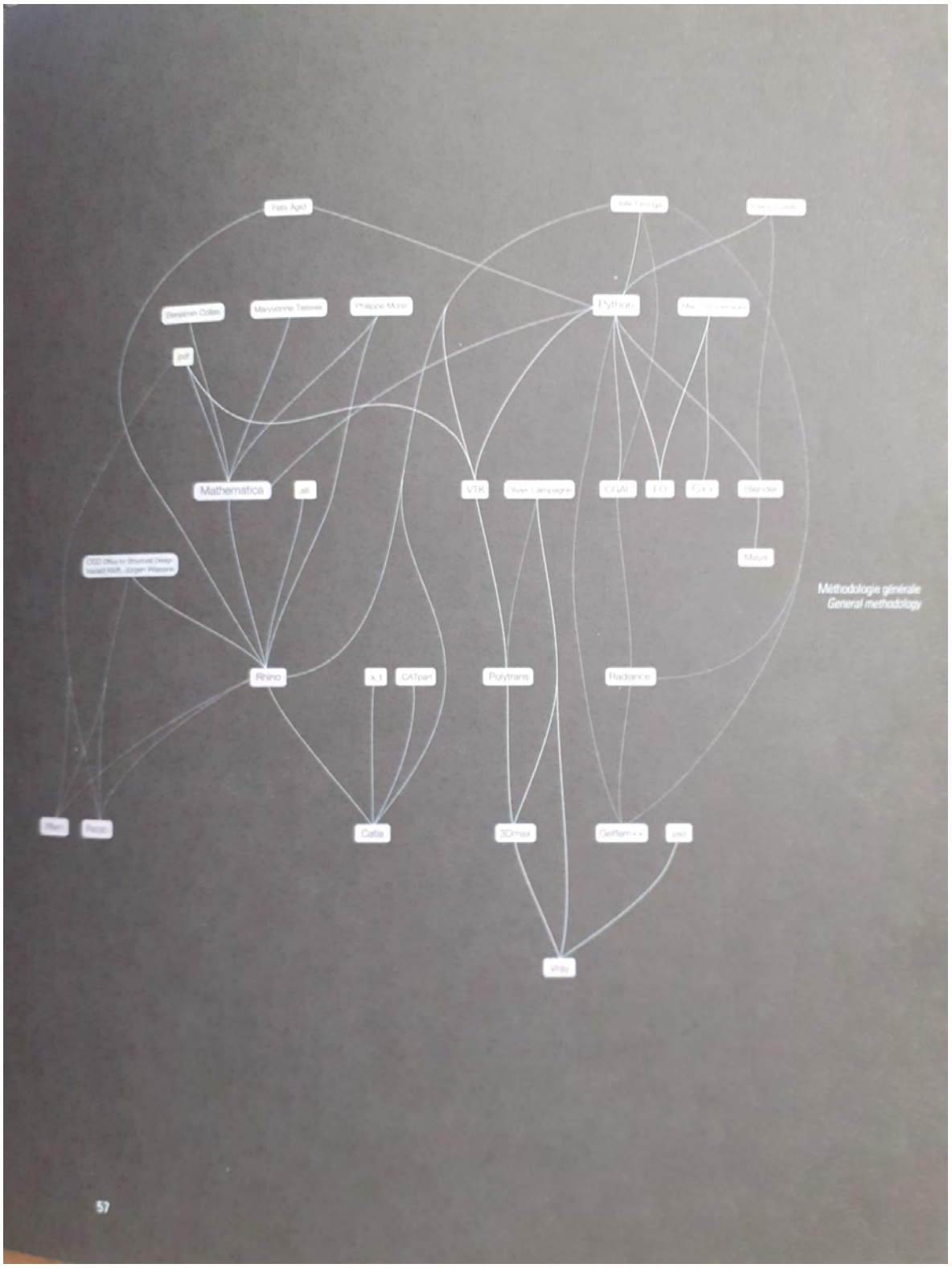


*Figure 35. Flux de travail du projet du Pavillon Seroussi*[155]

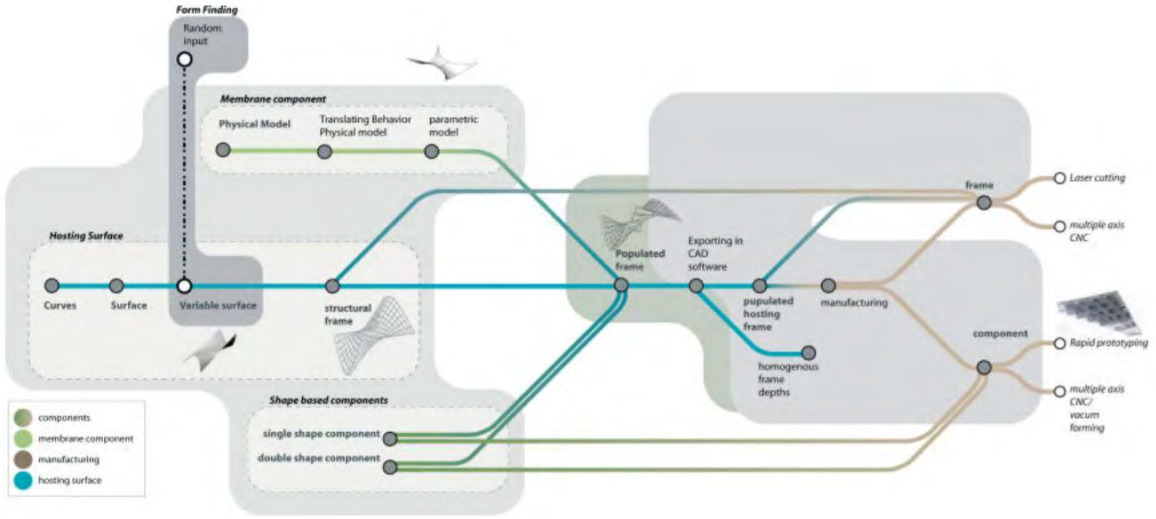



*Figure 36. Flux de travail du projet Associative Component Structure*[156]

### *5.3.3 Évolution chronologique*

En fonction des interfaces, l'utilisateur n'a pas les mêmes possibilités, et leur disponibilité influence donc les pratiques du champ computationnel. Pour mieux comprendre les évolutions récentes du champ, il faut donc se pencher sur la chronologie de développement des outils en fonction de leurs familles d'interfaces et de leur typologie. La figure 38 montre une chronologie globale des outils par familles d'interfaces. On y observe l'accumulation, au fil du temps, d'interfaces en ligne de commande généralistes, puis de premières interfaces utilisateurs notamment avec le développement des modeleurs 3D, puis d'interfaces en ligne de commande spécialisées. La phase la plus récente est caractérisée par une explosion des interfaces de programmation visuelle, mais aussi par le début de réapparition d'interfaces utilisateurs. Les premières apparitions d'interfaces utilisateur correspondent aux logiciels de modélisation 3D, mais pas seulement. C'est ce qu'on peut voir sur la figure 37, qui permet aussi d'observer la variation des types d'interfaces disponibles au cours du temps en fonction des typologies. Dans les années 2000, avant l'apparition d'interfaces de programmation visuelle,

[155] E. Guenoun (ed.): 2007, Pavillon Seroussi : biothing, DORA (Design Office for Research and Architecture), EZCT Architecture & Design Research, Gramazio & Kohler, IJP – George L. Legendre, Xefirotarch, HYX.
[156] Hensel, M., Menges, A.: 2006, Material and Digital Design Synthesis: Integrating Material Self-Organisation, Digital Morphogenesis, Associative Parametric Modelling and Computer-Aided Manufacturing, Architectural Design, Vol. 76 No. 2, pp. 88-97

nombre de scripts sont conçus avec une interface utilisateur, pour permettre à des étudiants ou des collègues d'en manipuler facilement les paramètres : c'est le cas pour plusieurs des programmes de biothing, de kokkugia ou de supermanoeuvre. Ce sont néanmoins des outils qui restent cantonnés à un petit cercle de praticiens - au sein d'une agence, d'un groupe de recherche ou d'un studio de master[157]. Nombre de ces objets à mi-chemin entre programme sur-mesure et outil transmissible ne sortent pas des mains des praticiens faisant le projet. C'est le cas par exemple pour le système de Seroussi ou pour celui d'Adaptative Component Structure, que nous avons décrit juste avant et qui n'ont jamais été transmis à d'autres. Il ne sont donc pas comptabilisés dans ces chronologies des outils du champ computationnel, puisqu'ils conservent un statut de programme sur-mesure. Les langages de programmation et logiciels modélisation sont par ailleurs essentiellement là pour servir de point de repère, et ne sont donc pas une liste exhaustive des sorties de langages et logiciels dans le domaine. Ceux recensés dans le corpus comme utilisés par le champ computationnel sont néanmoins indiqués. Quelques exemples des langages de programmation généralistes pris en compte : C++, Python, Java. Quelques exemples pour la programmation spécialisée : Generative Components, Dynamo, Grasshopper, Processing. Quelques exemples pour la modélisation : CATIA, Form*Z; Rhino, Revit, AutoCAD, mais aussi par exemple ARCHIMODOS, une des réalisations précédentes de Chris Yessios, mentionnée au chapitre II et en usage dans toute la faculté d'architecture de Ohio State University à la suite de son développement. Les rectangles vides indiquent les outils développés hors champ computationnel, mais néanmoins à disposition des praticiens. Aucun usage de ces outils par les praticiens n'a été recensé dans le corpus, mais les praticiens auraient pu si souhaité y recourir, ils sont donc intégrés comme points de repère ; les systèmes multi-agents notamment comptent de nombreux frameworks développés pour l'industrie de l'animation qui auraient pu être mobilisés, comme MASSIVE ou Golaem Crowd. Les systèmes de croissance eux aussi offrent une grande diversité de logiciels disponibles, existant en raison des pratiques artistiques nombreuses qui se basent sur le recours à cet ensemble d'algorithmes. Leur programmation est cependant suffisamment simple pour être faite directement dans un environnement généraliste ou spécialisé directement par les praticiens aguerris du champ computationnel qui ne recourent donc pas vraiment à ces logiciels. Les algorithmes génétiques, eux, disposent de très peu d'outils dédiés antérieurs aux pratiques du champ computationnel pour une raison proche : leur programmation se fait dans des environnements de programmation généralistes.

[157] Parmi ces outils expérimentaux dans des cadres restreints, seuls sont présents sur les figures ceux repérés au cours du travail de recherche et qui ont été mentionnés dans le texte. S'il en existe certainement d'autres, ils ne sont pas forcément identifiés dans les archives consultées. Il subsiste par ailleurs parfois un doute concernant certains programmes, puisque l'expression "telle typologie comme outil de conception" est très répandue dans les textes qui accompagnent les projets, bien que l'outil en question ne soit en fait souvent rien de plus qu'un programme sur mesure.

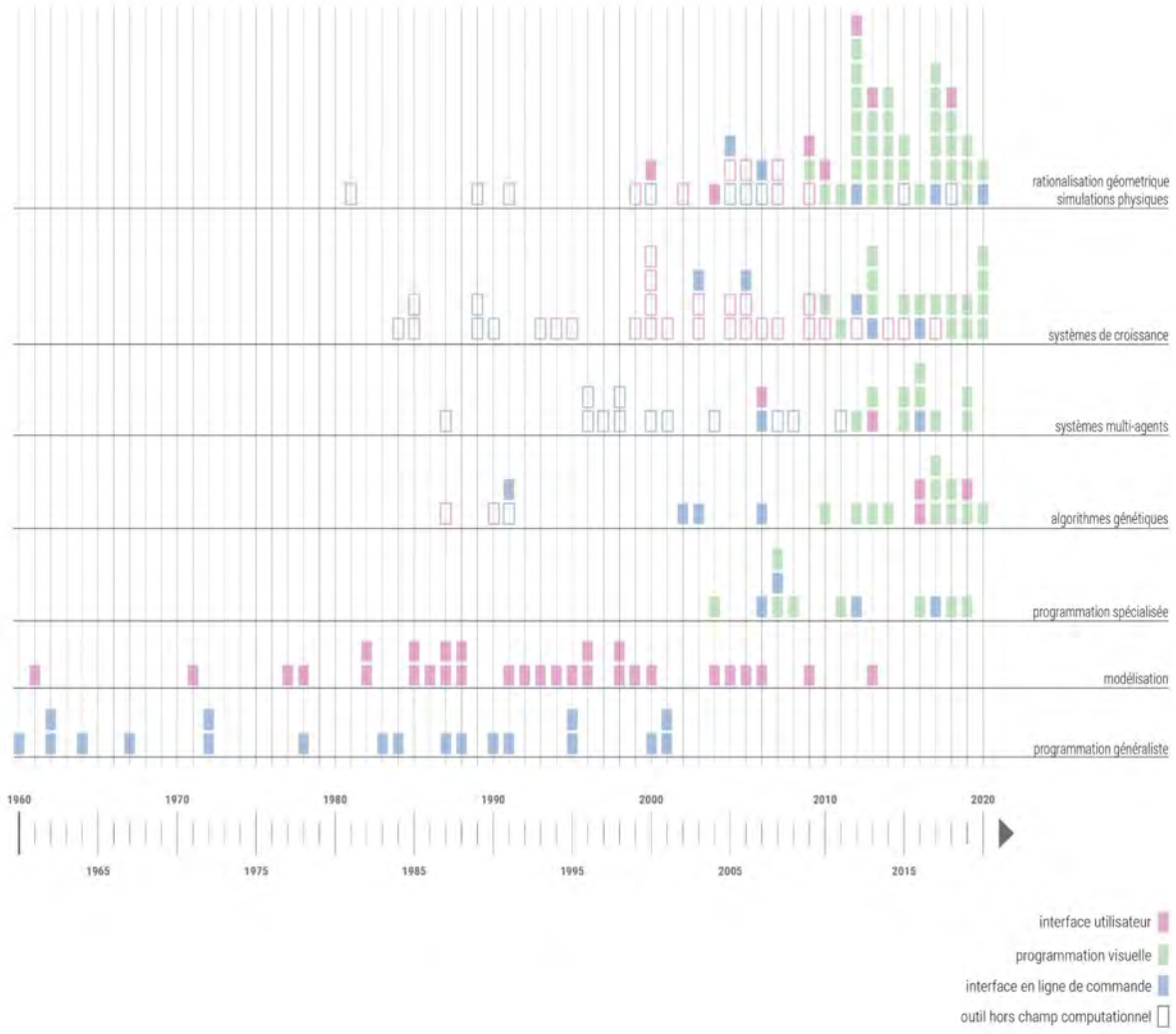



*Figure 37. Chronologie des apparitions d'interfaces par typologies*

Cette variation dans les outils disponibles en fonction des typologies se faisait déjà sentir dans la comparaison des différents setups montrée plus haut, ou l'épaisseur était comparable dans les familles d'interfaces en ligne de commande et d'interfaces de programmation visuelle, mais beaucoup plus variable d'une typologie à l'autre dans les interfaces utilisateur. Ces différentes dynamiques entre les familles s'observent aussi dans le nombre d'utilisateurs pour chaque typologie, présenté sur la figure 39. Celle-ci montre plus précisément l'évolution au fil du temps du nombre de téléchargements de plugins Grasshopper par typologie, avec la sortie des différents outils[158]. Ces nombres sont traduits

[158] Ceci ne nous donne pas exactement le nombre d'utilisateurs par typologies, très difficile à estimer, mais nous fournit néanmoins un aperçu de leur évolution.

ensuite en un pourcentage des téléchargements par typologie d'algorithme, qui indique la répartition des utilisateurs entre elles d'une année sur l'autre. Si les chiffres donnés sur cette dernière figure sont analysés par objectif de conception plutôt que par typologies, les trois sont équivalents : génération de forme (systèmes multi-agents, systèmes de croissance, simulations physiques), optimisation (algorithmes génétiques, outils d'évaluation) et constructibilité (rationalisation géométrique) s'équilibrent dans les usages. Mais la répartition par typologies montre que certaines sont plus présentes que d'autres en fonction des périodes et des types d'interfaces.

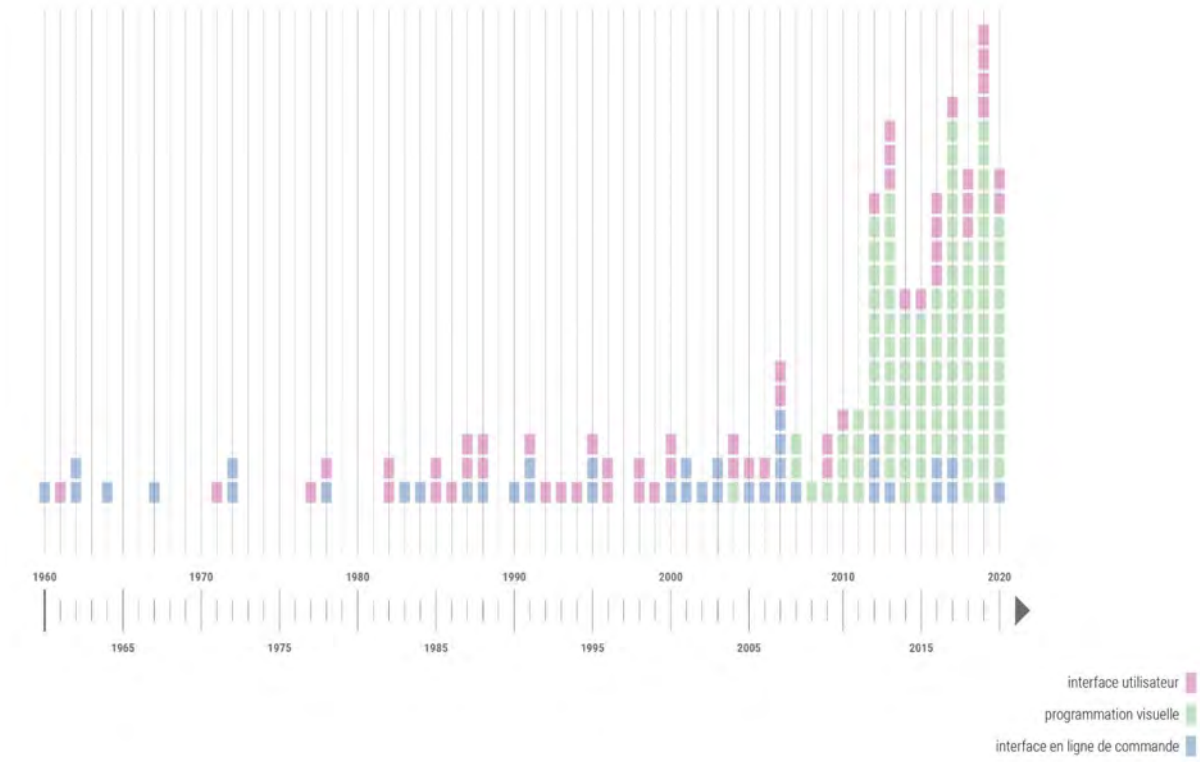
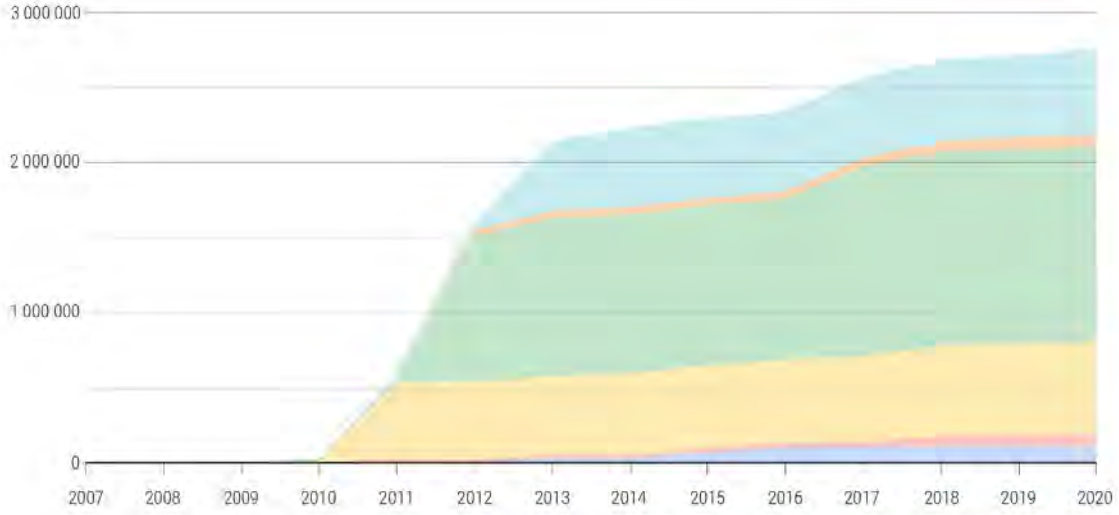


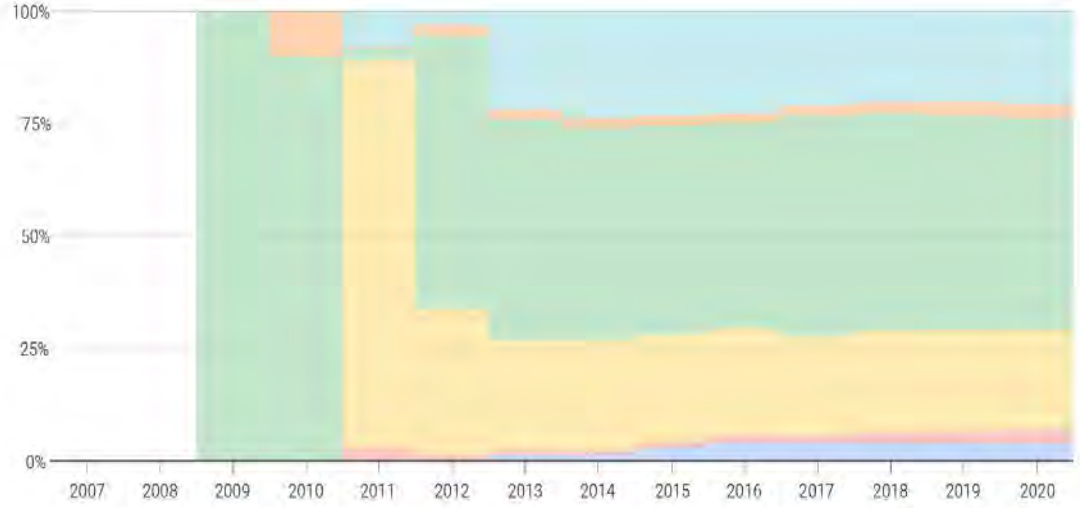

*Figure 38. Chronologie cumulée des apparitions d'interfaces*

Les outils de simulation physique et de rationalisation géométriques comptent ainsi beaucoup de nouveaux outils. Concernant les premiers, non seulement il s'agit d'une typologie ludique, assez facile à manipuler pour obtenir des résultats formels rapidement, mais c'est aussi une typologie qui permet de produire des formes plus aisément constructibles que celles résultant d'autres typologies. C'est aussi le cas des seconds, pour lesquels on observe une très forte progression, en même temps que les projets du champ computationnel passent de prototypes ou projets spéculatifs à des applications construites. Ces constructions sont d'abord des projets d'exception, mais gagnent aussi petit à petit la sphère des projets plus conventionnels, comme en attestent le grand nombre de façades conçues à

l’aide de modèles paramétriques s’appuyant eux-mêmes sur des algorithmes de rationalisation géométrique. Au cours du temps, les systèmes de croissance tendent en revanche à être moins fréquemment utilisés par les praticiens expérimentés : les projets produits par les praticiens dont nous avons montré le réseau en début de chapitre recourent à d’autres typologies algorithmiques. La figure 39 ne montre néanmoins pas de chute dans les téléchargements. Les systèmes de croissance restent en usage notamment parce que c’est une typologie qui se prête facilement à l’exploration par des débutants. C’est ce que nous avons vu à travers le travail de John Frazer, de Paul Coates ou de Marjan Colletti avec leurs étudiants. Les systèmes de croissances sont simples à implémenter et à combiner avec d’autres typologies, et offrent à partir d’un set-up assez simple une richesse formelle facile à exploiter, ce qui explique leur popularité au départ. Cependant, on observe tout de même dans le recours aux systèmes de croissance au sein des projets du champ une certaine perte d'intérêt des praticiens les plus chevronnés, qui se tournent peu à peu vers d’autres typologies. Ainsi, même si les systèmes de croissance sont une typologie toujours en usage, on observe un déplacement de ces utilisations. Le recours aux algorithmes génétiques est aussi le lieu d’un changement d'usage. Plus exactement, d’un changement d’outils. En effet, si les figures montrent que les algorithmes génétiques voient peu de nouveaux outils être développés, et peu d’augmentation dans les téléchargements, cela s’explique par deux éléments. D’abord, le fait qu’ils sont souvent programmés dans des environnements généralistes, ou dans l’un des deux plugins principaux de Grasshopper : Octopus et Galapagos. Or Galapagos est intégré à Grasshopper directement en même temps que Grasshopper est intégré à Rhinocéros. À partir de la, les algorithmes génétiques sont donc disponibles automatiquement à tout utilisateur de Rhinocéros, sans besoin d’aller télécharger un autre plugin. Or les projets qui ont recours à des algorithmes génétiques sont en constante augmentation, en raison d’un changement dans les pratiques que nous examinerons plus en détail au chapitres VI et VII. Ceci nous fournira un point de comparaison avec les systèmes multi-agents, une typologie qui voit le nombre de projets qui y ont recours diminuer, pendant que le recours aux outils associés stagne malgré le développement de nouveaux plug-ins ayant vocation à rendre les systèmes multi-agents plus faciles à utiliser. Avant d’examiner dans les mécanismes logiciels qui contribuent à la popularisation de certaines typologies, nous allons nous pencher plus en détail sur le cas des systèmes multi-agents, et proposer une explication à la perte de vitesse de cette typologie dans le champ computationnel.




évolution du nombre de téléchargements par familles d'algorithmes




pourcentage des téléchargements par familles d'algorithmes

10 plug-ins les plus téléchargés(1)

1 - Lunchbox (RG) : 538 191 tl
2 - Kangaroo 2 (SP) : 516 403 tl
3 - Ladybug (EV) : 419 231 tl
4 - Paneling Tools (RG) : 225 801 tl
5 - MeshEdit (RG) : 160 724 tl
6 - Pufferfish (RG) : 158 718 tl
7 - Mesh+ (RG) : 78 565 tl
8 - Karamba 3D (EV) : 64 895 tl
9 - Geco (EV) : 47 050 tl
10 - Octopus (AG) : 38 492 tl

- rationalisation géométrique : 1 295 951 téléchargements
- simulations physiques : 617 001 téléchargements
- évaluations : 570 586 téléchargements
- systèmes multi-agents : 105 052 téléchargements
- algorithmes génétiques : 87 714 téléchargements
- systèmes de croissance : 84 790 téléchargements

(1) Au sein des typologies examinées. L'ordre complet est le suivant : Lunchbox, Kangaroo 2, Ladybug, GhPython, Paneling Tools, Elk, MeshEdit, Pufferfish, Human, Anemone, Mesh+, Bifocal, Karamba 3D, TT Toolbox, VisualARQ, Elefront, Firefly, Hoopsnake, Heteroptera, Geco, RhinoPolyhedra, gHowl, HumanUI, shortestWalk, Fabtools, Octopus. Les autres plug-ins présents dans la liste sont des outils d'interface, d'optique, d'électronique, de BIM, des aides à la programmation ou des composants ponctuels.

*Figure 39. Chronologie des apparitions de plug-ins par typologies*

Pour commencer, quels chiffres permettent de dire que les systèmes multi-agents sont une typologie moins usitée que les autres. et de moins en moins utilisée ? Concernant l'évolution des outils disponibles, nous avons vu que le petit nombre d'outils dédiés aux systèmes multi-agents est dû notamment au recours à des environnements de programmation généralistes et à des outils personnels conservés dans de petits réseaux clos. Il faut tout de même noter qu'en regard de l'explosion de la programmation visuelle, peu de plug-ins dédiés aux systèmes multi-agents voient le jour. De plus, malgré les quelques plug-ins tout de même développés, le nombre de téléchargement ne décolle pas du tout. Par ailleurs, il faut noter qu'il n'existe pas de package Dynamo permettant de recourir aux systèmes multi-agents, alors qu'il en existe pour toutes les autres typologies. Un autre point de comparaison est le nombre de téléchargements du plugin le plus usité par typologie, ou les systèmes multi-agents se retrouvent bons derniers. Les huit plugins Grasshopper permettant l'utilisation de systèmes multi-agents affichent un nombre cumulé de 105 052 téléchargements, soit une moyenne de 13 131 par plugin - Lunchbox, le plugin Grasshopper le plus téléchargé, affiche un total de 538 191 téléchargements. Si cela laisse déjà entrevoir une petite communauté d'utilisateurs, les membres actifs des groupes Grasshopper correspondants ne représentent que moins de 2% de ce nombre, de même que les articles publiés sur le sujet. Malgré la difficulté d'évaluer correctement le nombre d'utilisateurs réguliers de ces typologies algorithmiques, ces chiffres mettent en évidence un élément clé les concernant : la différenciation entre les utilisations superficielles et extensives. Ces chiffres suggèrent que, si la présence de plugins Grasshopper laisse entrevoir des tentatives de popularisation des systèmes multi-agents par des interfaces plus faciles, le vivier d'utilisateurs reste faible, notamment par rapport à d'autres typologies.

La différenciation des utilisateurs, entre expérimentation superficielle des systèmes multi-agents et maîtrise de la typologie qui permet de l'implémenter dans un projet complexe, est un signe de la difficulté technique du recours au systèmes multi-agents. Là où l'accessibilité des fractales, des L-systèmes ou de la méthode de la relaxation dynamique permet d'obtenir très vite des objets géométriques exploitables dans un projet d'architecture, les systèmes multi-agents nécessitent un plus grand nombre d'étapes parfois délicates, pour parvenir à ce résultat. Même la visualisation géométrique du résultat n'est pas aisée. Par ailleurs, leur caractéristique principale étant l'apparition de comportements émergents, cette typologie peut s'avérer plus difficile à contrôler pour un utilisateur néophyte. On note d'ailleurs que les travaux de kokkugia s'appuient assez tardivement sur les systèmes multi-agents comme générateurs de forme : l'utilisation qu'ils en font consiste pendant un temps à les appliquer sur d'autres géométries, d'où le terme d'effets employé. Ainsi on distingue deux

phases de développement du recours aux systèmes multi-agents dans la chronologie du champ computationnel. D'abord un premier pic de développement par des utilisateurs versés dans la programmation , qui utilisent des environnements de programmation généraliste et dont nous avons décrit le travail notamment au chapitre IV. Les systèmes multi-agents représentent alors une part significative des projets. À cette période succède une tentative de popularisation par la création d'outils plus simples d'usages. Cette tentative n'est cependant pas couronnée de succès, puisque le recours aux systèmes multi-agents dans les années qui voient la publication de ces outils est en perte de vitesse. Cette perte de vitesse est liée à la difficile transposition dans ces outils de la maîtrise de la typologie, une difficulté liée à la structure algorithmique des systèmes multi-agents. Celle-ci se prête en effet mal à la simplification. La deuxième phase de développement des systèmes multi-agents se caractérise donc par des interfaces à forte épaisseur et des paramètres relégués dans les profondeurs, permettant difficilement une manipulation détaillée de la typologie et freinant l'appropriation tacite, étape pourtant cruciale à l'utilisation des systèmes multi-agents compte tenu de la complexité de cette typologie, et donc à leur démocratisation.

Pour les systèmes multi-agents comme pour les autres typologies, représenter les set-ups par le biais des différents diagrammes présentés permets de mieux comprendre ce à quoi sont confrontés les utilisateurs des outils du champ computationnel. Dans son livre The Interface Effect, paru en 2013, le chercheur Alexander R. Galloway définit les interfaces non comme des objets, mais comme une multiplicité de processus. C'est cette multiplicité que donne à voir l'analyse de leur structure, qui donne aussi un aperçu de l'expérience de travail des praticiens du champ. Plusieurs éléments du parcours des utilisateurs au sein de ces set-ups techniques permettent de décrire un peu mieux ce savoir-faire : positionner les paramètres clés, circuler dans les setups, adapter les paramètres dans leurs valeurs numéraires. C'est cette expérience de travail qui permet de constituer le savoir-faire propre à la conception computationnelle. Ou plutôt, c'est cette expérience qui le permet lorsque les interfaces ne brisent pas la tâche en des éléments trop éloignés les uns des autres, désengageant les utilisateurs du travail à accomplir. À ce titre, le fort développement de la programmation visuelle interroge. Très en vogue pour les bénéfices qu'elle offre pour la formation des utilisateurs novices, c'est aussi la famille d'interfaces qui enterrent les paramètres le plus et qui s'épaissit le plus. Ceci a une conséquence directe sur la manipulation par les praticiens des typologies. La question se pose donc de comprendre dans quel objectif les différentes familles d'interfaces sont mobilisés par les praticiens dans le cadre de l'enjeu de transmission qui se pose dans le champ.

## Conclusion

### La négociation nécessaire de l'apprentissage de la programmation

Nous avons vu au cours de ce chapitre que le recours aux outils dans le champ computationnel s'évalue selon plusieurs aspects. Faire ses propres outils de conception algorithmique est toujours un marqueur dans le champ. Mais plusieurs manières de structurer ces outils et leurs usages coexistent, caractérisées selon plusieurs critères : la typologie algorithmique employée, le format des outils et leur interface. Nous avons établi que les typologies d'algorithmes, dont l'usage varie au fil du temps, structurent le recours aux algorithmes dans le champ. Ces typologies proviennent d'autres domaines et sont détournées et adaptées par les praticiens du champ computationnel, dont ce travail est une des facettes du savoir-faire. Nous avons établi qu'il existe différents formats d'outils, en fonction de la volonté des programmeurs de se construire des raccourcis d'usage, ou de construire des raccourcis d'usage pour des utilisateurs à l'intention desquels l'outil est développé. Ces raccourcis peuvent avoir deux rôles différents : permettre de programmer plus vite, ou permettre d'utiliser l'outil sans tout comprendre de sa description technique - comme c'est souvent le cas avec les machines et outils en usage au quotidien. Le second point demande un travail particulier sur les interfaces, dont on trouve plusieurs types dans le champ. Ces différents types permettent des recours différents aux algorithmes en fonction de la position des paramètres clés des typologies. La forme des outils en usage dans le champ computationnel influe fortement sur leurs possibilités d'utilisation. Certains d'entre eux, examinés de près, révèlent des caractéristiques techniques qui mettent leurs utilisateurs novices dans une position de contrôle moindre des algorithmes.

La structure des outils algorithmiques révèle la relation qu'ils entretiennent au niveau de connaissance de leurs utilisateurs. Certaines des interfaces, des formats, des typologies permettent à des utilisateurs novices de s'emparer aisément des outils algorithmiques, quand d'autres restent d'un usage plus compliqué malgré les tentatives de facilitation. Le développement sifort de la programmation visuelle et le retour des interfaces utilisateur s'explique notamment par cette volonté de trouver une manière de transmettre le savoir-faire du champ tout en permettant aux utilisateurs de travailler dès le début sur des propositions spatiales, sans avoir à atteindre un très haut niveau de programmation en amont. Cet usage plus facile a cependant un prix : renoncer à une maîtrise détaillée de l'algorithme et de ses sorties. En effet, moins l'utilisateur est formé, moins il a de contrôle sur la mise en œuvre du modèle et de ses implications. Le développement des différents outils de programmation dans le champ met donc en évidence un enjeu clé : la négociation dont ils sont le lieu. Cette négociation a lieu entre facilité d'usage pour des utilisateurs néophytes et contrôle des algorithmes permis par des outils qui ne simplifient pas trop l'interface de manipulation de ces algorithmes. Ceci interroge les conditions de

transmission du savoir-faire propre au champ : faut-il tout savoir de la programmation pour pratiquer l'architecture computationnelle ? Le chapitre suivant va nous permettre d'explorer comment les praticiens se positionnent par rapport à cette question, et quelle gestion de cette négociation les cadres de transmission au sein du champ mettent en place. En effet les outils seuls n'expliquent cependant pas la trajectoire des différentes typologies. Celle-ci est aussi tributaire des cadres de transmission mis en place par le champ computationnel, que nous allons maintenant examiner.

# CHAPITRE VI.
# -
# Une mue du champ

## 6.1 Profils du champ computationnel

### *6.1.1 Profils de connaissance : de nouveaux acteurs*

Les interfaces logicielles qui permettent la manipulation des algorithmes sont aussi ce qui permet aux utilisateurs d' accéder à des modèles algorithmiques complexes et à les manipuler. Nous avons vu que de la structure de ces interfaces dépend en partie la connaissance et la maîtrise des outils. Mais cette maîtrise dépend non seulement des interfaces, mais aussi du savoir des utilisateurs. Pour mieux comprendre comment les connaissances des utilisateurs interviennent dans la définition de leurs pratiques computationnelles, nous allons examiner les différents profils de praticiens en présence dans le champ. Pour ce faire, nous allons nous baser sur la notion de profil de connaissance. Ces profils représentent différents types d'expertises, et donc différentes manières de mobiliser les outils computationnels. Les praticiens du champ, en fonction de leur formation et de leur profession, développent différentes expertises, résumées par des profils montrant les différents types de connaissances tacites et explicites mobilisées par chacun, mais aussi la capacité de traduction du tacite vers l'explicite. Trois types de connaissances tacites sont identifiées : celles liées à la conception architecturale, celles liées à l'usage d'un outil computationnel et celles liées à la structuration d'un modèle algorithmique et par extension au développement d'un outil. Grâce à l'observation de ces profils, il s'agit d'identifier si et comment les modalités d'usage des outils se répartissent en fonction des savoirs des praticiens. La formation et la pratique quotidienne ont une influence sur la capacité à utiliser des algorithmes, à les programmer, voire à programmer ses propres outils - un impératif qui pèse sur tous les praticiens du champ, quelle que soit leur formation initiale. Il s'agit aussi d'observer plus clairement les interactions entre praticiens dans les différents cadres d'exercice du champ computationnel, pour pouvoir saisir les dynamiques d'apprentissage et de transmission à l'œuvre.

Pour étudier les différents profils de connaissances, cinq points sont étudiés pour les différents praticiens du champ computationnel. Chacun de ces points est noté de un à trois en fonction des connaissances acquises par le praticien. D'abord, leur parcours de formation : celui-ci détermine en effet un premier ensemble de connaissances de programmation si la formation suivie intègre des cours

sur le sujet. Les trois possibilités pour les praticiens du champ sont donc :

- n'avoir reçu aucune formation à la programmation pendant leurs études ;
- avoir reçu une formation à la programmation comme partie de leur cursus ;
- avoir suivi une formation spécialisée dans la programmation.

Le second point est la profession des praticiens, qui influe sur leur manière de pratiquer, leur vision de l'architecture, leurs *a priori* sur les outils numériques. Les trois possibilités pour les praticiens du champ sont :

- programmeur ;
- ingénieur ;
- architecte.

Le troisième point est la fréquence d'utilisation d'algorithmes au quotidien. Ces profils visent à représenter différents types d'expertise sans se focaliser uniquement sur la formation et la profession, mais en prenant aussi la dimension pratique de la conception architecturale. Or la régularité de manipulation joue aussi dans l'acquisition des savoirs que nous observons. Les trois possibilités pour les praticiens du champ sont :

- une pratique occasionnelle ;
- une pratique régulière ;
- une pratique professionnelle.

Le quatrième point est le milieu de pratique des praticiens, qui influe lui aussi sur leurs habitudes de travail et leurs habitudes de pensée. Les deux possibilités sont :

- le milieu industriel[1] ;
- le milieu académique.

Enfin, le cinquième point est le format des outils utilisés. Celui-ci est lié aux capacités de programmation des usagers, mais aussi à la finesse de leur savoir-faire[2]. Les trois possibilités pour les

[1] L'industrie de la construction ou du logiciel en fonction des cas.
[2] Voir Chapitre IV.

praticiens du champ sont :

- utilisation d'outils prêts à l'emploi ;
- customisation d'algorithmes ;
- développement d'outils sur-mesure.

Sur le graphe ci-dessous (fig. 1) sont visibles les profils identifiés dans le champ. Les points listés sont répartis du centre vers l'extérieur en suivant l'ordre des listes. Sur le côté gauche du graphe sont concentrés les points influant majoritairement sur l'acquisition du savoir-faire de la conception architecturale. Sur le côté droit se trouvent les points en lien avec l'acquisition du savoir-faire de la programmation. L'ensemble de l'aplat représente donc l'intermédiaire que constitue le savoir-faire propre au champ, et la capacité de traduction qui émerge de la combinaison des savoirs-faire acquis en fonction des différentes caractéristiques du profil. Quatre profils de connaissances prédominants sont observés : les architectes-programmeurs, les ingénieurs-programmeurs, les développeurs et les architectes.

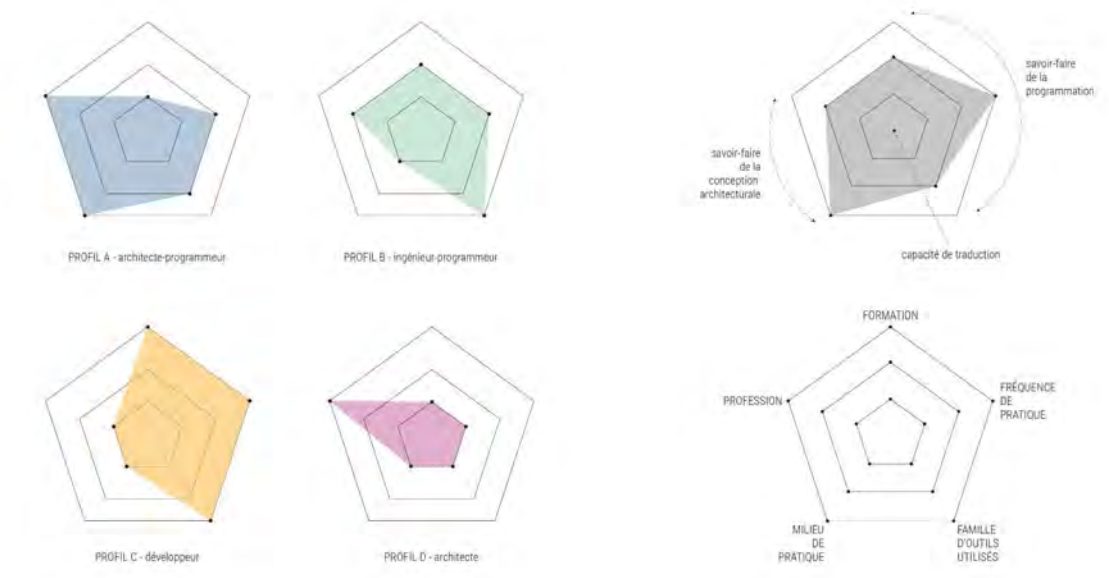



*Figure 1. Profils de connaissance*

Le premier profil relevé est celui des architectes-programmeurs. Plusieurs trajectoires de praticiens relevant de ce profil sont visibles sur la figure 2 à titre d'illustration. La profession de ces praticiens est celle d'architecte : ils sont diplômés d'une école d'architecture. Pour une majorité d'entre eux, leur parcours s'est fait dans deux, voire trois écoles d'architecture différentes. La première, où est effectuée la licence, est une école d'architecture située dans leur pays natal, usuellement en dehors des réseaux du champ computationnel. La seconde, où est effectué le master ou le second master, est une des écoles d'un pôle du champ. La troisième les accueille pour le doctorat, lorsqu'ils changent d'institution après le master. Ces praticiens n'ont donc pas de formation initiale à la programmation, ou du moins pas de formation classique. En effet, pour une partie d'entre eux, le passage dans une école faisant partie du réseau implique un apprentissage de la programmation au sein de cet établissement. C'est cependant un apprentissage qui a toujours lieu dans le cadre du projet d'architecture, et non un apprentissage de la programmation à part entière. Ces praticiens s'identifient comme architectes, mais n'exercent cependant pas en tant qu'architectes au sens conventionnel du terme. Même si leur parcours est parfois marqué par des incursions en agence, comme nous l'avons vu avec Bill Hillier, Alisa Andrasek et plusieurs autres, leur milieu d'exercice principal reste le milieu universitaire. Leur travail avec les outils relève essentiellement du détournement d'algorithmes et de leur customisation, parfois de la réalisation d'outils sur-mesure. Leur fréquence d'utilisation de la programmation, très élevée, leur confère donc une maîtrise au fil des années même si elle n'a pas été acquise au cours de leur formation initiale. Ainsi, leurs savoir-faire s'équilibrent entre les différents domaines, avec un pic vers les savoir-faire de la conception architecturale. Il s'agit du profil le plus équilibré parmi ceux repérés.

Le deuxième profil relevé est celui des ingénieurs-programmeurs. Plusieurs trajectoires de praticiens relevant de ce profil sont visibles sur la figure 3 à titre d'illustration. Après une formation en école d'ingénieur, usuellement dans le domaine du génie civil, ces praticiens exercent en tant qu'ingénieur, majoritairement dans des bureaux d'études, c'est-à-dire dans le milieu industriel. S'ils peuvent intervenir ponctuellement dans des écoles d'architecture, c'est rarement leur poste principal. Leur formation comprend usuellement des cours de programmation, sans que ce soit l'objet principal du cursus. Mais la présence de ces enseignements leur donne néanmoins des connaissances en programmation. Il faut par ailleurs noter que certains profils d'architectes-programmeurs ont une formation qui inclut une licence en génie civil : ils sont donc à mi-chemin entre architecte-programmeur et ingénieur-programmeur. Leur travail se fait usuellement à partir de l'appropriation ou de la customisation d'algorithmes, parfois avec des outils sur-mesure, et la fréquence régulière de leur travail de programmation renforce leurs connaissances dans le domaine. Il

s’agit donc d’un profil de connaissance plutôt équilibré, avec un pic du côté des savoirs techniques observés.

Le troisième profil relevé est celui des développeurs. Plusieurs trajectoires de praticiens relevant de ce profil sont visibles sur la figure 4 à titre d’illustration. Leur profession est celle de programmeur. Leur activité est l’écriture d’algorithmes et de programmes, d'où le terme de développeur puisque leur objectif principal est le développement d’outils (quel que soit leur format). Leur milieu de pratique est majoritairement celui de l’industrie de la construction, pour laquelle ils programment. Quelques exemples de programmeurs évoluant dans le milieu universitaire existent cependant aussi dans le champ. Formés à l’informatique, la programmation est l’objet principal de leur cursus. Ils en ont donc une connaissance poussée. Leur pratique quotidienne professionnelle et leur travail avec des outils sur-mesure ne fait que renforcer cette maîtrise. Néanmoins, malgré ce savoir technique très étendu, leur savoir-faire de la conception architecturale reste peu développé, puisqu’il ne s’agit pas de leur aire d’expertise. Leur travail de développement se fait en collaboration avec d’autres profils, qui sont la source des informations nécessaires dans ce domaine.

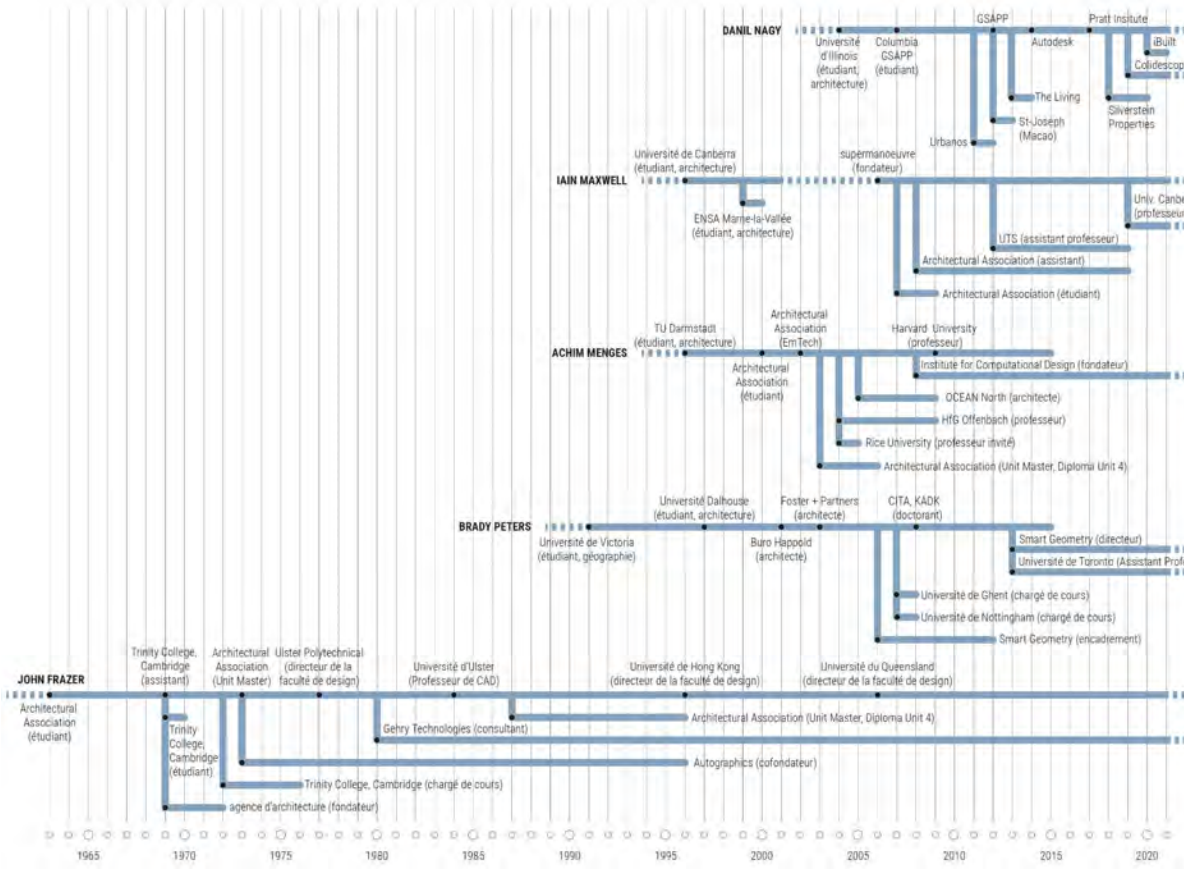



*Figure 2. Trajectoires exemples pour le profil architecte-programmeur*

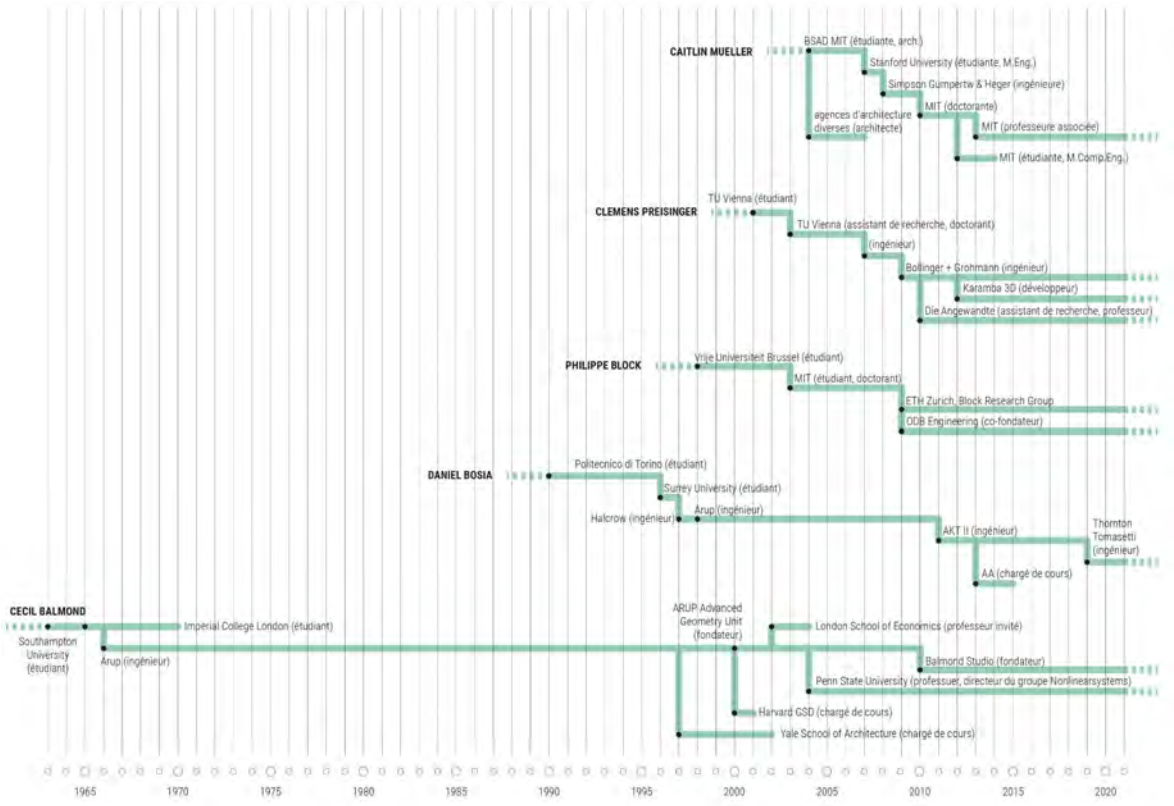



*Figure 3. Trajectoires exemples pour le profil ingénieur-programmeur*

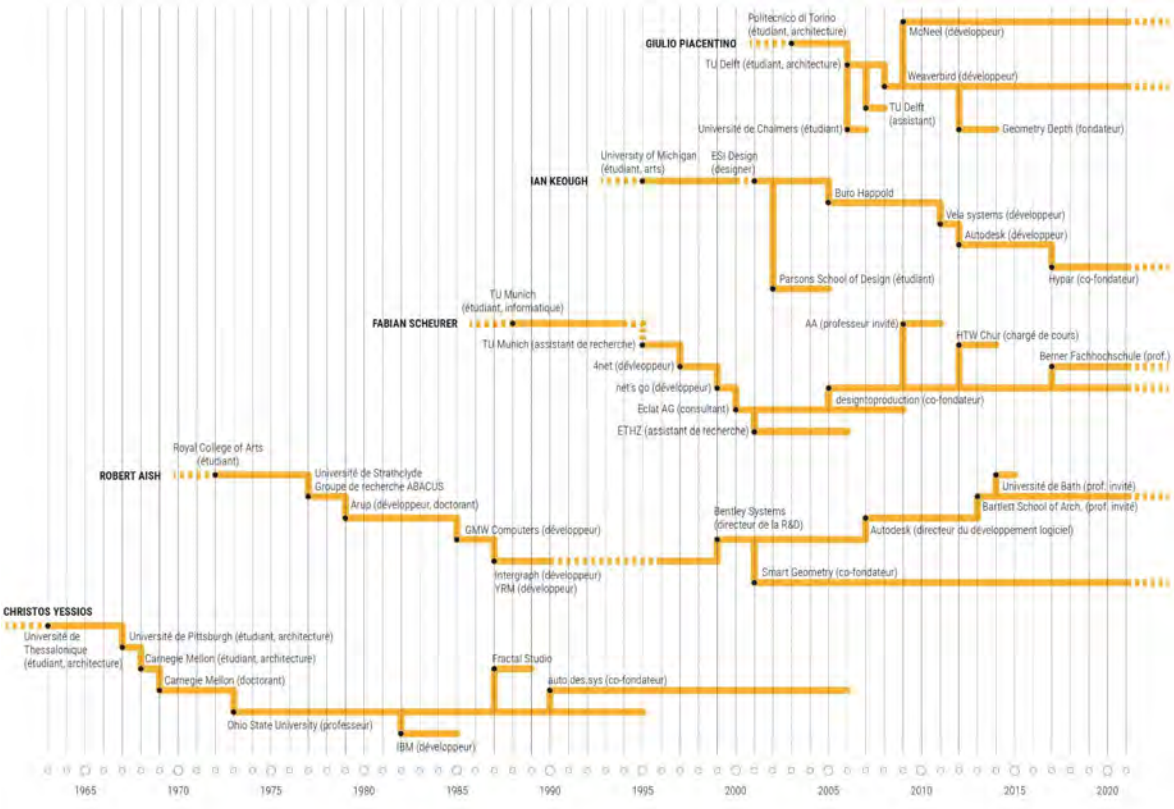


*Figure 4. Trajectoires exemples pour le profil développeur*

Le quatrième profil relevé est celui des architectes. Comme les architectes-programmeurs, ces praticiens sont formés en école d'architecture. Ils ne passent cependant pas par les institutions du champ computationnel, et suivent une formation classique d'architecture. Leur recours aux outils algorithmiques arrive dans un second temps. Il est motivé notamment par leur milieu de pratique, des agences d'architecture conventionnelles - ils relèvent donc de la sphère industrielle. Leur travail en agence motive le recours à des outils prêts à l'emploi, de manière plus occasionnelle que les autres profils. La promotion des outils algorithmiques par le champ computationnel, la mise à disposition de différents outils à travers leur intégration dans des modeleurs comme Rhinocéros ou Revit contribuent à y faire accéder ces praticiens. Les mouvements maker, open-source, participatifs encouragent par ailleurs également ces praticiens à se saisir de nouveaux outils, notamment d'outils algorithmiques permettant la customisation de leurs outils numériques et des projets résultants. Ayant suivi une formation d'architecture conventionnelle, ces praticiens ne sont absolument pas formés à la programmation, ce sont des néophytes dans le domaine. Comme le pic sur la figure 1 l'illustre, leur savoir-faire est entièrement concentré sur la conception architecturale.

Bien qu'en étudiant les trajectoires des différents praticiens on puisse identifier nettement ces quatre profils différents, leurs présences respectives dans le champ varient avec le temps. La répartition entre les profils change ainsi d'une génération à l'autre, surtout entre la troisième et la quatrième génération (fig. 5). Les profils historiques de l'architecte-programmeur et de l'ingénieur-programmeur, présents depuis le début dans le champ, s'y maintiennent au fil des générations. Cette longévité ne se fait cependant pas sans évolution. C'est du moins le cas pour les architectes-programmeurs, dont la trajectoire est marquée par des évolutions dans la formation et les activités professionnelles. Le profil des ingénieurs-programmeurs montre quant à lui peu d'évolutions. Concernant la formation des architectes-programmeurs, elle passe pour les praticiens les plus jeunes d'un cursus dans deux écoles d'architecture, dont une dans un des pôles du réseau, à un cursus effectué dans un seul établissement, qui fait déjà partie du réseau. L'intérêt des praticiens ne se traduit donc plus par ce changement en cours de cursus. Par ailleurs, la plupart des micro-agences qui permettaient aux praticiens de la troisième génération de conserver un espace de pratique professionnelle cessent leurs activités. Cela marque une séparation de plus en plus marquée entre les deux milieux de pratique du champ. Néanmoins, la transformation des trajectoires est aussi influencée par l'ouverture de nouvelles possibilités professionnelles vers des agences conventionnelles. Celles-ci portent un intérêt croissant aux outils algorithmiques, elles sont de plus en plus en demande de praticiens formés à leur usage. Les architectes-programmeurs peuvent donc occuper des postes de concepteurs spécialisés ou diriger au

sein des agences de petits groupes dédiés à la transformation des pratiques. L'éventail des modalités d'exercice s'agrandit donc pour les architectes-programmeurs au-delà du seul milieu universitaire.

Le profil des développeurs connaît également des changements profonds. En particulier, un changement d'échelle et de milieu se produit, puisque le profil passe peu à peu des programmeurs qui exercent à l'université aux éditeurs de logiciel. Certes des développeurs existent dès le début dans le champ. La première génération est marquée par des collaborations avec des informaticiens de formation comme Greg Bryant ou James Dips. Avec le développement industriel lié à l'infographie, de nombreuses entités privées émergent à la seconde génération en complément des groupes universitaires. Ces entités sont néanmoins aux frontières du champ, et non vraiment au sein de son réseau. La troisième génération est marquée par un plus petit nombre de développeurs, puisque la majeure partie des praticiens sont immergés dans le milieu de l'architecture et de la construction et en proviennent - ce sont essentiellement des profils d'architectes-programmeurs ou d'ingénieurs-programmeurs, ou alors des informaticiens universitaires. La quatrième génération est marquée par l'arrivée d'éditeurs de logiciel se prenant d'intérêt pour les outils de conception algorithmique qu'ils souhaitent intégrer à leur offre. Aux générations précédentes, ces éditeurs n'étaient pas très éloignés du champ computationnel : IBM, Dassault ou McNeel s'y intéressent déjà depuis un certain temps. Mais l'échelle de leur investissement change, et ils occupent un rôle de plus en plus dominant dans le champ. Par ailleurs, Autodesk et d'autres éditeurs ont su repérer tôt les profils d'architectes-programmeurs prometteurs du champ computationnel pour les inviter à contribuer au développement de leurs outils. Ils mettent également à contribution de longue date des développeurs issus du domaine de l'informatique - en toute logique puisque ce sont des éditeurs de logiciels informatiques. Mais la quatrième génération est marquée par l'apparition de développeurs indépendants directement dans le champ, sans passer par un intermédiaire comme Autodesk. De plus, ces acteurs s'intéressent désormais aux outils computationnels plutôt que simplement aux outils de modélisation. Ces structures de petite échelle, comme Hypar, Fologram, HAL ou MESH, sont pilotées par des développeurs issus du champ de l'informatique, mais aussi parfois par des architectes-programmeurs directement issus du champ de l'architecture computationnelle. Ces derniers se sont ainsi déplacés d'un profil vers l'autre.

L'intérêt croissant des agences conventionnelles se traduit également par l'apparition d'un nouveau profil : celui des architectes novices en programmation. Au fil des générations, le nombre de praticiens dans le champ grossit. Mais en comparaison de la très forte augmentation de ces architectes, les autres profils stagnent presque. Le champ continue donc à se développer, et les architectes-programmeurs et ingénieurs-programmeurs à former des héritiers. Mais ce nouveau profil

des architectes prend une très grande importance. De très nombreux utilisateurs amateurs se saisissent des modèles algorithmiques désormais proposés un peu partout en ligne et au sein des solutions logicielles les plus populaires de l'industrie de l'architecture. Le nombre des praticiens des courants computationnels se situe autour de 65 000, mais le nombre des utilisateurs ayant accès via les solutions logicielles toutes faites à des modèles algorithmiques s'élève- en fonction des estimations - jusqu'à 18 millions[3]. Une dynamique de démocratisation exponentielle est donc bien à l'œuvre, mais elle concerne seulement certains profils: si le nombre des amateurs augmente fortement, ce n'est pas le cas du nombre d'architectes capables de programmer seuls un algorithme sur-mesure. En effet, les autres profils, ayant des connaissances en programmation, sont capables de manipuler seuls des algorithmes. Et lorsque les utilisateurs sont majoritairement capables d'écrire un algorithme sur mesure et de concevoir leurs propres bibliothèques et interfaces, le profil de connaissances entre utilisateur et programmeur se chevauche, d'où une épaisseur d'interface faible et un maintien de la transmission des connaissances. Ce n'est pas le cas des architectes novices en programmation. Le chevauchement des profils de connaissances des utilisateurs d'outils et de leurs développeurs - auparavant réunis en une même personne, ou étant collègues - diminue.

Le réseau évolue à travers les changements observés dans les profils historiques et à travers l'apparition de nouveaux profils. La séparation va croissant entre développeur et utilisateur, le nombre des premiers augmentant par ailleurs peu alors que les seconds voient leur nombre exploser. Le nombre d'outils disponibles augmente lui aussi largement[4]. Le nouveau profil des architectes novices en programmation, le nouveau statut des développeurs, l'augmentation et la diversification des usagers sont autant de marqueurs d'une sortie du réseau tel qu'on l'a décrit dans les phases précédentes. Cette nouvelle génération du champ computationnel se distingue par une extension du réseau à de nouveaux profils - en particulier ceux issus du domaine de l'informatique plutôt que celui de l'architecture ou du génie civil. Parmi les développeurs qui gravitent autour de Grasshopper, on observe de plus en plus des profils d'architectes-programmeurs qui ne sont pas directement des héritiers de la génération précédente, c'est-à-dire qu'ils ne sont pas passés par l'une des institutions intégrées au réseau. De nouveaux développeurs d'outils apparaissent également, avec d'autres antécédents que l'architecture. Les nouveaux praticiens du domaine n'apparaissent plus seulement à partir d'une transmission de connaissances des générations précédentes au sein des institutions du réseau. Une sortie du réseau historique s'effectue donc à cette période. Pour ces nouveaux venus dans le réseau des pratiques computationnelles, une chose a changé. L'ordinateur n'est plus un objet de curiosité, c'est un objet du quotidien. Pour les praticiens des débuts, comme Nicholas Negroponte ou

[3] La méthode d'obtention de ces estimations est explicitée un peu plus loin dans le chapitre.
[4] Voir Chapitre VI.

Yona Friedman, l'ordinateur est une nouveauté scientifique qui requiert une exploration. Pour les praticiens plus jeunes, comme Georges Legendre ou Fabio Gramazio, les ordinateurs sont moins exotiques, ils sont disponibles à l'école voire à la maison mais il s'agit d'objets à peine importé dans le monde architectural[5]. La curiosité liée à la rencontre des deux domaines invite donc toujours à l'expérimentation. Mais dans les années 2010, l'ordinateur est naturalisé, il a sa place au quotidien dans toutes les disciplines, c'est une évidence pour les praticiens en âge de s'intéresser au champ computationnel, qui sont à la fois plus nombreux et plus exposés à l'idée. Le changement dans les interfaces observé au chapitre précédent s'accompagne donc d'une évolution dans les profils de connaissances des programmeurs et des utilisateurs, ainsi que du nombre d'utilisateurs, qui traduit un changement de tissu du réseau.

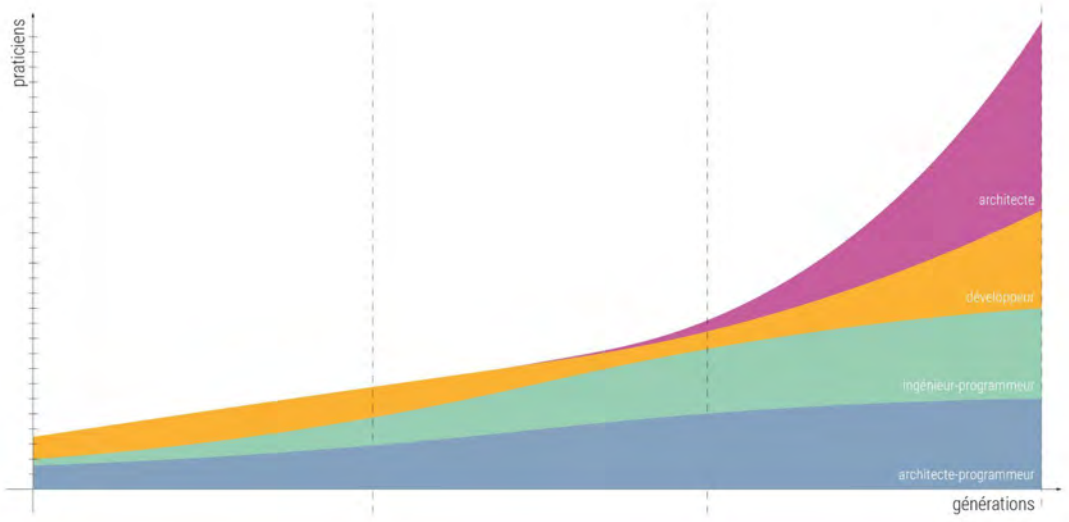



*Figure 5. Évolution des profils*

### *6.1.2 Un changement de tissu du réseau : le poids des éditeurs*

Dans le cours des récits de praticiens qui ont permis, par touches successives, de saisir le développement des courants computationnels en architecture, affleure parfois une hésitation. Une

[5] Au cours des entretiens, Fabio Gramazio a notamment raconté comment lui et Matthias Kohler ont eu accès à des ordinateurs dès l'enfance, grâce à l'acquisition par leurs familles d'ordinateurs Commodore 64. George Legendre a raconté quant à lui ses incursions au sein du département de génie civil pour explorer le recours qui y était fait des ordinateurs dont la faculté était équipée.

crispation apparaît lorsque la conversation dérive vers une ombre planant sur les multiples expérimentations qui ont bourgeonné depuis les années 60. Ou plus exactement depuis 1982, année de fondation du mastodonte qui projette cette ombre : Autodesk. Pour saisir les raisons de cette crispation, et l'ampleur de cette ombre portée, il faut revenir sur l'expansion d'Autodesk entre les années 1980 et aujourd'hui, ainsi que sur les stratégies qui ont présidé à cette expansion.

| Année | Entité | Domaine | Type d'achat | Devenir immédiat | Devenir à terme (2021) |
|---|---|---|---|---|---|
| 1992 | Micro Engineering Solutions | Manufacture | entreprise | assets réintégrés | absorbé |
| 1993 | Ithaca Software | Animation / Infographie | entreprise | assets réintégrés | absorbé |
| 1996 | Softdesk | AEC | entreprise | assets réintégrés | absorbé |
| 1998 | Genius CAD-Software | Manufacture | assets | assets réintégrés | absorbé |
| 1998 | Discreet Logic Inc. | Animation / Infographie | entreprise | assets réintégrés (3DSMax) | absorbé |
| 1999 | VISION* Solutions | GIS | entreprise | assets réintégrés | absorbé |
| 2001 | Gentry Systems | Manufacture | entreprise | assets réintégrés | absorbé |
| 2001 | Buzzsaw | Data Management | entreprise | logiciel commercialisé | discontinué (2019) |
| 2002 | Revit Technology Corporation | AEC | entreprise | logiciel rebrandé | maintenu |
| 2002 | CAICE Sotware Corporation | AEC | entreprise | logiciel commercialisé | maintenu |
| 2002 | trueInnovations, Inc. | Data Management | assets | assets réintégrés | absorbé |
| 2003 | Linius Technologies | Animation / Infographie | entreprise | assets réintégrés | absorbé |
| 2003 | VIA Development Corporation | Manufacture | assets | assets réintégrés | absorbé |
| 2004 | MechSoft, Inc. | Manufacture | entreprise | assets réintégrés | absorbé |
| 2005 | COMPASS systems GmbH | Data Management | assets | logiciel commercialisé | discontinué |
| 2006 | Alias | Manufacture | entreprise | logiciel commercialisé | maintenu |
| 2007 | Skymatter Inc | Animation / Infographie | entreprise | logiciel commercialisé | maintenu |
| 2007 | NavisWorks Limited | AEC | entreprise | assets réintégrés | absorbé |
| 2007 | Opticore AB | Manufacture | assets | assets réintégrés | discontinué |
| 2007 | PlassoTech | Manufacture | entreprise | assets réintégrés | absorbé |
| 2008 | Robobat | AEC | entreprise | logiciel commercialisé | maintenu |
| 2008 | Carmel Software Corporation | AEC | assets | assets réintégrés (?) | absorbé |
| 2008 | Moldflow Corporation | Manufacture | entreprise | logiciel commercialisé | maintenu |
| 2008 | Kynogon SA | Animation / Infographie | entreprise | logiciel rebrandé | discontinué |
| 2008 | REALVIZ S.A. | Animation / Infographie | entreprise | logiciel commercialisé (Stitcher), asse | discontinué |
| 2008 | Square One Research | AEC | entreprise | logiciel commercialisé | maintenu |
| 2008 | Softimage Co | Animation / Infographie | entreprise | logiciel commercialisé | discontinué |
| 2008 | BIMWorld | AEC | assets | assets réintégrés | absorbé |
| 2008 | ALGOR, Inc. | Manufacture | entreprise | logiciel rebrandé | maintenu |
| 2009 | VisualTAO (PlanPlatform) | AEC | entreprise | logiciel rebrandé | discontinué |
| 2010 | Illuminate Labs | Animation / Infographie | entreprise | logiciel commercialisé (Beast), assets | discontinué |
| 2011 | Blue Ridge Numerics | Animation / Infographie | entreprise | logiciel commercialisé | discontinué |
| 2011 | Pixlr | Animation / Infographie | entreprise | assets réintégrés | revendu |
| 2011 | Instructables | Manufacture | entreprise | site maintenu | maintenu |
| 2011 | Numenus | AEC | entreprise | assets réintégrés (?) | absorbé |
| 2011 | Grip Entertainment | Animation / Infographie | entreprise | assets réintégrés | absorbé |
| 2011 | Horizontal Systems | AEC | entreprise | assets réintégrés | absorbé |
| 2011 | Scaleform | Animation / Infographie | entreprise | logiciel commercialisé | discontinué |
| 2012 | Qontext, Inc. | Data Management | entreprise | assets réintégrés | absorbé |
| 2013 | Firehole Technologies | Manufacture | entreprise | logiciel commercialisé | discontinué |
| 2013 | Tinkercad | Manufacture | entreprise | logiciel commercialisé | maintenu |
| 2013 | Graitec | AEC | assets | logiciel commercialisé, assets reintég | maintenu |
| 2014 | Delcam | AEC | entreprise | logiciel commercialisé | discontinué |
| 2014 | Creative Market | AEC | entreprise | site maintenu | revendu |
| 2014 | NET Software | Manufacture | assets | assets réintégrés | absorbé |
| 2014 | Within Technologies | Manufacture | entreprise | logiciel rebrandé | discontinué |
| 2014 | Shotgun Software | Data Management | entreprise | site maintenu, logiciel commercialisé | maintenu |
| 2014 | Topolabs Technology | Manufacture | entreprise | assets intégrés | absorbé |
| 2015 | SeeControl | Data Management | entreprise | logiciel commercialisé, assets réintég | discontinué |
| 2016 | SolidAngle | Animation / Infographie | entreprise | logiciel commercialisé | maintenu |
| 2016 | CadSoft Computer GmbH | AEC | entreprise | logiciel commercialisé | maintenu |
| 2018 | Assemble Systems | AEC | entreprise | assets intégrés | absorbé |
| 2018 | PlanGrid | AEC | entreprise | logiciel commercialisé | maintenu |
| 2018 | BuildingConnected | AEC | entreprise | logiciel commercialisé | maintenu |
| 2020 | Aurigo Sotware | Data Management | investment | assets intégrés | absorbé |
| 2020 | Bridgit Inc | AEC | investment | assets intégrés | absorbé |
| 2020 | Pype | AEC | entreprise | assets intégrés | absorbé |
| 2020 | Spacemaker | AEC | entreprise | logiciel commercialisé | maintenu |

*Tableau 1. Acquisitions d'Autodesk entre 1992 et 2020*

La fondation d'Autodesk correspond à la conception et mise sur le marché d'AutoCAD. L'entreprise est donc initialement une firme "mono-produit", qui domine très vite le marché du dessin assisté par

ordinateur dans les secteurs de l'architecture et de la construction. Autodesk amorce ensuite au début des années 90, à l'arrivée de Carol Bartz à la tête de l'entreprise, une politique de diversification qui lui permet d'étendre ses activités, en particulier aux domaines de la manufacture et de l'animation. Une politique couronnée de succès, comme en témoigne non seulement l'augmentation du chiffre d'affaires d'Autodesk, mais aussi la nomination en 2001 et 2002 dans la catégorie FX des Oscars de six films pour lesquels la firme s'occupe des effets spéciaux[6]. La chronologie de développement du domaine de l'infographie vue au chapitre 3 permettait déjà de discerner la longévité d'Autodesk des années 1980 à aujourd'hui, en comparaison de nombre d'autres entreprises du domaine. Cette chronologie montrait également la montée en puissance d'Autodesk face aux entreprises concurrentes fondées les mêmes années[7], une montée en puissance qui culmine en 2006 avec l'achat d'Alias, dernière d'une série d'acquisitions ayant permis à Autodesk de survivre à la plupart de ses concurrents, voire de les absorber. Si ce dernier achat permet un triomphe symbolique sur les principales entreprises qui se partageaient le marché à leurs débuts, Autodesk ne s'arrête pas là : la politique de développement de l'entreprise est largement basée sur l'acquisition de multiples autres structures de l'industrie du logiciel - 56 entre 1992 et 2020 (Tab. 1), dans tous les domaines de la conception assistée par ordinateur : construction, design et manufacture, mais aussi jeux vidéo ou gestion des données.

Les rapports édités annuellement par le cabinet de conseil Jon Peddie Research (JPR), spécialisé dans le domaine du CAD, identifient une dizaine d'entreprises comme leaders sur le marché : Autodesk, Aveva, Bentley Systems, Dassault, Graebert, Hexagon, Nemetschek, PTC, Siemens Digital Industries, Trimble. Parmi ces dix entreprises, Siemens, Dassault et Autodesk dominent de loin, avec respectivement 31%, 20% et 18% des parts de marché[8]. Parmi les logiciels mentionnés dans les chapitres précédents, un certain nombre sont développés par Bentley Systems, Trimble, McNeel ou AutoDesSys - à titre de comparaison, ces entreprises possèdent aujourd'hui respectivement 5%, 2%, 0,36% et 0,11% des parts de marché. Le domaine du CAD est néanmoins très large, avec des sous-secteurs dans lesquels ces entreprises se spécialisent : ainsi Bentley et Hexagon proposent essentiellement des solutions logicielles pour le domaine des systèmes d'information géographique[9], Siemens se partage entre le secteur manufacturier et l'ingénierie énergétique et maritime, Dassault et Autodesk s'adressent essentiellement aux secteurs de la manufacture, de l'architecture et de la construction. Et si Autodesk n'a que 18% des parts du marché CAD généraliste selon JPR, d'autres

[6] "Autodesk, Inc. ." International Directory of Company Histories. . Encyclopedia.com. (June 16, 2021). Consulté le 11 Juillet 2021. https://www.encyclopedia.com/books/politics-and-business-magazines/autodesk-inc
[7] Alias, Wavefront, Softimage.
[8] Jon Peddie Research, 2020 CAD Report.
[9] SIG en français, GIS en anglais.

sources lui donnent jusqu'à 60% des parts du marché CAD pour la conception architecturale[10], les diverses études lui donnant toujours une assez large avance sur son principal concurrent dans le secteur, Dassault Systèmes. Au cours des dernières décennies, le paysage CAD du secteur de l'architecture et de la construction s'est beaucoup transformé, Dassault prenant une place de plus en plus importante, au détriment de Bentley Systems, Trimble et Nemetschek qui étaient jusqu'alors les principaux éditeurs de logiciels[11] - avec Autodesk, qui pour sa part n'a jamais quitté sa position de leader.

Une étude réalisée annuellement par CNC Cookbook montre par ailleurs les logiciels privilégiés par les utilisateurs. Sur les trois années 2018, 2019 et 2020, les logiciels édités par Autodesk rassemblent entre 36 et 41% des utilisateurs ; par Dassault 27 à 29% ; par McNeel 4,1% et par Trimble, 2,2%[12]. Autodesk annonce par ailleurs 18 millions d'utilisateurs pour ses produits, quand Dassault en annonce 1 million seulement et que le nombre de ceux de McNeel est estimé autour de 1 million également[13]. L'écart entre les parts de marché en fonction du revenu ou du nombre d'utilisateur s'explique notamment par la variation du prix des licences en fonction des logiciels et des éditeurs : si la plupart fixent ces prix entre 1000 et 2000$, les licences de Siemens démarrent à 5000$, celles de Dassault à 9000$, et Autodesk a mis il y a quelques années en place un système d'abonnement à ses logiciels, facturant annuellement jusqu'à 3500$[14]. Les chiffres exacts de la répartition des parts de marché comme du nombre d'utilisateurs varient selon les pays, les sources et les secteurs, mais Autodesk domine dans les endroits qui ont vu l'émergence du courant computationnel - et ce de plus en plus largement au fil des années.

Cette hégémonie ne s'est pas construite sans s'appuyer largement non seulement sur des pratiques d'acquisition agressives, mais aussi sur des pratiques d'appropriation de savoirs et d'outils développés par d'autres. Les accords passés par Autodesk avec d'autres entreprises permettent à la firme d'acquérir d'autres logiciels en complément de ceux déjà commercialisés, pour les distribuer ou les intégrer à d'autres en cours de développement. Mais ces pratiques de réassemblage ou de marquage s'ancrent parfois aussi dans des procédés plus douteux, comme en témoigne le procès intenté à Autodesk par Spatial Corporation, propriétaire du kernel géométrique ACIS, en 2003. Ces derniers

[10] étude
[11] Serraino, P. History of Form *Z. The IT Revolution in Architecture. Basel Berlin: Birkhäuser, 2002.
[12] CNCCookbook 2021 CAD Survey, https://www.cnccookbook.com/cnccookbook-2021-cad-survey-market-share-customer-satisfaction/, consulté le 13 Novembre 2021.
[13] Ces chiffres sont tirés de plusieurs estimations faites par des observateurs extérieurs à partir du nombre de licences vendues par Rhinocéros. //lien
[14] "Autodesk Details Subscription Transition for New Software Licenses", 04 Février 2015, https://investors.autodesk.com/news-releases/news-release-details/autodesk-details-subscription-transition-new-software-licenses, consulté le 13 Novembre 2021.

accusent alors l'entreprise d'avoir transmis à un de ses sous-traitants une version du code d'ACIS développée par Spatial Corporation quelques années plus tôt dans le cadre d'une collaboration avec Autodesk. Non seulement le contrat liant Autodesk à Spatial Corporation interdisait selon ces derniers un tel partage, mais Autodesk est également accusé par la suite d'avoir utilisé le code d'ACIS pour développer son propre kernel, ShapeManager - notamment pour éviter de s'exposer à la concurrence, ACIS ayant entre-temps été racheté par Dassault Systèmes. Et si à l'issue du procès, le jury a reconnu à l'entreprise le droit d'utiliser ce code, il n'en reste pas moins que le kernel d'Autodesk a été développé en s'appropriant les programmes d'autres[15]. Autodesk est également régulièrement la cible de critiques de la part de ses revendeurs et de ses utilisateurs, qui l'accusent de pressions juridiques et financières. Ainsi, la révocation des licences perpétuelles d'AutoCAD puis le développement tâtonnant de Revit ont été vivement attaquées, mais nombre d'utilisateurs préfèrent rester anonymes, en particulier de grandes entreprises qui acceptent de soutenir des lettres publiées à ce sujet mais ne souhaitent pas révéler leur identité, de peur de sanctions de la part d'Autodesk. Ces derniers pourraient notamment leur faire subir de coûteux audits de vérification des droits d'usage ou refuser de leur accorder des licences[16], voire les attaquer en justice, un acte auquel ils ont recours sans hésiter quand ils l'estiment nécessaire. Cependant, si témoignages et actions légales mettent en lumière des pratiques problématiques de la part d'Autodesk, l'entreprise n'a jamais été condamnée, naviguant toujours à la limite de la légalité malgré les abus de sa position, souvent dénoncée comme étant un monopole de facto tant son influence sur le domaine du CAD et de l'architecture conventionnelle est grande.

Ces pratiques douteuses accompagnent l'histoire d'Autodesk depuis sa création, et le champ de l'architecture computationnelle ne fait pas exception, comme en témoignent les praticiens. Le peu qu'ils acceptent de raconter lorsqu'ils sont interrogés porte en particulier sur la propension d'Autodesk à copier intégralement des logiciels ou systèmes concurrents - comme ACIS ou Grasshopper -, ou récupérer des morceaux de code un peu partout, en se servant notamment dans les répertoires open-source publiés par supermanoeuvre, kokkugia, plethora project ou encore ijp corporation[17]. Si cette réutilisation de codes open-source n'a rien d'illégal puisqu'ils sont justement libres d'accès, ne pas citer les développeurs et s'appuyer sur leurs travaux pour en commercialiser des dérivés n'en reste pas moins peu éthique.

[15] Spatial Corp. VS Autodesk, Inc., https://www.wsgr.com/publications/PDFSearch/autodesk0306.pdf, consulté le 13 Novembre 2021.
[16] Martyn Day, "Autodesk AEC customers demand better value", 25 Juillet 2020, https://aecmag.com/bim/letter-to-autodesk-aec-customers-demand-better-value/, consulté le 13 Novembre 2021.
[17] Entretiens correspondants.

Malgré la réticence des praticiens à raconter en détail leurs rencontres, de peur de s'exposer à de possibles représailles de la part d'Autodesk[18], des similitudes émergent au fil des allusions, qui elles concernent la méthode adoptée par la firme, le profil des entreprises visées ou encore les objets ciblés. Autodesk contacte ainsi régulièrement divers praticiens pour des visites de leurs lieux de travail respectifs ou des discussions sur l'objet de leurs recherches, comme l'a été Christos Yessios, régulièrement invité à échanger avec eux à la fin des années 1980 alors qu'il développait Form*Z, et qui souligne que si Autodesk n'avait pas considéré comme peu pertinents les principes de modélisation 3D qu'il explorait alors pour son logiciel, il n'aurait jamais pu éditer son outil faute de pouvoir concurrencer la firme[19]. Cette dernière offre ainsi des conditions de travail à première vue attractives pour des praticiens ayant déjà développé des outils qui ont fait leurs preuves lors de leur mise en application dans le cadre de projets reconnus, ou lors de collaborations avec des concurrents. Ainsi d'XtreeE invitée à plusieurs reprises à prendre part aux activités de recherche sur l'impression 3D commençant à être mises en place au technocentre d'Autodesk à Boston, une invitation survenant peu après l'annonce d'une collaboration logicielle d'XtreeE avec Dassault Systèmes[20]. Robert Aish lui aussi est invité à rejoindre l'équipe d'Autodesk à la suite de son travail sur Generative Components pour Bentley Systems. A terme, Autodesk absorbe même parfois complètement les structures ayant accepté une invitation à rejoindre un de leurs centres pour un séjour de recherche, comme c'est le cas pour l'agence The Living, qui est hébergée dans les locaux de l'entreprise et travaille depuis 2016 presque exclusivement pour Autodesk, testant les nouvelles solutions logicielles développées par l'entreprise et publiant les projets résultants dans la presse d'architecture généraliste[21] et dans des actes de conférences du champ computationnel, accolant à son nom la mention "an Autodesk Studio".

Les technocentres d'Autodesk, au nombre de quatre et situés à Boston, San Francisco, Toronto et Birmingham (UK), offrent des programmes de résidence avec accès à des équipements de conception et de fabrication de pointe, pour des praticiens se penchant sur des questions autour de la réalité augmentée, la réalité virtuelle, l'haptique, la robotique, l'automatisation de la conception, ou tout autre sujet clé pour l'édition d'outils qui seront les plus prisés du champ dans les années à venir[22]. Ces centres de recherches permettent à l'entreprise d'offrir un espace de collaboration à des profils du courant computationnel typiques du réseau à partir des années 2000, maîtrisant bien la programmation et développant régulièrement de nouveaux outils, soit dans le cadre de recherches à l'université, soit dans le cadre de leur pratique professionnelle. Ce sont d'ailleurs plutôt ces derniers qui sont courtisés

[18] Si l'on peut questionner le potentiel de nuisance réel d'Autodesk, leur influence sur le domaine professionnel reste néanmoins incontestable, ce qui explique le silence des praticiens.
[19] Serraino, P. *History of Form *Z. The IT Revolution in Architecture*. Basel Berlin: Birkhäuser, 2002.
[20] Archives personnelles.
[21] http://www.thelivingnewyork.com/, consulté le 13 Novembre 2021.
[22] https://www.autodesk.com/technology-centers, consulté le 13 Novembre 2021.

- peut-être notamment parce que ne disposant pas du confort d'une position universitaire, ils pourraient être plus sensibles à l'offre d'Autodesk d'un espace protégé des contraintes de l'industrie de la construction et de fonds pour poursuivre leurs recherches. Ainsi ijp corporation, kokkugia, supermanoeuvre ou encore plethora project, dirigées par George Legendre, Roland Snooks et Robert Stuart-Smith, Iain Maxwell et Dave Pigram ou Jose Sanchez, structures aux profils et productions similaires du point de vue des compétences techniques mobilisées[23], sont toutes approchées par Autodesk[24].

The Living ou XtreeE, structures plus jeunes, dont les équipes appartiennent à une génération plus tardive, sont néanmoins les héritières des précédentes selon le même mouvement de transmission décrit pour les générations précédentes[25]. XtreeE, co-fondée en 2015 par Philippe Morel, qui a enseigné à la AA, à la Bartlett, à Delft et en France, l'est avec un groupe d'anciens étudiants dans l'objectif de transférer un ensemble de processus de fabrication robotisés, en particulier l'impression 3D, des laboratoires de recherche à l'industrie de la construction française. Cela se traduit par un ensemble de collaborations avec de grands constructeurs français, mais l'entreprise s'inscrit aussi dans le monde de la recherche, dans le sous-domaine de la robotique architecturale, en participant aux conférences emblématiques du champ, en intervenant dans les institutions du réseau et en s'associant à certains de leurs laboratoires de recherche. C'est également le cas de The Living, dont le fondateur David Benjamin enseigne au GSAPP de Columbia après y avoir étudié, tout comme ses associés Damon Lau, Jim Stoddart, Ray Wang, Lorenzo Villaggi, Lindsey Wikstrom ou John Locke, également passés par SOM pour certains d'entre eux. L'exemple de The Living montre par ailleurs que l'espace de recherche ouvert à ceux qui acceptent l'offre d'Autodesk n'est pas totalement libre, dans la mesure où nombre des intérêts trouvés dans les premiers projets de l'agence, notamment pour la question du biodesign[26], ont ensuite complètement disparu, au profit de la conception d'espaces de bureaux ou de logements à l'aide des nouveaux plug-ins développés par Autodesk pour Revit[27]. Autodesk s'intéresse parfois également, bien que plus rarement, à des individus isolés, comme en témoigne le recrutement de Robert Aish en 2008. Par ailleurs, malgré le fait qu'il s'agisse de praticiens de différentes générations, de structures se spécialisant dans différents sous-domaines du champ computationnel, et

[23] Pour plus de détails sur ces profils voir le chapitre IV.
[24] Sans qu'une collaboration en résulte.
[25] Voir chapitre IV.
[26] Le projet Hy-Fi, exposé au MoMa. Voir http://www.thelivingnewyork.com/, consulté le 13 Novembre 2021.
[27] Par exemple Das, S., Day, C., Dewberry, M., Toulkeridou, V. et Hauck, A.. « Automated Service Core Generator in Autodesk Dynamo - Embedded Design Intelligence aiding rapid generation of design options ». In Herneoja, Aulikki; Toni Österlund and Piia Markkanen (eds.), Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 2, University of Oulu, Oulu, Finland, 22-26 August 2016, pp. 217-226. CUMINCAD, 2016.

qu'ils acceptent ou non de collaborer avec Autodesk, ces praticiens relèvent tous du même profil de connaissance - celui des architectes-programmeurs.

Cependant, si les architectes-programmeurs intéressent particulièrement Autodesk dans le cadre de leur expansion vers le champ computationnel, cette mainmise sur le travail de développement de fonctionnalités complémentaires ou d'outils nouveaux par des praticiens extérieurs ne leur est pas réservée : c'est une stratégie habituelle de développement pour l'entreprise. La politique d'expansion par acquisitions d'Autodesk ne se base pas uniquement sur le rachat d'autres entreprises : environ un cinquième des acquisitions sont uniquement de la propriété intellectuelle[28], et lorsque Autodesk rachète d'autres entreprises, ce n'est souvent pas tant pour leurs parts de marché que pour pouvoir s'approprier les outils développés et les commercialiser. Ainsi, par exemple, en 2003, Autodesk rachète un ensemble d'actifs à trois entreprises, truEInnovations, VIA Development et Linius Technologies, pour les combiner et permettre à ses clients de modéliser des systèmes de câblage électrique[29]. En 2006, le rachat d'Alias leur permet notamment d'acquérir le format de fichier FBX[30]. En 2008, Autodesk rachète une série d'entreprises pour obtenir le contrôle des logiciels Softimage, Stitcher, Ecotect et BIMWorld, mais aussi de l'intergiciel Kynapse, très utilisé dans l'industrie du jeu vidéo. Autres exemples encore, en 2013, c'est au tour du logiciel AdvanceSteel d'être racheté à Graitec, et en 2014, Autodesk rachète à NEi Software le solveur Nastran, ensuite intégré à Inventor. Sur les 56 acquisitions d'Autodesk entre 1992 et 2020, 25 logiciels ont été immédiatement discontinués et leurs technologies intégrées à d'autres produits de l'entreprise, la commercialisation de 14 s'est d'abord poursuivie avant que les logiciels ne soient à terme à leur tour discontinués une fois les technologies intégrées, et 17 ont été maintenus et sont encore commercialisés aujourd'hui. L'essentiel des outils proposés par Autodesk pour l'animation et la manufacture, mais aussi les ressources ayant rendu possible le passage aux processus B.I.M. et aux technologies cloud[31] sont issues de ces achats, qui ont permis à l'entreprise de construire une offre conséquente dans ces domaines.

L'entreprise n'édite de logiciels quasiment qu'en rachetant des logiciels, programmes ou plug-ins développés par d'autres, et en intégrant ensuite les technologies informatiques sur lesquelles ils

[28]Sur les 56 acquisitions d'Autodesk cataloguées, 47 sont des rachats d'entreprises, 9 sont des achats de technologies seulement.

[29] "Autodesk Moves Forward to Fulfill its Mainstream Product Lifecycle Management Strategy with New Asset Acquisitions", 04 Mars 2003, https://investors.autodesk.com/news-releases/news-release-details/autodesk-moves-forward-fulfill-its-mainstream-product-0, consulté le 13 Novembre 2021.

[30] Plus exactement l'ensemble des logiciels qui permettent de le traiter, le format n'étant pas protégé par un dispositif de propriété intellectuelle.

[31] Les technologies dites de cloud sont l'ensemble des technologies qui permettent de stocker, partager, travailler sur des fichiers en ligne.

reposent dans leurs propres solutions logicielles, en rassemblant plusieurs acquisitions en un seul nouveau logiciel ou parfois simplement en commercialisant directement une fois la marque renouvelée - ainsi des nombreux logiciels gardant leur nom, auquel la mention Autodesk est simplement accolée suite au rachat. Nombre des critiques techniques faites à Revit le sont d'ailleurs parce que l'entreprise n'a jamais réécrit les programmes permettant au logiciel de fonctionner après son acquisition en 2002, alors qu'au fil des évolutions informatiques ce dernier en aurait eu besoin[32]. Même AutoCAD, le premier et produit phare de l'entreprise, n'a pas été développé en interne mais racheté. Si les sources divergent concernant la liste exacte des fondateurs ayant pris part à une réécriture d'AutoCAD avant sa publication par Autodesk[33], il est cependant avéré que le prototype a été conçu et programmé par Mike Riddle, ensuite approché par John Walker, le fondateur d'Autodesk, pour en céder les droits à l'entreprise. Les tactiques de développement mises en œuvre par Autodesk reposent donc sur des stratégies visant à obtenir toujours la même chose : des programmes, morceaux de scripts, méthodes algorithmiques ou flux opérationnels (workflows). L'entreprise délègue depuis ses débuts presque systématiquement le développement de programmes et caractéristiques essentielles à des entités extérieures. Dès 1990, le système ADS (AutoCAD Development System) est mis au point : il s'agit d'une interface de programmation permettant d'écrire des modules complémentaires pour AutoCAD grâce à un langage spécifique, conçu en appoint d'AutoLISP pour rendre possible la création de plug-ins à part entière en complément de la simple customisation permise par ce dernier langage[34], et dès 1994, plus de 2000 développeurs ayant recours à ADS sont recensés[35]. L'existence de développeurs tiers est autorisée dès le début par Autodesk, et encouragée par la mise en place de ces systèmes, mais aussi par une politique d'établissement de partenariats officiels avec les structures tierces, qui se conclut pour les plus pérennes et prolifiques par un rachat. L'importance accordée par Autodesk aux développeurs tiers s'explique notamment par le choix de John Walker, dès les premières années de l'entreprise, d'équipes de management composées de programmeurs plutôt que d'autres profils, donc conscientes de la ressource que cela représente. Les équipes de développement des entreprises rachetées au fil des années sont d'ailleurs par la suite souvent directement intégrées aux équipes d'Autodesk, dont la masse salariale s'agrandit principalement ainsi.

---

[32] Martyn Day, "Autodesk AEC customers demand better value", 25 Juillet 2020, https://aecmag.com/bim/letter-to-autodesk-aec-customers-demand-better-value/, consulté le 13 Novembre 2021.

[33] Difficile de dire exactement à quel point le logiciel a été réécrit, si seul Mike Riddle s'en est chargé pour le compte d'Autodesk, si d'autres gens y ont travaillé - notamment John Walker. Les sources Autodesk, notamment celles écrites et éditées sous le contrôle de Walker, affirment que le PDG y a pris part, quand d'autres sources moins proches de l'entreprise affirment le contraire.

[34] "*AutoLISP is an easy to learn, easy to use macro language that encourages customization of the AutoCAD drawing editor by the non-programmer. ADS is a sophisticated application development environment for the professional programmer"* - "AutoCAD Development System", https://www.engr.uvic.ca/~mech410/ACAD_and_C/ads_arx/fd-1.pdf, consulté le 13 Novembre 2021.

[35] "Autodesk, Inc.", https://www.encyclopedia.com/social-sciences-and-law/economics-business-and-labor/businesses-and-occupations/autodesk-inc, consulté le 13 Novembre 2021.

Les tactiques de développement des programmes et le devenir des acquisitions au fil des années dépendent de fait des phases de développement de l'entreprise et de ce que les PDG successifs choisissent de favoriser. L'expansion vers le champ computationnel date ainsi de la phase de prospection pilotée par Carl Bass, PDG entre 2006 et 2017, une phase qui se caractérise par la prise de contact avec de nombreux praticiens du champ et par la mise en place d'expérimentations pour développer des outils algorithmiques de conception spatiale, ainsi que par un intérêt croissant pour la robotique architecturale. Son remplacement par Andrew Anagnost, plus conservateur et plus porté sur la stabilisation des offres pour garantir l'augmentation des revenus de l'entreprise, se traduit par un accent mis sur la construction, une première chez Autodesk depuis le départ de John Walker. Cet accent implique la transformation d'un certain nombre des expérimentations enclenchées par Carl Bass en des outils finalisés et intégrés en particulier à Revit, permettant l'automatisation d'étapes de la construction et le développement de ce que l'industrie de la construction aime appeler le design génératif - des outils de conception algorithmique, donc. Dynamo et ses modules d'algorithmes génétiques, de placement automatisé des noyaux de circulation ou de génération des volumétries deviennent rapidement chez Autodesk le fer de lance du développement des pratiques computationnelles, pensées en association avec l'expansion des pratiques B.I.M. via le *construction cloud* mis en place par l'entreprise.

Si les stratégies d'Autodesk passent parfois par des pratiques qui semblent douteuses, et si les détails de la mise en œuvre de ces stratégies varient selon les périodes, elles correspondent néanmoins dans l'ensemble à un modèle de développement stable depuis quarante ans. L'entreprise s'appuie donc sur une vision stratégique qui a fait ses preuves, avant tout parce qu'elle repose sur une analyse clairvoyante des domaines pour lesquels l'entreprise fournit des solutions logicielles - rares sont d'ailleurs les domaines abandonnés ou les entreprises revendues. Les similitudes observées dans les structures et les objets visés par Autodesk impliquent de la part de ces derniers une fine compréhension du réseau, qui va amener l'entreprise à jouer, avec d'autres, un rôle significatif dans les évolutions récentes du tournant computationnel en architecture. La nature des acquisitions d'Autodesk au fil des années montre d'abord la capacité de la firme à percevoir les développements à venir dans les champs auxquels elle fournit des logiciels : l'acquisition de Buzzsaw en 2001 témoigne d'un intérêt précoce pour les technologies cloud, celle de Revit en 2002 d'une perception du rôle à venir du paramétrique dans le B.I.M. bien avant que les cabinets d'analyse de l'industrie CAD ne le fassent apparaître dans leurs rapports sur les tendances à venir[36]. Autres signes, l'achat d'Instructables en 2011 qui reflète l'émergence du mouvement maker ou encore l'acquisition de Spacemaker en 2020,

[36] Jon Peddie Research, 2019 et 2020 CAD Report.

symbolique de l'importance que sont en train d'acquérir les méthodes de deep learning dans le domaine de la conception architecturale. La création des technocentres permet quant à elle à Autodesk une insertion dans le champ de la robotique architecturale alors qu'en plus des chercheurs du domaine académique, certains constructeurs commencent également à s'y intéresser. Ces infrastructures marquent la volonté d'Autodesk d'occuper le marché des outils de conception computationnels via le lien qu'ils permettent d'établir et d'entretenir avec les profils d'architectes-programmeurs. La création de ces technocentres dénote une clairvoyance d'Autodesk quant au rôle joué par les praticiens du champ computationnel à travers le développement de leurs propres outils, et fait figure de prolongement d'AutoLISP, d'ADS, et des autres systèmes permettant de cultiver un réseau de développeurs tiers déjà mis en place par l'entreprise.

Plus généralement, Autodesk est très bien parvenu à identifier les différents profils de connaissance en jeu dans la conception d'outils informatiques pour l'architecture et la construction, et à créer des espaces pour les transformer en contributeurs à leurs offres. Nous avons évoqué les développeurs tiers provenant du domaine de l'informatique et de l'industrie du logiciel, mais aussi des profils plus intermédiaires, comme les architectes-programmeurs, ce que montre la différenciation faite entre AutoLISP et ADS, qui trahit également une conscience de l'intérêt du format intermédiaire que sont les plug-ins et les bibliothèques[37]. Mais Autodesk s'intéresse également aux amateurs, comme en atteste l'acquisition d'Instructables qui s'adresse aux makers, essentiellement des amateurs, celle de TinkerCAD, que l'entreprise transforme en rampe d'accès qu'elle encourage les utilisateurs novices à employer[38], ou encore les versions LT[39] d'AutoCAD, sans AutoLISP, qui permettent une licence moins chère pensée comme une autre marche intermédiaire. Non seulement Autodesk a une conscience aiguë du rôle des amateurs dans cet écosystème, mais l'entreprise prête par ailleurs particulièrement attention à la facilité de manipulation de ses logiciels ainsi qu'à la courbe d'apprentissage que permettent ses ressources combinées ensemble. Ce choix, associé à la différenciation des formats d'outils proposés pour le champ computationnel, sera une contribution décisive à la démocratisation des procédés de conception algorithmiques à laquelle ils vont contribuer.

La domination d'Autodesk sur le champ du CAD en architecture et construction repose donc sur une stratégie de développement aux dessous parfois douteux, mais qui s'avère extrêmement efficace. Les similitudes observées dans les méthodes d'appropriation déployées comme dans les structures et les objets visés dénotent de la part de l'entreprise une fine compréhension des enjeux présents et futurs du

[37] Voir Chapitre IV.
[38] "Autodesk ne propose plus les produits 123D", https://www.autodesk.fr/solutions/123d-apps, consulté le 13 Novembre 2021.
[39] "lite", ou light - allégées.

domaine. Cette réceptivité explique le rôle de plus en plus essentiel joué par l'entreprise dans la période la plus récente de développement du champ computationnel : auquel Autodesk va commencer à s'intéresser au milieu des années 2000, alors que les pratiques de la génération précédente arrivent à maturité. Cet intérêt pour le champ, combiné à celui d'autres acteurs pour cette question de l'expansion du réseau et de l'accès d'un plus grand nombre de praticiens aux outils computationnels, va enclencher une mue du champ, à travers le développement d'outils d'un nouveau genre. En effet, Autodesk n'est pas le seul éditeur à devenir un acteur essentiel du champ.

Au moment où Autodesk se penche sur le champ computationnel en architecture, McNeel est un éditeur de logiciel bien installé dans ce réseau : Rhinocéros et Grasshopper y sont des outils essentiels, nous l'avons vu au chapitre V. Grasshopper, d'abord pensé comme une initiation à la programmation orientée objet, est un dispositif particulièrement populaire puisqu'il est en grande partie à l'origine d'une popularisation des procédures computationnelles pour la conception architecturale. Sa structure pédagogique, son interface ludique et son modèle ouvert lui assurent une place majeure dans le champ, McNeel a donc su s'assurer une place en tant qu'éditeur grâce au développement de Grasshopper, mais aussi grâce à l'utilisation de Rhinocéros comme modeleur dans d'autres domaines, comme la joaillerie. La première période de développement de Grasshopper correspond à une période florissante pour l'éditeur. Mais cette période ne dure pas : à partir de la fin des années 2000, McNeel entre dans une période de perte d'utilisateurs et de parts de marché. L'éditeur, qui doit faire face à ces difficultés. s'engage donc dans une transformation, qui touche en particulier Grasshopper. Le défi pour McNeel est alors de rendre l'environnement accessible au plus grand nombre pour étendre son marché, tout en gardant la communauté et les caractéristiques qui ont fait le succès de Grasshopper. McNeel a toujours occupé un espace différent de celui d'Autodesk dans le marché des éditeurs de logiciels. Le nombre d'utilisateurs que l'éditeur cherche à toucher n'est pas aussi grand, les possibilités géométriques offertes par son modeleur le place en position de toucher des professions différentes, et son statut d'entreprise détenue par des salariés lui donne une organisation très différente. Le prix des licences très varié vu un peu plus haut illustre déjà cette recherche d'autres catégories d'utilisateurs. Mais au sein du champ computationnel, McNeel rencontre cependant des enjeux similaires à ceux auxquels est confronté Autodesk : il lui faut également mobiliser des stratégies pour assurer son expansion, voire sa survie. Comme pour Autodesk, cela se traduit en une stratégie au niveau des outils et des ressources que nous allons examiner. Grasshopper évolue donc également à la faveur des dynamiques de développement du champ, pour devenir un logiciel et des ressources un peu différents de leurs premières versions.

Autodesk et McNeel ne sont pas les seuls éditeurs de modeleurs 3D sur le marché, et certains des autres éditeurs s'intéressent au champ computationnel depuis plusieurs années. C'est le cas de Bentley Systems, qui avait su s'imposer un temps avec Generative Components. Mais Grasshopper ayant peu à peu remplacé le logiciel, Bentley se positionne finalement sur une niche différente : Generative Components sert désormais essentiellement à la modélisation d'infrastructures et d'espaces de production divers. Parmi les éditeurs principaux de l'industrie de la construction et de l'architecture néanmoins, plusieurs montrent un intérêt croissant pour le champ computationnel. C'est le cas par exemple de Dassault Systèmes, qui lance une plateforme consacrée à la modélisation d'objets destinés à être imprimés en 3D, ou de Trimble, qui rachète Digital Project. S'ajoutent à cela les développeurs à plus petite échelle que nous avons abordés plus tôt, qui éditent eux aussi des logiciels spécialisés pour le champ computationnel, comm HAL robotics qui propose des outils de contrôle robot ou Fologram, qui propose des outils de réalité augmentée pour assister les ouvriers sur les chantiers. Bien que les objectifs en termes de parts de marchés ne soient pas exactement les mêmes chez ces différents éditeurs, leurs stratégies de développement ont une forte influence sur le champ. Elles se répercutent aussi dans les cadres de transmission existants.

## 6.2 Cadres de transmission

### *6.2.1 Description des cadres de transmission*

Les formats autour desquels se structure la transmission des connaissances dans le champ relèvent principalement de deux catégories : les offres pédagogiques institutionnalisées et les ressources en libre-service. Les premières, des ateliers aux studios, sont développées en présentiel au sein des institutions universitaires du réseau, avec des temporalités et des publics différents en fonction des formats, et s'adressent essentiellement aux étudiants et aux chercheurs du monde universitaire. Les secondes sont généralement mises à disposition en ligne sous forme de tutoriels vidéos, de documents écrits ou de plateformes d'échanges, et sont destinées à tout praticien professionnel ou amateur désireux de s'initier à l'un ou l'autre des outils et des méthodes computationnelles mises en œuvre dans le champ.

Participant de l'essor du champ décrit dans les chapitres précédents, les offres pédagogiques se sont multipliées à partir des années 2000, s'appuyant sur les nombreux praticiens alors en activité, mais aussi petit à petit sur des nouveaux venus, héritiers de ces praticiens et formés dans les institutions déjà en place. Les groupes de recherche qui se constituent à travers le monde anglophone et l'Europe, mais aussi l'Asie, articulent de plus en plus fréquemment leurs recherches avec l'encadrement d'enseignements portant sur le sujet au sein de l'institution qui les accueille : des cours spécifiques

dans les formations encore généralistes de licence, dispensés à des étudiants en amont de leur spécialisation en master, et des masters et des écoles doctorales consacrés aux pratiques computationnelles en architecture. En 2011 se crée ainsi à Barcelone l'IAAC, qui propose aujourd'hui sept masters consacrés à divers aspects du champ, en 2012 est créé à Paris le département Digital Knowledge de l'ENSAPM et l'université de Singapour intègre à sa faculté d'architecture le programme Architecture and Sustainable Design, qui fait la part belle aux techniques de conception computationnelles. A Copenhague, CITA ouvre en 2013 un cursus de master dédié, comme le fait en Serbie l'université de Novi Sad la même année. Dans les années qui suivent, c'est au tour des universités de Vienne, de Darmstadt, de Téhéran, de Penn State, du New York Institute of Technology ou encore de l'institut de technologie d'Israel Technion d'ouvrir des formations consacrées au champ computationnel en architecture.

En parallèle des institutions créant un master et/ou un programme doctoral, nombre d'autres intègrent l'apprentissage de savoirs du champ à leur programme de formation en architecture, sans faire de l'architecture computationnelle un sujet à part entière, mais en intégrant les techniques dans les pratiques de conception enseignées. Au-delà des trajectoires individuelles des praticiens et des groupes de recherches - qui demeurent de petites unités avec une certaine agentivité propre au sein des institutions qui les accueillent -, l'essor du champ permet de voir se dessiner différentes dynamiques de développement à plus grande échelle. Certaines institutions développent d'abord une offre d'enseignement conséquente avant de se tourner vers la structuration d'un groupe de recherche, aux Etats-Unis notamment : à Carnegie Mellon, Ohio State University ou Georgia Tech par exemple, où les pratiques computationnelles existent depuis longtemps autour de praticiens individuels - ceux-ci contribuant d'abord à l'enseignement général puis dans un second temps à des offres pédagogiques dédiées et enfin beaucoup plus tardivement, dans les années 2010, à des groupes de recherche "officiels", sur le modèle d'autres institutions dont ces unités ont fait le succès, comme CITA, l'ICD ou encore le BRG. Dans le même esprit, le pôle londonien, lui, donne naissance dès les années 70 à une série de masters dédiés, permettant aux écoles et en particulier à la AA d'être le pôle dominant depuis des années, mais très peu à des groupes de recherches à part entière - l'accent est ainsi mis sur l'enseignement plutôt que la recherche. Enfin, pour les plus tardifs, la recherche et l'enseignement démarrent en même temps. Les offres pédagogiques se structurent donc de plusieurs manières, tout en étant en nette augmentation ces dernières années : alors que seulement une vingtaine d'universités offraient des cursus autour de l'architecture computationnelle avant 2010, plus de soixante le font en 2020. Les offres pédagogiques se spécialisent par ailleurs de plus en plus, tout comme les groupes de recherche dont font partie les praticiens qui enseignent au sein de ces offres pédagogiques : l'ETHZ ou l'IAAC se concentrent sur les enjeux autour de la fabrication et la robotique, nombre des

universités américaines poursuivent leur travail sur les grammaires de formes, EmTech ou le Digital Structures Group s'intéressent particulièrement à la conception via des méthodes d'optimisation, CITA au champ du biodesign. On assiste donc à cette période à une multiplication et à une structuration plus nette des espaces de recherches et de leurs offres de formation, mais aussi à leur spécialisation.

Si les ressources en libre-service se multiplient elles aussi, dans un premier temps ce n'est pas accompagné d'une structuration : c'est même plutôt un ensemble assez désordonné d'informations concernant les divers outils que les praticiens du champ computationnel ont contribué à programmer au cours des années. Ces informations sont mises à disposition sur différents supports par des praticiens divers, et presque toujours en relation avec un outil donné, plutôt qu'avec une typologie ou une approche. Ces outils sont alors pour beaucoup des formats intermédiaires - plug-ins, bibliothèques, scripts pré-écrits -, avec toute la flexibilité d'usage qu'ils impliquent[40], mais aussi la difficulté à en saisir les modalités de fonctionnement. Ceci encourage certaines modalités de transmission, comme on l'a vu, mais complique le développement d'une documentation appropriée, car dans la mesure où ces formats favorisent l'appropriation et l'adaptation par les utilisateurs, il faut pouvoir expliciter la paramétrisation via des fichiers en libre service figés plutôt que des explications orales ou des démonstrations en direct. Cette documentation est de plus chaotique, quand elle existe au-delà de simples fils de discussions sur les forums, et composée d'éléments hétéroclites qui varient en fonction des outils : des descriptions écrites des possibilités, des illustrations diagrammatiques du fonctionnement ou des exemples de résultats, des vidéos explicatives, des fichiers exemples, des instructions d'installation. Celles-ci sont parfois particulièrement compliquées, comme c'est le cas des plug-ins Ladybug et Honeybee, qui nécessitent l'installation de quatre logiciels, d'un plug-in supplémentaire pour l'un d'entre eux, de deux formats différents de fichiers complémentaires pour l'installation du plug-in Grasshopper lui-même, mais aussi le déverrouillage de l'accès à certains de ces fichiers et à certains des programmes de logiciels, l'installation d'un plug-in Grasshopper complémentaire en fonction des versions du logiciel utilisée, et enfin, pour faire fonctionner ne serait-ce qu'un fichier exemple, le téléchargement d'un fichier de données météorologiques complémentaires à placer à un endroit bien particulier[41]. Ces instructions sont de plus disponibles dans un fichier annexe qu'il faut savoir aller consulter plutôt que de suivre directement la méthode usuelle d'installation d'un plugin Grasshopper - cette dernière semblant fonctionner au premier abord mais ne permettant en fait pas de faire fonctionner l'analyse climatique et solaire que permettent Ladybug et Honeybee. Si cet exemple est extrême, il n'en est pas moins illustratif de l'opacité du fonctionnement

[40] Voir Chapitre VI.
[41] Instructions d'installation de Ladybug et Honeybee. Voir https://honeybee.readthedocs.io/en/latest/getting_started/index.html, consulté le 13 Novembre 2021.

de toute une partie des outils disponibles, que la documentation disponible ne permet pas de dissiper. Les contenus proposés dans la documentation sont généralement ceux jugés pertinents par les développeurs plutôt que des explications systématiques, par exemple dans les fichiers de démonstration souvent (mais pas toujours) fournis en accompagnement des plugins Grasshopper : c'est au développeur de choisir ce qu'il va montrer des différentes possibilités d'utilisation de son outil, en l'absence de retours des utilisateurs.

La documentation est hébergée dans des endroits divers - sur les sites personnels des développeurs, sur des plateformes d'échange de programmes comme Github, sur différents sites en relation avec une même plateforme de programmation - dans le cas de Processing, le site de l'outil[42] et le forum des utilisateurs[43], dans le cas de Grasshopper, le forum des utilisateurs[44], le site de McNeel[45] et le site Food4Rhino[46] qui sert de bibliothèque des plug-ins disponibles. Les informations nécessaires pour apprendre à maîtriser un outil donné, bien que supposées accessibles à n'importe quel utilisateur intéressé, ne sont donc pas toujours faciles à dénicher. Ainsi des éléments disponibles pour comprendre, par exemple, comment utiliser Kangaroo Physics, le plug-in de simulation physique le plus populaire de Grasshopper[47] : fichiers d'installation et fichiers exemples, documentation, tutoriels vidéo et discussions sur les forums, ainsi qu'une poignée d'articles scientifiques. À l'exception de ces derniers, ces éléments sont recensés sur le site sur le site de Kangaroo[48] et téléchargeables ou consultables éclatés sur les sites suivants : Food4Rhino, McNeel, Grasshopper 3D, le site personnel, la page Github, la page Vimeo de Daniel Piker - programmeur du plugin -, ainsi que sur les grandes plateformes vidéo usuelles pour les tutoriels proposés par d'autres et sur les grandes plateformes de la recherche pour les articles - ces derniers étant réservés à un public de chercheurs plutôt qu'à un public d'utilisateurs amateurs. Food4Rhino propose une description en quatre phrases du plug-in, un lien vers les fichiers d'installation, un espace de discussion des problèmes d'installation et un lien vers le site de McNeel, ainsi qu'un lien, cassé, vers un forum d'assistance, sans même donner d'images de modélisations produites grâce à l'outil, comme cela se fait pourtant usuellement sur Food4Rhino. Le lien vers le site de McNeel renvoie vers la partie Kangaroo du forum de discussion McNeel, sachant qu'il existe également des archives de discussion concernant Kangaroo sur le forum de discussion Grasshopper 3D, qui contient aussi une description un peu plus longue de l'outil - une demi-page, avec un lien lui aussi cassé vers une autre page pour plus de détail. Cette description date néanmoins

[42] https://processing.org/, consulté le 13 Novembre 2021.
[43] https://openprocessing.org/, consulté le 13 Novembre 2021.
[44] https://www.grasshopper3d.com/, consulté le 13 Novembre 2021.
[45] https://discourse.mcneel.com/c/grasshopper/2, consulté le 13 Novembre 2021.
[46] https://www.food4rhino.com/en, consulté le 13 Novembre 2021.
[47] Voir Chapitre V pour les chiffres.
[48] http://kangaroo3d.com/, consulté le 13 Novembre 2021.

de 2015 alors que la version actuelle du plugin date de 2019. La page Vimeo de Daniel Piker présente des vidéos de démonstration de ses utilisations de Kangaroo, mais sans aucune hiérarchie des vidéos ou explication du plug-in rendue immédiatement disponible. Le dernier post du blog, qui mélange par ailleurs des posts sur plusieurs plugins historiques de Grasshopper, date de 2014[49]. Sur Github se trouvent enfin des fichiers exemples, un peu plus régulièrement mis à jour que le reste, et la documentation officiel du plugin : un pdf explicatif de neuf pages où sont décrits les différents composants - des informations qu'on retrouve aussi dans le répertoire des composants Grasshopper, accessible depuis le forum[50] - et quelques précisions sur les changements du programme pour la version de 2017[51], mais toujours aucun descriptif des principes de Kangaroo permettant d'éclairer un utilisateur totalement néophyte.

Ces documents sont donc non seulement éparpillés un peu partout en ligne, ce qui ne facilite pas leur consultation, mais sont par ailleurs dans l'ensemble assez sommaires, et ne permettent jamais vraiment d'obtenir une vue d'ensemble des principes de fonctionnement et donc des possibilités de l'outil, qu'un utilisateur néophyte ne peut donc que découvrir en observant d'autres utilisateurs plus chevronnés dans des contextes de rencontres, ou en écumant ces ressources en ligne hétéroclites. Tutoriels et discussions sur les forums sont une source importante d'informations plus détaillées, mais ne deviennent facilement exploitables qu'avec suffisamment d'expérience pour naviguer sur les plateformes pour trouver ces conseils, puis pour être à même de les mettre en œuvre. Kangaroo Physics n'est pas un exemple isolé en la matière, la plupart des plug-ins de Grasshopper sont documentés de la même manière et posent donc les mêmes problèmes. Si dans le cas de Grasshopper, les ressources disponibles sont particulièrement chaotiques, l'équipe qui gère Processing a su proposer des ressources structurées de manière un peu plus systématique, ce qui permet de trouver plus d'explications globales sur le rôle des bibliothèques, des ressources plus centralisées et une recherche plus facile sur le site ou les scripts sont partagés. La mise en ligne des ressources se fait cependant dans l'ensemble sur la base d'une participation individuelle peu encadrée. Au-delà des deux grandes familles de ressources autour des outils du champ computationnel que sont Grasshopper et Processing, le reste de ce qui est disponible compte essentiellement des outils uniquement gérés par leurs concepteurs et des morceaux de programmes qui circulent d'un utilisateur à un autre sans espace d'échange collectif en ligne qui centraliserait les ressources.

La documentation n'est par ailleurs pas toujours élaborée par les praticiens qui sont à l'initiative des outils eux-mêmes, il s'agit parfois simplement des ressources mises à disposition par d'autres, ainsi

[49] https://spacesymmetrystructure.wordpress.com/, consulté le 13 Novembre 2021.
[50] https://grasshopperdocs.com/, consulté le 13 Novembre 2021.
[51] Date du passage à Kangaroo 2 et de l'écriture du fichier.

des nombreux tutoriels disponibles pour Grasshopper, dont seulement une poignée est mise en ligne par David Rutten et son équipe, les autres étant proposés par divers autres praticiens. Les diverses documentations sont donc produites sans vraiment d'initiative de supervision globale, et l'apparition des ressources fluctue donc plus en fonction des contributeurs qu'en fonction des besoins, à l'exception des indications fournies sur les forums ou des scripts parfois transmis en réponse à une question posée sur ces plateformes. Certaines des ressources en libre-service disparaissent parfois au gré des évolutions techniques d'internet et de l'obsolescence de certains sites que l'absence de maintenance condamne, ou encore du vieillissement de certains scripts qui ne sont plus compatibles - comme par exemple les fichiers exemples fournis sur les forums pour Kangaroo 1, plus compatibles avec la seconde version du plugin alors qu'une bonne partie d'entre eux fait la démonstration de fonctions de base essentielles dans les deux versions, mais que peu s'attachent à mettre à jour alors que les fils de discussion en question sont vieux de plusieurs années. Nombre des plug-ins de Grasshopper ont été conçus par des praticiens simplement passés à autre chose après le développement et la mise en ligne de l'outil et de quelques éléments de documentation, qui restent donc sommaires et figés malgré les évolutions techniques des méthodes computationnelles liées.

Ces fluctuations sont une situation classique des milieux open-source et des modèles d'échange des connaissances qu'ils prônent, modèle qui caractérise l'émergence de ces premiers ensembles de ressources en lignes. Les espaces d'échange que sont les forums témoignent aussi de ce modèle : ce sont des espaces horizontaux, où seules les connaissances préalables sur un sujet ou l'autre hiérarchisent les échanges, et où tous les profils de connaissances sont présents et ont les mêmes possibilités de contribution. C'est aussi le cas d'une partie des espaces associatifs où se retrouvent physiquement les praticiens du champ, par exemple le Smart Geometry Group à ses débuts. Cette structure horizontale favorise les échanges informels entre tous, et ces espaces sont donc des communautés ouvertes où les discussions portent tant sur le développement des outils que sur leur usage, et ce au sein d'une même plateforme et communauté. Ainsi Grasshopper et son forum sont (initialement) comme les autres le support d'une communauté de pratiques ouverte à tous, permettant une collectivisation des connaissances de programmation et de conception algorithmique de l'espace. Les groupes d'échanges sont alors multiples, et permettent une circulation des connaissances comme des fichiers, la plupart des praticiens investis dans le développement d'outils étant ouverts au partage de ces outils, mais aussi à la modification de certains scripts en fonction des remarques et des besoins exprimés par certains collègues[52]. Les offres pédagogiques institutionnalisées et les ressources en libre-service telles qu'elles sont structurées au début de la phase de démocratisation qui s'ouvre dans les années 2010 participent donc d'un réseau dont l'ouverture garantit une forme d'équilibre et un

[52] Pas d'utilisateurs donc mais des collègues.

espace d'exploration assez libre des enjeux du champ computationnel.

Par le biais des offres pédagogiques et des ressources en libre-service telles qu'elles existent est assurée la transmission d'une certaine maîtrise de la programmation, mais seulement à certains profils, poursuivant les recherches prospectives du champ computationnel - l'équilibre est donc préservé à la condition d'un certain entre-soi[53]. Si ces praticiens sont somme toute peu nombreux et évoluent dans une bulle universitaire qui constitue toujours un espace préservé, une nette augmentation du nombre de praticiens est observée dans cette bulle, dont le réseau s'étend donc comme d'autres champs universitaires au moment de leur phase d'expansion. Les formats de transmission tels qu'ils existent sont donc satisfaisants dans la mesure où ils permettent de former des praticiens qui poursuivent l'objectif d'expérimentation du potentiel des outils algorithmiques en architecture établi par la génération précédente, et dans la mesure où les connaissances qu'ils permettent de transmettre permettent une symétrie entre les différents profils de connaissances présents au sein du champ, et donc de préserver l'équilibre du réseau tel qu'atteint à la génération précédente. La phase suivante de développement du champ computationnel est néanmoins marquée par l'apparition d'autres objectifs, qui accompagne celle des nouveaux profils décrits plus haut. Si la transmission de savoir-faire précis propres au champ est toujours d'importance, la progression professionnelle prends pour les praticiens individuels les plus anciens d'autant plus d'importance que la constitution des pratiques computationnelles comme un champ à part entière permets une reconnaissance plus grande, tandis que les nouveaux profils se penchent sur l'enjeu de démocratisation des outils. Transmettre, au sein du champ computationnel, c'est faire parvenir un savoir-faire qui repose sur des connaissances tacites à deux niveaux : celui de la programmation et celui de la conception architecturale. Démocratiser, d'autre part, c'est rendre accessible au plus grand nombre d'utilisateurs possible - quelles que soient les conditions d'utilisation et y compris si ces conditions d'utilisation sont en partie un renoncement à l'enjeu de transmission. Or ces nouveaux objectifs ne sont pas tous atteints par les dispositifs d'apprentissage déjà en place.

### *6.2.2 Un ensemble qui permet une transmission parfois trop minutieuse*

Nous avons vu les problèmes que posent les ressources en libre-service existantes pour l'apprentissage de la programmation appliquée à la conception spatiale - des problèmes que les échanges en direct des offres pédagogiques permettent en partie de contourner. Ces dernières soulèvent cependant également des difficultés : la transmission des savoirs tacites liés à la conception

[53] Comme c'est le cas dans les réseaux makers d'ailleurs. Voir Camille Bosqué, *La fabrication numérique personnelle, pratiques et discours d'un design diffus : enquête au cœur des FabLabs, hackerspaces et makerspaces de 2012 à 2015*, thèse de doctorat, Université Rennes 2, 2016.

par la programmation s'accommode mal des formats classiques d'enseignement en école d'architecture, que les praticiens sont néanmoins contraints d'adopter puisque c'est précisément dans ces milieux-là qu'ils évoluent[54]. L'enseignement des pratiques computationnelles a généralement lieu en studio, au cours d'un semestre d'initiation à ces sujets. Le problème principal est celui de la densité des connaissances à assimiler au cours d'un seul semestre, un temps assez court si l'on considère que le nombre de choses que les étudiants doivent saisir. Il s'agit d'apprendre à programmer alors qu'ils n'ont généralement pas de connaissances en la matière, mais aussi de saisir la logique qui préside au recours à des outils algorithmiques - la formulation d'instructions explicites de conception donc - alors même qu'il n'y a pas de manière claire collective d'exprimer les différences potentielles entre cette logique et la logique de conception usuelle enseignée en école d'architecture. Les étudiants ont par ailleurs souvent été formés durant plusieurs semestres à cette dernière puisque l'enseignement des pratiques computationnelles en projet se fait essentiellement en master. Il est rare que les enseignants insistent au cours des semestres précédents sur la formulation d'instructions explicites quelles qu'elles soient - l'exercice est donc nouveau pour les étudiants. Il faut également continuer la formation à la conception spatiale générale que les étudiants sont encore en train de suivre et qui est l'objectif premier des écoles d'architecture. Enfin il s'agit aussi pour les étudiants de se former à la culture du champ computationnel, d'en assimiler les références ainsi que d'acquérir un regard critique sur ces pratiques, or celui-ci est difficile à affiner sans une maîtrise pointue de la programmation, dont l'apprentissage est pourtant tout juste commencé. Si d'autres classes sont aussi consacrées aux pratiques computationnelles, elles comportent moins d'heures de formation, et leur présence dans la grille d'enseignement et donc la possibilité de les combiner avec un studio est par ailleurs déterminée en partage avec d'autres champs disciplinaires de la formation des architectes[55], les pratiques computationnelles étant considérées justement comme un champ et non comme un ensemble de techniques et méthodologies transversales. De plus, le parcours des étudiants au sein de l'école varie au gré des options sélectionnées, ce qui tend à cantonner leur apprentissage des pratiques computationnelles à ce seul studio d'initiation, et éventuellement à un ou deux autres cours, séminaires dédiés ou tutoriels techniques. Si les masters complets dédiés aux pratiques computationnelles permettent de contourner une partie de ces problèmes, cela ne fonctionne que pour les étudiants qui décident de s'investir vraiment dans le champ - il s'agit là du parcours des profils héritiers.

[54] Des milieux dont l'inertie face à des changements potentiels de formats d'enseignement est d'autant plus patente que d'autres spécialisations architecturales ne nécessitent aucun changement des formats, au contraire.
[55] À l'exception d'écoles comme l'IAAC, dédiées aux questions computationnelles, qui restent néanmoins très rares.

Les premiers studios du champ computationnel s'appuient sur les formats intermédiaires que sont les plugins et bibliothèques pour apprendre la programmation aux étudiants. Ces formats représentent cependant une marche d'apprentissage plutôt haute pour une initiation d'un semestre à la programmation destinée à des étudiants sans connaissances dans le domaine. Il faut donc proposer des étapes d'apprentissage commençant avec des interfaces plus simples d'utilisation ou des exemples d'utilisation des plugins et des bibliothèques très simples - ces derniers présentant le risque que les étudiants ne parviennent pas à des connaissances suffisamment poussées pour développer un projet architectural à l'aide de ces outils. Interfaces, outils et fichiers utilisés pendant le semestre prennent du temps à être élaborés pour garantir un bon équilibre entre diversité et profondeur des savoirs transmis. De plus, les praticiens qui enseignent sont encouragés à renouveler régulièrement le thème des studios et l'exercice de conception proposé aux étudiants. Cette pression provient des attendu des écoles sur le renouvellement de l'offre des studios, mais aussi de l'injonction à créer ses propres outils qui existe dans le champ computationnel, et de l'utilisation qui est faite de la production des studios par les enseignants-chercheurs, souvent considérée comme des résultats professionnels à part entière pour ces derniers. Renouveler le thème des studios chaque semestre permet donc à ces praticiens d'étendre leur portfolio. Or les algorithmes doivent être adaptés aux projets pour plus de pertinence : au projet de chaque étudiant, auquel la responsabilité échoit donc de cette adaptation, mais aussi, pour l'enseignant, au sujet du semestre. Les offres pédagogiques impliquent de ce fait un effort conséquent et répété de la part des enseignants pour mettre à disposition des outils avec une structure bien équilibrée pour l'apprentissage et sur mesure pour le sujet chaque semestre, alors même que les étudiants risquent de manquer de temps pour les maîtriser complètement.

Ces cours s'appuient donc par ailleurs parfois également sur les ressources en libre service, qui posent leurs propres problèmes concernant l'apprentissage des processus de conception computationnels. Ce recours s'explique aussi par le fait que l'effort demandé aux praticiens enseignants est peu reconnu. Ces derniers pointent leurs réticences croissantes à s'investir dans des formats de transmission qui ne sont pas du tout satisfaisants sur le plan de leur trajectoire et de leur reconnaissance professionnelle[56]. Les nombreuses bibliothèques, plugins, scripts et vidéos d'explications détaillées de ces derniers mis au point par ces praticiens et ensuite partagés non seulement à leurs étudiants mais aussi en ligne sont souvent reprises et utilisées par d'autres praticiens et étudiants du réseau, et ce sans créditer les auteurs initiaux - alors que c'est pourtant une des exigences principales des licences open-sources sous lesquelles sont partagées ces ressources. Ces derniers en conçoivent du ressentiment : les praticiens interrogés soulignent tous leur déception face au devenir de ces outils et scripts partagés[57].

[56] Entretiens.
[57] Entretiens.

Ceci touche en particulier ceux à l'origine des bibliothèques Processing les plus utilisées - par exemple Jose Sanchez, Iain Maxwell, Robert Stuart-Smith ou Ezio Blasetti. Si les développeurs de l'écosystème Grasshopper comme David Rutten, Giulio Piacentino ou Daniel Piker bénéficient d'un peu plus de reconnaissance en regard de leur apport à la banque d'outils du champ - tant dans la presse spécialisée qu'au sein des espaces de la communauté Grasshopper où ils sont très actifs -, ils font aussi partie d'un group plus restreint et plus centré autour de leurs champ et apports[58], et ils ont également une carrière correspondant mieux à leurs attentes.

En effet, les praticiens dont la réticence à contribuer régulièrement s'exprime le plus fortement dans les entretiens sont aussi ceux qui ont le plus de difficulté dans leur carrière, notamment parce qu'ils se sont attachés à concilier pratique et recherche, et que seule cette dernière s'est développée autant qu'espéré. Nous avons en effet vu que les trajectoires professionnelles des praticiens de cette génération relèvent majoritairement de deux milieux : professionnel, avec des praticiens évoluant au sein de grandes agences et bureaux d'études, ou universitaire, avec des praticiens dont la source de revenu et l'occupation principale sont des postes d'enseignants-chercheurs. Ces derniers combinent souvent leurs travaux de recherche avec une petite agence leur permettant de continuer à pratiquer et à produire des projets en dehors de l'espace universitaire : c'est le cas de Sixteen*(Makers), kokkugia, supermanoeuvre, biothing, EZCT, ou encore Mesne. Or l'activité de ces agences, qui se réclament de l'architecture computationnelle, est restreinte à certains types de projets seulement : à part quelques commanditaires visionnaires, des projets relevant plutôt du domaine artistique ou quelques expositions dédiées à ces pratiques, peu de personnes dans le monde de l'architecture consensuelle s'intéressent à ces sujets. Au contraire, on s'en méfie plutôt. Au-delà de projets ponctuels et de la participation à des concours, l'accession à la commande classique s'avère donc compliquée. En conséquence, le début des années 2010 voit ces agences se transformer pour s'adapter aux possibilités d'accession à la commande existantes.

Certaines, comme Matsys ou TheVeryMany, sont très à l'aise dans l'espace particulier des projets possibles et acceptent de s'y cantonner, rencontrant un certain succès. D'autres choisissent de changer de discours, et tout en continuant à recourir à des outils algorithmiques, n'en font plus le point central dans l'explication de leur pratique, à l'image d'asymptote, d'ijpcorporation ou de supermanoeuvre[59]. Les praticiens de ces agences continuent à évoluer au sein du champ computationnel, échangeant avec leurs pairs au sujet des méthodes et des enjeux de la pratique, enseignant au sein des institutions qui offrent un espace pour développer ces travaux et participant à des expositions avec des projets

[58] La communauté Grasshopper est très centrée autour de l'architecture et du design computationnels alors que Processing brasse une plus grande diversité d'usagers.
[59] Entretien Iain Maxwell.

spécialisés. Les projets portés par leurs agences en dehors de ces espaces ne contiennent cependant aucune mention du recours à des outils computationnels : le discours qui s'adresse aux milieux les plus consensuels est expurgé des arguments qu'on retrouve ailleurs à ce sujet. C'est une stratégie qu'ils assument, en réponse à la méfiance ou au désintérêt montré par ces milieux pour les questions computationnelles. D'autres agences se transforment plus profondément, renonçant non seulement au discours mais aussi aux pratiques computationnelles. Elles délaissent le milieu computationnel et ses projets alternatifs au profit de commandes classiques, et leurs pratiques de conception comme leur communication sont expurgées de ces questions. C'est par exemple le cas pour Aranda/Lasch, dont le dernier projet qui recourt à des outils algorithmiques est Morning Line, pavillon fractal réalisé pour la fondation d'art contemporain Thyssen-Bornemisza entre 2008 et 2013, ou de MOS, qui s'est largement tourné vers la construction de programmes résidentiels après 2008. C'est également ce qui se produit en partie au sein du réseau OCEAN, qu'Achim Menges quitte pour diriger l'ICD à Stuttgart, mais aussi pour travailler avec l'agence Scheffler + Partner, dont la pratique est beaucoup plus consensuelle que celle de l'association. Enfin, un assez grand nombre d'agences cessent leur activités, comme Sixteen*(Makers), Mesne, DR_D ou encore f-u-r. Si les praticiens ne quittent pas le champ computationnel, ils se tournent comme Phil Ayres, Bob Sheil ou Tim Schork vers la recherche et l'enseignement, ou, comme Ezio Blasetti, Axel Kilian ou Evan Douglis vers une position de freelance, oscillant entre workshops et positions dans l'enseignement, écriture de scripts pour d'autres agences encore en activité, projets de design, d'art ou d'architecture d'intérieur.

La difficulté dont les praticiens font part à être reconnus pour leur apport et à vivre de cette pratique, la faiblesse de la reconnaissance du travail engagé dans le modèle open-source engendre petit à petit la transformation ou la disparition des agences computationnelles nées au début des années 2000. Ceci, combiné aux difficultés rencontrées pour parvenir à enseigner la conception spatiale par la programmation, a pour conséquence l'affaiblissement ou la disparition des profils qui étaient attachés à une transmission du savoir-faire computationnel et de sa dimension tacite. Alors que les studios se multiplient, les formats intermédiaires sont de plus en plus souvent abandonnés au profit d'outils prêts à l'emploi qui facilitent l'enseignement. En agence aussi, les outils prêts à l'emploi rencontrent plus de succès que les formats intermédiaires. En effet, si la transmission fonctionne en partie dans les espaces pédagogiques universitaires, avec les étudiants les plus investis, sur le plan professionnel ces formats intermédiaires ne rencontrent que peu de succès. Évoluant en dehors des offres pédagogiques, les praticiens ne recourent donc qu'aux ressources en libre-service, sans forcément avoir le temps de s'y retrouver en raison des impératifs que leur impose la pratique en agence. Les environnements d'agences s'accommodent mal des formats de transmission en place : envoyer un petit groupe de chercheurs travailler sur des recherches prospectives n'est pas faire adopter un nouvel outil à une

agence de 200 personnes au planning régi par les commandes. Les outils prêts à l'emploi, conçus pour répondre à ce besoin, permettent de donner accès à ces praticiens aux outils algorithmiques, dont le fonctionnement nécessite néanmoins encore un peu d'apprentissage. L'objectif de démocratisation nécessite donc en complément des nouveaux outils d'autres cadres pour enseigner le recours aux algorithmes, ou au moins une évolution des cadres existants.

*6.2.3 Une mue des formats annonciatrice de changements dans le champ*

Dans un premier temps, une évolution de la documentation se produit. Les praticiens y ayant le plus contribué ayant conscience des faiblesses du modèle open-source ouvert et peu cadré, la première moitié des années 2010 voit l'apparition d'initiatives visant à l'établissement de ressources en libre-service un peu plus structurées. Ceci résulte d'abord des efforts de praticiens individuels produisant des contenus dans la continuité des logiques open-source précédentes, mais mieux structurés, comme les vidéos proposées par Plethora Project pour les langages de programmation les plus répandus dans le champ, de Grasshopper à Processing en passant par Python, ou les manuels d'utilisation pour Grasshopper publiés par Zubin Khabazi, accompagnés de scripts de démonstration. Ces offres sont pensées comme des introductions complètes et détaillées à la logique et aux possibilités des outils et permettent plus facilement aux débutants d'avoir une vue d'ensemble. Ensuite, les nouveaux plugins Grasshopper sont documentés de manière plus systématique également au niveau individuel, avec des ensembles de tutoriels vidéos eux aussi structurés pour permettre une approche plus globale aux utilisateurs. Les retours des utilisateurs sont aussi mieux pris en compte, via des plateformes dédiées qui permettent aux développeurs de mieux les centraliser. C'est le cas par exemple pour WASP, qui offre un ensemble très complet de vidéos d'initiation aux différentes méthodes d'agrégation de géométries que propose le plugin, ainsi qu'un espace Slack et une newsletter dédiée. La spécialisation dans les sous-champs, qui implique la mise au point par chaque groupe d'outils sur-mesure pour ses pratiques de prédilection, engendre également des changements : la distance entre ce que montrent les exemples d'utilisation et les possibilités de l'outil se réduit à mesure qu'ils se spécialisent, puisque les outils ciblent des opérations algorithmiques beaucoup plus précises et délimitées. La documentation de nombreux plug-ins prend également de plus en plus souvent la forme d'articles de conférence ou de journal, publications présentant l'avantage non négligeable pour les enseignants-chercheurs du champ d'être open-source, mais surtout d'être pris en compte comme métrique de leur efficience professionnelle, ce qui les pousse à en produire. Enfin, ces supports vont être proposés par la suite dans le cadre d'initiatives encadrées par les groupes les plus établis : ainsi des tutoriels vidéos de DDU, de Compas, la grande base d'outils génératifs de l'ETHZ supervisée par le Block Research Group, ou de son équivalent en cours de développement à l'ICD de

Stuttgart, tous deux accompagnés d'une documentation beaucoup plus développée que précédemment.

Les espaces d'échange et d'apprentissage accessibles aux praticiens évoluent également, en particulier ceux des ressources en libre-service et les forums. Le développement de Revit et le lancement de Dynamo par Autodesk s'accompagnent d'un site offrant de nombreuses ressources pour apprendre à manipuler les deux programmes, et d'un forum d'échange pour les utilisateurs. La réorientation de Grasshopper et de ses contenus par McNeel s'accompagne quant à elle à partir de 2017 d'un nouveau forum hébergé sur le site de McNeel et de la mise à l'arrêt du forum Grasshopper 3D, même si les ressources qui y sont proposées sont toujours accessibles. Ces nouveaux espaces sont bien plus fermés et rigides que les précédents, et encouragent surtout la présence de profils amateurs provenant d'agences d'architecture plutôt consensuelles, et de profils de professionnels du développement d'outils informatiques. La répartition des profils de connaissances au sein de cette nouvelle organisation est beaucoup plus hiérarchisée. La présentation de Dynamo sur le site de Revit est beaucoup plus directive que sur le forum Grasshopper, qui conserve malgré les changements d'orientation un coté fouillis et une invitation à la débrouille : contrairement au site de Revit, il n'y a pas de page *Explore* qui présente en détail les fonctionnalités, juste un discret *About* de quelques lignes, finissant par une invitation à télécharger le programme. Plutôt que d'en apprendre plus - "learn more" -, Grasshopper propose de s'essayer directement à l'exercice - "things to try" - et renvoie dès sa page d'accueil au forum et à des échanges avec d'autres utilisateurs plutôt qu'à des cours. Revit met par ailleurs en avant des cas d'études réalisés par des agences de grande taille plutôt que par des utilisateurs individuels, et fait de ses programmes une présentation très générique, là où Grasshopper montre des applications précises et personnalisées tout de suite. Enfin, chez Revit, les ressources sont toutes sur le même site et créées par Autodesk, quand chez Grasshopper la page *Learn* renvoie vers de nombreuses autres pages, et les supports d'apprentissage sont le fruit d'utilisateurs.

Les espaces sont par ailleurs scindés en deux : d'une part des groupes de développeurs restreints en nombres et en type de profils, contribuant à la mise au point des outils et d'autre part des groupes d'utilisateurs dans des espaces de discussion portant sur les usages. Dynamo et ses plug-ins ont par exemple été développés par un groupe avec des échanges sur le modèle de Smart Geometry, mais restreignant à l'inverse l'accès à des profils de développeurs avec une formation dans le domaine, et à des praticiens formés au génie civil s'étant tournés vers la programmation d'outils. Nombre des outils les plus récents, notamment ceux ayant recours à des réseaux neuronaux, sont développés par une ou deux personnes au sein d'une start-up, qui créent des groupes de testeurs constitués de ces mêmes profils, en nombre restreint et peu médiatisés sur les plateformes ouvertes aux autres types de profils.

Spacemaker a par exemple mis en place un espace Slack[60] pour ses bêta-testeurs qui n'est répertorié sur aucun des sites des associations du champ computationnel ni sur aucun des sites usuellement fréquentés par les utilisateurs. Sur le site de Revit, la section des développeurs leur est réservée, elle est séparée du reste du site, et elle requiert un processus d'inscription plus complexe - ce qui fait écho aux stratégies de développement par des tiers d'Autodesk beaucoup plus qu'à l'aide par les pairs mis en place par McNeel pour Grasshopper. Dans les espaces accessibles aux utilisateurs, les conseils se font de manière beaucoup plus verticale, avec le premier groupe - les experts -, répondant aux questions du deuxième - les amateurs -, les premiers étant identifiés par un badge Revit et une indication de leur rôle. En comparaison, sur les deux forums de discussion Grasshopper, même David Rutten a un profil identique à celui de n'importe quel autre utilisateur, et sur le premier il n'est même pas précisé qu'il s'agit du développeur du programme. Les formes d'interaction sont elles aussi limitées, et les conversations assez différentes, beaucoup plus fermées, avec des interventions limitées des développeurs et modérateurs là où la conversation était beaucoup plus ouverte précédemment. Dans l'ensemble, la structure beaucoup plus rigide de ces nouveaux espaces d'échange, leur verticalité et leur compartimentation montre une hiérarchisation beaucoup plus forte entre les praticiens, qui contribue là aussi à limiter l'autonomie des utilisateurs et l'agentivité des nouveaux arrivants.

Les offres pédagogiques évoluent également : on observe l'augmentation des masters professionnalisants et l'apparition d'offres de formation privées, en dehors du monde universitaire. Les éditeurs de logiciels proposaient déjà des formations à leurs solutions, mais les offres consacrées aux pratiques computationnelles se multiplient, tout comme les masterclass ou les workshops proposés par les agences - non seulement dans les conférences comme auparavant mais aussi en interne. Autodesk offre par exemple des formations dans ses technocentres, et organise même pendant un temps sa propre conférence, l'Autodesk Computational Design Symposium, dont le format est similaire aux autres conférences mais qui donne la parole essentiellement à des praticiens du monde professionnel. Cette privatisation de l'offre s'accompagne de stratégies de promotion pour étendre le pool d'utilisateurs, qui prennent le dessus sur les stratégies de transmission et favorisent d'autres outils. Certaines entreprises, comme Dassault ou Kuka[61], offrent un accès privilégié à leurs logiciels à des praticiens prescripteurs évoluant dans les milieux de formation et de recherche. Cet accès privilégié se fait souvent au moment du développement du logiciel ou d'une mise à jour importante, et permet aux entreprises de bénéficier des conseils des praticiens, mais aussi de les pousser à recommander ensuite l'outil. Dans le milieu étudiant, des dispositifs similaires d'incitation à l'usage

[60] En tout cas avant son rachat par Autodesk.
[61] Pour Dassault, archives personnelles ; pour Kuka (fabricant de robots dont les modèles sont assez répandus au sein du champ computationnel) conversation avec Samim Mehdizadeh.

existent : Adobe laisse les cracks de logiciels proliférer sans effectuer de contrôle, contrairement à la surveillance exercée sur les professionnels, Autodesk propose des licences étudiantes gratuites, McNeel offre un prix étudiant très bas pour Rhinocéros. Ceci donne l'habitude aux praticiens de se reposer sur ces solutions logicielles dès les premières années de leur formation, encourageant l'achat de licences une fois les étudiants passés dans le monde professionnel tout en conservant leurs habitudes de travail. Les outils les plus répandus au sein du champ computationnel sont d'ailleurs ceux pour lesquels cet accès est le plus facilité : d'autres sont moins populaires dans le milieu architectural, à la fois parce qu'ils ne mettent pas en place ce genre d'accès facilité[62], parce qu'ils ont des interfaces moins simples d'usage, parce que les méthodes de modélisation utilisées conviennent moins à une partie des architectes, et parce qu'ils se concentrent sur les pratiques de modélisation d'autres secteurs. La structuration des ressources open-source mentionnée plus haut témoigne elle aussi de l'apparition de stratégies de promotion en remplacement de celles de transmission : elle montre le passage d'initiatives pour faire vivre le partage à des initiatives pour communiquer sur ses activités, développement le plus récent du devenir vitrine de l'objet code.

Les nouveaux formats ont pour conséquence une rupture du cycle d'apprentissage et un bouleversement du transfert des connaissances tacites tel qu'il s'était organisé précédemment. Là où les structures précédentes permettaient de pousser des utilisateurs initialement novices vers une maîtrise fine des outils, leur évolution permet simplement un accès à plus d'utilisateurs, sans vraiment initier un apprentissage poussé de la programmation. Les conséquences de ces nouveaux formats portent essentiellement sur les connaissances tacites liées à la programmation, les connaissances tacites liées à la conception architecturale n'étant pas directement concernées[63]. Si les deux familles d'espaces analysées permettent l'acquisition de connaissances tacites liées à l'usage d'un outil, la seconde le permet seulement dans la limite des fonctionnalités identifiées et cadrées par le groupe de développeurs. Les connaissances tacites acquises ne portent donc pas sur les affordances que permet la conception algorithmique, il s'agit simplement d'une aptitude à manipuler les fonctionnalités d'un logiciel. Les connaissances tacites liées à la structuration d'un modèle computationnel et par extension au développement d'un outil, qui étaient favorisées par la première famille d'espaces, est empêchée ou restreinte dans la seconde. Ici aussi les discussions des forums sont parlantes : au sein des espaces de discussion Grasshopper, les explications techniques sur la structure de l'algorithme à mettre en oeuvre sont privilégiées, alors que dans les espaces consacrés à Dynamo et Revit, une information sur l'utilisation de l'interface - sur quel bouton cliquer - est fournie. De plus, la structure même de Grasshopper et le discours qui l'accompagne continuent à encourager la prise en main de l'utilisateur

[62] Du moins pas dans le champ computationnel.
[63] Même si nous allons plus loin examiner l'idée que les nouveaux outils et formats sont en fait des vecteurs épistémologiques et idéologiques.

et sa trajectoire vers la conception d'outils. La structure de l'outil n'a pas changé, ce sont plutôt l'accès aux éléments et les ressources d'apprentissage qui ont changé. Même si les ambitions de McNeel pour Grasshopper ont évolué, sa structure de base reste beaucoup plus favorable à une autonomie de ses utilisateurs[64].

Les communautés de pratiques sous leur forme la plus récente ne peuvent plus vraiment être qualifiées de communautés d'apprentissage. Certaines caractéristiques essentielles[65] disparaissent en effet : relation de pouvoir et responsabilité des apprenants, mise au centre ponctuelle de chacun des apprenants et co-construction du discours. Les situations identifiées comme clés pour la transmission des connaissances dans le modèle SECI[66] diminuent ou disparaissent également. L'externalisation est ciblée : seuls certains ensembles de connaissances explicites sont rendus accessibles. Les possibilités de combinaison de deux ensembles de connaissances explicites diminuent significativement, notamment en raison de l'accent mis sur les fonctionnalités des outils pour les utilisateurs. La socialisation, supposée permettre le transfert direct de connaissances tacites par l'observation, n'est pas permise aux amateurs novices qui évoluent en dehors de la bulle, alors que ce sont justement eux que visent les nouveaux outils et formats. Cette étape est pourtant identifiée dans le modèle SECI comme le point de départ du cycle d'apprentissage. Même si la linéarité du modèle et ce choix de première étape ont été critiqués par la suite[67], c'est pourtant assez conforme à ce qu'on observe dans les pratiques du champ computationnel : les praticiens racontent tous une expérience qui démarre par une phase de socialisation[68]. Le premier contact avec les pratiques du champ via les ressources en libre-service pourrait être associé à une phase de socialisation, mais au vu de l'expérience décrite par les praticiens, cela relève plutôt de l'internalisation. Cette dernière est donc la seule étape encore présente qui implique d'apprendre des connaissances tacites, mais sans socialisation les connaissances tacites n'ont aucune chance de devenir un édifice collectif, alors que c'est un enjeu clé pour le champ computationnel[69] - enjeu auquel l'objectif de démocratisation contraint donc de renoncer.

Les enjeux de démocratisation , de plus en plus présents dans le champ, impliquent une adaptation des formats dominants, qui y font obstacle au début des années 2010. Ce changement accompagne l'apparition des outils prêts à l'emploi, et hiérarchise fortement les espaces des communautés du

[64] Voir Chapitre IV.
[65] Bielaczyc K, Collins A. Learning Communities in Classrooms: Advancing Knowledge For a Lifetime. NASSP Bulletin. 1999;83(604):4-10.
[66] Nonaka, Ikujiro, R. Toyama, and N. Konno. (2000). "SECI, Ba, and leadership: a unified model of dynamic knowledge creation." Long Range Planning 33:5–34.
[67] Gourlay, Stephen (2003), "The SECI model of knowledge creation: some empirical shortcomings", 4th European Conference on Knowledge Management, Oxford, England, 18-19 Sep 2003
[68] Entretiens Fabio Gramazio, George Legendre.
[69] Voir Chapitre IV.

champ computationnel. Les pratiques open-source, les espaces flottants, et les praticiens engagés dans leur maintenance perdent de leur emprise, et la bulle dans laquelle ils évoluent se referme sur elle-même. S'il entraîne l'augmentation du nombre d'utilisateurs, il tend aussi à freiner l'acquisition des connaissances tacites et la prise d'autonomie menant à la conception de ses propres algorithmes et outils, ce qui ajoute à l'effet des nouvelles interfaces et engendre une rupture des chaînes de transmission. La démocratisation des outils computationnels se fait donc au détriment du développement individuel des connaissances tacites comme de leur devenir collectif. Cette nouvelle génération marque un tournant dans les outils mais aussi dans les pratiques. Bien que le réseau dans sa forme précédente encourage une diversité de pratiques, parmi lesquelles celles relevant de la rationalisation, ce sont essentiellement ces dernières qui seront reprises et popularisées après l'apparition des nouveaux acteurs et des nouveaux enjeux. En effet, si Autodesk s'intéresse au nouveau potentiel commercial de ces outils, ce n'est pas parce qu'une révolution paradigmatique et esthétique de l'architecture est en marche suite au tournant computationnel, mais parce que ces outils sont un vecteur extrêmement efficace de rationalisation des pratiques architecturales.

## 6.3 Une mue des outils

### *6.3.1 De nouvelles stratégies d'apprentissage et de popularisation*

Nous avons vu plus haut que Autodesk a bien compris, au cours des années 2000, le poids croissant pris par les pratiques computationnelles dans le champ logiciel architectural. Le développement de Generative Components puis de Grasshopper durant ces années n'est pas passé inaperçu, et le plus gros des éditeurs de logiciel s'intéresse à la mise en place d'une solution équivalente. Ou plus exactement, une interface de programmation visuelle, certes, mais avec l'approche d'Autodesk. La solution logicielle ambitionnée par l'éditeur est en effet plus proche de Generative Components que de Grasshopper, dans le sens où elle vise le même milieu d'utilisation : l'industrie plutôt que l'université. Le choix d'Autodesk pour le modeleur à associer à cette interface de programmation visuelle se porte donc sur Revit, que l'entreprise est alors en train de transformer en support de rassemblement de l'ensemble des techniques CAD dont elle dispose. Cette stratégie paye, puisque Revit devient au cours des années 2010 le logiciel CAD le plus utilisé du monde. Sélectionner Revit pour y intégrer cette nouvelle approche est donc un choix logique pour Autodesk, dans la continuité de sa stratégie de développement. La programmation visuelle rencontre donc le BIM, dont Revit est le fer de lance chez Autodesk. En effet, Revit n'a pas pour caractéristiques principales, comme Rhinocéros ou Generative Components, un kernel basé sur des splines permettant une puissante exploitation géométrique des courbes, mais plutôt une mise à disposition directe des sous-produits de la construction les plus répandus. L'approche de la conception architecturale y est donc très différente,

le rôle joué par le logiciel n'est pas du tout le même, et plutôt qu'une exploration des possibilités formelles apportées par la modélisation numérique et paramétrique, il facilite le respect des contraintes de l'industrie de la construction. De plus, penser un module de programmation visuelle pour le plus usité des logiciels CAD implique que Autodesk vise un public différent de celui qui est alors utilisateur de Generative Components ou Grasshopper. En effet, au-delà du petit noyau de programmeurs expérimentés dont les pratiques computationnelles sont essentiellement l'apanage quand Autodesk se lance dans ce projet, l'entreprise vise des programmeurs débutants qui se servent de langages comme AutoLISP, voire même des novices complets, qui n'évoluent que dans la première couche d'interface de Revit. Autodesk a donc pour son nouveau projet des ambitions proches de celles de ses deux grands frères sur le plan de l'intégration de la programmation visuelle, mais vise un public très élargi, se confrontant donc à un enjeu d'apprentissage plus important que jamais.

Autodesk recrute donc Robert Aish, le développeur à l'origine de Generative Components, pour le faire travailler sur le projet. Ce dernier a de grandes ambitions, qu'il articule autour de DesignScript, un nouveau langage de programmation qu'il imagine pour Revit. En effet, Aish n'imagine pas simplement réaliser un équivalent de Generative Components ou de Grasshopper pour Revit. Il souhaite mettre en place une plateforme beaucoup plus ambitieuse de formation aux pratiques computationnelles. Celle-ci s'articule autour d'une courbe d'apprentissage sur six niveaux. Le premier est celui de la programmation visuelle, qui rend facilement accessible la logique de base de la conception computationnelle. Le second niveau fait convertir par les utilisateurs certains nœuds de programmation visuelle en code écrit. Ceci permet de réduire la taille des fichiers de programmation visuelle et de les naviguer plus facilement lorsque le processus de conception se complexifie. Cela permet également aux utilisateurs d'apprendre la programmation en ligne de commande en comparant les deux logiques de travail à petite échelle[70]. La troisième étape est la conversion de l'ensemble des nœuds de programmation visuelle en code textuel. Le programme fonctionne toujours avec la même structure et toujours par associativité, c'est-à-dire en associant des caractéristiques à des ensembles d'objets, mais l'utilisateur passe d'une interface de programmation visuelle à une interface qui fait la part belle à l'écriture des instructions. L'étape suivante consiste à faire basculer ces instructions dans une logique de programmation différente, la programmation impérative. Plutôt que de décrire des objets et leurs caractéristiques, en laissant ensuite un autre programme déterminer la suite d'instructions à exécuter pour parvenir à ces caractéristiques comme le fait la programmation associative, la programmation impérative saisit directement les instructions à exécuter. La cinquième

[70] *Learning by observing the correspondence between the DesignScript notation and geometry, for example, by comparing the geometric model with the graph based symbolic model and with the DesignScript notation displayed in the IDE* - Robert Aish, "DesignScript: Origins, Explanation, Illustration." In: Gengnagel C., Kilian A., Palz N., Scheurer F. (eds) *Computational Design Modelling*. Springer, Berlin, Heidelberg, 2011

étape consiste à créer des fonctions à partir de ces séries d'instructions. Plutôt que de recourir aux fonctions programmées par d'autres, l'utilisateur est alors en mesure de se créer ses propres descriptions d'objets et de caractéristiques, et de maîtriser la réalisation de la conversion en instructions à exécuter. Il bénéficie donc des avantages qu'offre la programmation associative, du confort d'écriture que ce niveau de langage offre, tout en ayant la capacité d'intervenir plus finement. Enfin, la dernière étape est celle de la mise à disposition d'autres utilisateurs des outils ainsi créés[71]. L'objectif de cette courbe d'apprentissage est de combiner la pente douce que permet la programmation visuelle au début de son utilisation avec le progrès rapide que permet la programmation en ligne de commande dans la seconde partie de son utilisation. Ceci permet d'éviter la courbe d'apprentissage très raide de la programmation en ligne de commande, mais aussi le plateau de la programmation visuelle dans la seconde partie de son utilisation.

Un des éléments clés de la courbe d'apprentissage pensée par Robert Aish est la combinaison d'une interface de programmation visuelle avec un nouveau langage, qui combine plusieurs caractéristiques. Son concepteur le décrit comme "*un langage de programmation spécifique à un domaine, destiné à l'utilisateur final, multi-paradigme, indépendant de l'hôte et extensible*"[72]. Ces caractéristiques correspondent à différentes spécificités des pratiques du champ computationnel telles que nous les avons observées jusqu'ici. DesignScript est spécifique à un domaine au sens où c'est un langage de programmation destiné à être utilisé par des architectes et des ingénieurs, il possède donc des fonctions propres aux besoins de ce domaine d'application. Il est extensible au sens où ses utilisateurs peuvent y ajouter des fonctionnalités. Il est indépendant de l'hôte au sens où il est pensé comme un langage-colle, qui permet d'établir des passerelles entre de multiples logiciels ou environnements. Il est destiné à des utilisateurs finaux, donc pas à des programmeurs professionnels, ou plutôt pas seulement. Robert Aish l'explique ainsi :

---

[71] *1. Visual dataflow programming provides an easily accessible approach for simple computational design problems*
*2. node to code : as the number of nodes and arcs increases, a region of the visual data flow graph can be converted to text based code, thus reducing visual complexity and 'seeding' the user's code with the logic previously developed in visual data flow programming*
*3. Text based data flow programming*
*4. Hybrid data flow imperative programming*
*5. Encapsulation via user functions*
*6. C# classes added using zero touch*
Robert Aish and Emmanuel Mendoza. 2016. DesignScript: a domain specific language for architectural computing. In Proceedings of the International Workshop on Domain-Specific Modeling (DSM 2016). Association for Computing Machinery, New York, NY, USA, 15–21.

[72] *a concise but somewhat complex description of DesignScript might be as a domain-specific, end-user, multi-paradigm, host-independent, extensible programming language* - Robert Aish, "DesignScript: Origins, Explanation, Illustration." In: Gengnagel C., Kilian A., Palz N., Scheurer F. (eds) *Computational Design Modelling*. Springer, Berlin, Heidelberg, 2011

*DesignScript est destiné à être utilisé par des concepteurs expérimentés ayant un large éventail de compétences en programmation, allant des non-programmeurs (qui pourraient indirectement programmer via une manipulation interactive directe), aux programmeurs novices non professionnels (utilisateurs finaux), et aux concepteurs expérimentés qui ont une expertise substantielle en programmation. Plus généralement : Un langage d'utilisateur final ajoute une syntaxe simplificatrice au langage, tout en réduisant certaines des restrictions souvent associées aux langages à usage général (destinés aux programmeurs expérimentés)*[73].

Il identifie ici l'ensemble des profils de connaissances qui peuplent le champ, et DesignScript a donc pour objectif d'accommoder leurs différentes modalités de pratique en une seule plateforme. Le langage est aussi multi-paradigme, au sens où il associe plusieurs paradigmes de programmation. Nous avons vu que la courbe d'apprentissage est supposée permettre d'apprendre à naviguer entre un modèle de programmation associatif et un modèle impératif, les deux ayant leurs avantages pour la conception computationnelle selon Aish[74]. Certains détails dans les choix de structuration du langage faits par Aish trahissent cette volonté d'associer plusieurs méthodes. Il décrit par exemple dans un article dédié à DesignScript ce qu'il nomme le principe de réplication, une manière de saisir, en ligne de commande, une collection de valeurs numériques plutôt que des valeurs individuelles en entrée, qui s'inspire de la manière dont des utilisateurs novices le font en programmation visuelle. Ceci a pour but de créer des passerelles entre les différents niveaux de l'environnement de programmation qu'Aish conçoit, et de faciliter le voyage des utilisateurs entre ces différents niveaux. Comme Processing l'avait fait avec Java, DesignScript se saisit de la structure d'un langage déjà existant, en l'ajustant et en le simplifiant pour mieux l'adapter aux attendus d'une profession. DesignScript est ainsi la cristallisation de l'expérience d'Aish dans le champ, à la fois dans l'identification des enjeux que pose l'apprentissage de la conception computationnelle, et dans l'expérience acquise avec les différents langages et environnements de programmation en usage dans le champ.

Robert Aish résume ses ambitions pour DesignScript par les trois points suivants : ce doit être un outil de modélisation efficient pour produire et évaluer des géométries complexes, un langage de programmation complet pour programmeurs expérimentés, et un outil pédagogique pour aider les

---

[73] *DesignScript is intended to be used by experienced designers with a wide range of programming skill, ranging from non-programmers (who might indirectly program via interactive direct manipulation), to novice non-professional (end-user) programmers, and to experienced designers who have substantial expertise in programing. More generally: An end user language adds simplifying syntax to the language, while reducing some of restriction often associated with general purpose languages (intended for experienced programmers).* - Robert Aish, "DesignScript: Origins, Explanation, Illustration." In: Gengnagel C., Kilian A., Palz N., Scheurer F. (eds) *Computational Design Modelling*. Springer, Berlin, Heidelberg, 2011

[74] Robert Aish, DesignScript: a learning environment for Design Computation In : *Proceedings of the Design Modelling Symposium*, 2013.

designers à devenir des programmeurs compétents[75]. Ces ambitions s'inscrivent dans la continuité de son travail avec Generative Components, et pleinement dans la négociation entre accessibilité et liberté qui se trouve au cœur des pratiques computationnelles en architecture. Aish vise néanmoins très large avec DesignScript, puisque les usagers novices visés par Autodesk sont supposés apprendre la programmation sans aucune autre aide que le logiciel lui-même et un peu de documentation, c'est-à-dire sans formation véritable à la programmation pour devenir des programmeurs compétents. Ce qui différencie les profils de connaissance des programmeurs les plus aguerris du champ est pourtant souvent l'existence dans leur parcours d'une formation, même brève, à la programmation. Les profils comme David Rutten ou Robert Aish qui n'ont été formés qu'au design et se sont ensuite tournés vers une activité de développeurs existent, mais ne sont pas nombreux. La courbe d'apprentissage de DesignScript est donc un défi non négligeable, bien qu'elle soit le fruit d'une confrontation à l'enjeu principal du développement du champ computationnel. La question de l'apprentissage est cruciale dans la genèse de DesignScript, mais aussi celle du passage à l'échelle[76], c'est-à-dire de l'industrialisation. C'est notamment la-dessus que repose la stratégie d'Autodesk de développer une interface de programmation visuelle pour leurs propres logiciels et leur propre marché. La programmation visuelle rencontre au sein de Revit le BIM, ce qui permet à Aish de développer largement ses premières intuitions sur la question alors qu'il travaille sur DesignScript. En plus du besoin d'un système pragmatique et efficace, il identifie trois enjeux pour ce saut d'échelle : la multiplicité des niveaux computationnels, gérée par la courbe d'apprentissage, mais aussi la taille des projets[77] et la multiplicité des disciplines et des intervenants, caractéristiques de l'industrie. Le développement de DesignScript repose donc autant sur une ambition d'apprentissage que sur une ambition de transmission des pratiques computationnelles au sein de l'industrie de l'architecture conventionnelle[78].

Si Robert Aish arrive chez Autodesk en 2007, la gestation de DesignScript est très lente. Elle se fait de la même manière que pour Generative Components chez Bentley Systems, avec un petit groupe de praticiens aux premières loges pour tester l'environnement de programmation au fur et à mesure de

---

[75] *DesignScript is intended to be:*
*• a production modeling tool: to provide an efficient way for pragmatic designers to generate and evaluate complex geometric design models*
*• a fully-fledged programming language: as expected by expert programmers.*
*• a pedagogic tool: to help pragmatic design professions make the transition to competent programmer by the progressive acquisition of programming concepts and practice applied to design*
Robert Aish, "DesignScript: Origins, Explanation, Illustration." In: Gengnagel C., Kilian A., Palz N., Scheurer F. (eds) *Computational Design Modelling*. Springer, Berlin, Heidelberg, 2011

[76] "scalability"

[77] 10^6 composants donné comme ordre de grandeur.

[78] Robert Aish, "DesignScript: Scalable Tools for Design Computation", in Stouffs, Rudi and Sariyildiz, Sevil (eds.), *Computation and Performance – Proceedings of the 31st eCAADe Conference – Volume 2*, Faculty of Architecture, Delft University of Technology, Delft, The Netherlands, 18-20 September 2013, pp. 87-95.

son développement[79]. Autodesk n'organise néanmoins pas le moindre atelier ou conférence qui puisse ouvrir un peu ce groupe de praticiens à d'autres personnes du champ computationnel. Le développement de DesignScript se fait donc dans le plus grand secret. En 2011, quelques informations sur ce qu'est ce projet en cours de création sont finalement publiées par Autodesk, mais DesignScript en tant que tel ne verra jamais le jour. Robert Aish quitte l'entreprise peu après, sans que l'outil soit en mesure d'être publié[80]. En 2012, Autodesk recrute le développeur Ian Keough, qui prend la relève d'Aish dans la gestion du développement de ce nouvel outil. Mais surtout, Ian Keough, formé à la sculpture et à l'architecture à Parsons avant d'exercer chez Buro Happold, amène avec lui Dynamo, un outil de programmation visuelle qu'il a développé pour Revit. L'ensemble des niveaux de programmation envisagés par Robert Aish et basés sur DesignScript est donc complété par Ian Keough et son équipe chez Autodesk grâce à Dynamo. Le développement de Dynamo remonte en fait à la période où Ian Keough occupe un poste de développeur spécialisé chez Buro Happold à Los Angeles. Il y travaille sur de nombreux projets dans Grasshopper, produisant des modèles 3D pour des agences d'architecture variées. Une grande partie du travail consiste à intégrer dans Grasshopper les remarques des architectes et les évolutions de la géométrie en fonction de la structure, pour transmettre ensuite un modèle mis à jour à l'agence. A l'occasion d'une collaboration avec l'agence d'architecture HOK pour un projet de gare multimodale à Anaheim, en Californie, Ian Keough et les équipes de Buro Happold sont confrontés à une demande inédite. HOK leur demande de réaliser la modélisation de la gare entièrement sur Revit, malgré la géométrie du toit et des coussins d'ETFE qui le composent (fig. 6), qui rendent la modélisation manuelle très difficile. Ayant conscience de la facilité qu'il y aurait à exécuter la tâche avec un module de programmation visuelle intégrée à Revit, et de combien le dialogue serait facilité avec les architectes, Ian Keough se lance dans la programmation de ce nouvel outil. Il n'en est pas à son coup d'essai, puisqu'il a déjà développé en freelance, indépendamment de son travail chez Happold, plusieurs outils spécialisés, rachetés par divers éditeurs de logiciels. En 2011, une première version de Dynamo est testée par une poignée

[79] *Autodesk's 'acquisition' of CAD visionary Robert Aish is about to reach maturity and the return on investment is looking a little shaky. After three years working at Autodesk, Robert Aish has tentatively been previewing his latest thesis: DesignScript.*
*Although you wouldn't know it.*
*The release of DesignScript parallels Aish's release of Generative Components while at Bentley: presentations and papers at conferences; faux secrecy; and an extended private alpha with invited practitioners. It worked for Generative Components in 2003 when the computational design community was primarily limited to an old boys club of conferences and invited projects. In 2011 it remains to be seen if you can successfully launch a project while ignoring the expanding computational design community online – there is currently no details of DesignScript on the internet and it took me a month of emailing Autodesk to finally get a video of DesignScript (which they posted publicly here).*
Daniel Davis, "DesignScript – Autodesk", 27 Juin 2011, https://www.danieldavis.com/designscript-autodesk/, consulté le 13 Novembre 2021.

[80] Les sources consultées n'ont permis de donner d'explication ni au départ de Robert Aish de chez Bentley Systems alors que Generative Components sortait tout juste, ni à son départ de chez Autodesk avant que DesignScript ne soit achevé.

d'utilisateurs. L'outil est alors totalement indépendant d'Autodesk. Ian Keough en parle néanmoins à une de ses connaissances, Matt Jezyk, alors chargé du développement des outils de design génératif chez Autodesk. Alors que Keough, manquant de temps pour continuer à développer Dynamo, s'apprête à mettre tout le code de l'outil en open-source pour que d'autres puissent y contribuer, Jezyk lui propose de faire travailler quelqu'un de chez Autodesk directement sur Dynamo. Ian Keough rejoint l'entreprise peu après, et y reste cinq ans, finalisant cette pièce manquante pour le système imaginé par Aish. En effet, bien que Keough ait clairement dit que son intention initiale était de fournir à Revit une simple copie de l'interface de Grasshopper[81], son parcours entre architecture, ingénierie et développement logiciel lui procure une bonne expérience des enjeux de transmission qui se pose alors dans le champ computationnel. Non seulement son passage au sein de Buro Happold lui vaut une conscience aiguë des difficultés rencontrées en agence avec le recours aux outils computationnels, mais il a également travaillé sur le développement d'outils BIM. Aish fait partie des pionniers du BIM et en a imaginé quelques-uns des mécanismes principaux, mais Keough a l'expérience du quotidien des praticiens avec ces outils, ce qui lui permet de parfaitement saisir les ambitions d'Autodesk pour Dynamo. En témoigne cet extrait de la biographie utilisée pour le présenter un peu partout :

> *Il écrit du code depuis son garage pour automatiser la génération de l'environnement bâti afin d'aider les parties prenantes à prendre de meilleures décisions plus rapidement. (...) Ian est convaincu que l'efficacité engendre la qualité et que l'automatisation permet d'obtenir des produits de meilleure qualité*[82].

La courbe d'apprentissage mise au point par Robert Aish est donc ce sur quoi repose l'intégration de Dynamo à Revit et toute la structure actuelle des niveaux d'utilisation du logiciel. Autodesk le met néanmoins peu en avant dans sa documentation et sa communication, l'essentiel des informations fournies à ce sujet le sont par Robert Aish dans ses publications. Autodesk réserve plutôt ces informations aux profils disposant des connaissances en programmation les plus développées, c'est-à-dire aux profils qui ont la possibilité de contribuer à son écosystème logiciel. Des nouvelles stratégies d'apprentissage et de popularisation des outils algorithmiques font en effet partie de la hiérarchisation des espaces d'échange et des espaces d'accès aux ressources que nous avons vus plus

---

[81] Podcast Getting Simple, épisode 9 : "Ian Keough - How to make Better Decisions Faster", Aout 2018, https://open.spotify.com/episode/0Q5y5YWOdBW95XCZgtQ1xS?si=Z9YuDxY7QuSOcHGBn05pmg&nd=1, consulté le 13 Novembre 2021.

[82] *He writes code from his garage to automate the generation of the built environment to help stakeholders make better decisions faster. Trained as a fine artist and architect—and known as The Father of Dynamo—Ian believes efficiency breeds quality and automation yields better, higher-quality products.* - par exemple ici : https://podtail.com/da/podcast/the-getting-simple-podcast/-9-ian-keough-how-to-make-better-decisions-faster/, consulté le 13 Novembre 2021.

haut. Celle-ci s'accompagne également d'un travail de mise à disposition des outils pour de nouveaux utilisateurs, en fonction de cette hiérarchisation. L'effort de regroupement par Autodesk de toutes ses offres pour l'industrie de l'architecture au sein de Revit, y compris les outils computationnels a pour effet de mettre Dynamo et DesignScript à disposition d'un nombre beaucoup plus massif d'utilisateurs. En intégrant Dynamo d'office à Revit, pourtant utilisé par un grand nombre d'utilisateurs à d'autres fins que la conception computationnelle, les outils dédiés à celle-ci se retrouvent accessibles à tous les utilisateurs du logiciel, donc à de potentiels nouveaux utilisateurs pour Dynamo. McNeel applique le même principe en intégrant d'emblée Grasshopper à Rhinocéros 6 en 2018. L'intégration systématique par la suite de Galapagos, le solveur qui permet de recourir à des algorithmes génétiques, et de Kangaroo, le moteur physique qui permet de réaliser des simulations variées, dans Grasshopper relève de la même stratégie. Optimo, le solveur de Dynamo, est lui aussi intégré d'office à l'écosystème à la même période. Algorithmes génétiques et simulations physiques sont justement les plus prisées et les plus faciles  d'accès des typologies, et permettent de convaincre facilement des utilisateurs novices. Les premiers sont appréciés en raison des possibilités d'optimisation des contraintes industrielles qu'ils offrent, les seconds pour la combinaison entre respect des contraintes et recherche de forme qu'ils permettent. Les deux offrent par ailleurs des mécanismes plutôt ludiques pour des débutants : les algorithmes génétiques permettent de parcourir un grand nombre d'options différentes, et les rebonds et relâchements des modélisations physiques en fonction des paramètres non seulement permettent d'explorer des formes variées, mais sont aussi amusantes à regarder.

*Figure 6. Terminal d'Anaheim*

Alors que Processing, ou Grasshopper initialement devaient être téléchargés sciemment et installés pour que leurs utilisateurs commencent à explorer ces environnements, l'intégration permet à Autodesk ou McNeel d'augmenter leur nombre d'utilisateurs *potentiels*. La figure 8 permet de mesurer l'augmentation du nombre de personnes ayant accès à ces outils algorithmiques. Nous pouvons y lire que Processing plafonne autour de 250 000 utilisateurs, et que Grasshopper avant son intégration à Rhinocéros 6 comptait environ 500 000 utilisateurs, tandis que Kangaroo avait été téléchargé 450 000 fois. L'intégration à Rhinocéros 6 rend Grasshopper et Kangaroo accessibles à près de deux millions d'utilisateurs. De la même manière, Dynamo est accessible aux trois millions d'utilisateurs de Revit, et si McNeel rassemble avec Rhinocéros des praticiens déjà relativement spécialisés dans les pratiques computationnelles, Autodesk revendique par ailleurs un bien plus grand nombre d'utilisateurs de ses produits auxquels ces outils algorithmiques pourraient être rendus accessibles selon les mêmes méthodes : dix-huit millions. Avec la croissance du nombre d'utilisateurs de Rhinocéros au cours des années 2010 et les stratégies de développement des outils algorithmiques mis en œuvre par Autodesk, l'accès aux outils de programmation du champ computationnel augmente très fortement à cette période. Les choix des éditeurs bénéficient néanmoins à certains outils plutôt qu'à d'autres. Dynamo et Grasshopper sont des outils de programmation visuelle, pas des environnements de programmation en ligne de commande, malgré les passerelles qu'ils sont supposés pouvoir permettre. Par ailleurs, le choix d'intégrer Optimo ou Kangaroo plutôt que d'autres plug-ins montre aussi que les éditeurs ont voulu mettre en avant certaines typologies plutôt que d'autres. Si Kangaroo atteint seul 450 000 téléchargements, et deux millions d'utilisateurs potentiels dans Rhinocéros, et Optimo trois millions dans Revit, les plug-ins Grasshopper permettant de recourir à des systèmes multi-agents n'atteignent que 65 000 téléchargements tout confondu. Le nombre d'utilisateurs de Maya, qui permet aussi de recourir à cette typologie, est du même ordre de grandeur. Autodesk n'a par ailleurs pour l'instant témoigné d'aucune intention d'intégrer des systèmes multi-agents à Revit, alors que les L-systèmes y sont par exemple déjà disponibles, bien que très peu utilisés. Ceci s'explique notamment par la différence entre les typologies et la faciliter de se saisir de certaines plutôt que d'autres pour les praticiens, comme nous l'avons vu au chapitre précédent, mais cela implique aussi que certaines typologies sont "condamnées" à garder peu d'utilisateurs en fonction des choix des éditeurs. Enfin, observer les groupes d'échanges consacrés aux différents plug-ins et typologies sur les différents forums montre que ces groupes d'échange ne comptent que quelques milliers d'utilisateurs actifs, soit un très petit nombre en regard de la grande quantité d'utilisateurs qui ont désormais accès à ces outils de conception algorithmiques - d'où la notion d'utilisateurs *potentiels* que cette stratégie de l'intégration offre à McNeel et Autodesk.

Cette stratégie fonctionne en partie, puisque comme nous l'avons vu plus haut le nombre d'utilisateurs dans le domaine computationnel augmente nettement au cours de ces années avec la popularisation des outils et le développement du champ. Mais d'une part certaines typologies ne voient pas leur nombre d'usagers augmenter, et d'autre part le profil des usagers plafonne : ce sont essentiellement des amateurs, qui le restent, malgré les courbes d'apprentissage soigneusement élaborées par David Rutten ou Robert Aish. Celles-ci sont soumises aux limites que posent les cadres d'usage évoqués en début de chapitre, mais aussi aux limites des formats d'outils sur lesquels elles s'appuient. En effet, la littérature sur le sujet met en évidence des dynamiques d'apprentissage parfois difficiles pour les utilisateurs de la programmation en ligne de commande comme pour ceux de la programmation visuelle[83]. Les études sur l'apprentissage de la programmation en ligne de commande montrent en effet qu'elle impose une courbe d'apprentissage exponentielle, avec un long plateau de peu de progression au début, puis une fois que la logique de programmation et la maîtrise de la syntaxe sont acquises, des progrès très rapides. Or, dans les stratégies de popularisation des outils mises en œuvre par les éditeurs, le pari est fait de laisser les usagers se débrouiller essentiellement seuls au sein des différents niveaux. La logique de conception par la programmation qu'il est nécessaire d'acquérir n'est pas ou peu cadrée dans l'approche des éditeurs, qui laissent leurs usagers l'apprivoiser eux-mêmes. La programmation en ligne de commande, ingrate à ses débuts, est donc peu prisée par ces utilisateurs seuls qui doivent démêler cette logique par eux-mêmes, et survivre seuls au long plateau du début de l'apprentissage. La programmation visuelle, au contraire, leur permet de mettre le pied à l'étrier en facilitant l'abstraction et l'acquisition de la logique. La courbe d'apprentissage décolle donc beaucoup plus vite que celle de la programmation en ligne de commande. Cette courbe plafonne néanmoins très vite, bloquant les utilisateurs dans un champ de possibilités beaucoup plus restreint. La programmation visuelle pose donc un problème considérable de montée en niveau des utilisateurs. C'est notamment le constat sur Grasshopper dans des études spécialisées comme par des critiques d'architecture dans des textes plus généralistes ou encore par Robert Aish ou Ian Keough[84].

---

[83] Celani G, Vaz CEV. CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison from a Pedagogical Point of View. International Journal of Architectural Computing. 2012;10(1):121-137. Leitao, A., Santos, L., Programming Languages for Generative design.Visual or Textual?, in: Zupancic,T., Juvancic, M.,Verovsek., S. and Jutraz,A., eds., Respecting Fragile Places, 29th eCAADe Conference Proceedings, University of Ljubljana, Faculty of Architecture (Slovenia), Ljubljana, 2011, 549-557.

[84] *The comparison between textual and visual programming languages showed that the later can lead to better results with novice architecture students. However, without any textual programming knowledge, applications are restricted to parametric explorations*. Celani G, Vaz CEV. CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison from a Pedagogical Point of View. International Journal of Architectural Computing. 2012;10(1):121-137.

*While Grasshopper's visual scripting language and word-of-mouth technical support makes it accessible, the learning curve quickly steepens as one approaches more complex problems. Moreover, its ease of initial entry can lead to a bypassing of the basics of programming. A user might be able to assemble a set of components quickly to produce a seemingly complex and variable geometric figure, but that user will soon reach a limit imposed by a lack of understanding of the way in which information is stored and sorted, and will require a*

Au-delà de ce plafond de progression, Rhinocéros pose également d'autres problèmes, notamment dans la définition géométrique des objets. Malgré son puissant kernel et son interface intuitive, le modeleur n'offre par exemple pas d'outil de détection des conflits géométriques, contrairement à nombre d'autres modeleurs du domaine de la construction et de la manufacture. Ce qui contribue à créer des problèmes de modélisation des projets. Les nombreux outils de reconstruction des maillages disponibles dans l'environnement permettent certes de les gérer, mais il s'agit d'un palliatif par un post-traitement des géométries, quand nombre d'autres modeleurs gèrent ce besoin en amont. Le fait que Rhinocéros soit un écosystème très ouvert, sans cadre très rigide d'encadrement des pratiques, pousse par ailleurs les utilisateurs à développer des solutions individuellement. L'autre problème essentiel de la programmation visuelle, c'est le passage à l'échelle. Au bout d'un trop grand nombre de nœuds, ou composants, assemblés les uns aux autres, la navigation du fichier de programmation s'avère ingérable[85]. Pour de très grands projets, impossible d'utiliser efficacement la méthode de hiérarchisation des informations qu'impose la programmation visuelle. Processing, en dépit d'atouts indéniables pour l'apprentissage de la programmation en ligne de commande, avec ses animations très rapides à mettre en place et son langage à la syntaxe facile à apprivoiser, se heurte au même problème de mise à l'échelle. Il est inenvisageable de modéliser dans cet environnement plutôt pensé pour l'art génératif et l'animation un projet de l'envergure du terminal HOK ou de la Fondation Louis Vuitton.

*more thorough understanding of fundamental concepts such as lists and indices.* - Mark Ericson; Review: Grasshopper Algorithmic Modeling for Rhinoceros 5. Journal of the Society of Architectural Historians 1 December 2017; 76 (4): 580–583

[85] Robert Aish, "DesignScript: Origins, Explanation, Illustration." In: Gengnagel C., Kilian A., Palz N., Scheurer F. (eds) *Computational Design Modelling*. Springer, Berlin, Heidelberg, 2011.


*Figure 7. Utilisateurs potentiels*

La programmation visuelle considérée comme une méthode géniale de popularisation de la programmation informatique et des pratiques computationnelles dans le domaine de l'architecture, si elle présente des avantages significatifs, pose donc en réalité des problèmes tout aussi conséquents. Penser cet apprentissage passe donc par un dilemme, que Ian Keough résume en pointant le niveau d'abstraction et l'interface parfaitement adéquats dont jouissent ces outils, mais les limites pour des programmeurs expérimentés et le problème du passage à l'échelle.

*C'est juste le bon niveau d'abstraction pour que des designers s'en saisissent. Cela ne convient pas pour des programmeurs plus avancés, car cela n'autorise pas le passage à l'échelle, mais cela permet aux gens d'entrer*[86].

C'est une manière de gérer ce dilemme que Robert Aish vise avec sa courbe d'apprentissage et sa combinaison des différentes méthodes de programmation.

*Si les diagrammes de graphes sont une technique extrêmement puissante qui permet aux programmeurs novices de créer leurs premiers modèles informatiques avec un minimum d'expérience et de compétences, il est généralement admis que la représentation des nœuds de graphes ne s'adapte pas à une logique plus complexe. En effet, la complexité visuelle du graphe peut devenir écrasante et contre-productive. C'est précisément à ce stade que l'application doit encourager le programmeur novice à passer du diagramme de nœuds de graphe au script : littéralement, du 'nœud au code'"*[87].

Mais Dynamo pose malheureusement les mêmes problèmes que Grasshopper et les autres outils de programmation visuelle. La passerelle imaginée par Aish ne fonctionne pour l'instant pas vraiment. Par ailleurs, Dynamo présente également des limites techniques encore conséquentes, malgré les efforts de développement consentis par Autodesk dans cet environnement. Enfin, le dernier problème que soulève Dynamo, c'est sa communauté et les outils partagés qu'elle produit : les packages, nom donné par Autodesk aux plug-ins du logiciel. Ceux-ci sont essentiellement des composants uniques ou des groupes de quelques composants programmés par des utilisateurs pour leur usage personnel. La structure de la plateforme de Dynamo rend ces composants publics, mais sans aucune explication quant à leur utilité. Le forum de Dynamo ne permet de plus pas de constituer des groupes d'échanges autour de package spécifiques, alors que c'est une des forces du premier forum Grasshopper. La communauté est d'autant plus verticale entre Autodesk et ses usagers, et n'offre pas le même soutien à des débutants. Les difficultés inhérentes aux cadres de transmission se conjuguent donc aux difficultés résultant des formats d'outils, créant le problème d'un apprentissage peu efficace de la conception computationnelle. Pour poursuivre le développement du champ, il faut donc trouver une alternative.

---

[86] Podcast Getting Simple, épisode 9 : "Ian Keough - How to make Better Decisions Faster", Aout 2018, https://open.spotify.com/episode/0Q5y5YWOdBW95XCZgtQ1xS?si=Z9YuDxY7QuSOcHGBn05pmg&nd=1, consulté le 13 Novembre 2021.

[87] *While graph diagramming is an extremely powerful technique to enable novice programmers to create their first computational models with the minimum of experience and skill, it is generally recognised that the graph node representation does not scale to more complex logic. Indeed the visual complexity of the graph may become overwhelming and counter productive. It is exactly at this point where the application should encourage the novice programmer to make the transition from graph node diagramming to scripting: literally from 'node to code'* - Robert Aish, "DesignScript: Origins, Explanation, Illustration." In: Gengnagel C., Kilian A., Palz N., Scheurer F. (eds) *Computational Design Modelling*. Springer, Berlin, Heidelberg, 2011

Petit à petit s'impose dans le champ une généralisation d'algorithmes prêts à l'emploi, reposant sur d'autres mécanismes d'usage que l'apprentissage progressif de la programmation par les utilisateurs. Les algorithmes prêts à l'emploi se caractérisent par la possibilité d'y recourir sans se frotter à beaucoup de programmation, en réglant simplement quelques paramètres clés préalablement à l'exécution d'un ensemble préétabli et pré-écrit d'instructions. Les modeleurs vus en détail au chapitre III fonctionnent sur ce même principe, puisqu'ils permettent de manipuler facilement des algorithmes de modélisation des surfaces complexes. Les typologies aussi ont des caractéristiques communes avec les algorithmes prêts à l'emploi, puisqu'elles permettent d'implémenter facilement des programmes grâce à une structure pré-définie. Les algorithmes prêts à l'emploi permettent néanmoins à l'utilisateur de se passer de l'écriture de cette structure prédéfinie, et donc de sa compréhension ou de la maîtrise de la syntaxe. Mais les algorithmes prêts à l'emploi ne sont pour autant pas exactement des solutions logicielles comme les modeleurs, logiciels comme tous les autres au sein desquels l'utilisateur n'a pas besoin de se confronter à la moindre notion d'algorithme ou de programmation, mais se contente d'utiliser le logiciel *via* son interface. Ces algorithmes prêts à l'emploi conservent une logique de programmation en donnant toujours à voir paramètres et structure de l'algorithme, tout en supprimant les difficultés d'usage liées à l'apprentissage de la programmation. Ils constituent ainsi une sorte d'entre-deux, à mi-chemin entre solutions logicielles standards et Far West de la programmation brute. L'interface joue donc un rôle prépondérant dans la définition des algorithmes prêts à l'emploi : elle y est d'une grande épaisseur[88], et la plupart des paramètres sont localisés à une grande profondeur. Les algorithmes sont alors encastrés dans des dispositifs logiciels complets qui vont bien au-delà des simples environnements de programmation initiaux. Ces solutions logicielles permettent d'encadrer le recours aux algorithmes par des mécanismes de facilitation de leur utilisation. Or, ce qui caractérise la mue de McNeel et l'aboutissement du travail d'Autodesk sur Dynamo, c'est justement l'implémentation au sein de leurs environnements de programmation de ces nouveaux mécanismes.

### *6.3.2 Des mécanismes de facilitation qui se généralisent*

Des nouvelles stratégies qui émergent dans le champ fait aussi partie le développement de mécanismes de facilitation de l'utilisation des outils computationnels. Nous avons vu le rôle joué par les typologies et les interfaces dans cette question, mais trois autres axes de travail sont investis par les éditeurs pour les outils les plus récents du champ : le management des flux de travail, la fonctionnalisation, et le guidage des utilisateurs. Les développeurs vantent par ailleurs largement cette nouvelle facilité d'usage dans la description de leurs outils. Voici quelques exemples de leurs descriptions :

[88] Voir Chapitre V.

*[Le plug-in aide] les designers à travailler plus efficacement et à obtenir d'excellentes performances dans leur travail. Il peut aider les entreprises à améliorer leur souplesse opérationnelle et à répondre rapidement aux besoins [des clients ou des partenaires]*[89].

*Le Reflector est une machine à réflections qui reflète automatiquement un maillage pendant un temps fini. Cet outil peut générer facilement des maillages complexes organiques en quelques secondes*[90].

*Avec CAALA, des variantes peuvent être créées et comparées en quelques secondes. La conception des bâtiments peut ainsi être optimisée efficacement sans effort supplémentaire*[91].

*Phoenix 3D permet de créer et d'explorer intuitivement des structures optimales en termes d'impact environnemental et autres objectifs de conception. Ses principes de conception permettent d'obtenir un feedback en temps réel et des variantes de design rapides*[92].

*Digitalis 3d vise à faciliter la conception numérique de structures en treillis complexes avec des propriétés locales et globales contrôlées. L'outil permet de générer automatiquement des treillis 3D avec des propriétés mécaniques, structurelles et de porosité finement ajustées*[93].

*Parakeet est une collection de composants pour la génération algorithmique de motifs ; l'outil offre une approche facile et unique qui génère des motifs et des réseaux géométriques et naturels*[94].

---

[89] *"[The plug-in helps] designers improve work efficiency, and [to] have excellent performance in their work. It can help enterprises improve business agility and quickly respond to the needs of [clients or partners]".* https://www.food4rhino.com/en/app/sunflower, consulté le 13 Novembre 2021.

[90] "*The Reflector is a reflection machine that automatically reflects a mesh for finite times. This tool can easily generate intricated organic meshes within seconds*". https://www.food4rhino.com/en/app/reflector, consulté le 13 Novembre 2021.

[91] *"With CAALA, variants can be created and compared in a matter of seconds. Building designs can thus be optimized efficiently without additional effort".* https://www.food4rhino.com/en/app/caala-rhino, consulté le 13 Novembre 2021.

[92] *"Phoenix3D enables to intuitively create and explore optimum structures in terms of environmental impact and other design goals. Its [underlying] concept allows for real time feedback and fast design variations".* https://www.food4rhino.com/en/app/phoenix3d, consulté le 13 Novembre 2021.

[93] *"Digitalis 3d aims to facilitate digital design of intricate lattice structures with controlled local and global properties. It supports smart and flexible automatic generation of 3d lattices with finely tuned local structural, mechanical and porosity properties".* https://www.food4rhino.com/en/app/digitalis3d, consulté le 13 Novembre 2021.

[94] *"Parakeet is a collection of components focusing on Algorithmic Pattern Generation; it offers a unique and easy-to-use approach that Generates Geometrical and Natural Patterns/Networks".* https://www.food4rhino.com/en/app/parakeet, consulté le 13 Novembre 2021.

*Automatisez de nombreuses tâches répétitives dans Revit (...) Vous trouverez un ensemble d'outils remarquables pour automatiser votre flux de travail quotidien et augmenter votre productivité dans la conception comme dans la production de documents d'exécution.*[95]

Ces promesses se retrouvent tant dans la description des nouveaux plug-ins Grasshopper que dans celle des packages Dynamo. Rapidité, automatisation, facilité : plus besoin de fournir beaucoup d'efforts pour manipuler des outils algorithmiques et faire émerger des propositions architecturales. C'est particulièrement flagrant dans les descriptions des packages Dynamo. Elles sont beaucoup moins précises techniquement, au point qu'il est parfois difficile de savoir quelles opérations le package permet de faire exactement. En revanche, les descriptions incluent toujours des mentions de l'aide apportée par le plug-in, de l'automatisation des tâches répétitives qu'il permet… même quand il n'est pas possible de déterminer quelle tâche exactement il s'agit de s'épargner.

Parmi les mécanismes mis en œuvre pour permettre aux développeurs de tenir leurs promesses aux utilisateurs, le premier que nous allons examiner est le changement dans les opérations rendues possibles par les outils. Beaucoup des nouveaux outils disponibles sont non pas des algorithmes utilisés pour générer ou évaluer des formes, mais des outils de management. Pour permettre aux utilisateurs d'être plus productifs, la première étape à franchir est en effet de leur épargner les difficultés de l'interopérabilité. Nous avions vu au chapitre IV divers exemples de projets se reposant sur le recours à un très grand nombre de logiciels différents. Parvenir à ce résultat demande beaucoup de travail à l'aide d'un langage-colle, qui sert de passerelle entre ces différents logiciels. Or l'offre des logiciels va croissant, et les praticiens sont encouragés à prendre de plus en plus d'éléments en compte dans leur travail de conception. Les développeurs mettent donc au point de nombreux outils complémentaires de gestion des flux de travail. Pour Dynamo en particulier apparaissent de nombreux packages à cet effet. Au sein de Revit se rencontrent en effet le développement du B.I.M[96] et l'expansion du champ computationnel, qui partagent certaines stratégies d'outillages et certaines ambitions - notamment celle de l'interopérabilité et du management. Ces outils permettent de mieux se coordonner entre praticiens, comme Speckle, par exemple, qui permet de mettre en relation dans un modèle en ligne diverses informations développées en local sur différents logiciels[97]. Il s'agit aussi de

---

[95] Synthesize Toolkit, description dans la liste des packages Dynamo - https://dynamopackages.com/
Citation complète : *Created by Karam Baki to Automate many repetitive tasks in Revit, divided into sections in the extra folder. Browse to the extra folder in Dynamo Player to use the Main Toolkit features (K-Join, Parameters, K-Laser....etc) You will find set of remarkable tools, such as K-Attractor, which helps in parameteric computational designing in Revit based on Attractors, and others nodes for randomizing based on multiple set of factors, and many more tools to automate your daily workflow, and increase your productivity, in design aspects and construction documents.*

[96] Building Information Modelling - l'acronyme désigne les technologies logicielles mises en oeuvre pour la modélisation jointe de projets de bâtiments par l'ensemble des corps de métiers intervenant dans sa production.

[97] https://speckle.systems/, consulté le 13 Novembre 2021.

mieux visualiser les données disponibles, un point crucial puisque les modèles comportent de plus en plus de données concernant le projet. Layer Stalker par exemple permet de mieux saisir la structure d'un fichier Rhinocéros / Grasshopper, en visualisant la façon dont les différents éléments du modèle sont classifiés[98]. Surtout, il s'agit de mieux faire communiquer les différents logiciels utilisés, et nombre de plug-ins sont donc dédiés à l'échange de données entre eux. Il existe donc des outils pour automatiser le transfert de données entre Rhinocéros et Revit, entre Grasshopper et Dynamo, entre Grasshopper et des logiciels de calcul des structures, de simulation environnementale, d'analyse de cycle de vie, de rendu des images.

Autre possibilité pour diminuer le travail de structuration des échanges entre environnements logiciels, rassembler toutes les opérations en un seul environnement. C'est la stratégie qui se cache derrière le B.I.M, mais aussi derrière les opérations de développement d'Autodesk, comme nous l'avons vu plus haut. Revit ne fait pas exception. En quelques années, le logiciel est devenu une combinaison de modeleur 3D, d'outil de management et de communication B.I.M, de programmation et de programmation visuelle. Il inclut même des outils d'automatisation de la conception plus poussée : toujours du calcul des structures, de simulation environnementale, d'analyse de cycle de vie - les essentiels de la conception contemporaine. Revit n'est pas tout à fait une cathédrale logicielle[99] au vu du nombre de développeurs impliqués dans son écosystème, mais le fait que son développement soit centralisé chez Autodesk avec les ressources pour les utilisateurs l'en rapproche tout de même dans une large mesure. Les ambitions de l'éditeur correspondent d'ailleurs à celle d'une cathédrale et non d'un bazar, même si Autodesk s'est approprié certains traits du bazar. Ceci permet à l'éditeur de promouvoir, comme le font les développeurs des packages et plug-ins, l'efficacité de ses outils au moins autant si ce n'est plus que leurs capacité de travail en elles-mêmes. La même logique de développement et de promotion se retrouve chez des entreprises plus jeunes, comme SpaceMaker ou Hypar[100]. Les marqueurs des stratégies de régulation des flux de travail dans le champ sont le poids de ces cathédrales logicielles et la promotion non plus de la liberté de forme mais de l'efficacité de la production. Mais on observe également l'apparition de cabinets de conseil comme Parallax[101] ou bad monkeys[102] en complément des développeurs et des éditeurs. Ces cabinets sont spécialisés notamment dans la production de ce genre d'outils, parfois rendus disponibles à tous mais souvent créés en

---

[98] Poinet, Paul, and Al Fisher. "Computational Extensibility and Mass Participation in Design." *Design Transactions: Rethinking Information Modelling for a New Material Age,* edited by Bob Sheil et al., UCL Press, 2020, pp. 56–61,

[99] Nous avons vu au chapitre V cette notion imaginée par le hacker Eric Raymond. Eric Raymond et Bob Young, The Cathedral & the Bazaar, O'Reilly, 2001.

[100] https://hypar.io/, consulté le 13 Novembre 2021. https://www.spacemakerai.com/, consulté le 13 Novembre 2021.

[101] https://www.parallaxteam.com/, consulté le 13 Novembre 2021.

[102] https://www.badmonkeys.net/, consulté le 13 Novembre 2021.

interne pour des agences conventionnelles qui transitionnent vers l'usage de Revit. Pour les développeurs, il s'agit donc d'épargner à leurs utilisateurs la pénibilité de la gestion de l'interopérabilité. Mais cela revient par là même à s'empêcher de construire son propre environnement logiciel personnalisé - sa propre écologie, comme dirait Dominik Holzer[103]. Or nous avons vu au chapitre précédents combien cela favorise la transmission.

L'aide à la gestion des flux de travail n'est pas le seul mécanisme de facilitation mis en place dans ces nouveaux écosystèmes logiciels. Plutôt que de proposer à leurs utilisateurs des fonctions mathématiques qui permettent de manipuler des structures de données ou des géométries, les nouveaux outils proposent directement des fonctionnalités orientées vers les applications. Les bibliothèques deviennent ainsi des bibliothèques d'éléments et non de fonctions algorithmiques. Ces fonctionnalités peuvent être basées sur la méthode de fabrication qui sera employée. Cockatoo propose par exemple de la recherche de forme basée exactement sur les mêmes principes de relaxation dynamique que Kangaroo Physics, mais pour des applications spécifiques : le tissage et le tricot. Concr3DLab offre la possibilité de modéliser des parcours outils pour l'impression 3D comme Silkworm, mais uniquement pour l'impression 3D de béton. D'autres outils sont construits pour permettre de produire des projets concernant des bâtiments et des infrastructures spécifiques. On trouve ainsi dans Dynamo et Grasshopper des plug-ins pour modéliser des ponts (Dynabridge), des stades (eclipse, Toro), des facades (DEBB, FacadeModel, SkinDesigner) des plans (Marmot, Peregrine) ou des toitures (Hible Roof), voire tout simplement directement des bâtiments résidentiels (Spheniscidae)[104]. Chacun de ces outils permet à l'utilisateur non plus de construire ses algorithmes en assemblant des fonctions pré-écrites, mais directement de manipuler les paramètres de conception de l'objet visé. Ce glissement vers des applications s'observe à différentes échelles : par types de bâtiments ou types d'infrastructures, mais aussi par détails de construction. Eelish[105] se concentre par exemple sur le placement de poteaux et poutres plutôt que sur une analyse structurelle large comme

---

[103] Dominik Holzer, "Design exploration supported by digital tool ecologies", *Automation in Construction*, Volume 72, Part 1, p. 3-8, 2016.

[104] https://forum.dynamobim.com/t/dynabridge/26809, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/eclipse, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/toro, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/debb-architectural-skin-facades-minutes, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/facademodel, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/skindesigner, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/hible-roofs, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/spheniscidae, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/marmot, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/peregrine, consulté le 13 Novembre 2021.
Pour un exemple détaillé d'algorithme derrière ces outils, on peut consulter Anderson C, Bailey C, Heumann A, Davis D. Augmented space planning: Using procedural generation to automate desk layouts. International Journal of Architectural Computing. 2018;16(2):164-177.

[105] https://www.food4rhino.com/en/app/eelish, consulté le 13 Novembre 2021.

Karamba 3D, alors même que ce dernier permet aussi de déduire des résultats d'analyse des emplacements optimaux. D'autres plug-ins comme Revit Rebar ou T-Rex permettent d'automatiser la modélisation des armements dans le béton, le package Kitchautomation de concevoir des cuisines[106]. Les composants algorithmiques pour des applications augmentent très largement dans les environnements de programmation visuelle. C'est particulièrement flagrant dans le cas de Dynamo, qui dispose de beaucoup moins de composants pour des fonctions mathématiques, des typologies algorithmiques ou des opérations géométriques diverses que Grasshopper. Ceci souligne encore un peu plus la différence de vocation de l'environnement d'Autodesk. Les composants "adaptatifs" pour des applications précises ne sont qu'une extension du dessin automatisé des détails de murs et autres qui étaient les éléments clés de Revit au départ. Néanmoins, cette stratégie de facilitation par la fonctionnalisation s'observe partout.

Un autre élément intervient dans le développement de l'interaction avec les nouveaux outils du champ : l'orientation des utilisateurs. La nécessité de guider les utilisateurs émerge à partir d'un problème que crée la génération d'un grand nombre de variations, elle-même permise par les outils algorithmiques. Une première possibilité pour l'utilisateur consiste à paramétrer le modèle puis à explorer manuellement les solutions générées. Mais le nombre de solutions peut être extrêmement élevé, ce qui complique cette exploration manuelle. Il est donc possible pour l'utilisateur de mettre en place un système d'évaluation pour trier les solutions, et n'examiner que les plus intéressantes. Néanmoins, un utilisateur néophyte n'aura pas forcément les compétences pour mettre en place un système d'évaluation. Par ailleurs, deux solutions peuvent être aussi performantes l'une que l'autre structurellement, mais être très différentes formellement ou spatialement, donc nécessiter un examen tout de même. Il faut donc pouvoir avoir une vue d'ensemble des solutions disponibles dans tous les cas, ce qui n'est pas aisé quand il y en a des centaines ou des milliers. Pour permettre à la fois à un utilisateur d'avoir une vue d'ensemble, et de mieux comprendre les résultats d'une évaluation réalisée par un expert d'un autre domaine, l'une des stratégies développées est de guider l'utilisateur dans l'espace des solutions disponibles.

Le Digital Structures Group du M.I.T, par exemple, qui travaille entre autres sur des outils algorithmiques d'analyse structurelle pour différents environnements, s'est d'abord penché sur les retours à fournir à des utilisateurs architectes et non ingénieurs. Le groupe recherche une méthode qui permette à ces utilisateurs de comprendre le comportement structurel de la proposition évaluée, ses défauts et les changements possibles pour l'améliorer. L'objectif initial du groupe est de concevoir un

[106] https://www.kitchautomation.com/, consulté le 13 Novembre 2021.
https://www.food4rhino.com/en/app/t-rex, consulté le 13 Novembre 2021.
https://knowledge.autodesk.com/support/revit/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Revit-Model/files/GUID-4C8C6094-D734-43B0-ACD9-D727A2C2B9DC-htm.html, consulté le 13 Novembre 2021.

outil qui fournit des informations sur l'évaluation puis laisse l'utilisateur ajuster les paramètres pour obtenir une meilleure performance sur les critères évalués. Cette méthode de travail est cependant considérée comme trop lente et peu efficace. Le groupe développe alors une autre approche : au départ, il s'agit de prendre en compte la préférence formelle de l'utilisateur pour réajuster la proposition, tout en améliorant sa performance structurelle. L'algorithme propose une série d'options à l'utilisateur, qui choisit sa favorite - ce choix influence ensuite l'ajustement de la structure, pour obtenir une proposition équilibrée entre performance structurelle et formelle. Cette seconde approche est un basculement : l'utilisateur n'est alors plus conseillé par le biais d'un retour d'information, mais directement guidé dans son travail de conception. Ainsi, aucun besoin de saisir la moindre nuance des questions structurelles. Chacun peut avancer sur les bases de sa propre expertise et le projet peut avancer plus vite. Après le développement de plusieurs outils basés sur ce principe de sélection des formes, le Digital Structures Group pousse l'approche encore plus loin[107]. En effet, dans cette première configuration, l'utilisateur n'entre absolument pas dans les détails. Les premiers outils du Digital Structures Group sont par ailleurs très simples en regard du grand nombre de paramètres usuellement impliqués dans la conception d'un bâtiment - ces premiers exemples n'en mettent en balance que deux. Or les nombreux paramètres qui entrent en jeu ne peuvent pas tous être optimaux dans une même proposition - les options de configuration des bâtiments sont donc un espace à plusieurs dimensions, ce qui complexifie encore l'exploration pour l'utilisateur. Le Digital Structures Group conçoit donc plusieurs manières de guider l'utilisateur au sein de cet espace des solutions. Plutôt que de laisser l'utilisateur parcourir seul et une par une les propositions, il indique dans quelle partie de cet espace des solutions se trouvent les propositions les plus pertinentes. Cela peut être les propositions les plus proches de ses intentions formelles, les plus performantes structurellement, les plus performantes thermiquement, les plus performantes sur le plan environnemental ou encore les mieux équilibrées entre ces différents critères. Enfin, si l'utilisateur est architecte et non ingénieur, selon le Digital Structures Group, on ne peut s'attendre à ce qu'il maîtrise l'élaboration de méthodes d'évaluations de critères de performance relevant usuellement du domaine du génie civil. Les outils proposés reposent donc sur un autre mécanisme : proposer à l'utilisateur les paramètres et des modèles d'évaluation les plus pertinents, et lui demander de valider leur prise en compte au cours de l'évaluation des propositions.

Ce travail marque un retour de l'enjeu d'automatisation dans les pratiques: puisqu'il faut rendre les outils faciles à utiliser, alors il faut automatiser tout un ensemble de procédures pour pouvoir les reléguer en arrière-plan et ne pas encombrer l'utilisateur. Il s'agit alors d'automatiser l'ajustement de

[107] Mueller, C., & Ochsendorf, J. (2013). An integrated computational approach for creative conceptual structural design. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2013.

certaines règles typologiques, accessibles seulement dans les couches les plus profondes de l'interface, mais pas uniquement. En allant plus loin, il devient question d'automatiser l'ensemble de la démarche en demandant à l'utilisateur seulement quelques clics de confirmation. Ceci se fait sur le modèle proposé par le Digital Structures Group. Leur logique consiste à demander simplement à l'utilisateur quelques entrées relatives à la géométrie du site et aux intentions formelles, quelques validations des paramètres d'évaluation, puis à l'emmener directement observer les propositions supposées les plus pertinentes. C'est une approche qui n'est pas l'apanage du Digital Structures Group ou de l'évaluation des performances d'un bâtiment : Dynamo propose un système équivalent pour le dessins des plans[108]. La stratégie appliquée consiste néanmoins à guider l'utilisateur au point de lui éviter de prendre des décisions. Or faire des ajustements structurels ou de plan soi-même, adapter les paramètres de l'algorithme pour obtenir une structure à la fois performante et spatialement satisfaisante est certes chronophage, mais c'est justement ce temps passé à travailler sur les questions qui permet d'apprendre. Ceci implique aussi pour les développeurs des outils, de toujours ramener la conception en cours à des grands principes applicables systématiquement, plutôt que d'affiner un modèle au plus proche des enjeux du contexte[109]. Cette simplification implique enfin de standardiser les pratiques, menaçant là encore la démarche de programmation heuristique dont on a pourtant vu qu'elle est essentielle dans le champ.

Cette logique d'automatisation poussée à l'extrême et basée sur des mécanismes de facilitation du recours aux algorithmes pour des utilisateurs novices dans le champ de la programmation ou de l'ingénierie se retrouve dans les derniers nés des outils computationnels : ceux basés sur les techniques dites d'apprentissage profond[110]. Ces outils sont caractérisés par les mêmes ambitions de fonctionnalisation : les entrées et les sorties sont déterminées en fonction des applications visées. Ici, ce sont directement des plans qui sont obtenus. Il s'agit de nourrir le système d'un grand nombre de plans de bâtiments du même type, pour l'entraîner à faire des propositions similaires bien qu'adaptées au contexte du projet en cours. Le système est supposé repérer à travers les exemples les caractéristiques clés du type de bâtiment projeté pour pouvoir les réintégrer à la proposition de plan. Après avoir analysé un grand nombre de bâtiments aux fonctions identiques, le système fait donc une

[108] John Pierson, " Space Planning in Dynamo with DynaSpace – Using Generative Design in Revit", 24 Mai 2021, https://dynamobim.org/space-planning-in-dynamo-with-dynaspace-using-generative-design-in-revit/, consulté le 13 Novembre 2021.

[109] Voir les travaux de Tseranidis, S. (2015). Approximation algorithms for rapid evaluation and optimization of architectural and civil structures. MIT SM Thesis. Tseranidis, S., Brown, N. C., & Mueller, C. (2016). Data-driven approximation algorithms for rapid performance evaluation and optimization of civil structures, Automation in Construction. Voir aussi la discussion de cette idée chez Oxman Rivka (2008) "Performance-based Design: Current Practices and Research Issues", *International Journal of Architectural Computing*.

[110] Nous reviendrons en détail sur les techniques algorithmiques qui se cachent derrière ces outils au chapitre suivant.

proposition complète automatique. Les outils de complétion automatique participent de la même ambition d'automatisation. De la même manière que les outils de traitement de texte, à force d'analyses de textes existants, peuvent suggérer automatiquement comment compléter une phrase, ces outils ont pour objectif de suggérer comment terminer la modélisation d'un projet, à partir des éléments déjà esquissés par l'ordinateur[111] . L'idée est toujours de faciliter le travail de l'utilisateur, concepteur souvent confronté à une énorme quantité de données et de tâches à accomplir pour concevoir un bâtiment de grande échelle. L'argumentaire des outils qui se reposent sur des techniques d'apprentissage profond pour automatiser repose d'ailleurs exactement sur les mêmes points que les outils qui se proposent de guider l'utilisateur au plus près. Seulement, il ne s'agit plus simplement de laisser l'outil offrir des propositions en regard du résultat des analyses de performance, mais d'automatiser complètement la production de l'espace, le concepteur se transformant en vérificateur par le recours à ces outils.

## Conclusion
## Un nouvel objectif, démocratiser

Au cours des années 2000, le champ connaît une mue progressive, une transformation qui arrive à maturité à la fin des années 2010. Cette mue s'observe sur trois plans différents. Le nombre de praticiens dans le champ augmente, et leurs profils de connaissances évoluent. On trouve comme avant des architectes-programmeurs et des ingénieurs-programmeurs, mais surtout beaucoup d'architectes novices, et des éditeurs de logiciel dont l'activité dans le champ augmente significativement. Ces changements sont le signe d'une sortie du réseau historique et de vecteurs d'expansion du champ différents. Parmi ces vecteurs d'expansion, les cadres de transmission se modifient aussi. L'activité de transmission des praticiens est marquée par une confrontation aux difficultés que leur réservent les formats usuels d'enseignement de l'architecture. Les environnements logiciels sur le modèle du bazar, qui favorisent beaucoup d'échange entre les utilisateurs de différents profils et niveaux, et le travail en open-source sont par ailleurs en perte de vitesse. La mue du champ favorise à l'inverse les cathédrales logicielles, aux ressources beaucoup plus centralisées et hiérarchisées. Les changements dans le champ se reflètent aussi dans les outils développés par les éditeurs, qui s'appuient sur des stratégies nouvelles pour permettre aux utilisateurs novices de tirer le meilleur parti de l'usage des algorithmes. Ces stratégies s'appuient sur une automatisation beaucoup plus intensive des méthodes algorithmiques de génération et d'évaluation de la forme, qui font écho à l'épaississement des interfaces.

[111] https://github.com/ksobon/thesaurus, consulté le 13 Novembre 2021.

La mue du champ computationnel en architecture est le signe d'un changement d'objectif dominant. Plutôt que de transmettre le savoir-faire propre aux pratiques computationnelles, il s'agit désormais de démocratiser le recours aux algorithmes. Démocratiser, c'est augmenter le nombre d'utilisateurs. Dans le cas présent, cela implique principalement l'augmentation des outils disponibles et la simplification de leur interface, qui a pour conséquence l'augmentation du nombre d'utilisateurs, et notamment des profils d'architectes novices en programmation. Le terme de démocratisation peut cependant aussi impliquer que recourir à ces outils devient moins cher. Ce n'est pas nécessairement le cas ici puisqu'on observe des dynamiques d'abandon de l'open-source et de privatisation des outils, donc de capitalisation sur la mise à disposition. Le terme de démocratisation peut aussi impliquer qu'il existe un enjeu politique, ce qui est bien le cas, mais nous aborderons la question au terme du chapitre VII. Les nouvelles stratégies mises en œuvre par les éditeurs de logiciels témoignent de nouvelles ambitions pour leurs outils algorithmiques. Il s'agit d'augmenter le nombre d'utilisateurs, mais aussi de diminuer le temps d'apprentissage nécessaire. Le temps passé par un utilisateur individuel pour maîtriser un outil, mais aussi le temps passé par une agence pour intégrer cet outil à ses habitudes de travail. En changeant d'objectif dominant dans le champ, le statut des outils change : l'importance de faire les siens diminue et transmettre cette idée aux autres praticiens n'est plus au cœur des ambitions de ceux qui les développent. Les variations au chapitre observées précédent témoignent de ce changement, puisque ce sont les interfaces qui permettent transmission et / ou démocratisation en fonction de leur structure. Consacrer du temps à l'apprentissage, apprendre par soi-même à les manipuler et faire ensuite ses propres outils vont ensemble dans les activités des praticiens avant la mue, puisque le troisième impératif découle des autres. Ce passage vers une facilitation maximisée du travail des utilisateurs avec les outils algorithmiques pour qu'ils évoluent en toute autonomie est un aspect extrêmement important dans cette transformation du champ computationnel.

Le changement des réseaux fait écho à ce phénomène de démocratisation à l'œuvre. On observe bien une démocratisation, dans le sens ou il y a une augmentation des utilisateurs comme des utilisateurs potentiels, et que nombre de ces nouveaux utilisateurs sont des novices en programmation. Parmi les autres profils, chacun joue un rôle particulier dans le pilotage de cette vague de démocratisation. Après l'équilibre atteint à la génération précédente, l'expansion du réseau s'accompagne de la réinvention des outils pour permettre leur démocratisation vers le milieu de l'architecture conventionnelle. En effet, on a vu que la popularisation de l'idée de faire ses propres outils, de maîtriser l'ensemble du workflow date de la seconde phase du champ, et que l'appropriation de typologies et le développement d'outils de programmation propres aux pratiques computationnelles date essentiellement de la troisième phase. La quatrième phase marque un tournant dans la mise à disposition de ces outils. La troisième génération de praticiens a auparavant fait la preuve du potentiel

de ces outils, et contribué à populariser des méthodes, des représentations, des esthétiques, des théories nouvelles pour la pratique architecturale. Un plus grand nombre de praticiens commencent à s'intéresser aux outils algorithmiques. Dans les grandes agences, la jonction entre les générations se produit par un passage de relais entre les unités de recherche internes, qui donnent à voir les usages possibles des outils algorithmiques dans des pratiques architecturales conventionnelles, et le reste des praticiens qui évoluent au sein de l'agence. L'arrivée des éditeurs à une position dominante dans le champ entérine certaines stratégies de mise à disposition des outils et des ressources, mais les différentes pratiques mentionnées précédemment et le solide réseau qui s'est constitué servent de base à la dynamique de démocratisation. Les praticiens du milieu universitaire, architectes-programmeurs et ingénieurs-programmeurs, y contribuent en effet en poursuivant leur travail de formation des héritiers, de prospective, de popularisation, de défrichage de solutions techniques. Nous avons vu que l'institutionnalisation du champ, l'obtention de fonds et la création de nouveaux groupes de recherche sont la cristallisation des efforts de cette génération. La formation de profils similaires, héritiers de ces pratiques, en est un autre résultat. Ces praticiens héritiers contribuent aussi à cette expansion, mais tous ne répliquent pas la trajectoire de leurs aînés. Nous avons vu qu'une partie d'entre eux se tourne aussi vers le privé et contribue à développer des solutions techniques ultra accessibles à des utilisateurs non formés. Le travail de prospective transitionne donc vers un travail plus proche de la R&D. Le champ n'est donc plus majoritairement caractérisé par une bulle d'expérimentation protégée des contraintes. Cette mue se répercute aussi sur les pratiques de conception au sein du champ, qui s'appuie sur les nouveaux outils et les nouveaux cadres. C'est sur ces pratiques que nous allons maintenant nous pencher.

# CHAPITRE VII.
# -
# Démocratiser, rationaliser

## 7.1 Pratiques

Le champ computationnel porte depuis ses débuts plusieurs courants, mais la pression philosophique historique héritée du monde de l'IA et les pressions économiques contemporaines en favorisent un en particulier en le faisant apparaître comme légitime à de multiples niveaux. Ce courant, c'est celui qui promeut une rationalisation des pratiques de conception architecturale. Le Larousse donne de la rationalisation la définition suivante : c'est l'action de rationaliser quelque chose, c'est-à-dire *d'organiser un processus de manière à accroître son efficacité*, ou de *donner à quelque chose une explication rationnelle, logique*. Le mot provient du latin *ratio*, « calcul », et désigne le fait de privilégier l'étude des liens de cause à effet pour la compréhension du monde et la prise de décision. Pour René Descartes, penseur fondateur de la rationalisation, il s'agit de privilégier la raison sur tous les autres modes de pensée et de perception[1]. Max Weber, qui a beaucoup contribué à penser le rôle de la rationalisation dans le développement du monde capitaliste occidental, ajoute à cette définition l'idée de la rationalité comme finalité en soi, plutôt que comme un outil privilégié seulement[2]. Dans les pratiques du champ computationnel, la rationalisation se traduit par la volonté d'expliciter toutes les étapes de conception de l'espace – les instructions explicites de programmation fournies à l'ordinateur ne sont donc pas seulement ce qui permet à celui-ci d'exécuter une série de calculs dont le résultat entre en jeu dans la conception du projet, elles deviennent l'ensemble du projet. Le besoin d'explicitation inhérent au recours aux outils algorithmiques devient encore plus littéral, total : les étapes de conception doivent trouver leur place dans une structure explicite logique, sans quoi elles ne sont pas légitimes. La nécessité d'expliciter s'accompagne donc d'une volonté de quantifier le maximum d'aspects du projet, et ce en vue de les évaluer : l'approche de rationalisation implique de recherche la plus grande efficacité donc d'évaluer en vue d'optimiser. Enfin, il s'agit de systématiser le processus de conception, de le construire à partir du recours à des modélisations scientifiques donc la validité est collectivement reconnue, et de pouvoir ensuite le répéter pour différents projets, quel que soit l'architecte, quels que soit le client et la commande, quel que soit le contexte. La phase de

[1] René Descartes, *Les principes de la philosophie*, Louis Elzevir, Amsterdam, 1644.
[2] Max Weber : La profession et la vocation de savant (texte rédigé en 1917), in "Le savant et le politique", La Découverte 2003 ; Économie et société (édition originale, publiée peu après la mort de l'auteur : 1922), Pocket, 2003

démocratisation du champ computationnel voit donc apparaître de nouveaux profils et de nouveaux outils, mais aussi de nouveaux projets, qui sont le fruit des pratiques des deux premiers. Si beaucoup plus de projets ont recours aux algorithmes dans les champs de pratique les plus consensuels, symptôme de la démocratisation, ce sont des projets différents de ceux qui ont vu le jour auparavant. La phase la plus récente du développement du champ n'est en effet plus celle d'une prospective spatiale, mais celle d'une standardisation des pratiques, par le biais de quantifications systématiques et sur fond de course à la performance. La quantification se fait dans de multiples domaines, des plus évidents aux plus obscurs : on mesure les performances structurelles, thermiques, acoustiques et environnementales, mais aussi l'adéquation du plan au programme et aux activités prévues dans l'espace en cours de conception ou l'esthétique du projet. Les évaluations techniques relevant du domaine de l'ingénierie s'appuient sur des modélisations physiques classiques et utilisent les critères standard usuels du domaine. Il s'agit de mesurer la performance structurelle pour évaluer si la structure résiste aux cas de charge prévue, la performance thermique pour évaluer les pertes de chaleur, la performance acoustique pour évaluer si les ondes sonores sont suffisamment absorbées. Ces critères sont ensuite généralement croisés avec un critère économique, parfois avec un critère esthétique, permettant de visualiser le rapport de force entre les deux en fonction des variations du projet envisagé. Par exemple, si la structure résiste largement au cas de charge prévu, il devient possible d'envisager de diminuer la quantité de matière dans cette structure, ce qui permet ensuite de diminuer le coût financier et environnemental.

### *7.1.1 Des aspects aisément quantifiables*

Critère d'importance, la description géométrique des éléments du bâtiment est ce qui permet sa construction. La complexité formelle est toujours plus prisée, dans la continuité du style digital ou paramétrique dont les courbes étaient le marqueur[3]. De la tour Leeza à l'hôtel Morpheus, les projets les plus récents réalisés par l'agence Zaha Hadid Architects en témoignent, tant par leur audace formelle que par les multiples récompenses reçues par leurs concepteurs[4]. Si les capacités de fabrication progressent, le champ des possibles en comparaison des possibilités de description mathématique reste néanmoins restreint. Les outils dédiés à cette question permettent donc de garantir une description géométrique rigoureuse indispensable à la construction, mais également de rester dans le champ du fabricable - qu'il soit défini par les lois de la physique ou par une enveloppe budgétaire.

[3] Voir chapitre III.

[4] La tour Leeza a été récompensée par le Council on Tall Buildings and Urban Habitat, par les Architizer A+ Awards et par les AIA UK 2020 Excellence Awards. Le Morpheus Hotel a été récompensé aux Quality Building Award 2020, aux Asia Pacific Property Awards 2019, aux 2019 Archdaily Awards, au classement 2018 du Time Magazine *World's Best Places*, par un National Geographic Best Sleep Award, au classement de Popular Science Magazine *Best of What's New*, et par le Council on Tall Buildings and Urban Habitat.

Les développeurs capitalisent sur le travail de post-rationalisation entamé à partir de la fin des années 1990 et sur la logique de pré-rationalisation développée au début des années 2000, comme sur l'expérience de la fabrication numérique accumulée par les praticiens du champ[5]. Ils produisent des outils dont l'objectif assumé est de cantonner les architectes à un espace de conception réaliste physiquement et financièrement[6]. Contraindre les explorations formelles pour les contenir dans un espace géométriquement raisonnable, ou les formes ne risquent ni erreurs de modélisation ni impossibilités constructives, permet aux équipes de projets de gagner du temps et de l'argent. Les outils de gestion de la géométrie encadrent la conception par les architectes. Le travail de doctorat de Romain Mesnil s'inscrit par exemple dans cette lignée : il présente un ensemble de méthodes de génération de forme à partir de courbes dont les paramètres mathématiques garantissent que la surface résultante sera divisible en panneaux quadrangulaires plans, y compris si il s'agit d'une surface à double courbure[7] - une solution bien moins coûteuse que les maillages courbes auxquels les formes complexes des dernières décennies nécessitent de faire appel. Le Digital Structures Group du M.I.T., le laboratoire Navier de l'ENPC ou encore le Block Research Group à l'ETHZ s'investissent tous dans le développement de modèles mathématiques garantissant la constructibilité des formes explorées. Ces modèles sont ensuite mis en oeuvre dans des outils qui rejoignent la liste de plus en plus longue des logiciels dédiés à l'évaluation et à la rationalisation constructive, des plus précoces programmes de manipulation des maillages comme Weaverbird ou PanelingTools aux plugins dédiés à des techniques de fabrication précise, comme Cockatoo ou GluLamb, en passant par ceux qui facilitent la création de fichiers de fabrication, comme Ivy ou OpenNest. De plus en plus de projets construits de grande échelle font appel à ces outils - si ce sont encore souvent des projets d'exception, avec des budgets qui permettent d'investir du temps dans la réalisation de modèles informatiques complexes, l'usage des algorithmes se répand néanmoins dans les pratiques constructives. C'est le cas de la tour Leeza ou de l'hôtel Morpheus, mais aussi de la Philharmonie de Paris, de la Fondation Louis Vuitton, de la couverture du département des arts islamiques au Louvre, des structures intérieures du magasin Hermès Rive Gauche à Paris, des projets les plus récents de Shigeru Ban comme le centre Pompidou Metz ou le Haesley Nine Bridge Golf Club House, et de bien d'autres encore.

[5] Voir chapitres III et IV.
[6] Conversation O. Baverel. C'est aussi une approche formulée notamment par le Digital Structures Group dans les publications que la suite du texte aborde plus en détail.
[7] Mesnil, R., Explorations structurelles de domaines de formes constructibles pour l'architecture non-standard, 2017, Thèse de Doctorat, ENPC.

*Figure 1. a. Hôtel Morpheus, Zaha Hadid Architects ; b. Tour Leeza, Zaha Hadid Architects*

D’autres projets voient le recours aux algorithmes augmenter, dans des applications légèrement différentes : il ne s’agit pas de les utiliser pour explorer des formes constructibles, mais pour évaluer

la performance structurelle et adapter la géométrie à cette évaluation. Le laboratoire Navier ou le Digital Structure Group se spécialisent dans ces questions de structure et proposent là encore des outils permettant de faciliter autant et aussi tôt que possible la prise en compte de cette question, par exemple structureFIT, qui permet de développer des treillis sur son navigateur internet. Bollinger + Grohmann, bureau d'étude allemand déjà renommé dans le champ pour son travail et pour le développement de Karamba 3D, rare plugin Grasshopper à faire sortir l'analyse structurelle du domaine réservé des outils de génie civil spécialisés, collabore depuis les années 2000 avec des architectes en s'appuyant sur des outils algorithmiques. Les années 2010 voient le nombre de projets construits ou ils accompagnent la conception via Karamba ou d'autres programmes développés spécialement pour l'occasion augmenter. Des concours comme l'Infobox aux projets réalisés comme le pavillon CIAB, le pavillon FFM ou le centre Frans Masereel, en passant par les collaborations avec des artistes comme Tomas Saraceno, l'analyse structurelle comme point de départ de la conception spatiale se donne à voir dans de multiples bâtiments. La performance thermique ou acoustique, en fonction des programmes, ont également une place prépondérante. La première est particulièrement importante dans la conception de logements ou l'esquisse de volumétries à l'échelle urbaine. La deuxième joue bien sûr un rôle clé dans la conception des salles de concerts et voit sa popularité comme point de départ du projet augmenter à mesure que les outils algorithmiques comme Pachyderm rendent les modélisations acoustiques accessibles. Le prototypage de panneaux acoustiques grâce à l'impression 3D ou à l'usinage de bois popularisent également l'analyse acoustique en offrant un aperçu de possibilités formelles[8]. La salle de concert de l'Elbphilarmonie de Hambourg, concue par Herzog et de Meuron en collaboration avec One to One, est par exemple tapissée de panneaux en fibre de gypse fraisés individuellement pour composer une mosaïque d'absorbeurs et de diffuseurs régulant la trajectoire des ondes sonores, et sa façade fait la part belle au verre courbé - tout ceci tirant largement parti de l'évaluation permise par les outils numériques désormais à portée de clic.

[8] Voir Burry, J, Davis, D, Peters, B, Ayres, P, Klein, J, Pena de Leon, A and Burry, M 2011, 'Modelling Hyperboloid Sound Scattering: The Challenge of Simulating, Fabricating and Measuring', in Gengnagel, C, Kilian, A, Palz, N and Scheurer, F (eds), Modelling Symposium, Springer-Verlag, Berlin, pp. 89–96, ou encore Reinhardt D., Cabrera D., Jung A., Watt R. (2016) Towards a Micro Design of Acoustic Surfaces. In: Reinhardt D., Saunders R., Burry J. (eds) Robotic Fabrication in Architecture, Art and Design 2016. Springer, Cham et la bibliographie qui y est listée.

*Figure 2. a. ElbPhilarmonie, Herzog & De Meuron ; b. Pavillon FFM, Bollinger + Grohmann*

Malgré la difficulté à mesurer correctement l'impact environnemental d'un projet, l'impératif écologique se fait sentir, et les analyses du cycle de vie (ACV) se font plus présentes dans les mesures également. De plus en plus d'outils cherchent à simplifier la visualisation d'une évaluation de cet ordre, tout en permettant la production rapide d'évaluations sur-mesure pour chaque projet : Ecotect, Geco, Tortuga, Bombyx ou One Click LCA. Les publications prennent cependant encore souvent la

forme de principes généralistes plutôt que d'évaluations chiffrées précises, comme dans les publications sur le sujet de Guillaume Habert et d'Isolda Agusti-Juan à l'ETHZ ou d'Achim Menges à l'ICD[9]. L'analyse de cycle de vie en tant que méthode implique de documenter l'ensemble des étapes de fabrication d'un objet, pour pouvoir mesurer ce que consomme ou ce que rejette chaque processus, chaque geste intervenant dans la réalisation de cet objet. Les bases de données existantes permettent d'utiliser des évaluations accomplies par d'autres en amont. Il faut cependant pour chaque projet reconstituer tout ce qu'il faudra faire pour construire le bâtiment, chercher s'il existe des évaluations chiffrées de certaines des étapes, chiffrer les étapes qui ne sont pas encore documentées sur le plan de leur coût environnemental, et enfin cumuler les coûts recensés pour obtenir une évaluation d'ensemble. Et chaque variation du projet implique des modifications dans cette évaluation qu'il faut de nouveau calculer. Enfin, toutes les informations nécessaires ne sont pas toujours rendues disponibles par les fabricants : certaines étapes du calcul d'impact nécessitent donc une approximation ou une équivalence qu'il faut bien peser, pour ne pas risquer de fausser les résultats. Fournir plus que des principes généralistes concernant l'impact du recours aux technologies numériques demande donc un travail long et minutieux, auquel le champ s'attelle tout juste. Ces principes s'accompagnent cependant de l'ambition d'intégrer l'évaluation de l'impact environnemental dès le début de la conception, comme pour les contraintes constructives et structurelles. Cette intégration se fait à l'aide de principes standards de l'ACV pour la construction, parfois réducteurs au vu de l'ampleur de la tâche que représente une analyse de cycle de vie fiable, mais permettant en effet de fournir des informations sur ce point très tôt dans le processus de conception. Par exemple, les spécialistes de l'ACV de bâtiments considèrent qu'il est possible de négliger l'impact du processus de fabrication en lui-même[10], car les ressources consommées pour construire un bâtiment sont telles que cela représente un impact plus grand de plusieurs ordres de grandeur. Les bases de données utilisées pour accomplir une ACV contiennent par ailleurs beaucoup d'informations sur l'impact des matériaux au mètre cube ou à la tonne. Il est donc possible de se faire une idée de l'impact d'un projet de bâtiment en mesurant la quantité de différents matériaux qu'il nécessite. Cette méthode est appliquée par certains praticiens du champ computationnel dans leurs propres projets[11]. Autre exemple, l'utilisation de la mesure des

[9] Agustí-Juan, I., Müller, F., Hack, N., Wangler, T., Habert, G., "Potential benefits of digital fabrication for complex structures: Environmental assessment of a robotically fabricated concrete wall", *Journal of Cleaner Production*, 154, p. 330–340, 2017. //ref Menges

[10] Ce qui n'est pas le cas par exemple pour fabriquer des objets en verre ou pour faire du ciment, cas de figure dans lesquels l'énergie consommée pendant le processus est aussi voire plus importante que la consommation des ressources.

[11] CITA ou ICD. Des travaux ont cependant montré que ce n'est pas forcément une estimation fiable au vu du coût environnemental des équipements informatiques impliqués dans les processus de fabrication numérique, plus élevés que ceux d'une grue ou d'un coffrage. Voir à ce sujet Kuzmenko K., Gaudillière N., Feraille A., Dirrenberger J., Baverel O., "Assessing the Environmental Viability of 3D Concrete Printing Technology". In: Gengnagel C., Baverel O., Burry J., Ramsgaard Thomsen M., Weinzierl S. (eds) *Impact: Design With All Senses. DMSB 2019*. Springer, Cham, 2020.

gains solaires, qui permettent de diminuer l'énergie utilisée pour chauffer un espace au cours de l'année, et donc l'énergie consommée par le bâtiment au cours de sa vie. Comme un article de Danil Nagy et ses collègues l'illustre, il est possible d'analyser la volumétrie d'un projet dès son esquisse pour évaluer le gain solaire en fonction des variations, et de comparer l'économie d'énergie réalisée au coût de construction pour la volumétrie évaluée. L'article, qui utilise comme cas d'étude la conception d'un quartier résidentiel aux Pays-Bas, a par ailleurs vocation à montrer comment cette technique peut être généralisée à n'importe quel exercice de conception grâce aux outils utilisés[12]. Ainsi, bien que la complexité de l'exercice d'ACV rende l'intégration de la mesure du l'impact environnemental peu aisé, l'importance de l'enjeu mène un nombre croissant de projets à prendre en compte cette évaluation en complément des autres évaluations techniques, par le biais de modèles simplifiés.

*7.1.2 Des aspects plus complexes à quantifier*

Au-delà des évaluations permises par des modèles scientifiques éprouvés et reconnus, les praticiens du champ computationnel s'essaient également à des exercices de quantification visant à permettre des évaluations fonctionnelles ou programmatiques des projets. Tombés en désuétude pendant un temps après leur développement dans les années 70 et 80, les outils de mesure de l'espace par la visibilité et le mouvement reviennent sur le devant de la scène, désormais utilisés comme critères d'évaluation de la fonctionnalité d'un espace. Les systèmes experts auxquels ils devaient être intégrés n'ont jamais vraiment vu le jour, mais les techniques développées s'avèrent néanmoins toujours utiles pour mesurer des caractéristiques spatiales, et sont intégrées à un grand nombre de systèmes algorithmiques plus récents, souvent pour des programmes similaires aux premières expérimentations qui les utilisaient. Les isovists par exemple, dont les mesures de visibilité depuis un point donné dans l'espace étaient particulièrement prisées dans la conception de centres commerciaux, car elles permettent d'établir quelles publicités seront vues par les consommateurs à quel endroit, sont régulièrement utilisés dans l'analyse et la conception d'espaces d'exposition[13]. En complément du principe de départ de mesure de la visibilité des isovists, des modèles de calcul similaires permettent d'examiner d'autres caractéristiques spatiales, comme la luminosité d'un espace ou le temps de marche d'un point à un autre. Cette dernière mesure, combinée aux isovists classiques, est entre autres utilisée dans la conception d'hôpitaux. Dans un article publié par Jisun Lee et Hyunsoo Lee par

[12] Nagy, Danil, Lorenzo Villaggi et David Benjamin. "Generative urban design: integrating financial and energy goals for automated neighborhood layout." (2018). Brown, N.C. & Mueller, C. (2016). Design for structural and energy performance of long span buildings using geometric multi-objective optimization. *Energy and Buildings*, 127: 748-761.

[13] Voir par exemple Choi YK. "The Morphology of Exploration and Encounter in Museum Layouts", *Environment and Planning B: Planning and Design.* 26(2):241-250. 1999.

exemple, les isovists sont utilisés pour évaluer la visibilité directe entre le bureau des infirmières et les chambres des patients. Ils sont combinés à une mesure de la distance de marche nécessaire aux infirmières pour faire le tour des chambres qu'elles doivent accomplir plusieurs fois dans la journée. Les deux mesures sont utilisées ensemble pour définir une configuration optimale, permettant aux infirmières de voir les patients dans leurs chambres autant que possible depuis leur bureau tout en leur évitant une trop longue distance de marche pour leur tour[14]. L'assemblage de plusieurs mesures de ce type est supposé permettre de saisir la fonctionnalité d'un espace en regard d'un programme donné, mais aussi l'expérience sensorielle offerte.

Les systèmes fonctionnant notamment sur des matrices de relations, elles aussi des modèles mathématiques utilisés de longue date[15], sont intégrés de plus en plus à des outils permettant une hiérarchisation spatiale et une répartition programmatique, en particulier chez Autodesk. DynaSpace[16] permet par exemple de placer automatiquement des espaces dans le plan à partir d'un fichier Excel qui spécifie le nombre d'espaces, leur aire, et ceux auxquels ils doivent être liés. Le résultat est un diagramme de la répartition dans le plan d'un ensemble de cercles symbolisant les différents espaces, et qui peut ensuite être utilisé comme base pour dessiner le plan du projet selon la géométrie envisagée. L'outil permet aussi de pondérer lesquels des critères doivent être rigoureusement respectés et lesquels peuvent être modifiés légèrement, pour permettre la résolution du problème posé. Autre exemple, le Core Generator permet de placer automatiquement les noyaux de circulation verticaux au sein d'un projet[17]. La volumétrie du bâtiment et le nombre d'étages sont d'abord fournis comme point de départ par l'utilisateur. Un ensemble d'indications permet ensuite de décider de la composition et de former la volumétrie des noyaux de circulation : dimensions des escaliers, nombre de personnes circulant ou encore limite de transport des ascenseurs. Un second ensemble de paramètres permet de les placer dans le plan des étages : nombre de noyaux, distance maximale de marche depuis n'importe quel point de l'étage, alignement avec un des axes du bâtiment. Ces paramètres peuvent être définis en fonction des normes de construction et permettent ainsi une prise en compte rapide de ces contraintes, souvent perçues comme rendant le processus de conception très fastidieux par les architectes.

---

[14] Lee, J., Lee, H., "Agent-driven accessibility and visibility analysis in nursing units" in *Intelligent and Informed, CAADRIA 2019*; Faculty of Architecture and Design, Victoria University of WellingtonWellington; New Zealand; 15 April 2019 through 18 April 2019; Volume 1, p. 351-360, 2019.

[15] Voir chapitre II.

[16] Dynamo Team, "Space planning in Dynamo with DynaSpace", https://dynamobim.org/space-planning-in-dynamo-with-dynaspace/, consulté le 12 Novembre 2021.

[17] Das, S., Day, C., Dewberry, M., Toulkeridou, V. et Hauck, A.. « Automated Service Core Generator in Autodesk Dynamo - Embedded Design Intelligence aiding rapid generation of design options ». In Herneoja, Aulikki; Toni Österlund and Piia Markkanen (eds.), Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 2, University of Oulu, Oulu, Finland, 22-26 August 2016, pp. 217-226. CUMINCAD, 2016.

En complément de ces programmes, Autodesk publie une série de cas d'étude montrant l'intérêt d'y avoir recours en agence, dans des projets de conception classique pour lesquels ils sont conçus. DynaSpace est par exemple utilisé par l'agence américaine Shepley Bulfinch pour générer rapidement des possibilités à partir des fichiers Excels de programme[18]. DynaSpace est associé dans ce cas-ci à un fichier Dynamo plus étendu, qui permet une fois les espaces hiérarchisés entre eux, d'ajouter des informations de dimensions verticales et générer directement une volumétrie rectangulaire. Enfin, un module permet d'enregistrer dans un fichier Excel toutes les configurations successives, permettant aux agences de grande taille comme Shepley Bulfinch[19], ou de nombreuses personnes peuvent intervenir successivement sur le même projet et le même fichier, de garder trace des étapes de conception. L'utilisation faite par l'agence MT Højgaard de Dynamo est également mise en vitrine[20] : celle-ci génère des volumétries sur le même principe que Shepley Bulfinch, en y combinant une évaluation du coût de construction. Autodesk se sert également de ces techniques pour aménager ses nouveaux bureaux à Toronto en 2017, projet démonstrateur phare des nouveaux outils développés par l'éditeur et confié à The Living. Puisqu'il s'agit de bureaux, un aspect essentiel du projet est le placement des postes de travail. The Living met donc en place une série de six mesures pour évaluer les qualités que présentent ces postes et les répartir au sein du bâtiment : la proximité, le style de travail, l'agitation, la productivité, la luminosité et la visibilité[21]. Ces métriques reposent en fait sur des calculs basés sur la mesure de la lumière, des vues (toujours grâce aux isovists) et de la distance à certains espaces, cette dernière permettant en plus une estimation du bruit. Les employés dont il est prévu qu'ils viennent travailler dans le bâtiment remplissent un questionnaire permettant d'obtenir des scores idéaux sur-mesure pour les différentes métriques. Ensuite un algorithme de hiérarchisation de l'espace en six étapes est utilisé pour placer les bureaux au sein du bâtiment, d'abord divisé en

---

[18] "Dynamo for Space Planning", https://dynamobim.org/home_usecases/use-case-1/, consulté le 12 Novembre 2021.

[19] 300 salariés. La majeure partie des agences que nous avons observées qui s'intéressent aux outils computationnels et commencent à devoir se poser la question de comment gérer leur arrivée dans la pratique quotidienne de l'agence, tout en ayant les moyens de le faire, ont un nombre d'employés qui tournent autour de 200 ou 300.

[20] "Dynamo for Space Planning", https://dynamobim.org/home_usecases/use-case-1/, consulté le 12 Novembre 2021.

[21] En anglais : Adjacency, Work Style, Buzz, Productivity, Daylight, Views to outside ; définies comme suit *"1. Adjacency preference, which measures the travel distance from each employee to their preferred neighbors and amenities. 2. Work style preference, which measures the suitability of an assigned neighborhood's daylight and distraction measurements to the assigned team's surveyed preferences. 3. Buzz, which measures the amount and distribution of high-activity zones. 4. Productivity, which measures concentration levels at individual desks based on sight lines to other desks and other noise sources. 5. Daylight, which measures the total amount of natural daylight entering the space throughout the year. 6. Views to outside, which measures the ratio of workspaces with an unobstructed view to the exterior glass façade."* Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.

quartiers[22] : placer un point central pour chaque quartier, en tracer les frontières en fonction de la répartition de ces points, placer le long d'une ces frontières une zone d'espaces de services pour chaque quartier - ensuite aménagée avec sa propre logique intérieure, placer les postes de travail à l'intérieur des quartiers, puis enfin assigner les équipes à un quartier et les salariés à un poste de travail, en fonction de la correspondance des métriques à leurs souhaits. L'algorithme permet de générer de multiples options, parmi lesquelles celle qui offre la meilleure correspondance entre les caractéristiques des postes de travail et les souhaits des employés sera retenue.

La standardisation de la construction et du design du début du XX$^{e}$ a elle aussi produit des métriques : mesure des corps, dimensionnement du mobilier, et conception de l'espace à cette mesure. En témoigne le Modulor de Le Corbusier, mais aussi le travail obsessionnel de Ernst Neufert et du DIN allemand pour normaliser les dimensions de tous les objets du quotidien, de la feuille de papier au logement[23]. Comme le reste du champ architectural, les courants computationnels héritent des normes de construction auxquelles ont mené ces dimensionnements, mais aussi de l'idée que ces systèmes induisent d'une évaluation possible du corps dans l'espace. De nouvelles technologies de scan, devenues abordables dans les années 2000, comme les kinects, rendent possible le repérage numérique du corps dans l'espace. Ceci permet de mesurer les déplacements du corps et d'en monitorer les positions, et certains praticiens se donnent pour objectif de comprendre à partir de ces données comment les personnes se comportent dans l'espace, et de concevoir en conséquence. Positions et mesures du corps sont associées à des activités, et le programme prévu d'un espace peut donc ensuite être traduit en nombre de corps, dont les mesures et positions sont utilisées pour en dériver des mobiliers, eux-mêmes utilisés ensuite pour définir l'espace. Autre méthode de mesure du corps dans l'espace : les systèmes multi-agents, utilisés pour simuler un groupe de gens et anticiper leur comportement dans un espace donné, que ce soit un aménagement urbain, ou une simple pièce dans laquelle la position du mobilier influe sur leurs déplacements dans l'espace.

L'esthétique elle aussi devient un critère d'évaluation auquel les praticiens ont régulièrement recours, mettant en place des systèmes de quantification de l'appréciation. Au sein du Digital Structures Group, par exemple, sont développés des outils qui permettent à l'utilisateur de cliquer pour sélectionner les morphologies les plus appréciées, une fois qu'une famille de possibilités a été générée et avant que leur performance soit évaluée[24]. Sélectionner une possibilité a pour effet de faire augmenter sa note et est pris en compte dans l'évaluation. Dans le cas d'un algorithme proposant

[22] *neighborhoods* dans l'article.
[23] Anna-Maria Meister, From Form to Norm : the Systematization of Values in German Design. Thèse de doctorat, Princeton University, School of Architecture, 2013.
[24] Caitlin T. Mueller, John A. Ochsendorf, "Combining structural performance and designer preferences in evolutionary design space exploration", *Automation in Construction*, Volume 52, p. 70-82, 2015.

plusieurs générations de possibilités, les morphologies suivantes proposées à l'utilisateur sont supposées se rapprocher de celles sélectionnées comme favorites précédemment[25]. Ce genre de dispositif se systématise ensuite dans les travaux du groupe, mais aussi ailleurs, intégrant la préférence de l'utilisateur comme un critère parmi d'autres[26]. Cette préférence peut être évaluée par un système de choix parmi diverses propositions, mais aussi par exemple en invitant l'utilisateur à choisir parmi des ensembles d'images ses préférées, à partir desquelles des caractéristiques spatiales ou de style sont dérivées et appliquées au projet en cours de conception[27]. Ces approches de prise en compte de l'opinion de l'utilisateur sont développées notamment pour la co-conception avec de futurs utilisateurs du bâtiment[28], mais qu'on retrouve petit à petit aussi développés par des ingénieurs ou des informaticiens à destination des architectes eux-mêmes.

Prisés aussi pour évaluer des aspects peu formalisés jusqu'ici, les réseaux neuronaux sont des systèmes algorithmiques devenus extrêmement populaires dans les années 2010, ou leurs capacités dans le domaine de la reconnaissance d'image, de texte ou vocale en ont fait les favoris des annonceurs du tout proche avènement de l'intelligence artificielle. Inspirés du fonctionnement des neurones biologiques et développés depuis les années 60 à partir des modèles de neurones artificiels proposés par Warren McCulloch et Walter Pitts[29], les réseaux de neurones consistent en de multiples neurones artificiels liés entre eux, selon des configurations variables. Le principe de ce modèles est de rechercher de configurations optimales pour faire correspondre le résultat du programme à l'attendu indiqué, ce qui les rend particulièrement efficaces pour faire des classifications et donc reconnaître un code postale manuscrit, distinguer un chien d'un chat sur une photo ou encore lier une phrase énoncée et une action sur un smartphone. L'idée est de ne pas construire un programme d'instructions de calcul que l'ordinateur exécutera ensuite, mais d'indiquer un point de départ et le résultat visé, et de laisser le système rechercher parmi diverses instructions lesquelles permettent le plus fréquemment d'obtenir

---

[25] Mueller, C., & Ochsendorf, J. (2013). An integrated computational approach for creative conceptual structural design. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2013. Brown, N. & Mueller, C. (2017). Designing with data: moving beyond the design space catalog. ACADIA 2017.

[26] Cunha, A., Loyens, D., & van Hattum, F., "Aesthetic design using multi-objective evolutionary algorithms. In Evolutionary Multi-Criterion Optimization—Lecture Notes" in *Computer Science* (pp. 374-388). Berlin Heidelberg: Springer, 2011. Marsault, Xavier. "A multiobjective and interactive genetic algorithm to optimize the building form in early design stages." Building Simulation (2013). Paola Sanguinetti, Marcelo Bernal, Maher El-Khaldi, and Matthew Erwin. 2010. Real-time design feedback: coupling performance-knowledge with design iteration for decision-making. In Proceedings of the 2010 Spring Simulation Multiconference (SpringSim '10). Society for Computer Simulation International, San Diego, CA, USA, Article 192, 1–8

[27] Louise Deguine, Nadja Gaudillière, Laya Hermelin, Ines Rogriguez-Porcel, "Wikitecture, a collaborative and open-source platform for a decentralized architecture production". https://www.academia.edu/10391518/Wikitecture_a_collaborative_and_open-source_platform_for_a_decentralized_architecture_production, consulté le 12 Novembre 2021.

[28] Pour des exemples d'application de ces dispositifs dans la co-conception, voir (articles)

[29] McCulloch, W. S., Pitts, W., A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

le résultat espéré. Relevant d'une approche probabiliste décriée par les premiers pontes de l'intelligence artificielle, ce qui en retarde leur développement, les réseaux de neurones prennent leur revanche à partir des années 1990, où ils commencent à démontrer leur efficacité notamment car le monde informatique dispose enfin de données en quantité suffisante pour que les analyses statistiques sur lesquels ils reposent portent leurs fruits[30]. Annoncés comme la voie royale pour enfin parvenir à l'intelligence artificielle forte car ils apprennent et ajustent leur configuration d'eux-mêmes, ils se sont répandus ces dernières années dans tous les domaines, et le champ de l'architecture computationnelle ne fait pas exception.

Stanislas Chaillou propose par exemple dans le cadre d'un travail de recherche mené à la Graduate Design School de Harvard un réseau neuronal entraîné à partir de plans classés dans quatre bases de données différentes à appliquer à un plan un style architectural différent : baroque, manhattanite (*Manhattan*), victorien ou maison de ville sur deux niveaux (*row house*)[31]. Le réseau neuronal transfère ensuite à un plan schématique les caractéristiques de l'un ou l'autre des styles, permettant d'obtenir une répartition spatiale globalement similaire mais une géométrie des cloisonnements variables. Les murs deviennent ainsi des successions de blocs épais et ouvragés qui fragmentent l'espace sans le fermer totalement dans le cas du style baroque. Dans les trois autres configurations, les cloisonnements sont plus similaires, mais le traitement des détails varient : la ou le style victorien offre de multiples placards dans les recoins, le style manhattanite transforme ces recoins en pleins non utilisés. Chacun des styles est ensuite documenté par Stanislas Chaillou à partir des résultats proposés par le réseau neuronal, l'évaluant sur six critères : la profondeur, la compacité, l'orientation (unique ou plurielle), l'angularité et le programme[32]. Selon l'auteur, l'influence du style sur la spatialité est majeur et joue un rôle clé dans la conception architecturale ; le réseau neuronal permet d'en saisir les particularités de chacun au-delà des critères identifiables par un humain[33]. La proposition finale du projet est un immeuble de logement ou les styles alternent en fonction des configurations spatiales pour proposer des appartements les plus appropriés possibles. Chaque appartement - dont le découpage s'est fait au préalable en travaillant avec un autre réseau neuronal - est évalué par le chercheur en fonction des paramètres identifiés, et ce dernier applique ensuite le style aux caractéristiques les plus proches pour adapter le dessin du plan.

---

[30] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.

[31] Stanislas Chaillou, "Architecture & Style. A New Frontier for AI in Architecture", https://towardsdatascience.com/architecture-style-ded3a2c3998f, consulté le 12 Novembre 2021.

[32] "*Depth, Compactness, Single-Orientation or Multi-Orientation (number of facades), Acute Angle (sharp geometry of the boundary), Program Spectrum (breadth of the program)"*

[33] Stanislas Chaillou, "Architecture & Style. A New Frontier for AI in Architecture", https://towardsdatascience.com/architecture-style-ded3a2c3998f, consulté le 12 Novembre 2021.

Dans le même esprit, mais sans s'intéresser à la différence de fonctionnalité qu'apporte l'adoption d'un style ou d'un autre, le projet *Artificial Zurichness*, mené par le Media and Design Laboratory à l'ETHZ entre 2020 et 2021 propose de saisir l'essence de la ville de Zurich[34] pour pouvoir générer des projets la distillant à leur tour. Les chercheurs ont d'abord entraîné un réseau neuronal[35] à partir d'un ensemble d'images de la vieille ville de Zurich, dont ils ont évalué à quel point chacune était représentative du style de la ville. Le réseau neuronal extrait ensuite[36] de nouvelles vues de la ville représentatives, à partir desquelles un collage est fait pour obtenir une enveloppe de bâtiment : façades et toit d'un immeuble. Dans un second temps, cette enveloppe est remplie grâce à une stratégie de dessin automatisé de plans étage par étage, en fonction des circulations et de la luminosité, pour créer un immeuble de logement[37]. Les recours de ce genre aux réseaux neuronaux explosent à partir du moment ou des structures comme Google Colab ou Open AI mettent à disposition des programmes divers pour entraîner facilement des réseaux neuronaux. En cinq ans, le nombre d'articles consacrés au développement d'applications de ces techniques dans le domaine de l'architecture computationnelle a été multiplié par plus de vingt, d'un article par an ou moins entre 2010 et 2015 à 21 en 2019 et 30 en 2020. Bien que les bases de données soient plus difficiles à établir dans le domaine en raison du nombre d'éléments nécessaires pour entraîner un réseau neuronal, les praticiens parviennent quand même régulièrement à trouver ou générer plusieurs centaines d'images de façades ou de plans qui puissent servir de point de départ. Évaluer l'architecture en soi se généralise donc, que ce soit sur des critères techniques, esthétiques, sur une préférence personnelle ou des critères plus flous comme la diversité[38], la vastitude, la continuité, la performativité[39], la résilience[40]. L'évaluation mystère des réseaux neuronaux fait de plus en plus autorité car ces dispositifs techniques sont supposés repérer des motifs invisibles à l'œil humain. Ils permettent donc de mesurer l'esthétique dans

[34] "Giro - Deep Facade", https://www.epfl.ch/labs/ldm/artificial-zurichness/, consulté le 12 Novembre 2021.
[35] Plus exactement, il s'agit d'une architecture ou plusieurs réseaux sont entrainés séparément avant d'eux-mêmes entraîner un réseau qui est celui qui est utilisé pour obtenir les résultats. "Sampling Zurich", https://www.epfl.ch/labs/ldm/sampling-zurich/, consulté le 12 Novembre 2021.
[36] depuis Google Street View.
[37] Travail en studio avec un groupe d'étudiants.
[38] Brown, N.C., & Mueller, C. T. (2018). Quantifying diversity in parametric design: a comparison of possible metrics. Artificial Intelligence for Engineering Design, Analysis and Manufacturing.
[39] *connectedness, spaciousness, convenience of access, continuity, adjacency* - Assem, Ayman; Abdelmohsen, Sherif; Ezzeldin, Mohamed; "A Fuzzy-Based Approach for Evaluating Existing Spatial Layout Configurations", p. 35-44 . In: Proceedings of 37 eCAADe and XXIII SIGraDi Joint Conference, "Architecture in the Age of the 4Th Industrial Revolution", Porto 2019, Sousa, José Pedro; Henriques, Gonçalo Castro; Xavier, João Pedro (eds.). São Paulo: Blucher, 2019. *adjacency, buzz, productivity, work style* - Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.
[40] Brown NC, Mueller CT. Design variable analysis and generation for performance-based parametric modeling in architecture. International Journal of Architectural Computing. 2019;17(1):36-52.

certains cas, mais aussi parfois un critère de pertinence indéfini, que les systèmes d'apprentissage profond sont censés être capables d'identifier.

### *7.1.3 Évaluer ses propositions*

Des mesures les plus rationnelles, faisant autorité dans le domaine scientifique, aux mesures les plus obscures, dont personne ne semble vraiment savoir comment elles sont construites, en passant par une myriade de mesures chiffrées diverses combinées en des critères aux noms parfois intrigants supposés capturer des caractéristiques spatiales jusqu'alors intangibles, le moindre aspect de la conception architecturale devient quantifiable. Quantifier et évaluer permet ensuite d'optimiser, c'est-à-dire de rechercher la valeur minimum ou maximum d'une fonction d'évaluation, valeur à laquelle correspond une configuration donnée des paramètres sur lesquels repose la conception du projet[41]. Par exemple, dans le cas de la Fondation Louis Vuitton, si l'objectif est de minimiser la différence de géométrie entre les voiles en verre initiales et les nouvelles voiles, fabriquées à partir de portions de cylindres, il est possible de mesurer la distance entre un ensemble de points de repère répartis sur la surface de la première version des voiles et leur équivalent sur la seconde version, et de chercher à minimiser cette distance : c'est cette mesure qui devient la fonction d'évaluation. Dans le cas du projet Alkmaar Housing, l'objectif étant de diminuer la consommation d'énergie en maximisant le gain solaire, il faut mesurer ce dernier en fonction de l'orientation des surfaces, leur matériau et leur taille, qui varient selon la volumétrie proposée pour le projet. Il faut ensuite en déduire l'économie d'énergie qui peut être réalisée, et la soustraire à la consommation d'énergie nécessaire en fonction de la température de confort souhaitée et de la volumétrie du projet, qu'il faut calculer en parallèle. La fonction d'évaluation est donc la minimisation de la consommation d'énergie et est formulée en fonction de la consommation d'énergie initiale et du gain solaire, ces deux dernières étant elles-mêmes formulées en fonction de la volumétrie du projet, dont les paramètres sont donc ceux qu'il faut faire varier pour rechercher l'optimum.

Pour mener à bien la recherche des minimums ou maximums d'une fonction d'évaluation, et notamment pour éviter le piège des minimums ou maximums locaux, les praticiens ont à leur disposition des modèles d'optimisation divers, mis en œuvre dans des algorithmes donc l'usage est de plus en plus facile[42] et de plus en plus dominant. Derrière les outils prêts à l'emploi et les scripts pré-écrits auxquels les praticiens du champ computationnel ont recours se cache une typologie principale, toujours la même : les algorithmes génétiques. Ceux-ci sont usuellement associés à une méthode de recherche de forme, qui permet de produire de nombreuses variations dont la performance

[41] Voir la section sur les algorithmes génétiques au chapitre V.
[42] voir 5.3

sur divers plans est ensuite évaluée. Le recours aux algorithmes génétiques perd de plus sa dimension de détournement : il ne s'agit plus comme pour EZCT ou supermanoeuvre, chez qui on retrouvait déjà souvent cette association de deux typologies dont des algorithmes génétiques, de jouer avec la fonction d'évaluation en fonction des attentes architecturales[43] : les fonctions d'évaluation sont désormais presque toujours simplement des implémentations des modèles mathématiques d'un phénomène physique ou géométrique disponible. Si les praticiens du milieu universitaire continuent à rechercher des manières d'évaluer d'autres caractéristiques architecturales, les praticiens du milieu consensuel se cantonnent à des évaluations techniques « objectives » implémentées automatiquement dans les nouveaux outils prêts à l'emploi. Les praticiens du milieu expérimental se penchent par ailleurs sur d'autres manières d'évaluer l'espace dans l'objectif de produire des modèles d'évaluations eux aussi objectifs et reconnus.

Les algorithmes génétiques sont prisés car ce sont des outils d'optimisation dit multi-critères : ils permettent de prendre en compte un grand nombre de facteurs en même temps, et de rechercher une solution qui ne soit pas forcément la meilleure sur le plan d'un seul critère, mais qui offre des performances satisfaisantes sur tous les plans. C'est cette idée qui est à l'œuvre dans les travaux sur les espaces d'exposition ou les bâtiments hospitaliers décrits plus haut : l'objectif est de maximiser la visibilité tout en minimisant la distance de marche. Les évaluations structurelles faites par les chercheurs qui s'intéressent à l'impression 3D, elles, sont faites dans l'objectif de minimiser la quantité de béton tout en maximisant la performance structurelle. Autre exemple encore dans DynaSpace, ou il s'agit de minimiser la différence entre le résultat en plan et le fichier Excel de départ : il faut donc créer une fonction d'évaluation dont la minimisation correspond à la minimisation de tous les critères, en les pondérant en fonction de leur importance. Ce système de pondération entre les critères est souvent présent, et permet de garantir que si certains sont plus importants que d'autres, leur optimisation passe en premier : minimiser le coût environnemental ou financier oui, mais pas au détriment d'une performance structurelle qui deviendrait insuffisante. L'ambition est que ces optimisations multicritères parviennent à terme à tout prendre en compte et combiner ensemble : minimiser le coût de fabrication, l'impact environnemental, la quantité de matière, les déperditions thermiques, la difficulté de réparation (en garantissant une géométrie spécifique), maximiser l'apport de lumière et la productivité, correspondre parfaitement aux goûts de l'architecte ou du client et au programme prévu dans le bâtiment, et bien d'autres encore. Et pour permettre l'implémentation de cet idéal tout-optimisé auquel aspirent les tenants de cette approche de rationalisation, après les outils permettant l'évaluation, les développeurs se concentrent sur la mise en place de flux de travail

[43] Voir chapitre IV et VI sur l'idée de trouver une expression de la fonction d'évaluation qui ne repose pas juste sur une mesure de certains éléments aussi scientifique que possible, mais qui repose sur des intentions architecturales.

(*workflow*) permettant de les combiner jusqu'à atteindre cette évaluation et optimisation totale. C'est dans cette optique que sont développés et promus la nuée d'outils de coordination des praticiens et des données décrits plus tôt[44].

Avec le développement de la rationalisation de la conception architecturale par la mesure et par l'optimisation, l'accent mis sur la question de l'évaluation a pour conséquence le devenir projet en soi de cette évaluation. Élément essentiel voir unique du processus, ses résultats sont le projet d'architecture. Dans le cas des outils de recherche de forme comme Design Space Exploration ou RhinoVault, celle-ci se fait à partir de l'évaluation de la performance structurelle, et les formes obtenues sont fréquemment directement transformées en projet : les images de rendu les montrent alors telles quelles, insérées dans un site et agrémentées de silhouettes, sans autre travail de conception. C'est particulièrement sensible dans le cas des outils de form-finding, mais la rationalisation pousse les pratiques vers une quantification systématique qui pousse en permanence à l'optimisation, quelque soit l'aspect le plus important du projet : on peut toujours mesurer quelque chose qui permettra d'évaluer la réussite du projet dans cet aspect, et c'est ceci qu'il faut à tout prix accomplir. Le principe d'exploration d'un espace fini de solutions régi par des contraintes quantifiables de constructibilité, de performance structurelle, thermique ou environnementale et l'intégration de cette approche aussi tôt que possible dans le processus de conception favorise également le devenir projet en soi de l'évaluation. Les métriques deviennent un aspect essentiel du projet et doivent être développées sur mesure à chaque fois pour pouvoir faire le projet ; c'est souvent effectivement le cas chez les praticiens les plus avertis. Mais elles sont néanmoins de plus en plus souvent directement intégrées à l'outil : le praticien novice n'a même plus besoin de s'interroger sur la mise au point de métriques et de fonctions d'évaluation pertinentes dans le contexte de son propre projet. En quelques clics, les métriques et les évaluations standards disponibles qui sont proposées à l'utilisateur dès l'ouverture du logiciel permettent à partir d'un fichier excel et d'un emplacement géométrique d'obtenir des variations dont le logiciel suggère ensuite automatiquement laquelle retenir. Puisque l'évaluation devient projet, il ne reste plus aux praticiens qu'à concevoir des options plutôt que des espaces. Il faut imaginer la modélisation, en programmer les instructions, parcourir les résultats pour en sélectionner, mais ce n'est plus vraiment nécessaire de se pencher sur les caractéristiques spatiales de ce dernier pour les examiner de près et les ajuster - ce sont les nombres qui les représentent qu'il faut étudier. L'évolution des représentations des projets dans le domaine computationnel est symptomatique de ce changement : les diagrammes de relations entre espaces, les façades floues des réseaux neuronaux et les grilles d'options vues en axonométries miniatures

[44] Voir Chapitre V partie 5.1.2.

prennent le dessus sur les plans, les coupes et les élévations. Il ne s'agit plus de concevoir des espaces, mais des espaces de conception[45].

### 7.2 Discours

Dans la pratique en agence comme dans les recherches, bien que les détails des projets, les outils utilisés et la nature des applications varient d'un milieu à l'autre, on retrouve des marqueurs communs de ces nouvelles pratiques de rationalisation. Mais malgré la rationalisation des pratiques observée dans les projets, le tacite n'est en réalité jamais loin. Il n'est jamais vraiment absent non plus quand les ingénieurs conçoivent et évaluent une structure, ni quand un chercheur d'une discipline dite des sciences dures mène ses expériences, alors même que ce sont des activités dont il est parfois présumé qu'elles ne nécessitent que des savoirs explicites. Les exercices parfois acrobatiques de construction des métriques et des fonctions d'évaluation à partir du savoir architectural qu'impliquent les exemples décrits plus haut en témoignent. Mais la dimension tacite de la pratique, pourtant actée par les praticiens des générations précédentes, disparaît du discours produit pour accompagner et promouvoir projets et outils, au profit d'objectifs de rationalisation clairement formulés. Les techniques d'explicitation du savoir architectural observées sont pour beaucoup des techniques qui existaient déjà avant, mais elles sont maintenant mobilisées dans une optique d'optimisation systématique, que trahit le discours tenu à leur sujet. Celui-ci s'articule autour de quatre arguments clés : quantification, options, efficience, et aide apportée par l'ordinateur.

#### *7.2.1 Optimiser l'architecture*

La quantification, qui permet fonctionnalisme des espaces et objectivité dans la conception, est le premier point sur lequel les praticiens insistent, Au-delà des mesures observées dans les algorithmes et décrites plus haut, nombre d'articles se targuent de pouvoir prendre la mesure de l'empreinte sociale et culturelle de l'architecture, discours récurrent qui dépasse la numérisation de critères spécifiques pour s'inscrire dans une ambition bien plus large de quantification complète des systèmes humains. Le travail de Lee et Lee sur la planification des hôpitaux décrit plus haut donne un exemple typique des ces ambitions :

> *Les premiers résultats indiquent que le cadre permet d'étudier les valeurs d'intégration et les propriétés isovist des principaux espaces fonctionnels. Il permet également de saisir les*

[45] *design space.*

*caractéristiques spatiales et sociales dans la conception des établissements de soins pour personnes âgées*[46].

Un discours qu'on retrouve en préambule de nombreux travaux, comme par exemple chez Li et al, pour qui il s'agit de "*quantifier l'activité humaine*"[47]. Signe de l'importance prise par le sujet, le nombre de numéros du magazine AD s'intéressant à des évaluations techniques au cours de la décennie est conséquent : entre 2000 et 2020, plus d'un numéro sur six est consacré à des considérations sur le rôle des technologies numérique dans la rationalisation de la conception et de la construction[48] et donnent à voir différentes facettes de cette approche. Le numéro *The New Structuralism. Design, Engineering and Architectural Technologies*, édité en 2010 par Rivka et Robert Oxman, les compile ensemble et est à ce titre annonciateur des discussions qui vont caractériser la décennie. L'éditorial rédigé par Helen Castle annonce "*un nouvel ordre dans le design et la construction*", grâce aux technologies numériques et aux géométries non-standards qu'elles permettent de créer[49]. Le texte appelle également à repenser la place de l'ingénieur dans le processus de conception, et du travail d'évaluation de la performance qu'il fournit, et d'en augmenter la prise en considération dans les décisions préliminaires[50]. Les articles rassemblés dans le numéro, écrits par des ingénieurs mais aussi par quelques architectes, traitent non seulement de cette nouvelle approche

[46]"*The initial results indicate that the framework allows the investigation of integration values and isovist properties of the main functional spaces. It also enables the capturing of spatial and social characteristics in the design of aged-care facilitie*s." Lee, J., Lee, H., "Agent-driven accessibility and visibility analysis in nursing units" in *Intelligent and Informed, CAADRIA 2019*; Faculty of Architecture and Design, Victoria University of WellingtonWellington; New Zealand; 15 April 2019 through 18 April 2019; Volume 1, p. 351-360, 2019.

[47] "*movement as the measurement of space, quantifying human activity*" - Li, Lezhi; Renyuan Hu, Meng Yao, Guangwei Huang and Ziyu Tong, "Sculpting the Space: A Circulation Based Approach to Generative Design in a Multi-Agent System", in : Rethinking Comprehensive Design: Speculative Counterculture, Proceedings of the 19th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2014) / Kyoto 14-16 May 2014, pp. 565–574.

[48] Sur 62, 11 sont dédiées à ces questions et 6 à la fabrication. Sur ces 17 numéros, 6 sont parus avant 2010 et 11 après. Il s'agit par ailleurs du deuxième sujet le plus traité après le biodesign.

[49] *The New Structuralism announces a new order in design and construction. With the onset of digital technologies, existing parameters have shifted. The old order of standardised design and its established processes no longer hold sway; contemporary architectural design can now be characterised by irregularity, and an appetite for producing customised non-standard, complex, curvilinear forms.* Castle, H. (2010), Editorial. Archit Design, 80: 5-5.

[50] *In this issue of AD, Rivka Oxman and Robert Oxman are eloquently calling for a new model of architectural production in which architects and engineers work together in a higher level of collaboration. The structural engineer is no longer the fixer brought in during the late design stage to make a design work, but integral to the earliest generative stages. Design is no longer wholly dictated by form with structure following behind; structure becomes integral to form-finding.* Castle, H. (2010), Editorial. Archit Design, 80: 5-5.

globale[51] mais aussi en détail du rôle prépondérant qui y est joué par l'analyse structurale[52], par la science des matériaux[53] et par les processus numériques.

Ces derniers, dans l'article *Optioneering: A New Basis for Engagement Between Architects and their Collaborators*, écrit par Dominik Holzer and Steven Downing[54], sont présentés comme particulièrement utiles pour ce qu'ils nomment l'"*optionnage*"[55]. Présenté comme une nouvelle méthode de management de la collaboration entre architectes et ingénieurs, il s'agit en fait essentiellement d'utiliser les outils numériques pour permettre de discuter des critères les plus importants dans la réalisation du projet et de résoudre les conflits potentiels en examinant les conséquences de la variation de l'un ou l'autre de ces critères sur le projet, grâce à une modélisation paramétrique. La modélisation d'un grand nombre d'options pour un même contexte et cahier des charges est en effet un point clé du discours, qu'on trouve par exemple déjà dans le numéro de AD *Versioning : Evolutionary Techniques in Architecture,* édité en 2002 par l'agence SHoP/Sharples Holden Pasquarelli, qui mettait en avant l'attrait des algorithmes génétiques précisément parce qu'ils permettent d'examiner un grand nombre de versions différentes. Bien qu'il ne soit pas formulé exactement de la même manière, l'argument de la nécessité et de la facilité de démultiplication des options possibles est au fond également au cœur de la définition du *paramétrisme*[56] donnée par Patrik Schumacher. Bras droit de Zaha Hadid depuis les années 1990, et directeur après la mort de cette dernière de leur agence, ou il a importé les technologies numériques et instauré une pratique intensive dès son arrivée, Patrik Schumacher s'investit depuis 2008 dans une articulation théorique des pratiques numériques et computationnelles[57]. Celle-ci se construit autour de l'idée d'un nouveau style architectural, le paramétrisme, qu'il définit comme suit : chaque propriété de chaque élément ou système est soumise à des variations paramétriques, et ce grâce aux outils numériques, mais aussi grâce à l'introduction d'une nouvelle philosophie . L'architecture y est considérée comme une pratique d'ordonnancement des communications sociales grâce à des méthodes de différentiation et de

---

[51] Sobek, W. (2010), Radical Sources of Design Engineering. Archit Design, 80: 24-33. Kara, H. (2010), On Design Engineering. Archit Design, 80: 46-51.
[52] Mangelsdorf, W. (2010), Structuring Strategies for Complex Geometries. Archit Design, 80: 40-45. Bollinger, K., Grohmann, M. and Tessmann, O. (2010), Structured Becoming: Evolutionary Processes in Design Engineering. Archit Design, 80: 34-39.
[53] Oxman, N. (2010), Structuring Materiality: Design Fabrication of Heterogeneous Materials. Archit Design, 80: 78-85. Scheurer, F. (2010), Materialising Complexity. Archit Design, 80: 86-93.
[54] Holzer, D. and Downing, S. (2010), Optioneering: A New Basis for Engagement Between Architects and Their Collaborators. Archit Design, 80: 60-63.
[55] *optioneering*
[56] *parametricism*
[57] Patrik Schumacher, *The Autopoiesis of Architecture*, John Wiley & Sons Ltd., London 2010. Patrik Schumacher, "Parametricism as Style - Parametricist Manifesto", Transcription d'une intervention au Dark Side Club1 , 11th Architecture Biennale, Venice 2008. https://www.patrikschumacher.com/Texts/Parametricism%20as%20Style.htm, consulté le 12 Novembre 2021.

corrélation[58]. Patrik Schumacher faisant partie des praticiens les plus influents de sa génération, dans le champ computationnel comme ailleurs, la notion de paramétrisme est reprise par de nombreux autres. Stanislas Chaillou considère le paramétrisme comme un des "*4 grands mouvements de la systématisation en architecture*"[59]. Rivka Oxman revient elle aussi dans plusieurs de ses textes sur les caractéristiques de différenciation et de corrélation, dont elle fait la clé de voûte de sa théorie de la pensée computationnelle en architecture[60]. Danil Nagy s'y réfère également dans plusieurs articles consacrés au travail de The Living, revenant en détail sur l'espace de design[61] que l'ensemble des options créées constitue, devenu l'élément principal du processus de conception :

*Plus important encore, l'approche paramétrique permet au concepteur de réfléchir à des solutions de conception d'une manière plus profonde et plus dynamique que ne le permettent les méthodes traditionnelles. Dans une approche traditionnelle, le concepteur étudie le problème de conception, intériorise toutes ses contraintes et tous ses objectifs, puis utilise ses compétences et son expérience pour élaborer une seule solution de conception, ou tout au plus une poignée. Avec l'approche paramétrique, les contraintes et les objectifs du problème de conception peuvent être directement intégrés dans le modèle paramétrique, qui peut ensuite être utilisé pour générer automatiquement une variété de solutions. Au lieu de concevoir une solution unique, le concepteur peut maintenant penser à concevoir un "espace" multidimensionnel de conception. Chaque dimension de cet espace de conception représente l'un des paramètres critiques exposés par le modèle paramétrique, et chaque*

---

[58] *In principle every property of every element or complex is subject to parametric variation. The key technique for handling this variability is the scripting of functions that establish associations between the properties of the various elements. However, although the new style is to a large extent dependent upon these new design techniques the style cannot be reduced to the mere introduction of new tools and techniques. What characterizes the new style are new ambitions and new values - both in terms of form and in terms of function - that are to be pursued with the aid of the new tools and techniques. Parametricism pursues the very general aim to organize and articulate the increasing diversity and complexity of social institutions and life processes within the most advanced centre of post-fordist network society. For this task parametricism aims to establish a complex variegated spatial order. It uses scripting to lawfully differentiate and correlate all elements and subsystems of a design. The goal is to intensify the internal interdependencies within an architectural design as well as the external affiliations and continuities within complex, urban contexts. Parametricism offers a new, complex order via the principles of differentiation and correlation.* Patrik Schumacher, "The Parametricist Epoch: Let the Style Wars Begin", AJ - The Architects' Journal, Number 16, Volume 231, 06. May 2010 https://www.patrikschumacher.com/Texts/The%20Parametricist%20Epoch_Lets%20the%20Style%20Wars%20Begin.htm, consulté le 12 Novembre 2021.

[59] Chaillou, Stanislas. Conférence Intelligence artificielle & Architecture au Pavillon de l'Arsenal, le 27 février 2020. Visible sur Dailymotion. Consulté le 11 Juillet 2021. https://www.dailymotion.com/video/x7snxh1.

[60] Theories and Models of Parametric Design Thinking eCAADe 2015 ; Thinking difference: Theories and models of parametric design thinking Design Studies 2017

[61] Voir Nagy, Danil, Lorenzo Villaggi et David Benjamin. "Generative urban design: integrating financial and energy goals for automated neighborhood layout." (2018) ou Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

*variation de conception individuelle peut être trouvée quelque part dans cet espace hyperdimensionnel*[62].

La production de nombreuses versions d'un même projet en fonction des contraintes sert à trouver, si ce n'est la meilleure solution, une des plus performantes, et de garantir le processus de conception le plus efficient. Ces raisons sont résumées comme suit dans la description du récent Centre de l'ETHZ Design ++ :

*Pourquoi ? Augmenter la productivité de la construction, améliorer la qualité de l'environnement bâti et réduire considérablement l'impact écologique. Quoi ? ACV, performance énergétique, ingéniosité structurelle, conception consciente de la fabrication, qualité acoustique, conception innovante, estimations de coûts adaptées. Comment ? Conception informatique, IA et apprentissage automatique, réalité étendue*[63].

Rivka Oxman intègre elle aussi cette idée, en particulier dans un article de 2008 qui résume la définition qu'elle donne du rôle de la performance dans la conception architecturale :

*Le terme [design performatif] implique en outre que la performance peut en soi devenir un facteur déterminant et une méthode de création de la forme architecturale. Dans de telles circonstances, la*

---

[62] *"Most importantly, the parametric approach allows the designer to think through design solutions in a deeper and more dynamic way than possible with traditional methods. In a traditional approach, the designer studies the design problem, internalizes all of its constraints and objectives, and then uses their skill and experience to craft a single design solution, or a handful at most. With the parametric approach, the constraints and goals of the design problem can be directly embedded within the parametric model, which can then be used to automatically generate a variety of solutions. Instead of designing a single solution, the designer can now think of designing a multi-dimensional 'space' of design. Each dimension of this design space represents one of the critical parameters exposed by the parametric model, and each individual design variation can be found somewhere within this hyper-dimensional space." ;* Voir aussi Beyond Heuristics *"À mesure que les outils de conception générative seront plus largement adoptés, le rôle du concepteur passera de la conception d'objets tridimensionnels statiques à la conception d'espaces de conception hautement multidimensionnels pouvant être explorés et optimisés par des systèmes artificiellement intelligents tels que les algorithmes génétiques."- "As generative design tools gain wider adoption, the designer's role will be transformed from designing static three-dimensional objects to designing highly multi-dimensional design spaces that can be explored and optimized by artificially intelligent systems such as genetic algorithms."* Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.

[63] "*Why? Increase construction productivity, improve the quality of the built environment and substantially reduce the ecological impact. What? LCA, Energy performance, structural ingenuity, fabrication aware design, acoustic quality, innovative design, adaptive cost estimates. How? Computational Design, AI & Machine Learning, Extended Reality".* "Design++ - Center for Augmented Computational Design in Architecture, Engineering and Construction" https://ic.ibi.ethz.ch/centres/design--.html, consulté le 12 Novembre 2021. On notera que le centre fait partie du département d'architecture, et non de génie civil.

*conception numérique passe d'un paradigme de conception dans lequel les compétences et les préférences de manipulation formelle du concepteur humain contrôlent le processus de l'extérieur à un paradigme dans lequel la conception est informée par des processus internes d'évaluation et de simulation*[64].

Dans cette approche, que Rivka Oxman baptise *design performatif*[65] mais qu'on retrouve chez de nombreux autres sous le terme *design basé sur la performance*[66], deux entités principales sont mises en opposition dans la description du processus de conception : la géométrie et sa performance. Comme d'autres praticiens du domaines, elle insiste en particulier sur la prise en compte de la performance d'un dispositif spatial avant ou pendant la conception géométrique et non après : on retrouve la logique de pré-rationalisation plutôt que de post-rationalisation mise en place par les praticiens de la période précédente[67]. Dans le numéro AD *Performance oriented architecture*, édité par Michael Hensel en 2013[68], la définition donnée par Rivka Oxman du design performatif est élargie : la performance est définie assez largement, en la liant à l'agentivité d'un ensemble d'éléments liés entre eux par des relations et évoluant au sein d'un environnement dynamique : le monde. Il s'agit donc d'évaluer la performance de l'architecture au sein de son l'environnement, de la rendre spécifique à son contexte et réactive grâce à un ensemble de mesures climatiques qui permettent ensuite d'ajuster structure et matériau en conséquence. L'enjeu est de s'inscrire grâce à l'adaptation que ces mesures permettent à la fois dans l'impératif écologique et dans l'impératif disciplinaire d'unicité du projet architectural, et c'est pour y parvenir qu'il faut tout évaluer. Ce discours est également repris par beaucoup de praticiens dans le sous-champ du biodesign, dont l'argument devient que grâce au biomimétisme, leur processus de conception est une forme de maximisation des performances. Les projets de l'ensemble du champ sont de plus en plus fréquemment décrits suivant un ensemble de mots-clés du champ lexical de la performance, utilisés dans une majorité des

---

[64] "*Digital design processes support transformation and generation of a geometrical model and they support analytical evaluation of environmental performance based upon simulating physical conditions such as solar or structural loadings. It is the potential of an integration of evaluative simulation processes with digital 'form generation' and 'form modification' models that is implied by the term Performative Design. The term further implies that performance can in itself become a determinant and method for the creation of architectural form. In such circumstances digital design diverges from a design paradigm in which the formal manipulative skills and preferences of the human designer externally control the process to one in which the design is informed by internal evaluative and simulation processes*". Voir aussi *"Performance-based design may be generally considered an approach in which building performance becomes the guiding factor in design. Performance-based models in architecture may be defined as the exploitation of building performance simulation for the modification of geometrical form towards the objective of optimizing a candidate design"*. Oxman Rivka (2008) "Performance-based Design: Current Practices and Research Issues", *International Journal of Architectural Computing*.

[65] *performative design*

[66] *performance based design*

[67] Voir chapitre IV.

[68] Michael Hensel (ed.), *Performance-Oriented Architecture: Rethinking Architectural Design and the Built Environment*, Wiley & Sons, 2013.

publications des praticiens mentionnés jusqu'ici dans le chapitre : il s'agit de résoudre des problèmes en définissant des objectifs, de prendre en compte des contraintes, d'itérer pour produire de multiples solutions et de les analyser pour concevoir des bâtiments les plus performants[69].

*7.2.2 L'apport de l'ordinateur*

La course à la performance s'engage à tous les niveaux, de l'espace conçu en lui-même à la puissance de calcul de l'outil - "*10 modèles en 4h*", vante Autodesk au sujet de l'estimation complète des coûts en fonction de la volumétrie réalisée dans Dynamo par MT Højgaard[70]. En effet, c'est bien l'ordinateur, assistant tout-puissant et clairvoyant, qui permet de maximiser les performances et de dépasser les limites de l'exécution humaine. L'emphase est de mise chez la plupart des praticiens, qui soulignent sans cesse la capacité à créer "*des géométries sans précédent*"[71], à traiter de très grandes quantités de données, à générer un nombre d'options vertigineuses, comme par exemple au sujet de la Elbphilharmonie :

> *il fallait générer environ un million de cellules, de deux à six pouces de diamètre, sur la base d'une série de spécifications données par l'acousticien, qui devaient finalement consister en des cellules placées au hasard et de forme individuelle pour les régions de la salle de concert. Cette tâche considérable, impossible à réaliser par des moyens conventionnels, a finalement été résolue par le*

[69] Un bel exemple ici : *""Early stages of the design process require the various stakeholders to define objectives, iterate multiple solutions, evaluate, and analyze these solutions with respect to the pre-defined objectives and make decisions with a high degree of uncertainty based on the information generated (Gerber et al. 2012). In addition, a central challenge for designers is to design buildings which perform on multiple fronts (i.e. which work economically, socially and technically) using what are often conflicting and competing objectives (Maver 1987). Researchers have used Multi-Criteria Design Optimization (MCDO) methods based on Genetic Algorithms (GA) to rapidly generate and evaluate multiple design solutions through domain integration especially in the early stages of design (Shea et al. 2004; Koeugh and Benjamin 2010; Lin and Gerber 2013; Bradner et al. 2014).Even with the increasing use of MCDO in the building industry, it has yet to be fully embraced as a vital part of the design process (Evins 2013). There are several reasons for the slow assimilation of MCDO tools in design. Designers must visually and analytically compare solutions resulting from MCDO in order to fully evaluate design trade-offs with regards to quantifiable goals, preferences, and constraints (Michalek a d Papalambros 2002; Roy et al. 2008; Haymaker and Flager 2009; Cunha et al. 2011; Marsault 2013). Rigorous analysis and comparison of the design solutions using both quantitative and qualitative criteria becomes essential to identify 'good/sub-optimal' solutions, therefore improving decision making (Cunha et al. 2011; Turin et al. 2011; Benjamin 2012). "* Ashour, Yassin et Kolarevic, Branko, "Heuristic Optimization in Design", in *ACADIA 2015: Computational Ecologies: Design in the Anthropocene* [Proceedings of the 35th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-53726-8] Cincinnati 19-25 October, 2015), p. 357-369.

[70] "MT Højgaard automates modeling and pricing using Dynamo", https://dynamobim.org/home_usecases/use-case-3/, consulté le 12 Novembre 2021.

[71] "*produce unprecedented geometries*" Akizuki, Yuta, Bernhard, Mathias, Kakooee, Reza, Kladeftira, Marirena and Dillenburger, Benjamin, "Generative Modelling with Design Constraints - Reinforcement Learning for Object Generation", in D. Holzer, W. Nakapan, A. Globa, I. Koh (eds.), *RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th CAADRIA Conference - Volume 1*, Chulalongkorn University, Bangkok, Thailand, 5-6 August 2020, pp. 445-454, 2020.

*développement par ONE TO ONE d'algorithmes personnalisés utilisant des définitions paramétriques pour les cellules*[72].

Autre exemple chez Danil Nagy, au sujet du projet de quartier résidentiel d'Alkmaar :

*La conception générative permet aux concepteurs d'exploiter la puissance de calcul pour explorer de vastes espaces de conception et dériver des solutions de conception qui sont à la fois nouvelles et performantes par rapport à un ensemble d'objectifs choisis. Ce processus s'appuie sur un ensemble de technologies, notamment un logiciel de conception paramétrique pour modéliser l'espace de toutes les solutions possibles, un logiciel de simulation pour dériver des mesures permettant d'évaluer chaque conception potentielle, et des solveurs d'optimisation tels que l'algorithme génétique (AG), qui peuvent rechercher automatiquement dans l'espace de conception les conceptions les plus optimales*[73].

Les capacités de calcul de plus en plus grandes des ordinateurs permettent de traiter des modélisations ingérables auparavant par l'esprit humain, et donnent donc accès à une grande complexité géométrique - au stade de la conception comme au stade de la fabrication - et à des processus eux aussi de plus en plus complexes, donc à une rationalisation beaucoup plus large de la pratique architecturale car beaucoup plus de paramètres peuvent être pris en compte. La vitesse du calcul permet d'intégrer dès l'étape de conception des résultats d'analyse et donc de les prendre en compte, puisqu'à chaque modification de la géométrie répond immédiatement une mise à jour des performances. Le travail est ainsi plus efficace, libéré des limites de l'esprit humain - non seulement en termes de capacités de calcul, mais aussi en ce qui concerne la subjectivité des évaluations. Après tout, l'esprit humain est faillible, comme en témoigne la myriade de commentaires sur le processus de

---

[72] pdf one to one "*this meant generating approximately one million cells, two to six inches in diameter, based on a range of specifications given by the acoustician, which would ultimately consist of randomly placed, individually shaped cells for regions of the concert hall. This momentous task, impossible to achieve by conventional means, was ultimately resolved by ONE TO ONE's development of custom algorithms employing parametric definitions for the cells".* ONE TO ONE, "One Million Cells and Ten Thousand Panels: Digital Fabrication of Elbphilharmonie's Acoustic Interior", http://onetoone.net/wp-content/uploads/2017/01/161128_PR_Elbphilharmonie.pdf, consulté le 12 Novembre 2021.

[73] "*Generative Design allows designers to tap into the power of computation to explore large design spaces and derive design solutions which are both novel and high-performing relative to a chosen set of goals. This process relies on a set of technologies including parametric design software for modeling the space of all possible solutions, simulation software for deriving metrics to evaluate each potential design, and optimization solvers such as the Genetic Algorithm (GA) which can automatically search through the design space to find the most optimal designs"* Nagy, Danil, Lorenzo Villaggi et David Benjamin. "Generative urban design: integrating financial and energy goals for automated neighborhood layout." (2018). Brown, N.C. & Mueller, C. (2016). Design for structural and energy performance of long span buildings using geometric multi-objective optimization. *Energy and Buildings*, 127: 748-761.

design humain et ses jugements peu rationnels, par exemple chez Danil Nagy ou chez Robert Youmans et Tomasz Arciszewski :

> *Comme les algorithmes n'ont pas d'intuition inhérente ou de parti pris pour la résolution de problèmes spécifiques, ils peuvent aider les architectes à s'affranchir des heuristiques que l'on trouve dans les processus de conception traditionnels, ce qui permet de découvrir non seulement des solutions de conception très performantes, mais aussi des solutions inédites*[74].

> *L'heuristique peut également conduire à une "fixation du design", c'est-à-dire que les designers limitent leur créativité en raison d'une dépendance excessive à l'égard de caractéristiques de designs préexistant*s[75].

Les technologies numériques offrent une productivité renouvelée, alors même que c'est la bête noire de l'industrie de la construction depuis longtemps[76] : les diagrammes montrant l'intérêt de l'intervention des ingénieurs plus tôt dans le processus de conception pullulent[77], comme les arguments sur la robotique qui serait plus efficace pour la fabrication. Sous la plume de Danil Nagy, c'est la productivité des concepteurs de projets qui augmente, qui grâce aux outils numériques peuvent se concentrer sur d'autres tâches que l'évaluation technique, prise en compte en arrière plan par les ordinateurs :

---

[74] *"Because the algorithms have no inherent intuition or bias for solving specific problems, they can help architects break free of the heuristics found in traditional design processes, leading to the discovery of not only high-performing design solutions but also novel ones" Voir aussi "Since human designers are limited in their ability to consider all these decisions at once, we tend to break down complex problems into a sequence of smaller problems that can be more easily solved based on "rules of thumb," or strategies that have been proven to work in the past. In computer science, simple proven strategies for solving complex problems are called "heuristics." Although heuristics can be useful for efficiently generating workable solutions, when applied to complex problems, they are not guaranteed to produce the best overall solution."* - Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

[75] "*[heuristics] can also lead to "design fixation," where "designers limit their creative output because of an overreliance on features of preexisting designs"* Youmans, R., & Arciszewski, T. (2014). Design fixation: Classifications and modern methods of prevention. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 28(2), 129-137.

[76] "La construction, mauvais élève en matière de productivité", batiactu, 27 Juin 2019. https://www.batiactu.com/edito/construction-mauvais-eleve-matiere-productivite-56801.php, consulté le 12 Novembre 2021.

[77] Par exemple ici : Rolvink, A., Mueller, C., & Coenders, J. (2014). State of the art of computational tools for conceptual structural design. Proceedings of the IASS-SLTE 2014 Symposium, ou il sert de figure principale.

*La recherche prouve la notion que les tâches répétitives nécessitant une expertise technique prescrite sont instanciables, ce qui permet aux architectes d'investir du temps dans d'autres aspects du projet, améliorant ainsi la productivité de l'équipe*[78].

D'ailleurs, même celle des employés est augmentée par le placement informatisé de leurs postes de travail[79]. Patrik Schumacher lui, asserte régulièrement que le paramétrisme, en faisant appel aux outils numériques, permet d'accéder à une complexité seule à même de permettre l'ordonnancement du monde contemporain[80], la gestion de cette complexité du monde, la production d'un algorithme y répondant suffisant à leur tour à faire projet. En cas de problème, nous indique à sa suite Danil Nagy, il suffit simplement de plus de puissance de computation, ou d'un autre algorithme :

*Enfin, le flux de travail peut être amélioré en intégrant d'autres types de modélisation, notamment l'apprentissage automatique, pour quantifier les aspects des conceptions qui sont difficiles ou impossibles à calculer directement. Cela est particulièrement intéressant car cela pourrait permettre à l'ordinateur d'acquérir des connaissances sur divers facteurs de conception tels que le confort, la beauté ou la nouveauté, qui sont essentiels à une bonne conception mais qui ont toujours été difficiles à associer à un ordinateur*[81].

---

[78] "*The research proves the notion that repetitive tasks needing prescribed technical expertise are instantiable, allowing architects to invest time in other aspects of the project enhancing team productivity*." - Das, S., Day, C., Dewberry, M., Toulkeridou, V. et Hauck, A.. « Automated Service Core Generator in Autodesk Dynamo - Embedded Design Intelligence aiding rapid generation of design options ». In Herneoja, Aulikki; Toni Österlund and Piia Markkanen (eds.), Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 2, University of Oulu, Oulu, Finland, 22-26 August 2016, pp. 217-226. CUMINCAD, 2016.

[79] Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.

[80] *"Parametricism is architecture's answer to contemporary, computationally empowered civilization. Parametricism is the only style that can take full advantage of the computational revolution that drives contemporary civilization. More specifically it is the only style congenial to recent advances in structural and environmental engineering capacities based on computational analytics and optimization techniques. All other styles are incapable of working with the efficiencies of the adaptive structural and tectonic differentiations that issue from the new engineering intelligence, i.e. they force its adherents to waste this opportunity and thus to waste resources. So, once contemporary architects take those performance conditions seriously they are nearly inevitably led to Parametricism and the geometric transcoding of parameter variations into differentiated geometries. This much pertains to Parametricism's obvious superiority in terms of the built environment's technical functionality. What is perhaps less obvious but by no means less compelling is Parametricism's superiority with respect to the advancement of the built environment's social functionality. Due to its versatile formal and spatio-organisational repertoire Parametricism is the only contemporary style that can adequately address the new societal tasks posed to architecture by the new social dynamics engendered by the information age. Accordingly, Parametricism is by now addressing all major urban building tasks, on all scales."* - Schumacher, P., « Social Performativity - Architecture's Contribution to Societal Progress ». Consulté le 11 juillet 2021. https://www.patrikschumacher.com/Texts/AU%20Parametricism.html.
(idée reprise par exemple par Alisa Andrasek)

[81] "*Finally, the workflow can be improved by integrating other types of modelling, particularly machine learning, for quantifying aspects of the designs that are difficult or impossible to compute through direct*

Une partie des citations données jusqu'ici proviennent de textes décrivant des outils nouvellement publiés[82]. En effet, si ces textes commencent par décrire quelles pratiques de conception ces outils permettent, la description des outils en elle-même, dans ces textes et ailleurs, est le prolongement de cette rhétorique de la rationalisation. Ceci est visible également dans la courte description des outils sur les sites ou ils sont mis à disposition avec leur documentation, par exemple dans la description de DynaSpace, qui *"emprunte des idées au diagramme à bulles et les combine avec les algorithmes de résolution de contraintes géométriques (du paquet DynaShape) pour appliquer et optimiser les exigences concernant les entités spatiales et leurs interrelations"*[83]. De plus en plus de projets s'inscrivent avant tout dans une mécanique d'optimisation, mais de plus en plus d'outils dédiés à cela sont aussi disponibles. Un très grand nombre des plugins publiés sont annoncés comme des outils à mobiliser dans une optique de performance design, même au sein de Grasshopper dont l'histoire montre pourtant que c'est un logiciel aux ambitions autres initialement[84]. Les outils deviennent des outils d'optimisation de quelque chose, comme par exemple Ameba, "*un outil d'optimisation topologique (...), qui permet l'optimisation de modèles géométriques 2D et 3D. (...) Les utilisateurs peuvent, selon les exigences de conception, appliquer différentes conditions de chargement et de limite au domaine de conception initial. Au cours du processus de calcul par le logiciel, le domaine de conception évoluera vers différentes formes, pour finalement atteindre une forme organique structurellement efficace*"[85]. De manière générale, la description des outils intègre la notion de performance et/ou d'optimisation et le vocabulaire associé, même quand ce ne sont pas des outils construits directement sur des typologies algorithmiques d'optimisation, comme par exemple EvoMass, qui sert à concevoir rapidement des volumétries, mais dont la description indique :

---

*calculation. This is particularly interesting because it might allow the computer to develop knowledge of various design factors such as comfort, beauty, or novelty that are crucial to good design but have traditionally been difficult to relate to a computer."* - Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.

[82] Notamment les citations de Danil Nagy, qui proviennent d'articles présentant les nouveaux outils créés par Autodesk pour Revit et Dynamo.

[83] *"DynaSpace borrows ideas from the bubble diagram and combines it with the geometric constraint solving algorithms (from the DynaShape package) to enforce and optimize for the requirements regarding the space entities and their inter-relationships."* - Dynamo Team, "Space planning in Dynamo with DynaSpace", https://dynamobim.org/space-planning-in-dynamo-with-dynaspace/, consulté le 12 Novembre 2021.

[84] Voir chapitre IV et chapitre V partie 1.2

[85] *Ameba is a topology optimization tool based on the BESO method, which provides optimization for 2D and 3D geometrical models. Ameba is being developed by Yi-Min Xie's teams. Users may, according to design requirements, apply different loading and boundary conditions to the initial design domain. During the computational process by the software, the design domain will evolve into various shapes, and eventually reach an organic form that is structurally efficient."* - https://www.food4rhino.com/en/app/ameba ; consulté le 11 Juillet 2021.

*EvoMass (...) est un outil intégré pour la génération, l'optimisation et l'exploration de la conception de masse de bâtiments basée sur la performance, en particulier pour l'exploration de la conception basée sur l'optimisation. L'exploration de la conception basée sur l'optimisation a pour but de tirer parti de l'optimisation comme moyen d'extraction de l'information pour réaliser un processus de synthèse de la conception tenant compte de la performance / informé de la performance, plutôt que de rechercher purement une solution de conception*[86].

Autre exemple encore, Zebra, "*utilisé pour atteindre les objectifs de conception en utilisant des agents*"[87] alors qu'il s'agit d'un plug-in pour recourir à des systèmes multi-agents classiques. Il s'agit désormais, pour qu'ils soient utilisés, de produire des outils qui soient capables soit d'automatiser[88], soit d'optimiser - et de préférence les deux à la fois.

Au-delà de la capacité technique croissante à produire des outils permettant la recherche d'une meilleure solution de manière systématique, cette obsession de l'optimisation est le marqueur d'une pensée de la conception à part entière. L'heuristique doit y être quantifiée, cadrée et intégrée à un processus rationnel et maîtrisé, et les contraintes matérielles servir de fil conducteur[89]. Chez Danil Nagy, le design génératif devient exclusivement la pratique de l'optimisation par le recours aux algorithmes : "*L'application d'algorithmes d'optimisation métaheuristiques à des problèmes de conception architecturale est communément appelée "conception générative".*"[90] - la définition du design génératif telle que donnée précédemment par David Stasiuk, par exemple, est dépouillée de sa dimension créative[91]. Chez Caitlin Mueller, le design sur catalogue devient un paradigme à part

---

[86] *EvoMass (formerly known as PBMDGO system) is an integrated tool for agile performance-based building massing design generation, optimization, and exploration, particularly for optimization-based design exploration. Optimization-based design exploration is aimed to leverage optimization as a means of information extraction for achieving a performance-aware / performance-informed design synthesis process rather than purely looking for a design solution."* - https://www.food4rhino.com/en/app/evomass ; consulté le 11 Juillet 2021.

[87] "*used to achieve design goals using agents"* - https://www.food4rhino.com/en/app/zebra ; consulté le 11 Juillet 2021.

[88] Voir le registre de la facilitation au chapitre VI.

[89] Par exemple ici Kilian A. Design Innovation through Constraint Modeling. International Journal of Architectural Computing. 2006;4(1):87-105. ou ici Aish R., Woodbury R. (2005) Multi-level Interaction in Parametric Design. In: Butz A., Fisher B., Krüger A., Olivier P. (eds) Smart Graphics. SG 2005. Lecture Notes in Computer Science, vol 3638. Springer, Berlin, Heidelberg.

[90] "*The application of metaheuristic optimization algorithms to architectural design problems is commonly referred to as generative design"* - Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

[91] Stasiuk, D. « Design Modeling Terminology ». PROVING GROUND, 13 juin 2018. Consulté le 11 Juillet 2021. https://provingground.io/2018/06/13/design-modeling-terminology/.

entière[92], et les commentaires sur la place des ingénieurs et de leur équipes en regard de celle de l'architecte sont bien le marqueur d'une vision différente du processus de conception elles aussi . Pour un certain nombre de praticiens, il ne s'agit pas simplement d'une manière d'envisager la conception, mais d'une conception du monde, dont l'organisation découle de principes rationnels que l'architecte doit admettre pour ensuite s'y conformer dans sa pratique C'est le propos que tient Elias Guenoun dans sa préface au catalogue du pavillon Seroussi : «*l'instabilité formelle du monde ne doit pas masquer l'extraordinaire rationalité de ses mécanismes internes de formations*»[93]. Ce qui se trouve en dehors n'est qu'un détail dont il n'est pas nécessaire de discuter, voire qu'il faut éradiquer, comme on le lit chez Patrik Schumacher :

> *Bien que chaque architecte ait une compréhension intuitive des protocoles d'interaction normatifs qui s'attachent aux différentes zones désignées par le cahier des charges et qu'il en sache généralement assez sur les modèles d'occupation des utilisateurs attendus et souhaités, de telles intuitions ne peuvent pas donner une orientation sûre sur la performance sociale relative des conceptions alternatives pour les environnements d'entreprise vastes et complexes. L'intuition doit ici être remplacée par des simulations capables de traiter des milliers d'agents interagissant dans un environnement de plusieurs centaines d'espaces. Lorsque les comparaisons quantitatives et l'optimisation sont visées, l'intuition échoue déjà dans des environnements beaucoup plus petits et plus simples*[94].

### *7.2.3 Un discours parfois biaisé*

Quantification, options, efficience et aide apportée par l'ordinateur sont des éléments de discours déjà anciens, présents dans les propos des praticiens des années 60[95]. Dans son livre *Evolutionary*

---

[92] Brown, N. & Mueller, C. (2017). Designing with data: moving beyond the design space catalog. ACADIA 2017.

[93] Guenoun, E. "Préface", In Seroussi, Natalie, Elias Guenoun, et Maison rouge - Fondation Antoine de Galbert, éd. *Pavillon Seroussi: biothing ; DORA (Design Office for Research and Architecture); EZCT Architecture & Design Research ; Gramazio & Kohler ; IJP - George L. Legendre ; Xefirotarch*. Orléans: Éditions HYX, 2007. Une assertion qu'on retrouve aussi ailleurs alors même qu'on a vu que les projets présentés dans le catalogue relèvent d'une maîtrise de la pratique architecturale au-delà de ces principes rationnels.

[94] "*While every architect has an intuitive grasp of the normative interaction protocols that attach to the various designated areas that the design brief indicates and usually knows enough about the expected and desired user occupancy patterns, such intuitions cannot give a secure guidance on the relative social performance of alternative designs for large, complex corporate environments. Intuition must here be substituted by simulations that can process thousands of agents interacting across an environment of hundreds of spaces. When quantitative comparisons and optimization is aimed at, then intuition fails already in much smaller, simpler settings."* - Schumacher, P. « Advancing Social Functionality via Agent Based Parametric Semiology ». Consulté le 11 juillet 2021. https://www.patrikschumacher.com/Texts/Operationalising%20Architectures%20Core%20Competency.html.

[95] Pour plus de détails voir chapitre II.

*Architecture*, John Frazer pointe l'importance de l'optimisation, et recourt fréquemment à l'analogie de la sélection naturelle pour expliquer le rôle des critères de sélection et appuyer la nécessité d'une structure algorithmique hiérarchique qui permette d'évaluer les propositions[96]. Dans les explications fournies au sujet du projet Whitehall, Lionel March et Leslie Martin envisagaient déjà la computation au service d'une approche scientifique de la conception architecturale[97]. Les pratiques d'automatisation de la génération de plan explorées par Per Galle ou William Mitchell avaient elles aussi une vocation systématique[98]. Les évaluations spatiales par le biais d'isovists datent des travaux de Michael Benedikt ou John Peponis, et le discours qui les accompagne met en avant les mêmes possibilités de quantification de l'espace[99]. Avec le concept de machine à architecture, Nicholas Negroponte promouvait le même rôle de conseil à travers la communication de données précises, les mêmes avantages offert par l'ordinateur que Caitlin Mueller ou Danil Nagy[100]. Christopher Alexander décrivait l'ordre et l'harmonie, le sentiment comme des paramètres quantifiables, mesurables grâce à une approche scientifique méthodique[101]. Le discours des praticiens du champ computationnel contemporain, bien qu'il souligne les avantages indéniables du recours aux algorithmes, perd cependant de ses nuances en comparaison du discours des praticiens plus anciens. Dans les propos les plus récents, en effet, le discours insiste sur la productivité / la performance, plutôt que de discuter des conditions de l'explicitation informatique comme avant. Toute une littérature sur le sujet, auparavant intégrée au même champ de travail, a entre-temps migré vers les design studies et ne contribue plus à cette réflexion[102]. Bien que le savoir-faire architectural soit reconnu, le reste du propos vise essentiellement à éliminer ce qui est associé purement à de l'intuition. L'expertise architecturale évolue également : elle n'est pas souvent niée, mais elle est diminuée ou passe au second plan. Danil Nagy en donne dans l'article Beyond Heuristics un bon exemple :

> *Après la sélection d'une stratégie de base d'aménagement de l'espace, il reste encore beaucoup de travail de raffinement et de conception, notamment le choix des matériaux architecturaux et la conception des détails de connexion, pour atteindre le niveau d'une conception finale constructible*[103].

---

[96] Frazer, J., *An evolutionary architecture*, Architectural Association Publications, 1995.
[97] March, L., "Mathematics and Architecture since 1960", in Williams, K. and Rodrigues, J.F. (eds.), *Nexus IV : Architecture and Mathematics*, pp. 7-33, Kim Williams Books, 2002.
[98] Per Galle. 1981. An algorithm for exhaustive generation of building floor plans. Commun. ACM 24, 12 (Dec. 1981), 813–825.
[99] Benedikt, M., "To Take Hold of Space: Isovists and Isovist Fields", in *Environment and Planning B Planning and Design,* Vol.6 n.1, pp. 47-65, 1979.
[100] Negroponte, N., *The Architecture Machine*, The MIT Press, 1970.
[101] Alexander, C., *Notes on the Synthesis of Form*, Harvard University Press, 1964. ; La notion de sentiment est exprimée par le mot *feeling*.
[102] Voir les praticiens mentionnés au chapitre II.
[103] "*After a basic space-planning strategy is selected, there is still much refinement and design work to be done, including selecting architectural materials and designing connection details, to get it to the level of a final constructible design."* - Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design

Sélectionner les matériaux et dessiner les détails ne sont pourtant pas les seuls savoirs des architectes, ni les seuls points du processus de conception à ne pas être résolus par le recours à des procédés algorithmiques automatiques. Il indique par ailleurs en conclusion :

*Cependant, lorsque les concepteurs commenceront à adopter ces outils, ils auront besoin de théories et de directives claires sur la façon de concevoir un bon modèle d'espace de conception, d'évaluer sa qualité et de prédire sa capacité à être optimisé par un système autonome. En d'autres termes, la conception d'un modèle de conception générative requiert de l'expérience, de la créativité, un esprit analytique et même des "typologies", tout comme la conception traditionnelle*[104].

L'essentiel de l'article consiste pourtant à détailler le fonctionnement d'un système qui retire aux praticiens non formés à la programmation leur capacité à mettre en place un processus de conception critique - une contradiction qu'on retrouve aussi chez Caitlin Mueller. Ailleurs, Nagy acte l'existence d'un monde de conception en dehors des algorithmes d'optimisation:

*Une fois qu'un ensemble de designs intéressants est sélectionné, elles peuvent être analysées par le concepteur humain, discutées avec les parties prenantes et développées en une conception finale. Il est important de noter que, puisque le MOGA suit un processus stochastique basé sur l'échantillonnage d'un nombre limité de conceptions dans l'espace de conception, la conception optimale globale ne sera pas nécessairement trouvée par le processus de recherche. En outre, comme nous l'avons vu précédemment, tous les aspects importants d'un projet architectural ne peuvent pas nécessairement être représentés sous forme de métrique dans le modèle de conception générative. Certains aspects, tels que la beauté, ne peuvent être quantifiés et doivent donc être pris en compte une fois le processus de conception générative terminé*[105].

---

Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

[104] *"However, as designers begin to adopt these tools, they will need clear theories and guidelines for how to design a good design space model, evaluate its quality, and predict its ability to be optimized by an autonomous system. In other words, designing a generative design model requires experience, creativity, analytical thinking, and even "typologies"—just as traditional design does"* - Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

[105] "*Once a set of interesting designs is selected, they can be further analyzed by the human designer, discussed with the stakeholders, and developed into a final design. It is important to note that since the MOGA follows a stochastic process based on sampling a limited number of designs from the design space, the overall optimal design will not necessarily be found through the search process. Furthermore, as discussed previously, not all*

Pourtant, rares sont les démonstrations récentes qui intègrent ces étapes supplémentaires, ou qui décortiquent tout le processus de projet quand elle existe, en précisant bien le déroulé et notamment la partie qui se déroule après les calculs accomplis par l'ordinateur - une partie pourtant extrêmement importante, comme vu avec le projet *Artificial Zurichness* ou le concours du Pavillon Seroussi[106].

L'idée de l'ordinateur comme un assistant tout-puissant à la conception architecturale, un assistant de l'architecte servant d'intermédiaire entre lui et l'ingénieur, n'est pas exempte de contradictions non plus. Chez Caitlin Mueller et Nathan Brown, on lit par exemple que "*l'optimisation a été proposée comme un moyen d'aller au-delà du retour d'informations et de s'orienter vers le guidage, qui fait de l'ordinateur un participant plus actif au processus de décision*"[107] alors que l'essentiel de leurs travaux montre justement qu'il y a quelqu'un à l'origine des modèles informatiques - cet ingénieur, ces équipes techniques dont il est souhaité qu'ils aient une plus grande place dès le début de la conception. Chez Danil Nagy, la contradiction se fait dans l'autre sens : "*comme l'ordinateur n'a pas d'intuition inhérente en matière de conception, le concepteur humain doit décrire explicitement à l'ordinateur comment déterminer les conceptions les plus performantes*"[108] alors que l'article décrit des algorithmes qui emmènent l'utilisateur là où c'est supposé être le plus pertinent dans le design space, c'est-à-dire des algorithmes qui visent à automatiser presque jusqu'au choix de la meilleure option. Ces contradictions se trouvent à l'articulation entre l'objectif d'automatisation affiché et le besoin d'expertise néanmoins toujours nécessaires en raison des limites techniques des technologies informatiques disponibles[109]. La conception architecturale fait bien appel à des savoir-faire a priori par encore complètement automatisables. En effet, l'assistant si efficace promu par cette nouvelle vague de praticiens n'est pas un ordinateur, c'est un programmeur, toujours aux commandes de la mise en

---

*aspects that are important to an architectural design can necessarily be represented as a metric in the generative design model. Some aspects, such as beauty, cannot be quantified, and thus need to be considered once the generative design process is complete"* - Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.

[106] Voir plus haut pour *Artificial Zurichness* et au chapitre IV pour le Pavillon Seroussi.

[107] "*Optimization has been offered as a way to move beyond data feedback and push towards guidance, which makes the computer a more active participant in the decision-making process."* - Brown, N. et Mueller, C. « Designing with Data: Moving beyond the Design Space Catalog ». In Disciplines and Disruption - Proceedings Catalog of the 37th Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA 2017, 154-63. ACADIA, 2017.

[108] "*Since the computer does not have any inherent intuition about design, the human designer must explicitly describe to the computer how to determine which designs perform better than others."* - Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, Wang, R., Zhao, D. et Benjamin, D. « Project Discover: An Application of Generative Design for Architectural Space Planning ». In Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017). Toronto, Canada: Society for Modeling and Simulation International (SCS), 2017.

[109] Voir Chapitre II et III.

place des algorithmes. Simplement, contrairement aux générations précédentes, programmeur et architecte sont de moins en moins souvent la même personne. L'utilisateur architecte est devenu un amateur, qui ne maîtrise pas forcément la programmation. Cette superposition n'est cependant que rarement traitée depuis Gordon Pask, qui en avait perçu et souligné les différentes dimensions[110]. Malgré la nécessité pratique qu'impliquent les processus informatiques de se pencher dessus, ces enjeux n'ont plus vraiment leur place au sein du discours. L'objectif de démocratisation, qui se fait pressant, mène néanmoins à contourner ces questions dans le discours. L'ordinateur y est présenté directement comme assistant plutôt que d'acter l'existence d'un programmeur derrière. Un programmeur qui n'a souvent pas la même formation, pas la même expertise et pas la même vision.

S'impose notamment chez les ingénieurs et les informaticiens une vision de l'architecte comme artiste dont le rôle porte essentiellement sur les qualités esthétiques du design. De fait, les architectes eux-mêmes nourrissent cette idée. Dire, comme Stanislas Chaillou, que "*l'architecture c'est designer des formes*"[111], en laissant l'ingénieur ou le logiciel s'occuper des questions de structure et de thermique, c'est encore un peu plus ramener l'architecture à une pure question d'esthétique, c'est nourrir ce paradigme de l'architecte-artiste. Le principe de catalogue mis en place par Caitlin Mueller, Danil Nagy ou leurs collègues, en plus de permettre de simplifier l'interaction avec le logiciel, est une construction qui découle directement de cette conception. Le catalogue ne donne pourtant que l'illusion du choix aux utilisateurs, puisque c'est seulement parmi un ensemble de variantes qu'il peut évoluer, sans plus pouvoir entrer dans le détail des paramètres clés de la modélisation[112]. L'expression *design by shopping* que Mueller reprend à Baller le dit bien. Non seulement l'architecte n'a plus que le choix de l'option finale en un clic parmi une grille de possibilités, mais en plus en cas de problèmes, il n'est même plus encouragé à aller mettre les mains dans le cambouis de la modélisation. Même le bricolage pour ajuster la modélisation aux besoins est retiré des mains de l'utilisateur. Il est guidé dans la sélection des paramètres pertinents, le choix des options les plus pertinentes, et l'ajustement de l'algorithme. Un groupe de chercheurs de chez Autodesk explique par exemple dans un article que "*si le système ne parvient pas à définir une conception structurelle réaliste, une série d'options de modification sont présentées à l'utilisateur, notamment le déplacement de la masse, le changement des types de construction ou de l'espacement, ou encore la direction principale des éléments*"[113]. C'était pourtant précisément ce besoin d'ajuster, de bricoler en fonction des ambitions

[110] Voir Chapitre II
[111] Chaillou, Stanislas. Conférence Intelligence artificielle & Architecture au Pavillon de l'Arsenal, le 27 février 2020. Visible sur Dailymotion. Consulté le 11 Juillet 2021. https://www.dailymotion.com/video/x7snxh1.
[112] Voir Chapitre V.
[113] *If the system fails to define a realistic structural design, a range of modification options were presented to the user, including shifting the massing, changing constructions types or spacing, or the primary direction of the member*" Beorkrem, C. ; J. Ellinger. « Multivariate Schematic Design Tooling ». In Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference on

techniques et architecturales de l'utilisateur, qui faisait de Grasshopper ou Processing des vecteurs efficaces de transmission du savoir-faire de l'architecture computationnelle[114]. Bryant le souligne dans le récit de son travail avec Christopher Alexander : "*nous avons longuement parlé des choix, de la structure, de la liberté et de la restriction dans la conception - du fait que trop de choix peuvent conduire à l'apathie, à l'absurdité ou à l'absence totale d'expression*"[115].

Avec la perte de ces nuances, la vision du monde sur laquelle le discours tenu sur les outils algorithmiques s'appuie devient si fortement ancrée qu'elle dérive parfois vers scientisme et positivisme chez certains praticiens du champ computationnel de cette dernière génération. D'une prise de position sur ce que doit être l'architecture - en l'occurrence une pratique scientifique raisonnée -, le glissement se fait parfois vers une foi absolue en la rationalité scientifique, une croyance en un processus historique qui va vers davantage de rationalité scientifique, une croyance que le développement des sciences et techniques est source de progrès. Une forme de positivisme est perceptible dans ce discours, comme cette citation de Shajay Bhooshan le laisse entrevoir : "*dans un avenir à long terme, il semble tout à fait plausible qu'une intelligence artificielle domine et, de manière plus pragmatique, dans un avenir proche, il y a une période de progrès symbiotique d'une richesse exaltante à exploiter et à capitaliser*"[116]. Un positivisme assumé pour certains, mais sans doute aussi parfois un positivisme plus naïf pour d'autres praticiens. Lorsque ce n'est pas une volonté de rationalisation consciente, le champ computationnel contemporain se caractérise par des démarches scientistes, qui cherchent à se calquer sur des pratiques scientifiques originaires d'autres disciplines, sans plus s'interroger sur les qualités propres de la pratique architecturale. La mystique de la toute puissance des algorithmes, de leur objectivité, refait surface[117] : le déterminisme du monde implique que tout est calculable, et l'ordinateur est le moyen de parvenir à cette modélisation totale. Il faut également pointer l'omniprésence de cette mystique dans le discours alors même que les pratiques sont beaucoup plus nuancées. Les textes analysés plus haute parlent régulièrement de design traditionnel en l'opposant au design computationnel : pourtant les pratiques examinées dans le chapitre précédent montrent qu'on peut utiliser les outils computationnels au sein d'une démarche traditionnelle, en les utilisant comme n'importe quels autres outils de la palette, et les pratiques

---

Computer-Aided Architectural Design Research in Asia (CAADRIA 2016) / Melbourne 30 March–2 April 2016, pp. 383-394. CUMINCAD, 2016.

[114] Voir chapitre IV.

[115] "*We talked at length about choices and structure and freedom and restriction in design -- how too many choices could lead to apathy or senselessness or no expression at all."* « Gatemaker or "The Aspen Summit" ». Consulté le 11 juillet 2021. https://www.gregbryant.com/grogbrat/aspen97/index.html.

[116] *"in the long-term future it seems entirely plausible that an artificial intelligence will dominate and, more pragmatically, in the near future there is an exhilaratingly-rich period of symbiotic progress to be worked and capitalized on"* Bhooshan, Shajay. « Realizing Architecture's Disruptive Potential ». Oz 38, no 1 (1 janvier 2016). https://doi.org/10.4148/2378-5853.1556.

[117] Comme aux premières heures de l'IA - voir chapitre II.

examinées ici ont toujours une part de savoir tacite - que ce soit dans la conception structurelle ou dans le développement d'une fonction d'évaluation. Cette mystique se résume en fait à une croyance en l'objectivité absolue des sciences, qui s'exprimerait dans les algorithmes. Cette croyance en l'objectivité des systèmes algorithmiques pourtant mise à mal tant par la philosophie des sciences que par les réalités de l'activité de modélisation.

La philosophie et la sociologie des sciences ont pourtant beaucoup contribué à remettre en question la notion d'objectivité dans les sciences, en étudiant la science en train de se faire et ses conditions d'exercice. Ce sont les pratiques des scientifiques qui font la science, et ces pratiques sont subjectives, reposent sur des connaissances tacites, sont l'objet de dynamiques sociologiques. C'est cet ensemble d'influences sur la science en train de se faire que Bruno Latour décortique dans son ouvrage La Science en Action[118]. Il y examine les habitudes de travail des collectivités scientifiques, et notamment les choix épistémologiques sur lesquelles elles reposent. Les cycles de développement et d'adoption de nouveaux paradigmes scientifiques identifiés par Thomas Kuhn dans La Structure des révolutions scientifiques[119] témoignent aussi du rôle joué par la conviction dans le développement des sciences. Les changements de paradigme tels qu'ils les décrit reposent en effet sur l'intuition de certains praticiens que le paradigme en vigueur ne correspond pas, sur la conviction nécessaire à ces praticiens pour creuser des hypothèses qui paraissent pourtant farfelues à leur communauté scientifique. C'est encore la variété des pratiques en train de se faire que souligne John Pickstone dans son travail sur les sciences, les techniques et la médecine. Il y identifie différentes manières de savoir, qui correspondent chacune à différents modes d'assimilation des connaissances et méthodes de travail[120]. La pensée de Paul Feyerabend représente une position radicale et souvent critiquée, mais l'idée qu'il développe que les faits eux-mêmes ne sont qu'idéologie relève elle aussi de la prise en compte des subjectivités individuelles dans la production des discours et des connaissances collectives[121]. Enfin, l'épistémologie de la modélisation montre que dans le domaine informatique également, les modèles utilisés suivent un processus de construction qui est le lieu de décisions qui les structurent, des décisions qui même lorsqu'elles sont fondées relèvent en partie de l'expertise et de la subjectivité des praticiens qui les prennent[122]. Les algorithmes du champ computationnel ne font pas exception. La notion de traduction sur laquelle nous nous sommes reposés jusqu'ici pour commenter le

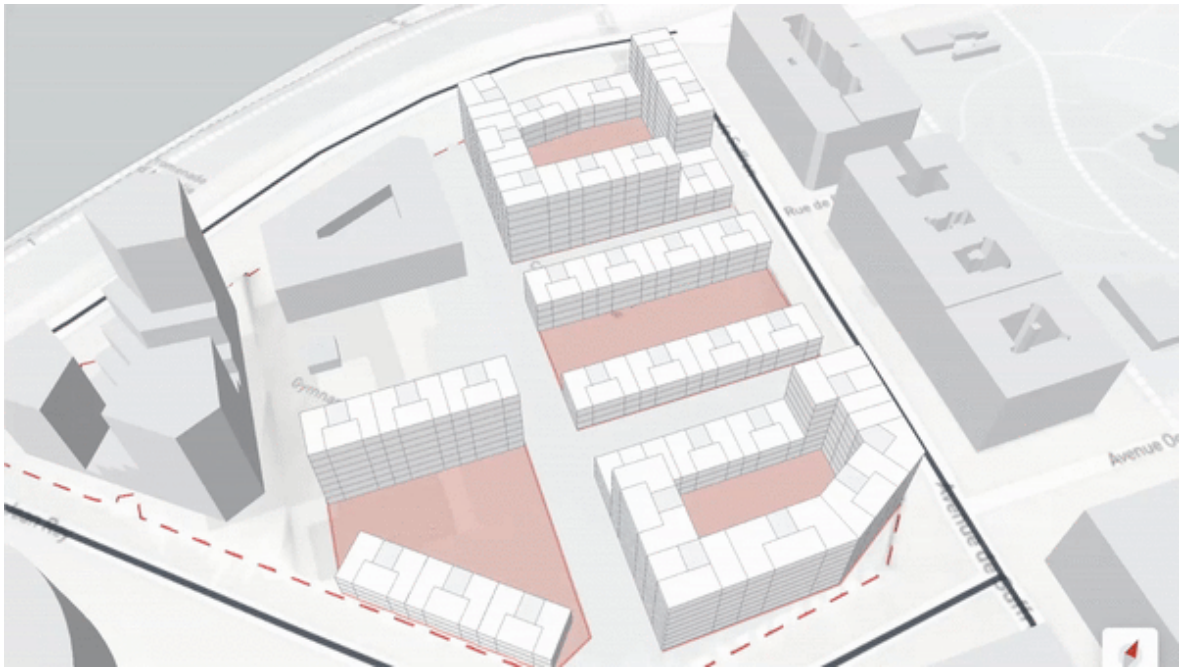
[118] Bruno Latour, *La Science en action*, traduit de l'anglais par Michel Biezunski ; texte révisé par l'auteur, Paris, La Découverte, 1989.
[119] Thomas Kuhn (trad. de l'anglais par Laure Meyer), *La structure des révolutions scientifiques*, Paris, Flammarion,, 2008 (1re éd. 1962).
[120] John Pickstone, *Ways of Knowing: A New History of Science, Technology, and Medicine*, University of Chicago Press, 2001.
[121] Paul Feyerabend, *Contre la méthode, Esquisse d'une théorie anarchiste de la connaissance* (1975), Paris, Le Seuil, 1979 ; éd. poche, Paris Le Seuil, 1988, coll. "Points sciences".
[122] Eric Winsberg, *Science in the age of computer simulation,* University of Chicago Press, 2010.

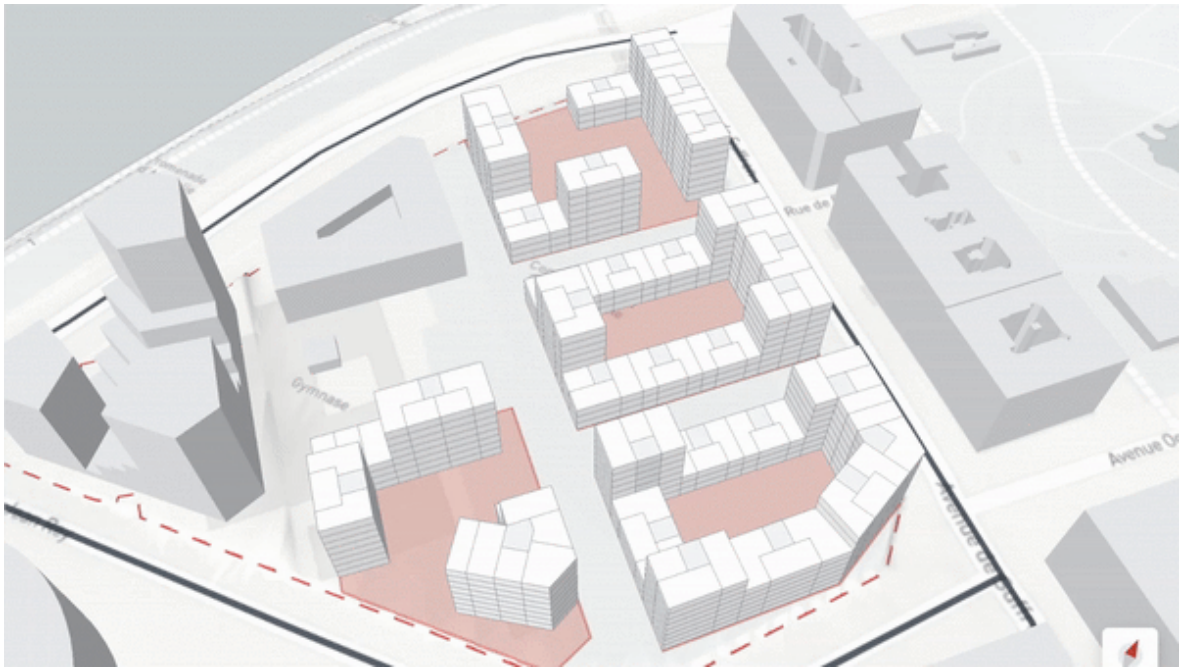
développement de ce champ est cruciale pour cette raison : bien qu'elle disparaisse des débats, elle ne disparaît pourtant pas des pratiques. Ainsi, même dans les projets qui s'affirment comme relevant d'une objectivité produite par la machine, le savoir-faire des architectes a encore un rôle à jouer, et le propos sur le rôle des sciences, bien qu'il dénote une position théorique plus affirmée, peut être questionné en regard du corpus existant en philosophie des sciences. Les nouvelles dynamiques de formation et les nouveaux outils disponibles encouragent un rapport déséquilibré aux sciences au sein du champ computationnel. Ceci se traduit par des pratiques de rationalisation conscientes pour les praticiens dont les connaissances en programmation sont préservées par leurs trajectoires, ou pour les praticiens amateurs par des pratiques de l'ordre de ce qu'on appelle le culte du cargo[123] : une imitation des pratiques scientifiques sans savoir vraiment pourquoi et dans quel objectif.

Au fur et à mesure que les outils computationnels sortent du réseau et du champ de pratiques initial, et plus ils arrivent dans des agences conventionnelles, plus les projets réalisés sont eux aussi conventionnels. Les praticiens s'y appuient sur les boîtes noires que constituent pour les programmeurs amateurs les outils computationnels. Nous avons vu au chapitre VI que ces nouveaux outils du champ reposent sur des mécanismes de fonctionnalisation. Les extensions sont des modules algorithmiques beaucoup plus appliqués, qui offrent le recours directement à des fonctionnalités ou programmes prédéfinis. Ces outils prennent aussi beaucoup plus souvent en compte les normes en vigueur dans l'industrie de la construction. C'est le cas du Autodesk Core Generator que nous avons décrit plus haut, mais aussi par exemple des plug-ins Grasshopper CAALA[124] ou eclipse[125]. Le premier permet de calculer la consommation énergétique d'un bâtiment et de s'assurer de sa conformité en regard des normes thermiques, le second est un outil de conception des stades dans le respect des normes de circulation. Les nouveaux outils se font ainsi le vecteur d'une normalisation et d'une standardisation de la production. En comparaison des projets décrits dans les chapitres précédents, en particulier ceux mentionnés au chapitre IV, la production s'uniformise. Formellement, les volumes deviennent systématiquement rectangulaires, cubiques ou angulaires, bien loin de la complexité et des volutes tortueuses des projets spéculatifs de la génération précédente. L'iconographie utilisée pour illustrer les packages Dynamo et les plug-ins Grasshopper les plus récents témoigne de cette uniformisation formelle (Fig. 3). On assiste donc à la fonctionnalisation,

[123] L'expression vient du champ de l'anthropologie et désigne l'activité d'aborigènes imitant l'attitude d'opérateurs radio occidentaux requérant du ravitaillement, pour faire arriver les avions-cargos apportant le ravitaillement. En 1974, Richard Feynman reprend l'expression dans une intervention à CalTech pour dénoncer des pratiques scientifiques approximatives. Enfin, en informatique, on parle de culte du cargo lorsqu'un bout de code est copié-collé sans etre compris par le programmeur dans l'espoir qu'il fasse la chose attendue.
Richard Feynman, « Cargo Cult Science », Caltech, 1974, http://calteches.library.caltech.edu/51/2/CargoCult.pdf, consulté le 12 Novembre 2021.
[124] https://www.food4rhino.com/en/app/caala-rhino, consulté le 07 Novembre 2021.
[125] https://www.food4rhino.com/en/app/eclipse, consulté le 07 Novembre 2021.

l’uniformisation, et la standardisation des pratiques computationnelles. Cette évolution reflète les exigences de l'industrie de la construction, et le positionnement théorique devenu dominant. En effet, malgré l’ambition qui perce dans les discours de faire passer cette rationalisation des pratiques pour un progrès scientifique, il s’agit pourtant d’un positionnement, donc nous allons examiner l’origine de la domination à cette période la plus récente du champ.





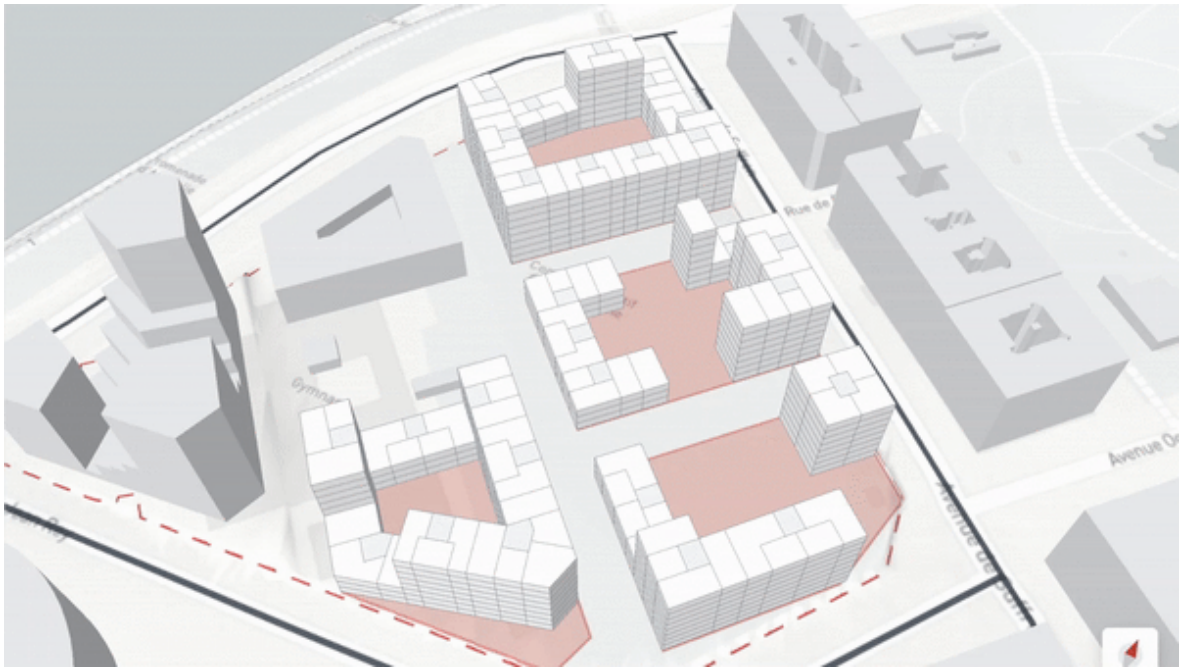


*Figure 3. "Get creative with generative design"*

### 7.3 Origines

Le développement actuel du champ informatique, tout en ayant assuré dans le passé un espace d'existence à l'approche heuristique et à l'approche rationaliste, crée pourtant un biais favorisant cette dernière dans la dynamique de démocratisation contemporaine. Plusieurs facteurs influencent ce biais de rationalisation. D'abord, l'héritage d'un biais mécaniste, avec une résurgence récente. Des glissements sémantiques dans la description de la nature des calculs effectués par un ordinateur mènent de plus à une évaluation erronée des capacités actuelles d'automatisation de la prise de décision. Deuxièmement, la bifurcation entre l'industrie et l'académie, l'expansion de la branche industrielle et la structure des réseaux qui en découle dans le domaine transforment les outils algorithmiques en boîtes noires pour de nombreux utilisateurs. Troisièmement, la dynamique de démocratisation actuelle est à l'œuvre sans remettre en cause ces boîtes noires et sans réflexion épistémologique poussée sur la nature de la discipline. Tout cela contribue à renforcer le biais de rationalisation dans les pratiques actuelles, au détriment de la diversité des pratiques qui caractérise non seulement le domaine informatique depuis ses premiers développements, mais aussi l'architecture en tant que discipline elle-même.

#### *7.3.1 Une proximité renouvelée avec l'IA*

Nous avons vu au chapitre II comment les liens entre intelligence artificielle et architecture computationnelle se sont forgés à la naissance des deux champs. Nous avons également vu au chapitre III comment ces liens se sont déliés à la faveur des hivers de l'intelligence artificielle et du développement de l'infographie. Enfin, nous avons aussi vu comment le champ de l'intelligence artificielle a fait émerger au cours des années 1950 à 1980 deux approches théoriques distinctes de la computation, dont le champ computationnel en architecture a hérité. Mais au-delà de ces deux interprétations du processus de pensée et de sa possible mécanisation, vastes projets théoriques, le champ computationnel a aussi une filiation avec le champ de l'intelligence artificielle à une échelle plus fine. La théorisation du processus de conception lui-même est formulée dans le champ sémantique de l'optimisation et issue du domaine de l'intelligence artificielle. Le terme de *problème malicieux*, ou *problème mal structuré*[126] est largement utilisé dans la littérature pour décrire les tâches auxquelles sont confrontés les concepteurs. Cette terminologie est l'héritière d'une chronologie débutant en 1956 avec un texte de John McCarthy, *The inversion of functions defined by Turing machines*. Cette histoire se poursuit en 1961 avec un texte de Marvin Minsky, *Steps toward Artificial Intelligence*[127]. Les deux chercheurs mettent en avant la notion de problèmes bien définis, c'est-à-dire des problèmes formalisés de manière à pouvoir être traités et résolus par un ordinateur. En 1964, Walter Reitman théorise dans *Heuristic decision procedures, open constraints, and the structure of ill-defined problems* la notion opposée de problèmes mal définis. Celle-ci est ensuite développée plus avant en 1967 par Alan Newell dans *Heuristic programming: ill-structured problems*[128]. En 1969, Charles Eastman fait le lien entre problèmes mal définis et conception dans son texte *Cognitive processes and ill-defined problems: a case study from design*[129]. Cette articulation est reprise par Herbert Simon[130] en 1973 dans T*he Structure of Ill-structured Problems*, dont une longue section est consacrée à l'examen du processus de conception architecturale. La même année, Horst Rittel et

[126] En anglais, *wicked problem* ou *ill-structured problem*. Parmi les traductions françaises proposées dans la littérature, en complément de l'utilisation du terme anglais, on trouve les propositions suivantes : problèmes vicieux, pernicieux, tordus, persistants, épineux, retors, malicieux. La recherche consacrée à l'origine du terme problème malicieux sont parus dans (cit design in translation)

[127] McCarthy. J. "The inversion of functions defined by Turing machines". In *Automata studies*, edited by C. E. Shannon and J. McCarthy, Annals of Mathematics studies no. 34, Princeton University Press, Princeton 1956, pp. 177–181. Minsky, M. « Steps toward Artificial Intelligence ». Proceedings of the IRE 49, no 1 (janvier 1961): 8‑30. https://doi.org/10.1109/JRPROC.1961.287775.

[128] Reitman, W. "Heuristic decision procedures, open constraints, and the structure of ill-defined problems", in G. L. Bryan and M. Shelly (eds.) Human Judgments and Optimallty. Wiley, N. Y. 1964. Newell, A. « Heuristic programming: ill-structured problems ». In The Soar papers (vol. 1): research on integrated intelligence, 3–54. Cambridge, MA, USA: MIT Press, 1993.

[129] Eastman, Charles M. « Cognitive processes and ill-defined problems: a case study from design ». In Proceedings of the 1st international joint conference on Artificial intelligence, 669–690. IJCAI'69. Washington, DC: Morgan Kaufmann Publishers Inc., 1969.

[130] Simon, Herbert A. « The Structure of Ill Structured Problems ». Artificial Intelligence 4, no 3‑4 (1 décembre 1973): 181‑201. https://doi.org/10.1016/0004-3702(73)90011-8.

Melvin Webber[131] adaptent la notion en inventant le terme de problèmes malicieux dans *Dilemnas in a General Theory of Planning*. À la suite de cet article, le concept se répand dans la recherche en design et grandit en importance au cours des décennies suivantes. Richard Buchanan publie notamment une vingtaine d'années plus tard, en 1992, un article remarqué, *Wicked Problems in Design Thinking*. Voici deux extraits des deux derniers articles donnant la définition des problèmes malicieux selon leurs auteurs:

*Les problèmes sur lesquels les scientifiques et les ingénieurs se sont généralement concentrés sont pour la plupart "inoffensifs" ou "bénins". Prenons par exemple un problème de mathématiques, comme la résolution d'une équation, ou la tâche d'un chimiste organique qui analyse la structure d'un composé inconnu, ou encore celle d'un joueur d'échecs qui tente de faire échec et mat en cinq coups. Pour chacun d'eux, la mission est claire. Il est clair, à son tour, que les problèmes ont été résolus ou non. Les méchants problèmes, en revanche, n'ont aucun de ces traits de clarté ; et ils incluent presque toutes les questions de politique publique - que la question concerne l'emplacement d'une autoroute, l'ajustement d'un taux d'imposition, la modification des programmes scolaires, ou la confrontation avec la criminalité[132].*

Horst W.J. Rittel & Melvin W. Webber, 1973

*Comme décrit dans le premier rapport publié sur l'idée de Rittel, les problèmes malicieux sont une "classe de problèmes de systèmes sociaux mal formulés, où l'information est confuse, où il y a beaucoup de clients et de décideurs avec des valeurs contradictoires, et où les ramifications dans l'ensemble du système sont complètement confuses". Il s'agit d'une description amusante de ce à quoi les concepteurs sont confrontés dans chaque nouvelle situation. Mais surtout, elle met le doigt sur une question fondamentale qui se cache derrière la pratique : la relation entre le déterminisme et l'indétermination dans le design thinking. Le modèle linéaire du design thinking est basé sur des problèmes déterminés dont les conditions sont définies. La tâche du concepteur consiste à identifier ces conditions avec précision, puis à calculer une solution. En revanche, l'approche des problèmes*

[131] Rittel, Horst W. J., et Melvin M. Webber. « Dilemmas in a General Theory of Planning ». Policy Sciences 4, no 2 (1 juin 1973): 155‑69. https://doi.org/10.1007/BF01405730.

[132] *"The problems that scientists and engineers have usually focused upon are mostly "tame" or "benign" ones. As an example, consider a problem of mathematics, such as solving an equation; or the task of an organic chemist in analyzing the structure of some unknown compound ; or that of the chessplayer attempting to accomplish checkmate in five moves. For each the mission is clear. It is clear, in turn, whether or not the problems have been solved. Wicked problems, in contrast, have neither of these clarifying traits; and they include nearly all public policy issues--whether the question concerns the location of a freeway, the adjustment of a tax rate, the modification of school curricula, or the confrontation of crime"* Horst W.J. RITTEL et Melvin W. WEBBER, "Dilemnas in a General Theory of Planning", Policy Science 4, p. 155-169, 1973.

*malicieux suggère qu'il existe une indétermination fondamentale dans tous les problèmes de conception, à l'exception des plus triviaux*[133].

R. Buchanan, 1992.

Les titres successifs des articles donnent à voir l'évolution de la notion depuis les mathématiques jusqu'au design thinking en passant par les méthodes computationnelles de résolution de problèmes. Les auteurs successifs donnent à voir le ping-pong entre intelligence artificielle et théorie du design. Le concept est indissociable de l'émergence du champ de la recherche en design anglophone des années 1960 à 1980. Il est associé à l'idée que les pratiques de design s'accompagnent d'une expertise qui leur est propre, théorisée par Donald Schön dans son livre *The Reflective Practitioner*, ou encore par Nigel Cross qui la désigne sous le nom de *designerly ways of knowing* – « la connaissance par le design »[134]. Aujourd'hui, la notion de problèmes malicieux reste prisée dans le champ du design, et est utilisée également en sciences sociales, dans le cadre de travaux sur les méthodes de traitement des enjeux environnementaux, de management de la santé, ou encore de développement des politiques publiques. Horst Rittel et Melvin Webber comme Richard Buchanan, pointent en effet dans leurs travaux que la résolution des problèmes malicieux passe par l'implication d'un grand nombre des acteurs concernés, une dimension participative particulièrement pertinente pour aborder les problèmes socio-environnementaux. On peut comprendre la notion de problèmes malicieux telle que développée en théorie du design comme découlant d'un effort de prise en compte, lors des tentatives de description du processus de conception, des savoirs tacites qui en font partie. Nombre de praticiens et de théoriciens impliqués dans cette entreprise reconnaissent en effet l'existence d'une expertise particulièrement difficile à qualifier dans les pratiques de design. Concevoir quelque chose implique de saisir le contexte dans lequel s'inscrit le problème qu'on cherche à résoudre - un problème qui change à chaque occurrence, et dont les méthodes de résolution changent également, non seulement en fonction de ce contexte mais aussi en fonction des praticiens, qui développent des stratégies personnelles[135]. C'est ce cadre de travail changeant que la notion de problèmes malicieux cherche à pointer.

---

[133] "*As described in the first published report of Rittel's idea, wicked problems are a "class of social system problems which are ill-formulated, where the information is confusing, where there are many clients and decision makers with conflicting values, and where the ramifications in the whole system are thoroughly confusing". This is an amusing description of what confronts designers in every new situation. But most important, it points toward a fundamental issue that lies behind practice: the relationship between determinacy and indeterminacy in design thinking. The linear model of design thinking is based on determinate problems which have definite conditions. The designer's task is to identify those conditions precisely and then calculate as solution. In contrast, the wicked-problems approach suggests that there is a fundamental indeterminacy in all but the most trivial design problems"* Richard BUCHANAN, "Wicked Problems in Design Thinking", Design Issues, Vol. 8 N. 2, p. 5-21, 1992.

[134] Voir chapitre II. Cross, N., "Designerly Ways of Knowing", in Design Studies Vol. 3 n. 4, pp. 221-227, 1982. Schön, D., The Reflective Practitioner. How professionals think in action, Temple Smith, 1983.

[135] Ce qui n'est pas sans rappeler Alexander, et ce que nous avons vu au chapitre II sur ces questions.

Il s'agit néanmoins d'une théorisation de l'activité de conception directement héritée de l'intelligence artificielle, puisque c'est dans ce cadre que les premières descriptions de ces problèmes malicieux émergent. Mais cette origine tombe assez vite dans l'oubli, que ce soit en théorie du design ou dans le champ computationnel, puisque seuls les articles de Eastman, de Simon, de Horst et Rittel et de Buchanan sont fréquemment cités. Ce sont ceux qui prennent comme exemple ou objet le design et la conception architecturale. Les articles de McCarthy, Minsky, Reitman et Newell sont bien moins mentionnés. Ce sont pourtant les premiers à avoir fait émerger la notion, et dans un cadre bien précis, qui n'est pas du tout celui du design. Les praticiens qui en sont les auteurs n'ont d'ailleurs jamais exercé en tant que designers ou architectes. La notion de problème mal structuré développée dans ces premiers textes est par ailleurs utilisée en lien avec la question de la méthode employée pour résoudre un problème. L'objectif est toujours de trouver une méthode universelle pour la transcrire en instructions de programmation et ainsi obtenir un programme capable de résoudre un problème quel qu'il soit. La description par Allen Newell du choix fait avec Herbert Simon de remplacer dans leur modèle la notion de prise de décision par la notion de résolution de problèmes est instructive : "*la résolution de problème comme un substitut plus apte à la prise de décision*"[136]. Newell indique par cette phrase l'idée que la recherche d'une méthode de résolution des problèmes est plus adéquate pour ses travaux avec Simon que la recherche d'un modèle de prise de décision. Ces prises de décisions humaines, étudiées en particulier par Simon avant sa collaboration avec Newell, sont intimement liées aux connaissances tacites possédées par les personnes. Elles sont donc difficiles à implémenter dans un programme informatique. La méthode de résolution des problèmes au sens où elle est appliquée par Newell et Simon vise à réduire des problèmes mal structurés à leurs composantes décrivables par des opérations explicites - des opérations logiques pouvant être programmées. C'est par l'application systématique de cette méthode aux champs les plus divers que Newell et Simon cherchent à parvenir à un programme d'intelligence artificielle global. Ainsi, non seulement les problèmes malicieux sont une notion qui trouve son origine dans le monde de l'intelligence artificielle, mais par ailleurs la réflexion sur son application à des problèmes de design ou d'architecture relève initialement d'une approche computationnaliste[137].

---

[136] "*problem solving a more apt substitute to decision making*". Allen Newell dit par ailleurs de lui-même : "*I was a problem solve*r". McCorduck, P., Machines Who Think (2e édition), Natick, MA: A. K. Peters, Ltd., 2004, p156 et 140.

[137] dire au moins en note de bas de page que ca marche très bien pour certains trucs, par ex le Logic Theorist, les échecs, et plus récemment les succès des réseaux neuronaux montrent que la logique d'un ordi permet de faire plein de trucs cools. Mais explicites. - le présent txt interroge simplement la prédominance d'une méthode sur l'autre dans le discours, quand les pratiques sont moins nettes

Au sein du champ computationnel, la méthode développée par Newell et Simon est largement reprise, comme vu en début de chapitre, ou les exemples donnés relèvent de son application. Le terme de problème malicieux est initialement lui aussi beaucoup repris. Nous avons en effet vu au chapitre II la superposition existante entre le champ computationnel en architecture, la théorie du design contemporaine et l'intelligence artificielle. La notion de problème malicieux est particulièrement prisée des théoriciens du design, mais cette superposition favorise sa reprise également dans le champ computationnel à la naissance des trois champs et jusque pendant l'hiver procédural. Par la suite néanmoins, si l'idée de *problème* figure en bonne place dans les textes, la notion de problème malicieux ou mal structuré fait place à la notion de "problème complexe", de "problème de design", voire de problème tout court. Il est parfois aussi question de "problème architectural". Ci-dessous sont donnés quelques exemples d'usage de ces termes.

*Pour illustrer les capacités et la flexibilité de structureFIT, cette section présente une série d'études de cas qui utilisent l'outil de plusieurs manières pour l'exploration évolutive d'un problème de conception donné.*[138].

> *L'inadéquation entre les capacités de l'optimisation et les besoins du processus de conception pratique est l'une des raisons probables de l'adoption limitée de l'optimisation dans l'industrie du bâtiment. Une deuxième raison, liée à la première, concerne le concept d'optimum lui-même : étant donné que de nombreux objectifs qualitatifs sont également subjectifs, il est impossible d'affirmer qu'une solution de conception est sans équivoque la meilleure dans la plupart des problèmes de conception*[139].
>
> C. Mueller & J. Ochsendorf, 2015.

*Un projet réel a été sélectionné pour permettre aux auteurs d'être directement exposés à un problème de conception réel d'un niveau de complexité élevé*[140].

D. Holzer et al., 2007

---

138 *"To illustrate the capabilities and flexibility of structureFIT, this section introduces a series of case studies that use the tool in several ways for evolutionary exploration of a given design problem"* Mueller Ochsendorf 2015

139 *"The mismatch between optimization's capabilities and the needs of the practical design process is one likely reason for the limited adoption of optimization in the building industry. A second, related reason deals with the concept of the optimum itself: since many qualitative goals are also subjective, it is impossible to say that one design solution is unequivocally the best in most design problems"* Caitlin T. Mueller, John A. Ochsendorf, "Combining structural performance and designer preferences in evolutionary design space exploration", *Automation in Construction*, Volume 52, p. 70-82, 2015.

140 *"A live project has been selected to allow the authors with direct exposure to an actual design problem with a high level of complexit"* Holzer D, Hough R, Burry M. Parametric Design and Structural Optimisation for Early Design Exploration. International Journal of Architectural Computing. 2007;5(4):625-643.

*Dans une approche traditionnelle, le concepteur étudie le problème de conception, intériorise l'ensemble de ses contraintes et objectifs, puis utilise ses compétences et son expérience pour élaborer une solution de conception unique, voire une poignée tout au plus. une poignée tout au plus. Avec l'approche paramétrique, les contraintes et les objectifs du problème de conception peuvent être directement intégrés dans le modèle paramétrique, qui peut ensuite être utilisé pour générer automatiquement une variété de solutions*[141].

*Comme pour tout projet de conception architecturale, le processus commence par l'étude du problème de conception, la compréhension de ses objectifs et contraintes, et la formulation d'une vision et d'un concept pour la conception*[142].

D. Nagy et al., 2017.

*Cette capacité à donner la primauté à la structure des relations et des qualités qui existent dans le contexte des problèmes architecturaux fait des techniques de modélisation topologique l'une des composantes impératives de l'architecture performative*[143].

R. Oxman, 2008.

L'idée de problème complexe rappelle la définition du problème malicieux, dont l'enjeu est justement de cerner la complexité du processus de design. De la même manière, on trouve dans les textes des mentions de différents types de problèmes, une idée qui rappelle la distinction initialement faite entre problèmes bien et mal structurés. Enfin, le problème de design peut lui aussi être associé au problème malicieux. Il est en effet possible de comprendre l'expression comme pointant vers l'idée que les problèmes de design sont particuliers, revenant là encore à la définition donnée des problèmes

---

141 *"In a traditional approach, the designer studies the design problem, internalizes all of its constraints and objectives, and then uses their skill and experience to craft a single design solution, or a handful at most. With the parametric approach, the constraints and goals of the design problem can be directly embedded within the parametric model, which can then be used to automatically generate a variety of solutions"* Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

142 *"As with any architectural design project, the process begins by studying the design problem, understanding its goals and constraints, and formulating a vision and concept for the design"* Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

143 *"This ability to give primacy to the structure of relationships and qualities that exist in the context of architectural problems makes topological modeling techniques one of the imperative component of performative architecture"* Oxman Rivka (2008) "Performance-based Design: Current Practices and Research Issues", *International Journal of Architectural Computing*.

malicieux. Ce premier terme est beaucoup utilisé en théorie du design pour caractériser l'activité de design, et le "problème de design" peut être interprété comme un glissement de langage chez les praticiens du champ computationnel qui se réfèrent en fait à travers cette expression ce corpus et à cette idée. Au fil des textes, la référence précise au corpus et à l'idée initiale se perd cependant. La dimension heuristique, l'idée d'un savoir-faire associé au traitement de ces problèmes malicieux qui est exprimée dans les corpus de théorie du design se perd au profit de considérations sur la méthode de résolution des problèmes que l'on peut implémenter par le biais des algorithmes. La méthode algorithmique de résolution des problèmes est présentée dans ces textes comme une alternative à la méthode heuristique, qui n'est donc plus nécessaire pour traiter des problèmes de design[144]. Le recours à des expressions alternatives à celle de "problème malicieux", voire au terme de problème tout court accompagne ainsi l'évolution de cette notion vers celle d'un simple problème à résoudre au même titre que n'importe quel autre grâce au développement des méthodes d'optimisation.

Une seconde notion, originaire elle aussi des débuts de l'intelligence artificielle, est alors également beaucoup reprise : celle d'*espace de design* (*design space*). Ce terme désigne l'ensemble des possibilités produites par un algorithme. Chacune de ces possibilités correspond à un ensemble de valeurs fixées pour les différents paramètres. A chaque variation de l'un des paramètres, le résultat de l'algorithme est différent et constitue une nouvelle solution potentielle pour le problème examiné. L'enjeu des méthodes d'optimisation en usage dans le champ computationnel et ailleurs est donc d'évaluer cette multitude de solutions possibles. Se promener dans l'espace de design est aussi, pour un utilisateur de ces algorithmes, une manière d'observer la diversité des configurations spatiales produites. Nous avons pointé plus haut l'intérêt des praticiens du champ computationnel pour cette grande diversité. L'espace de design algorithmique peut donc servir à la fois pour un processus d'optimisation et pour un processus d'exploration plus heuristique. La notion provient d'un ensemble de textes communs ou voisins de ceux qui ont formalisé la notion de problème mal structuré. L'un des premiers à décrire l'espace de design est Marvin Minsky, dans son texte de 1961 :

> *Un ordinateur ne peut faire, en un sens, que ce qu'on lui demande de faire. Mais même lorsque nous ne savons pas exactement comment résoudre un certain problème, nous pouvons programmer une machine pour qu'elle cherche dans un vaste espace de tentatives de solutions*[145].

[144] Par exemple dans les travaux de Danil Nagy et de son équipe.

[145] "*A computer can do, in a sense, only what it is told to do. But even when we do not know exactly how to solve a certain problem, we may program a machine to search through some large space of solution attempts"* Minsky, M. "Steps toward Artificial Intelligence," in Proceedings of the IRE, vol. 49, no. 1, pp. 8-30, Jan. 1961, doi: 10.1109/JRPROC.1961.287775.

L'idée de l'espace de design s'inscrit ainsi dans la continuité du développement de la méthode de résolution algorithmique des problèmes proposée par Newell et Simon. L'idée suit la même trajectoire que les problèmes dont elle permet d'examiner les solutions potentielles, de l'intelligence artificielle vers l'architecture computationnelle. Elle est cependant bien plus reprise par les praticiens contemporains du champ dont nous examinons les pratiques dans ce chapitre que par les théoriciens du design. Les praticiens contemporains retiennent en effet l'espace de design à la fois comme un outil d'optimisation et comme un outil de facilitation attractif pour les praticiens plus amateurs, qui peuvent ainsi se promener d'une solution à l'autre, comme vu dans le chapitre précédent. Voici de nouveau quelques exemples d'usage de l'expression dans leurs textes.

*De quelle manière les concepteurs peuvent-ils explorer l'espace des solutions possibles à un problème de conception à la recherche de bonnes conceptions ?*[146]

C. Mueller & J. Ochsendorf, 2013.

*Cette méthode est préférable lorsque l'utilisateur a trouvé une partie de l'espace de conception qui l'intéresse et qu'il souhaite affiner la conception en explorant de petites variations. Les taux de mutation élevés augmentent la probabilité que la progéniture saute dans des régions de l'espace de conception éloignées de ses parents*[147].

C. Mueller & J. Ochsendorf, 2015.

*Chaque dimension de cet espace de conception représente l'un des paramètres critiques exposés par le modèle paramétrique, et chaque variation de conception individuelle peut être trouvée quelque part dans cet espace hyperdimensionnel*[148].

D. Nagy et al., 2017.

---

[146] "*In what ways can designers survey the space of possible solutions to a design problem in search of good designs?"* Mueller, C., & Ochsendorf, J. (2013). An integrated computational approach for creative conceptual structural design. Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2013.

[147] *"This is preferable when the user has found a part of the design space of interest, and wishes to fine-tune the design by exploring small variations. Large mutation rates increase the likelihood of offspring to jump to regions of the design space far from their parents"* Caitlin T. Mueller, John A. Ochsendorf, "Combining structural performance and designer preferences in evolutionary design space exploration", *Automation in Construction*, Volume 52, p. 70-82, 2015.

[148] *"Each dimension of this design space represents one of the critical parameters exposed by the parametric model, and each individual design variation can be found somewhere within this hyper-dimensional space."* Nagy, D., Villaggi, L.,Zhao D. et Benjamin, D., « Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture ». In ACADIA 2017: Disciplines & Disruption [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017), pp. 436- 445. CUMINCAD, 2017.

*En les combinant dans un seul modèle d'espace de conception et en optimisant pour les deux objectifs à la fois, nous pouvons mieux comprendre le compromis entre ces objectifs concurrents et trouver les conceptions optimales qui résolvent ce compromis de la meilleure façon possible*[149].

D. Nagy et al., 2018.

Les problèmes de design sont caractérisés par leur complexité, mais aussi par le grand nombre de solutions potentiellement adéquates. A travers le recours à la notion d'espace de design, c'est cette caractéristique qui est retenue, bien plus que la spécificité des savoirs nécessaires pour résoudre un problème malicieux[150]. Trouver comment parcourir ce très vaste espace des solutions devient donc un enjeu majeur. L'exploration des diverses solutions manuellement pour évaluer leurs qualités architecturales et leur pertinence en regard du contexte devient difficile lorsque l'espace de design comporte des milliers d'alternatives. L'évaluation automatisée de ces solutions devient ainsi une composante nécessaire et inévitable du processus. Le recours aux deux notions de problème de design et d'espace de design accompagne l'application systématique de la méthode algorithmique de résolution des problèmes à la conception spatiale chez une partie des praticiens du champ. La conception architecturale devient avec l'utilisation renouvelée de ces notions un problème à résoudre, usuellement dans le cadre du recours à un algorithme génétique ou un processus d'optimisation. Et si la conception architecturale peut être décrite comme un problème à résoudre, cela implique que des paramètres et une fonction d'évaluation bien choisis sont la bonne voie pour parvenir à son automatisation. Ainsi non seulement la conception architecturale par la résolution de problème se popularise, mais un biais d'optimisation se constitue également. Les pratiques du champ deviennent donc de plus en plus à cette quatrième période de son développement des pratiques d'optimisation.

Même si la question occupe peu de place dans les réflexions des praticiens qui y ont recours, la méthode de résolution des problèmes popularisée par ces pratiques peut être analysée comme une manière alternative de se confronter à la traduction tacite-explicite. Celle-ci se fait bien plus en amont du processus de conception que dans le cadre de l'approche heuristique. L'idée derrière le recours à la méthode de résolution des problèmes est de formuler l'exercice de conception sous forme de

---

[149] *"By combining them within a single design space model and optimizing for both objectives at once, we can better understand the tradeoff between these competing goals and find the optimal designs which solve this tradeoff in the best way possible"* Nagy, Danil, Lorenzo Villaggi et David Benjamin. "Generative urban design: integrating financial and energy goals for automated neighborhood layout." (2018)

[150] Par exemple Holzer cite le travail de Bryan Lawson, mais se concentre sur cet extrait : "*since design problems defy comprehensive description and offer an inexhaustible number of solutions the design process cannot have a finite identifiable end"*. Bryan Lawson, *How designers think*, Routledge, 2006, cité par Holzer D, Hough R, Burry M. Parametric Design and Structural Optimisation for Early Design Exploration. *International Journal of Architectural Computing*. 2007;5(4):625-643.

problème, en termes explicites, dès le début. Une proposition pour l'exercice de conception découle ensuite de la résolution de ce problème par une méthode rationnelle d'évaluation des possibilités. L'approche heuristique consiste à expliciter la proposition finale en gérant une traduction graduelle au fil du processus d'écriture du programme et de l'ajustement de ses résultats. L'approche par la méthode de résolution des problèmes consiste à expliciter le problème de départ, en partant du principe que cette méthode, adossée au recours aux algorithmes, garantit d'y trouver une solution adéquate. Le désintérêt pour l'approche heuristique au profit de l'approche par la résolution de problème est de ce fait renforcé par un mythe de l'intelligence artificielle, lui aussi hérité de ce champ parent de l'architecture computationnelle. Dans les propos des praticiens du champ computationnel transparaissent, nous l'avons vu plus haut, une foi certaine en les capacités de l'ordinateur. Cette foi va main dans la main avec une forme de personnification de l'ordinateur. Celui-ci est souvent présenté comme doté d'une autonomie dans le processus de conception.

Le champ computationnel cultive ainsi l'idée que l'intelligence artificielle est déjà là ou presque, et que l'ordinateur sait déjà mieux que l'humain prendre des décisions pour la conception spatiale. Ce mythe prend lui aussi racine dans la proximité initiale entre les deux champs, et se superpose à la méthode de résolution des problèmes en un double héritage. Le monde de l'intelligence artificielle est en effet caractérisé par un très grand optimisme lorsque ses praticiens sont questionnés sur le moment où les ordinateurs manifesteront autant d'intelligence que les humains[151]. Malgré les débats, le champ semble généralement le lieu d'une assurance perpétuelle que la singularité est proche[152]. Cette conviction est soutenue par les progrès des recherches dans le domaine. Les succès des programmes informatique en logique et aux échecs, puis au fil des années dans des jeux de plus en plus complexes, ou ils parviennent maintenant régulièrement à battre des humains entretiennent l'espoir[153]. Le passage du test de Turing par un nombre croissant de programmes également, bien que cela se fasse des années plus tard que ce que son inventeur avait prédit. Les progrès en reconnaissance du langage et des images, qui rendent les programmes plus à même que jamais de communiquer avec leurs utilisateurs en langage naturel, font aussi beaucoup pour entretenir ces convictions. Les promesses du champ de l'intelligence artificielle, auxquelles le domaine revient très régulièrement malgré les échecs essuyés au cours de ses différents hivers, infusent dans la perception collective de ce qu'est l'intelligence artificielle aujourd'hui. La vague de succès rencontrés par les techniques d'apprentissage profond a récemment remis sur le devant de la scène les ambitions des chercheurs du domaine, et leur confère de nouveau une certaine crédibilité.

[151] Voir chapitre III.
[152] La singularité est le moment où l'IA se mettra à exister.
[153]Jeopardy, mais aussi des jeux vidéos divers, notamment DOTA, qui est considéré comme parmi les plus complexes. OpenAI et al., "Dota 2 with Large Scale Deep Reinforcement Learning", CoRR, 2019.

La personnification de la machine dans les discours a pour effet de lui prêter une forme d'intentionnalité, au point de parfois sembler impliquer que l'ordinateur est déjà intelligent. Ces projections se construisent dans un lien fort avec la question de l'intentionnalité de la machine, dont aucune preuve n'existe néanmoins jusqu'ici. On peut donc bien parler de mythe au sujet de ces idées, qui forment pourtant la toile de fond du recours aux outils algorithmiques contemporains. Celui-ci est renforcé par un glissement dans le discours qui s'amorce très tôt. Dès les premières tentatives de formulation du rôle de la machine dans le processus de conception architecturale, celle-ci en est présentée comme une figure à part entière. Ces tentatives visent à formuler le rôle de la machine à long terme, indépendamment des difficultés techniques auxquelles les praticiens sont confrontés. Il s'agit de penser la place de la machine indépendamment de la limite de ses capacités actuelles. Dès les années 1960, la personnalisation de l'ordinateur et le développement de la possibilité d'une IA prenant des décisions sur la conception au mettre titre que les architectes humains font partie du discours qui vise à articuler le rôle de la machine dans le processus de conception. Nous avons évoqué au chapitre II les réflexions sur les rôles respectifs de l'architecte et de l'habitant, de l'utilisateur et du programmeur, tous acteurs d'un projet d'architecture computationnelle. Mais pour nombre de praticiens l'ordinateur aussi est une figure de l'échange. Chez Nicholas Negroponte, le principe de machine à architecture se base sur l'idée de conseils (et non de données) fournis à l'architecte par l'ordinateur. Chez John Frazer, l'interaction entre le bâtiment-programme et l'habitant-utilisateur fait aussi de l'entité informatique une figure à part entière. Le discours de Danil Nagy ou de Caitlin Mueller s'inscrit dans la même lignée. L'idée que l'ordinateur participe au processus de conception à l'égal de l'humain est clairement formulée : *dans le cadre de la conception axée sur les performances, l'objectif est de mettre en œuvre l'interaction homme-ordinateur de manière à maximiser les forces synthétiques et analytiques des deux entités.*[154]. L'ordinateur est présenté comme assistant ou associé plutôt que d'acter l'existence d'un programmeur derrière. La ligne se brouille entre les opérations exécutées par un ordinateur et les instructions fournies par le programmeur. Combiné à une manière de parler des machines qui leur prête intelligence, créativité, innovation, ceci contribue à installer l'idée d'une égalité entre les deux entités, malgré une réalité technique bien moins avancée que prévu par les chantres de l'intelligence artificielle. L'omniprésence de ce mythe de l'intelligence artificielle relègue de ce fait les enjeux de la traduction tacite-explicite au second plan dans le champ computationnel.

[154] "*when pursuing performance-based design, the goal is to implement human–computer interaction in a way that maximizes the synthetic and analytic strengths of both entities"* Brown, N. & Mueller, C. (2017). Designing with data: moving beyond the design space catalog. ACADIA 2017. Le Digital Structures Group est d'ailleurs au M.I.T, au sein d'infrastructures façonnées entre autres par Nicholas. Negroponte.

L'héritage théorique laissé par le domaine de l'intelligence artificielle au champ computationnel en architecture est donc d'une grande importance dans la formulation des enjeux de la pratique de ce dernier. Or cet héritage tend à favoriser l'un des deux appareils théoriques. Cette préférence, un temps atténuée par l'évolution autonome du champ computationnel, fait ensuite une réapparition très marquée. Cette résurgence n'est pas seulement le résultat du lien dès leur naissance entre les deux champs. C'est aussi le résultat d'une proximité renouvelée avec le monde de l'intelligence artificielle, qui ravive la préférence. L'histoire récente de l'intelligence artificielle est marquée par le tournant vers l'apprentissage automatique et le big data au début du XXI$^{ème}$ siècle. Ce tournant se caractérise par la montée en puissance de nouvelles méthodes de traitement des données. Celles-ci ont des résultats impressionnants dans le domaine de l'analyse d'images dans un premier temps, puis progressivement dans tous les domaines du traitement de l'information. Les réseaux de neurones font dans ce cadre leur grand retour. Les technologies de l'informatique sont désormais bien plus avancées et permettent enfin de recourir à ces méthodes de traitement des données esquissées dans les années 1950 avant d'être abandonnées fautes de résultats[155]. Cette fois-ci, les résultats sont tels que c'est tout le projet d'intelligence artificielle qui retrouve sa popularité. L'expression est remise au goût du jour et les travaux universitaires à ce sujet pullulent de nouveau, tout comme nombre de recherches privées[156]. Les réseaux neuronaux ont d'autant plus le vent en poupe que Google ou Microsoft mettent à disposition des outils d'entraînement de ces systèmes qui les rendent très faciles à utiliser par des amateurs. Or de manière générale, avec la mise à disposition d'outils algorithmiques de conception spatiale de plus en plus nombreux, la tradition de détournement des architectes-programmeurs a diminué. On observe donc une recrudescence des collaborations avec des informaticiens, d'autant que départements du champ computationnel et d'intelligence artificielle sont désormais voisins sur de nombreux campus. Ceci favorise des collaborations autour des réseaux neuronaux, unique point d'intérêt du champ informatique ou presque. A l'ère de l'apprentissage automatique, la proximité entre intelligence artificielle et champ computationnel en architecture se renouvelle donc. Ce renouveau des réseaux neuronaux et sa transpiration dans le champ computationnel n'est pas sans conséquences sur la manière dont les praticiens envisagent le rôle de l'ordinateur dans la conception architecturale. Le monde de l'informatique contemporain favorise en effet lui-même largement une des deux approches, tout en étant par ailleurs le lieu de difficultés théoriques significatives. Ainsi, à la fois par la remise au goût du jour d'un corpus théorique ancien et par la recrudescence de biais théoriques dans le champ de l'intelligence artificielle, la proximité renouvelée entre ce champ et celui de l'architecture computationnelle ravive la préférence pour l'approche rationaliste mécaniste.

[155] Chapitre II.
[156] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.

L’histoire de l’intelligence artificielle est marquée par des changements de paradigme successifs. L'approche connexionniste est d’abord popularisée par les cybernéticiens dans les années 1930 à 1950. Elle est ensuite enterrée par l’approche symboliste qui lui succède dans les années 1960, avant que le champ ne condamne cette seconde approche en périclitant pour un temps. Le tournant de l’apprentissage profond marque un retour au paradigme connexionniste. Dans leur article de 2018 *La revanche des neurones*, Dominique Cardon, Jean-Philippe Cointet et Antoine Mazières résument ce changement d'orientation dans le monde de l'IA. Ils y décrivent les paradigmes symboliste et connexionniste en définissant ce que la machine traite dans chaque cas et le rapport au monde qu'elle favorise[157]. Dans l'approche symboliste, l'objectif est de donner à la machine des capacités de raisonnement abstrait et logique, en lui fournissant une entrée et un programme, et en obtenant la sortie. Les règles fournies à la machine par le programme créent un espace de raisonnement interne, un "monde-jouet" isolé du monde réel et au sein duquel la machine évolue. Dans l'approche connexionniste, en revanche, la machine reçoit une entrée et la sortie correspondante. C'est alors à la machine de trouver le programme qui lui permet de faire correspondre au mieux l'une à l'autre. L'objectif de ce mécanisme est de sortir la machine de son monde-jouet et de rétablir un lien avec le monde réel, via des quantités massives de données brutes. Le passage dans les années 2000 aux techniques d'apprentissage profond marque donc un renouvellement des ambitions dans le domaine de l'intelligence artificielle, à travers un changement technique profond, qui déclenche à son tour un changement paradigmatique. Cardon, Cointet et Mazières affirment cependant dans leur article que malgré ce changement dans les techniques utilisées dans le domaine de l'intelligence artificielle, les chercheurs ont encore tendance à appliquer aux pratiques contemporaines des théories héritées de la période de domination de l'approche symbolique. Bien que ces pratiques s'ancrent dans un paradigme connexionniste renouvelé, le domaine de l'IA lui-même manque d'explorations théoriques qui reconnaissent ce changement et fournissent des théories appropriées pour ces nouvelles pratiques d'apprentissage profond. Ce manque tend de fait à se refléter dans l’adoption dans le domaine de l'architecture computationnelle de théories anciennes sur lesquelles appuyer les pratiques du champ.

Le monde de l’informatique contemporaine reste non seulement marqué par les travaux des précurseurs des années 60, mais aussi par de nombreuses assomptions du genre de celles que nous avons vu affleurer dans les discours du champ computationnel sur les bénéfices du recours à l’ordinateur. Les pratiques ne sont pas exemptes d’un certain positivisme, ni de certains biais. Ces derniers sont au contraire parfois très marqués, comme en témoigne l’émergence récente des travaux

[157] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.

d'analyse des biais sociaux et raciaux des algorithmes et leurs constats sur l'état du champ. On peut en donner comme exemple le travail de Judith Duportail[158], qui a montré que les algorithmes des sites de rencontre sont soumis à de nombreux biais de genre : ils sont programmés pour favoriser la mise en contact de femmes plus jeunes avec des hommes plus vieux, ou d'apparier les hommes avec des femmes dont le niveau de revenu comme le niveau intellectuel sont évalués comme inférieurs. Autre exemple, le travail de Virginia Eubanks[159], qui a montré que les algorithmes auquel le gouvernement américain recours dans différents états pour gérer la distribution des aides financières ou l'intervention des services sociaux discriminent massivement les personnes de couleur et ce au mépris de l'expérience des fonctionnaires supposés s'en servir, qui en connaissent très bien les inégalités, sans plus pouvoir intervenir depuis la mise en place de cette gestion informatisée. Les réseaux neuronaux sont eux de plus en plus souvent pointés du doigt dans le monde de la reconnaissance d'image : en raison des inégalités d'accès aux technologies numériques, ils sont entrainés à partir de datasets ou les hommes blancs sont surreprésentés, et ont donc de bien plus grandes difficultés à reconnaître les femmes noires[160]. Des travaux les plus légers aux plus préoccupants, il est devenu indéniable que les algorithmes sont biaisés. Ce constat suscite depuis quelques années des réactions dans le monde informatique, par exemple la création du groupe Fairness, Accountability, Transparency in Machine Learning (FAT-ML) en 2014, qui se réunit depuis annuellement pour discuter à la fois de questions techniques et de questions d'éthique. Des laboratoires de recherche spécialisés dans cette question ont vu le jour, notamment chez Google et chez Microsoft, mais aussi dans le monde universitaire, ou le laboratoire de recherche new-yorkais AI Now Institute produit par exemple un rapport annuel examinant les implications au sens large du recours aux technologies dites d'intelligence artificielle. La communauté IA reste cependant très monoculturelle, et très fermée à ces remises en question de ses pratiques, notamment dans le secteur professionnel. "*L'idée prédominante était que les humains étaient biaisés, pas les algorithmes. Les gens pensaient qu'il était ridicule de travailler la-dessus*"[161], déclare Moritz Hardt, professeur en informatique à Berkeley et cofondateur de FAT-ML - une idée prédominante que l'on retrouve aussi chez de nombreux praticiens du champ computationnel, on l'a vu. La prise en compte de ces questions reste encore beaucoup cantonnée à un petit espace au sein du

[158] Duportail, Judith. L'amour sous algorithme. Paris: le Livre de poche, 2020.
[159] Eubanks, Virginia. Automating inequality: how high-tech tools profile, police, and punish the poor. First Edition. New York, NY: St. Martin's Press, 2017.
[160] Lohr, Steve. « Facial Recognition Is Accurate, If You're a White Guy ». The New York Times, 9 février 2018, sect. Technology. Consulté le 11 juillet 2021. https://www.nytimes.com/2018/02/09/technology/facial-recognition-race-artificial-intelligence.html.
[161] "*The more predominant idea was that humans were biased and algorithms weren't," says Moritz Hardt, now a UC Berkeley computer science professor who cofounded the workshop (FAT-ML) with a researcher from Princeton. "People thought it was silly to work on this*." - Simonite, Tom. « What Really Happened When Google Ousted Timnit Gebru ». Wired. Consulté le 11 juillet 2021. https://www.wired.com/story/google-timnit-gebru-ai-what-really-happened/.

monde universitaire, alors même que les applications de ces technologies se répandent de plus en plus, tout comme le recours aux algorithmes se répand dans le monde architectural.

En effet, les pratiques de l'IA sont de plus en plus orientées vers l'exploitation commerciale, ce qui favorise la rationalisation et la propagation d'outils dans lesquels cette approche rationnelle est intégrée. Les nouvelles technologies d'IA utilisées par Google et ses concurrents le sont essentiellement en fonction des parts de marché qu'elles permettent de gagner, sans jamais vraiment exiger que les principes théoriques sous-jacents correspondent aux pratiques promues. En effet, encourager la diffusion de techniques sans discussion des enjeux théoriques est un scénario plus favorable au développement dans une économie de marché. En raison de sa proximité de longue date avec le domaine de l'IA, ainsi que de sa trajectoire similaire dans l'économie de marché contemporaine, l'architecture computationnelle reproduit ce scénario et sa focalisation sur la technique. Le propos est parfois même amplifié étant donné le statut de boîte noire des outils lorsque les praticiens en sont de simples utilisateurs. L'évolution du champ informatique y implique de plus en plus un positionnement sur comment on utilise un algorithme, sur ce que les algorithmes sont capables de faire, plutôt qu'une théorie de ce qu'est l'esprit humain et comment parvenir à l'imiter, comme c'était le cas aux débuts du champ. Or seule une petite partie des praticiens qui y évoluent se penchent sérieusement sur la question[162]. La vision contemporaine qui prédomine aujourd'hui en informatique est une vision centrée sur l'automatisation d'un point de vue technique, comme en témoignent les propos de Nicholas Carr dans *The Glass Cage. Automation and Us*. Il y expose l'idée que peu importe la méthode employée par une machine pour remplir l'objectif déterminé. Seul cet objectif compte, et si la machine parvient à imiter l'être humain, peu importe que la méthode soit différente de celle employée par ce dernier pour y parvenir. C'est notamment en raison de cette idée que l'apprentissage profond rencontre autant de succès. Certes, il ne s'agit que de calculs statistiques. Mais l'ordinateur parvient tout aussi bien à trier des colis en fonction de leur code postal[163] ou à conduire une voiture[164] qu'un humain qui ferait de même en mobilisant son savoir tacite. Seule compte donc la recherche de la bonne technique pour parvenir au résultat recherché - l'automatisation d'une tâche en particulier. En effet, les recherches en intelligence artificielle s'attaquent désormais à des tâches de beaucoup plus petite échelle qu'auparavant. C'est ce que Searle nomme l'*IA faible*, qui

[162] L'article consacré à Timsit Gebru et au temps qu#elle a passé chez Google en témoigne. Simonite, Tom. « What Really Happened When Google Ousted Timnit Gebru ». Wired. Consulté le 11 juillet 2021. https://www.wired.com/story/google-timnit-gebru-ai-what-really-happened/.

[163] LeCun, Y., Matan, O., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jacket, L.D., & Baird, H.S. (1990). Handwritten zip code recognition with multilayer networks. [1990] Proceedings. 10th International Conference on Pattern Recognition, ii, 35-40 vol.2.

[164] Voir par exemple T. Do, M. Duong, Q. Dang and M. Le, "Real-Time Self-Driving Car Navigation Using Deep Neural Network," 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), 2018, pp. 7-12, doi: 10.1109/GTSD.2018.8595590.

cherche à automatiser seulement des actions spécifiques, par contraste avec l'*IA forte*, qui imite les processus de pensée humains dans leur entièreté[165]. Les recherches contemporaines relèvent dans leur grande majorité de l'IA faible. Peu importent donc les ambitions à grande échelle, le modèle de pensée qui permettrait de parvenir à une IA forte. Seuls comptent les résultats sur l'automatisation de tâches spécifiques, des résultats mesurés au remplissage des objectifs. Avoir recours à des algorithmes n'est donc plus vraiment un projet théorique, mais plutôt un projet technique[166]. Notamment parce que le monde de l'informatique, comme celui de l'architecture computationnelle, n'est plus seulement un monde d'universitaires : c'est devenu un monde industriel à part entière, avec des contraintes différentes.

### *7.3.2 Des pressions économiques*

Nous avons vu au chapitre III que depuis les années 1980 le monde de l'intelligence artificielle se développe de manière croissante comme une industrie plutôt que comme un milieu académique. Le domaine de l'intelligence artificielle se crée à l'orée de la globalisation. Celle-ci a évidemment un impact sur le champ au fil du temps. Christopher Alexander mentionne déjà l'effet sur le développement des logiciels de conception qu'il perçoit : il critique vivement le fait que les attentes se concentrent sur la production de belles images sophistiquées plutôt que sur les enjeux de la conception de l'espace en elle-même. Greg Bryant lui aussi ressent une pression similaire, il souligne sa difficulté à s'investir dans des activités de programmation qui lui paraissent avoir du sens dans l'environnement de la Silicon Valley où il exerce.  Cette tendance ne fait que se renforcer dans les décennies qui suivent. L'hégémonie contemporaine des GAFA[167] est l'aboutissement du développement industriel des technologies de l'informatique. Ce que nous avons abordé un peu plus haut sur les visions contemporaines de l'Intelligence Artificielle s'ancre dans les pratiques de ces entreprises, au sein desquelles beaucoup de la recherche est maintenant conduite. Quant au champ computationnel en architecture, nous avons vu qu'une première bifurcation entre industrie et recherches universitaires avait débouché dans les années 1990 sur l'essor des logiciels de modélisation 3D dans la conception architecturale. La sortie du noyau du réseau et la démocratisation amorcée correspondent à une dynamique d'industrialisation des outils et des pratiques similaires. La prise d'importance des éditeurs de logiciels, les nouveaux profils d'amateurs provenant des agences conventionnelles, les impératifs intégrés au développement des nouveaux outils sont autant de signes que la sortie du noyau qu'on

[165] Bringsjord, Selmer and Naveen Sundar Govindarajulu, "Artificial Intelligence", The Stanford Encyclopedia of Philosophy (Summer 2020 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/sum2020/entries/artificial-intelligence/>., consulté le 12 Novembre 2021.
[166] Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». Réseaux n° 211, no 5 (16 novembre 2018): 173‑220. https://www.cairn.info/revue-reseaux-2018-5-page-173.htm?ref=doi.
[167] Google Apple Amazon Facebook

observe dans le réseau se fait en alignement avec les dynamiques capitalistes, d'où la transformation du tissu du réseau[168]. Par ailleurs, la pratique architecturale elle-même s'inscrit depuis longtemps dans des dynamiques économiques identiques. L'industrialisation de la construction, amorcée au début du XXe siècle avec le développement de la construction métallique et du béton armé, se poursuit tout au long du siècle[169]. La reconstruction qui suit la deuxième guerre, puis l'augmentation de la population mondiale créent une demande très forte pour construire vite et beaucoup. Le développement du style international contribue à fixer des standards pour la conception architecturale, les instituts de normalisation également[170]. À la fin du siècle, les changements sociétaux ont fait de la construction un marché comme un autre, soumis aux mêmes exigences économiques de rentabilité. Rien de nouveau dans le propos tenu ici mais ces évolutions pèsent sur le devenir des outils algorithmiques dans le champ computationnel et encouragent la trajectoire que nous venons d'étudier.

Alors que la construction passe aux mains de l'industrie, se transformant d'architecture en "environnement bâti"[171], les exigences qui portent sur sa production changent. La propriété intellectuelle prend beaucoup d'importance, d'une part au début du XXe avec la protection par brevet des méthodes de construction[172] et d'autre part fin XXe avec le développement de la protection des algorithmes et des logiciels. Ainsi le développement de l'architecture de marché joue un rôle important dans le délaissement des pratiques open-source dans le champ computationnel . En témoigne ce commentaire au sujet d'un plug-in Grasshopper mis à disposition de tous en ligne :

> *Globalement un bon plug-in. [...]. Malheureusement sous licence Creative Commons. Sans option pour usage commercial. Donc presque totalement inutile dans le "monde réel". Vous devriez réfléchir à modifier cela*[173].

Nous avons pointé le fait que les architectes du champ computationnel tendent au départ à se retrancher dans un microcosme protégé des contraintes industrielles. Il est intéressant de noter qu'Alan Colquhoun fait exactement la même remarque au sujet du mouvement moderne en Europe

---

[168] Voir Chapitre VI

[169] Cyrille Simonnet, *Le Béton, histoire d'un matériau. Économie, technique, architecture*. Paris, éd. Parenthèses, 2005.

[170] Anna-Maria Meister, From Form to Norm : the Systematization of Values in German Design. Thèse de doctorat, Princeton University, School of Architecture, 2013.

[171] *built environment*, un terme aujourd'hui très répandu dans le monde anglophone pour désigner toutes les constructions qui nous entourent.

[172] Cyrille Simonnet, *Le Béton, histoire d'un matériau. Économie, technique, architecture*. Paris, éd. Parenthèses, 2005.

[173] *"Generally a good plugin. Thumbs up.*
*Unfortunately licenced under [Creative Commons].*
*With no option for commercial usage. Therefore almost useless in the 'real world' You should consider changing that."* https://www.food4rhino.com/en/app/meshedit?page=1, consulté le 29 Novembre 2021.

dans les années 1960[174]. Mais la sortie du microcosme implique de se confronter aux exigences en vigueur : protection intellectuelle, rentabilité économique, construction à grande échelle. Pour parvenir à ces objectifs se crée un mécanisme de rationalisation. Une volonté de rationalisation qu'on retrouve en intelligence artificielle, où elle est présente dès les débuts du champ, mais devient un objectif beaucoup plus crucial ensuite, en raison du développement industriel qui suit. Ainsi Russell et Norvig, dont nous avons vu qu'ils sont la référence principale pour la définition de l'intelligence artificielle, la présente comme un système devant agir rationnellement, et rencontrent depuis les années 1990 beaucoup de succès dans leur travail de propagation de cette idée[175]. Le computationnalisme n'a plus guère d'importance, et les théories de l'intelligence artificielle ont de toute façon perdu beaucoup de leur poids, comme nous l'avons vu plus haut. Ce qui compte désormais, c'est la rationalisation des pratiques, bien mieux en lien avec les nouvelles exigences des industries contemporaines. Le champ computationnel se retrouve donc pris entre deux industries, dont les exigences orientent son développement.

Mais la révolution industrielle n'a pas uniquement pour impact d'altérer les contraintes auxquelles est soumise la conception architecturale. Bien plus tôt que cela, les conséquences de ce bouleversement sur la structuration de la profession moderne d'ingénieur accentuent aussi le recours à des processus rationnels de définition de l'espace et de son enveloppe. Antoine Picon montre dans son travail sur l'histoire de l'École des Ponts et Chaussées entre 1747 et 1851 les changements qui marquent l'avènement de la profession d'ingénieur moderne. Ces changements débouchent sur l'exercice de la profession tel qu'il se fait encore aujourd'hui, et influent de ce fait sur la professionnalisation des architectes[176]. La première évolution qui marque le corps des ingénieurs est la transition de leur formation vers une culture scientifique beaucoup plus étendue. Ceci se fait en raison du développement d'outils physico-mathématiques nombreux et divers au cours du XVIIIe siècle. Ces outils sont mis en œuvre pour mieux comprendre et projeter voûtes, tunnels ou barrages. Ils sont le résultat de la poursuite des efforts de quantification et d'explication entamés à la période des Lumières et mènent au dépassement des anciennes méthodes. L'application de maximes de conception et de

---

[174] Alan Colquhoun, *L'Architecture Moderne*, InFolio, 2006, p 234.

[175] Voir chapitre II pour les détails de leur définition. *What about Russell and Norvig themselves? What is their answer to the What is AI? question? They are firmly in the "acting rationally" camp. In fact, it's safe to say both that they are the chief proponents of this answer, and that they have been remarkably successful evangelists. Their extremely influential AIMA series can be viewed as a book-length defense and specification of the Ideal/Act category* - Bringsjord, Selmer and Naveen Sundar Govindarajulu, "Artificial Intelligence", The Stanford Encyclopedia of Philosophy (Summer 2020 Edition), Edward N. Zalta (ed.), https://plato.stanford.edu/archives/sum2020/entries/artificial-intelligence, consulté le 14 Octobre 2021.

[176] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989. Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

proportions validées par l'expérience fait place au recours à ces nouveaux outils. Antoine Picon évoque à ce sujet la mise en place de "*modèles plus mathématiques qu'architectoniques*"[177]. Ces progrès scientifiques donnent dans un premier temps lieu à des questionnements sur la place respective de la théorie et de la pratique : "*Comment définir en d'autres termes les modalités d'application de la science physico-mathématique aux techniques ?*"[178].

L'École des Ponts et Chaussées est alors le lieu d'un débat vivace auquel nombre d'enseignants prennent part. La question du lien entre théorie et pratique dans l'exercice du génie civil, cruciale au début des échanges, fait pourtant rapidement place à une autre. En effet, l'ère industrielle naissante implique pour les ingénieurs une nouvelle forme de pratique[179]. Le débat sur le rôle des savoirs analytiques dans la pratique de l'ingénieur se fait simultanément aux débuts de l'industrialisation. Cette simultanéité a un impact sur l'issue de ces débats et sur la structuration de la profession d'ingénieur moderne. Antoine Picon fait dans son travail le récit de l'affrontement à l'École des Ponts et Chaussées de deux visions de la profession d'ingénieur : l'une tournée vers la connaissance physique et mathématique, l'autre vers les enjeux économiques[180]. Or c'est cette seconde vision qui triomphe à terme[181]. Ce choix de privilégier la prise en compte des contraintes financières avant tout chose doit beaucoup au contexte dans lequel il est fait. La révolution industrielle renforce d'autant plus l'attrait du savoir-faire économique des ingénieurs qu'il leur garantit d'obtenir des commandes. En effet, le développement de leur pratique dans ce sens permet aux ingénieurs de l'École des Ponts et Chaussées de se forger très vite une réputation de bons gestionnaires économiques, et a pour effet de faire augmenter leur accession à la commande au détriment des architectes[182]. Les enjeux économiques de la révolution industrielle et l'invention en parallèle de leur profession offrent donc un rôle prépondérant aux ingénieurs dans la construction à partir du XVIIIe.

---

[177] Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

[178] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.

[179] "*Rendu possible par la domestication discrète de l'outil mathématique, ce projet marque en tout cas l'extinction d'un certain type de débat entre théorie et pratique, caractéristique de la transition entre l'âge classique et l'ère industrielle naissante*". Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

[180] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.

[181] "Résistance des matériaux et travail humain : cette double direction de quantification renvoie à la constitution d'un nouvel espace de référence que l'on peut qualifier de proto-économique". Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.

[182] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.

La complexité des grands projets de construction transforme dans le même mouvement le métier d'ingénieur. Le praticien solitaire des arts de l'ingénieur devient un chef de projet socialement reconnu s'appuyant sur la rationalité des sciences de l'ingénieur pour asseoir son autorité et son influence[183]. Ce changement n'est pas sans conséquences sur la philosophie de la profession. L'ingénieur devient "*l'incarnation d'un progrès indéfini. (...) Le caractère instrumental de son savoir se renforce au détriment de l'intime sympathie qu'il éprouvait autrefois à l'égard de son environnement. Il est un domaine, cependant, où ce caractère instrumental débouche sur une compréhension plus attentive de la réalité : ce domaine est celui de l'économie*"[184]. Antoine Picon souligne par ailleurs ce moment de transition comme un basculement de l'histoire de la rationalité technique "*de l'imitation raisonnée (...) à la recherche délibérée de l'innovation*"[185]. Enfin, il faut noter que l'analyse qu'Antoine Picon propose est fondée essentiellement sur l'exemple des ingénieurs français et de l'École des Ponts et Chaussées, qu'il compare aux écoles Polytechnique et Centrale. Les bouleversements de la profession d'ingénieur dont il fait le récit concernent néanmoins plus largement les ingénieurs occidentaux, ou au moins européens, à condition de faire quelques nuances. Les ingénieurs de l'École des Ponts et Chaussées sont des ingénieurs d'Etat, dont Antoine Picon souligne bien la vocation : exercer au sein de l'administration française. La place donnée à la gestion économique du projet correspond aux exigences de cette vocation. Néanmoins, Antoine Picon note que les ingénieurs d'État, notamment ceux de l'École des Ponts et Chaussées, font figure de référence. Leur structuration de l'enseignement et la manière d'exercer la profession d'ingénieur qu'ils impulsent a donc des répercussions bien au-delà de leur corps. Les modalités d'accession à la commande des ingénieurs d'État français sont elles aussi particulières, puisqu'ils sont missionnés par l'État sur les projets dont ils ont la charge. Néanmoins, la crédibilité scientifique qu'ils acquièrent au fil de l'intégration des nouveaux outils physico-mathématiques à leurs ressources n'est ni une spécificité de l'École des Ponts et Chaussées, ni une spécificité française[186]. Leur nouveau statut social ne l'est pas non plus. Ceci favorise donc partout la prise de responsabilité des ingénieurs dans la conception et construction de l'environnement bâti. Ils obtiennent ainsi la responsabilité d'un grand nombre de projets de construction, même à travers des mécanismes d'accession à la commande

[183] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.

[184] Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993., p. 623

[185] Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

[186] Antoine Picon note par exemple que les ingénieurs anglais reçoivent une formation beaucoup moins technique qu'en France au départ. Elle le devient ensuite, et au moment ou commence notre récit, bien plus tard, les savoirs et pratiques sont ancrés ailleurs qu'en France. Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

différents. En effet, en complément des projets dont l'État charge directement ses corps, les ingénieurs qui répondent à des appels d'offres se voient de plus en plus souvent attribuer ces projets, auxquels les architectes accédaient jusqu'alors. Les enjeux de l'industrialisation se font sentir ailleurs qu'en France avec les mêmes conséquences, celles qui vont impacter bien plus tard le champ computationnel en architecture. L'émergence de leur profession crée pour les ingénieurs une nouvelle approche de la construction, basée sur l'économie et la rationalité, tout en leur garantissant dans la conception de l'espace une place prépondérante. Celle-ci, tout comme la question de l'accès à la commande, crée des tensions avec les architectes, mais aussi autour de l'appropriation des savoirs techniques.

Les maximes employées par les ingénieurs avant le tournant de la fin du XVIIIe siècle sont, selon Antoine Picon, des outils mathématiques assez simples. Les nouveaux outils mathématiques le sont moins.

> *Entre la science et les problèmes concrets subsiste un important décalage que révèle assez bien Bélidor dans son Architecture hydraulique, en reprochant à Parent, l'un des premiers physiciens à s'être intéressé au problème du frottement avec Araontons, de s'être contenté de « calculs algébriques à perte de vue », capables d'effrayer lès professionnels les mieux disposés, au lieu d'en avoir « déduit des conséquences en forme de maximes » que l'on aurait suivies « avec la confiance que l'on a ordinairement pour tout ce que l'on sait être établi sur des principes de mathématiques, quoique l'on ignore la voie par laquelle on y est arrivé »*[187].

La professionnalisation des ingénieurs s'accompagne néanmoins de l'adoption de ces nouveaux outils mathématiques. Les ingénieurs en apprivoisent en quelques années les complexités avec l'aide des changements apportés à leur formation. La professionnalisation des architectes, légèrement postérieure à celle des ingénieurs, ne s'accompagne pas de ce tournant technique. Pour reprendre l'expression d'Antoine Picon, contrairement aux ingénieurs, les architectes restent dans des modèles architectoniques plus que mathématiques. Leur formation se poursuit également dans cette lignée, sans voir les mêmes changements que celle des ingénieurs. En France, la formation des architectes continue à être menée à l'École des Beaux-Arts jusqu'en 1968. Cette formation se caractérise notamment par la structure en ateliers et par la pédagogie par le projet qui y est adoptée. Plutôt que le calcul, ce sont le dessin et notamment les outils de géométrie projective qui occupent une place prépondérante dans le cursus. Ailleurs en Europe, l'enseignement se fait aux Beaux-Arts aussi, comme par exemple au Danemark, ou l'école des Beaux-Arts est fondée en 1754 et conserve l'école

[187] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.

d'architecture en son sein jusqu'en 2011[188]. Dans certains pays sont constituées assez tôt des écoles d'architecture autonomes privées, ou la formation est néanmoins construite sur un modèle similaire à celui de la formation des Beaux-Arts. C'est le cas de la AA de Londres, fondée en 1890[189]. Enfin, le cas des facultés d'architecture intégrées à de grandes structures universitaires existe aussi. Les premiers collèges d'architecture américains datent de cette même période de la fin du XIXe, la Bartlett de Londres naît en 1841. Nous avons vu que ces cadres d'enseignement offrent une proximité aux autres disciplines, notamment au génie civil et mécanique. Cette proximité induit une relation un peu différente aux savoirs techniques pour les architectes qui sont formés dans ces facultés. L'industrialisation de la construction et la professionnalisation du métier d'architecte en relation avec celle des ingénieurs ont cependant des conséquences similaires dans ces environnements. Concernant la question de la formation des architectes, le cas français est encore une fois plus marqué qu'ailleurs par la tension entre les deux professions et le délaissement des nouveaux savoirs techniques du XVIIIe. Il fournit cependant un bon support pour comprendre les dynamiques menant à la définition des professions d'architecte et d'ingénieur modernes en Europe et aux Etats-Unis. Ces dynamiques montrent l'apanage des savoirs techniques par la formation des ingénieurs, tandis que la formation des architectes reste une affaire de dessin plutôt que de calcul.

La révolution industrielle et la professionnalisation de la construction qui en découlent ont également pour conséquence d'éloigner les architectes des dimensions techniques de la production de l'architecture. L'augmentation de la taille des chantiers, l'industrialisation des processus de construction et le développement des brevets signent entre la fin du XIXe et le début du XXe la fin de l'implication des architectes dans le développement des machines de construction. Ceci constitue le second volet du désinvestissement technique des architectes. Au-delà de ce désinvestissement pratique, cette période est aussi celle d'ancrage de la profession d'architecte dans des fondements épistémologiques très différents de ceux de leurs confrères ingénieurs. L'analyse d'Antoine Picon suggère que la professionnalisation des architectes se fait en partie en réaction à celle des ingénieurs, en réaction donc à l'appropriation des savoirs scientifiques de la construction. La place des savoirs techniques dans la formation et la pratique architecturale prend donc racine dans un contexte qui engendre une forme de rejet de ces connaissances, les architectes se recentrant sur d'autres pans de leur pratique préexistante. La démocratisation des outils computationnels à des cercles d'utilisateurs au-delà de praticiens formés à la programmation s'accompagne de dynamiques qui font écho à ce rejet, qui perdure dans la pratique contemporaine des architectes. Le tournant de la

[188] "History of Royal Danish Academy - Architecture, Design, Conservation" https://royaldanishacademy.com/kadk-history, consulté le 12 Novembre 2021.
[189] Julie André-Garguilo, *La fabrique de l'architecte extraordinaire : L'Architectural Association School, 1964-1983*, Thèse de doctorat, Université Paris-Est, 2020.

professionnalisation génère chez les architectes une relation beaucoup plus conflictuelle aux techniques que chez leurs collègues ingénieurs, dont on trouve par la suite des traces au moment de l'expansion du champ computationnel.

Ces difficultés se superposent aux tensions sociales que génère la professionnalisation et la séparation des rôles, des tensions que l'on peut également observer à travers la crispation des architectes à cette période. En effet, la reconnaissance nouvelle de l'ingénieur et son regain d'accession à la commande se font au détriment des architectes, dont le métier, qui n'est pas encore protégé par un diplôme, se trouve menacé. Mais avec la création du diplôme d'architecte, qui conditionne la possibilité de pratiquer, c'est aussi le statut d'auteur de l'architecte qui est protégé. Le diplôme constitue ainsi une réponse à la professionnalisation des ingénieurs, car il permet celle des architectes. Mais il s'agit aussi d'une réponse à une source majeure de crispation des architectes : la menace sur l'autorat des bâtiments. Dans son ouvrage *L'alphabet et l'algorithme*, Mario Carpo définit les pratiques modernes des architectes à partir de ce qu'il nomme le paradigme d'Alberti[190]. Il fait remonter aux canons de représentation établis en architecture à partir de la perspective développée par Alberti une autre caractéristique de ces pratiques : l'idée que c'est le dessinateur d'un bâtiment qui en est l'auteur. Et en effet, au fil des siècles la figure de l'architecte-constructeur se transforme en celle de l'architecte-dessinateur. La mise en exergue des croquis initiaux de starchitectes comme Frank Gehry ou Renzo Piano en est aujourd'hui un des témoignages les plus évidents. Or ce statut d'auteur est mis en danger par la professionnalisation des ingénieurs et déclenche un repli sur soi des architectes, dont la professionnalisation est caractérisée par des manœuvres de protection ce statut d'auteur[191]. La création du diplôme d'architecture par l'École des Beaux-Arts en 1867, puis la création de la Société des architectes diplômés par le gouvernement en 1877 en sont les premiers jalons. Les crispations autour du statut d'auteur des architectes sont par ailleurs très liées aux difficultés rencontrées pour nommer les savoir-faire propres à la profession. Le statut d'auteur par le dessin devient ainsi presque le seul étendard de la discipline tant le reste semble dur à formuler de manière adéquate. Le moment de débat analysé par Antoine Picon a notamment trait à l'organisation des savoirs des ingénieurs, c'est-à-dire à une définition précise de la profession, de se.s objectifs, de ses méthodes, de ses activités[192]. Mais la professionnalisation des architectes ne mène pas vraiment à une définition équivalente, aussi précise, de la discipline. Tout au plus les enjeux spatiaux revêtent-ils une

[190] Mario Carpo, *The Alphabet and the Algorithm*, The MIT Press, 2011.
[191] Antoine Picon, "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction",Revue d'histoire des sciences, Vol. 42, No. 1/2, La mathématisation 1780-1830 (JANVIER-JUIN 1989), pp. 155-172, 1989.; AM Chatelet ?
[192] Voir le récit par Antoine Picon du débat Brisson-Navier entre une science unique de l'ingénieur et un art de la pratique. Le pragmatisme des enjeux financiers l'emporte néanmoins à la fin. Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

importance renouvelée en architecture, alors qu'ils sont délaissés par l'ingénierie[193]. Ce sont pourtant ces enjeux spatiaux qui sont les plus difficiles à cerner, comme nous l'avons vu dans les premiers chapitres de ce texte.

Certains pans des réflexions du champ computationnel portent de plus sur des notions qui pourraient là encore mettre en danger le statut d'auteur de l'architecte. La relation entre le bâtiment et ses usagers développée par Gordon Pask est ainsi une relation directe qui ne nécessite plus l'architecte comme intermédiaire, mais lui substitue un programme pour définir les espaces occupés. La juxtaposition des interrogations sur le rôle de l'utilisateur et du programmeur, de l'architecte et de l'habitant, de l'humain et de la machine constitue un questionnement systématique du statut d'auteur de ces différentes entités, et donc potentiellement une remise en question du statut de l'architecte. Nous l'avons vu au chapitre II, le champ computationnel s'ancre depuis sa naissance dans ces interrogations. Celles-ci sont le reflet de questionnements plus profonds sur le savoir-faire architectural, sur la possibilité de le partager avec des machines, de l'automatiser. Cette recherche peut être perçue comme une mise en danger supplémentaire des architectes, d'où la difficulté de certains à accepter le recours aux algorithmes. En dehors du microcosme initial des premières générations du champ computationnel, les praticiens font preuve d'une certaine défiance face aux outils algorithmiques. Cette défiance prend racine dans le questionnement du statut d'auteur de l'architecte qu'amènent ces outils, un statut que les architectes tiennent pourtant par-dessus tout à conserver puisqu'il est aujourd'hui le garant de leur profession. La méfiance induite implique peu d'investissement de la part de ces praticiens dans la tâche d'apprivoiser la programmation et laisse les architectes dépendants d'outils simples d'utilisation pour des amateurs. La démocratisation du champ computationnel se construit donc sur une fragilité épistémologique autant que technique. En effet la crise de la profession architecturale ne s'est pas vraiment résolue, ni au XIXe siècle, ni après. C'est une profession en perte de repères depuis longtemps, et la mue du champ computationnel appuie précisément sur ses failles, créant un terreau fertile pour la prolifération d'outils prêts à l'emploi peu questionnés. Le rôle prépondérant de l'ingénierie dans les dynamiques observées après la mue du champ, dans les pratiques et le discours qui se sont généralisés au sein du champ n'est pas un hasard, c'est la poursuite d'une expansion des pratiques de rationalisation de l'architecture depuis plusieurs siècles.

[193] Antoine Picon, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées*, 1747-1851, Presses de l'École Nationale des Ponts et Chaussées, 1993.

**Conclusion - Optimiser l'architecture ?**

Le champ computationnel en architecture s'est transformé au cours des deux dernières décennies. Dans les pratiques les plus récentes, des deux appareils techniques vus au chapitre II pour intégrer les outils computationnels à la conception architecturale, l'un domine très largement, pour de multiples raisons. Le poids des contraintes économiques et industrielles, tout comme la répartition moderne des savoirs entre les différentes professions et l'héritage du champ de l'intelligence artificielle font dominer de plus en plus une approche de rationalisation des pratiques architecturales. Cette rationalisation s'accompagne d'un discours de plus en plus dominant sur le devenir science de l'architecture, et sur l'aide que représentent les outils computationnels pour atteindre cet objectif. Ceci favorise la mise en œuvre d'une méthodologie de quantification et d'évaluation systématique des propositions architecturales qui tend à standardiser la production, à rebours des expérimentations formelles et spatiales de la génération précédente. Le développement des outils est désormais majoritairement entre les mains de l'industrie du logiciel et de la construction. Leurs développeurs y incorporent directement des mécanismes d'optimisation et de rationalisation. Le recours à ces outils, que la démocratisation met entre les mains de praticiens expérimentés comme de novices, propage donc des biais d'optimisation et de rationalisation. Parce que les utilisateurs n'ont plus toujours la maîtrise de la programmation et le savoir-faire associé aux pratiques computationnelles, la popularisation de cette rationalisation du travail de l'architecte se fait au prix d'un certain scientisme. La vague de démocratisation des outils computationnels qui caractérise cette récente phase de développement du champ ne favorise en effet que l'augmentation du nombre d'utilisateurs et leur autonomie, aux dépens de la transmission du savoir-faire propre au champ.

Les outils computationnels se font ainsi le vecteur d'une rationalisation des pratiques, faisant basculer la négociation entre démocratisation et transmission largement en faveur du premier enjeu. Nous avons vu au chapitre précédent qu'à la faveur de cette vague de démocratisation, deux types de profils prennent de l'importance dans le champ : les amateurs issus de la pratique architecturale conventionnelle et les développeurs issus de l'industrie du logiciel. Parmi les autres profils du champ, certains architectes-programmeurs et ingénieurs-programmeurs ont une volonté de rationalisation de la pratique assumée. Mais la rationalisation des pratiques se répand bien au-delà des activités de ces praticiens individuels. Cette volonté est partagée par certains des éditeurs de logiciels, devenus parmi les profils de développeurs les plus importants du champ. Eux ne font en revanche que répondre à une contrainte économique. Les outils récents développés par ces éditeurs intègrent donc cette pratique de la rationalisation, alors que leur objectif est par ailleurs de démocratiser les pratiques computationnelles. Les outils logiciels agissent ainsi comme des passeurs, non comme des vecteurs de transmission du savoir-faire - ce qui était le cas des outils de programmation et des algorithmes aux

interfaces plus minces. Les nouveaux outils du champ sont porteurs d'un biais de rationalisation car la maîtrise de l'approche computationnelle et de ses enjeux par les praticiens qui y ont recours via ces outils est parfois questionnable. La croissance du champ computationnel implique l'existence des groupes de praticiens investis dans la recherche, et d'autres simples utilisateurs, qui entretiennent un rapport différent au développement des outils.

Nous avons vu les mécanismes de développement qui se cachent derrière ces nouveaux outils du champ au chapitre précédent. Nous y avons aussi exploré la notion de design malveillant des interfaces. S'il ne s'agit pas de design malveillant dans le cas des outils qui nous occupent, il s'agit pourtant souvent d'un design intentionnel. Les intentions des développeurs d'outils sont en effet très claires. Démocratiser implique de faciliter : tout doit pouvoir être fait en quelques clics, comme évoqué au chapitre VI. Les intentions des développeurs sont parfois même plus agressives : certains considèrent qu'il est nécessaire de retirer des mains des architectes certaines décisions de conception, car leurs connaissances techniques et leurs efforts pour se confronter aux contraintes industrielles sont insuffisants. Or un développeur - ou un éditeur - structure son modèle, et donc son algorithme, son programme, son logiciel, à partir de sa vision épistémologique, qui imprègne les décisions prises, donc ce modèle et l'outil qui y est associé. Conjugué à l'épaississement des interfaces provoqué par la démocratisation jusqu'à des utilisateurs amateurs, ceci contribue à créer des boîtes noires propageant le biais de rationalisation au sein du champ computationnel.Ce biais est encouragé par la structure des outils eux-mêmes et par le poids des contraintes économiques, mais aussi par la difficulté de la traduction tacite-explicite à laquelle les praticiens sont confrontés. En effet, concilier la démarche de conception tacite de l'architecte et le besoin d'expliciter qu'impliquent les outils algorithmiques n'est pas aisé. Les modèles scientifiques sur lesquels s'appuient la démarche de rationalisation popularisée dans la phase la plus récente du champ constituent un recours aisé, puisqu'ils proposent une démarche de conception déjà explicitée. Ils constituent donc pour les praticiens amateurs une alternative attractive face à la difficulté de l'acquisition du savoir-faire propre au champ computationnel.

Malgré cette facilité, les biais désormais en vigueur au sein du champ doivent être interrogés. La place accordée aux mécanismes d'optimisation en particulier questionne la pratique architecturale. Le recours aux outils d'optimisation est une approche particulièrement satisfaisante pour les praticiens qui partagent l'ambition de faire de la conception architecturale une science. Les procédures d'optimisation font en effet appel à des modèles quantitatifs et des méthodologies explicites qui permettent de faciliter cet objectif. La conceptualisation de la conception architecturale par la notion de problème de conception, particulièrement populaire chez ces praticiens, offre un modèle théorique auquel adosser ces ambitions. Comme son parent l'automatisation, ce travail d'optimisation suppose

de parvenir à faire émerger des règles logiques de conception universelles. Nous avons vu que c'est ce qui se cache derrière la notion de problème de conception et de définition de méthodologies de résolution de ces problèmes. Le devenir science du travail de conception s'ancre dans une recherche d'efficacité, de productivité, que les outils d'optimisation semblent permettre d'atteindre. Pourtant, même les praticiens dont l'ambition est de parvenir à des sciences de la conception bien structurées font un usage des outils de conception qui relève d'un certain savoir-faire et de la mobilisation de connaissances tacites. L'optimisation multicritère demande en effet pour la formulation des fonctions d'évaluation une certaine inventivité, pour pouvoir retranscrire les impératifs du projet : trouver comment mesurer, pondérer en fonction des nécessités mais aussi des ambitions, pénaliser les solutions qui ne sont pas viables. Les critères des algorithmes d'optimisation ne relèvent pas directement d'une évaluation scientifique qui fait autorité, mais plutôt d'un bricolage pour évaluer certaines qualités spatiales. Automatiser à travers le recours à un processus d'optimisation fonctionne pour produire des bâtiments. Mais seulement un certain type de bâtiment, et encore faut-il tout de même une forme d'adaptation du processus d'optimisation au cas par cas, une adaptation qui ne se fait pas seulement par la modification des entrées.

Or ce travail d'adaptation est justement ce qui fait la richesse du processus d'optimisation. Malgré son adéquation en regard des discours tenu sur la place des outils computationnels dans le processus de conception architectural, c'est le sign que jouer avec la notion d'optimisation peut s'avérer tout aussi riche qu'avec d'autres procédures algorithmiques, et s'éloigner de la standardisation des propositions résultantes. C'est d'ailleurs ce que nous avons vu avec certains praticiens de la troisième période de développement du champ, au chapitre IV, dont les travaux spéculatifs utilisent pour certains des algorithmes génétiques. L'appropriation de ces méthodes d'optimisation y est visible par le jeu avec les fonctions d'évaluations et les gènes, mobilisés plus seulement pour mettre en oeuvre respect des contraintes financières, diminution de la quantité de matière ou régulation thermique, mais aussi pour intégrer des clins d'oeils formels au site ou des références à d'autres projets d'architecture. Optimiser l'architecture peut être abordé par le prisme de la rationalisation. Mais cela peut aussi être le lieu d'une exploration de la notion d'optimisation. Qu'est-ce que le mieux en architecture, quand la définition de la discipline et des ambitions du projet dépend du praticien à l'œuvre ? S'approprier les fonctions d'évaluations et les gènes, c'est pouvoir intégrer sa propre vision du mieux architectural dans un algorithme génétique. Bricoler ces paramètres devient donc un prétexte à une réflexion sur l'objet architectural produit. Les praticiens détournent ainsi l'optimisation pour répondre non aux codes d'une rationalisation des pratiques, mais à une exploration architecturale personnelle. Cela revient aussi à replacer l'optimisation au sein de diverses approches algorithmiques de la conception architecturale, diversité qui fait la richesse des pratiques computationnelles, elle-même reflet de la

richesse architecturale qui caractérise la discipline. L'appropriation des techniques d'optimisation par ce détournement de ses applications, c'est enfin ramener les outils computationnels à leur place d'outils, mobilisés par des praticiens pour développer leur propre approche.

# CONCLUSION
# -
# Computation et épistémologie de l'architecture

*Démocratiser, c'est rationaliser ?*

Le développement du champ computationnel s'articule autour de quatre moments clés. Les deux premières périodes du champ sont celles de sa constitution en un domaine à part entière. Celle-ci possède une double filiation : elle s'ancre dans le domaine de l'intelligence artificielle et dans le domaine de l'animation. Les premières périodes du champ sont celles de son appui sur ces domaines pour constituer un savoir-faire et des outils propres. C'est aussi le moment du passage de la manipulation d'outils primitifs créés en collaboration avec des informaticiens à l'utilisation d'outils développés par et pour eux-mêmes par les praticiens du champ. Les deux périodes suivantes sont le théâtre d'un changement profond dans la place dévolue à ces outils. Ce changement se produit dans la réponse apportée à la nécessaire négociation entre garantie de la transmission du savoir-faire et garantie de la démocratisation des outils. Si le savoir-faire l'emporte initialement, la période la plus contemporaine du champ est celle d'un basculement en faveur de la démocratisation des outils à travers l'industrie. Ce basculement s'accompagne de l'implémentation d'un mécanisme de rationalisation des pratiques architecturales au sein des outils computationnels. À ce titre, la situation contemporaine dans le champ constitue une victoire de l'industrie. En effet, le mécanisme de rationalisation à l'œuvre est celui d'un respect des contraintes industrielles pour garantir productivité et rentabilité dans les productions du champ. Cette ambition de rationalisation n'est jamais très loin du champ computationnel au fil des différentes périodes de son développement, et elle en vient à dominer largement les pratiques. L'évolution du champ computationnel peut donc se lire comme une double domination de l'industrie de la construction et du logiciel sur les pratiques. Les outils algorithmiques sont rendus accessibles à davantage de praticiens, et ils permettent un respect beaucoup plus efficace des contraintes industrielles dans la production architecturale.

Constructibilité, optimisation, maîtrise de la nature : dans le discours aussi en vient à dominer une ambition de contrôle de la production. Beaucoup plus qu'une reconnaissance de la dimension expérimentale, prospective pourtant observée dans les pratiques. Avec la disparition de cette reconnaissance disparaît la richesse de la confrontation du savoir tacite de la conception architecturale avec la formalisation algorithmique. La question se pose de savoir si les explorations spéculatives des architectes qui ont caractérisé un temps les productions du champ sont elles aussi en voie d'extinction. Il est possible que la disparition de la reconnaissance, la domination d'une approche de rationalisation entraîne la disparition du savoir-faire propre au champ qui avait émergé. Le développement du champ tel qu'il s'est produit signerait donc la disparition d'une possibilité d'existence autre pour

l'architecture computationnelle. A ce titre, le récit que nous faisons est-il l'histoire d'un échec pour les architectes ? Le déploiement récent du champ computationnel pourrait aussi se comprendre ainsi. Un échec des architectes à s'emparer véritablement des pratiques de programmation, malgré la démonstration faite par certains de ces praticiens de l'existence d'un espace qui leur est propre. Pourtant les pratiques heuristiques ne disparaissent pas totalement, elles sont simplement confinées dans un espace de pratique restreint. Elles existent d'ailleurs depuis longtemps dans le champ, signe d'une certaine ténacité. Leur cantonnement à cet espace restreint est cependant peut-être inévitable, un coût nécessaire à la démocratisation des outils computationnels. Définir l'architecture par la performance qu'elle doit atteindre n'est pas une condition de l'usage des outils computationnels. C'est un glissement par rapport à la question de leur démocratisation. Ce glissement s'est produit dans le champ, avec les conséquences discutées. Il interroge cependant sur les raisons des pertes observées, dont il est difficile de trancher si elles sont le fait des ambitions de performance ou le fruit de la démocratisation. Fournir des outils algorithmiques faciles à utiliser est nécessaire à leur démocratisation. Mais la capacité des utilisateurs à discerner l'injonction à la performance, ou tout autre injonction intégrée aux outils dont ils se servent n'est peut-être pas forcément mise en danger par cette facilitation. Il est peut-être possible de permettre aux utilisateurs-architectes de manipuler des paramètres de manière à pouvoir implémenter l'intuition architecturale plus facilement, tout en conservant leur capacité d'action sur le modèle.

*Une appropriation alternative de la traduction*

Le devenir boîtes noires ou non des outils de conception algorithmiques dépend des stratégies de négociation entre tacite et explicite à l'œuvre dans le champ et de leur transmission. En fournissant un recours plus facile aux outils de conception algorithmique, les interfaces recourent à un mécanisme de boîte noire : elles deviennent des outils standards de la conception architecturale, et les utilisateurs n'ont plus besoin de s'interroger en détail sur leur fonctionnement et leur pertinence, ni de creuser au-delà de l'interface pour saisir pleinement les modèles mathématiques, physiques et informationnels complexes sur lesquels elles sont basées. Si la simplification par une interface plus facile à manipuler est la clé de la démocratisation des outils de conception algorithmique en architecture, les tactiques de négociation dépendent en grande partie de la compréhension des outils par leurs utilisateurs, et des biais permis par la structuration des typologies algorithmiques, des outils et des interfaces. Il nous faut cependant interroger ce que veut dire exactement comprendre son outil. Nous pouvons supposer qu'il est nécessaire que les praticiens du champ computationnel maîtrisent la programmation. C'est ce que l'attente installée dans le champ que les praticiens développent leurs propres outils sous-entend. La notion de boîte noire pourrait impliquer l'idée que la maîtrise des outils de programmation est nécessaire à la compréhension des algorithmes. Pourtant rien n'indique que la maîtrise si poussée de la programmation de certains praticiens reflète une quantité minimale de savoir technique nécessaire

pour pouvoir pratiquer l'architecture computationnelle. Il n'est donc pas tout à fait certain que le développement d'un savoir-faire propre à la conception computationnelle ne puisse se faire indépendamment du savoir détaillé de la programmation. Les activités d'une partie des développeurs du champ computationnel montrent la conscience qu'ils ont de cette question. Les courbes d'apprentissages qu'ils programment au sein des interfaces témoignent d'une exploration des intermédiaires possibles entre capacité à développer des outils pour les autres et aveuglement d'un utilisateur trop candide. Considérés indépendamment des intentions architecturales de leurs développeurs, les outils et les algorithmes prêts à l'emploi pourraient n'être qu'un palier parmi d'autres dans ces courbes d'apprentissage. Ces paliers sont autant d'étapes d'apprivoisement de la programmation. Il est possible que les utilisateurs puissent se libérer des boîtes noires trop contraignantes à chacune des paliers de cette courbe d'apprentissage. Déterminer si comprendre la logique de la programmation, la logique de traduction du savoir-faire architectural tacite en instructions explicites et non en formes est nécessaire aux pratiques computationnelles est la clé pour que les outils prêts à l'emploi puissent fonctionner comme des boîtes noires convenables.

"*On donne à la machine*", "*on laisse à la machine*", "*si on demande à la machine dessine moi une façade*", "*les formes soudainement deviennent de l'architecture*" nous annoncent les nouveaux développeurs du champ, tout à leur enthousiasme. Pourtant, "la machine" ne fait pas encore toute seule, au service de son utilisateur. Si elle semble faire tout seule, c'est que quelqu'un d'autre lui a donné des instructions à la place de l'utilisateur. Si l'architecte reste un utilisateur candide, cela modifie les enjeux de la pratique computationnelle tels que nous les avons vus. La première possibilité est que les connaissances tacites détenues par ce praticien, dont nous avons pourtant présumé qu'elles sont nécessaires à la production d'une architecture pertinente, ne sont pas intégrées au dispositif algorithmique. La seconde est qu'un autre dispositif de traduction des connaissances tacites est mis en place, un dispositif qui n'est pas exactement celui que nous avons étudié de leur transcription en instructions de programmation. Ceci implique qu'il y aurait d'autres manières d'acquérir le savoir-faire propre aux pratiques computationnelles. La programmation serait donc un artisanat quelque soit l'épaisseur de l'interface. L'idée que la pratique architecturale ne change peut-être pas tant que ça, même lorsqu'elle recourt aux outils de programmation, est un argument en faveur de cette possibilité. Il y aurait donc des boîtes noires convenables pour les pratiques computationnelles. Après tout, il existe des boîtes noires convenables dans le monde de la programmation. La création de langages de programmation de plus haut niveau répond justement à une volonté des programmeurs de se créer des raccourcis efficaces, sans que cela ne menace leur savoir-faire. Au-delà de l'architecture, nos sociétés sont construites sur de multiples boîtes noires considérées comme convenables. La conception computationnelle voyagerait donc de l'énonciation d'instructions en langages de niveau à

la mobilisation d'affordances logicielles[1]. La programmation de l'architecture serait simplement l'affaire d'autres affordances que le dessin papier ou la modélisation, mais des affordances tout aussi bien apprivoisées. Les outils de programmation s'intercaleraient toujours comme une étape de formalisation particulière dans le processus de conception. Le lieu de la traduction du tacite se déplacerait. Ce serait une formalisation par l'énonciation d'instructions de programmation ou la mobilisation d'affordances logicielles - sans distinction entre l'une et l'autre de ces possibilités. Encore faut-il parvenir à mettre en place ces outils de conception computationnelle d'un autre genre.

*Un choix épistémologique déguisé*

Les décisions de programmation qui guident la structuration des algorithmes et des outils sont des décisions de conception au même titre que n'importe quelle autre. C'est le cas que les architectes et autres concepteurs s'en saisissent ou non. Les programmeurs qui travaillent à l'écriture de ces algorithmes et de ces outils en ont bien conscience : ces décisions sont le cœur de leur pratique quotidienne. Au fil de la démocratisation du champ computationnel et de ses outils, la pratique quotidienne des architectes intègre de plus en plus des décisions de modélisation similaires. Les architectes n'en ont pourtant pas toujours conscience. Ces décisions ont néanmoins trait à la nature de leur discipline et à ses modalités de pratique. Ce qui se cache derrière l'évolution au cours des décennies des outils computationnels est en effet un choix épistémologique. C'est ce que trahit la coexistence dans le champ des deux approches de la conception computationnelle de l'architecture, quel que soit leur équilibre au fil du temps. Ces deux approches illustrent deux possibilités d'évolution pour l'architecture en tant que discipline. La rationalisation des pratiques s'ancre en effet pour une partie des praticiens dans l'ambition de transformer la pratique architecturale en science. Les utilisations plus prospectives des outils computationnelles s'ancrent quant à elles dans une approche heuristique plus classique en architecture : à chaque praticien sa manière de faire. Devenir science ou continuer contre la méthode ? Formulées ainsi, ces deux approches peuvent sembler relever de deux extrêmes possibles pour l'architecture. Ce sont néanmoins deux réponses à un vieux débat, dont les tenants sont déjà tiraillés depuis longtemps entre empirisme et rationalité. Un débat tout aussi vieux que le problème corps-esprit, qui interroge les possibilités d'automatisation qu'offre la machine et qui revêt à ce titre une grande importance dans le champ computationnel. Cette bataille computationnelle et architecturale s'inscrit donc dans une controverse épistémologique ancienne.

Cette bataille pose en fait une question cruciale à toute discipline et à laquelle l'architecture n'échappe pas. Cette question est celle des cadres communs d'échange au sein d'une discipline. Les débats dans le champ de la théorie du design, domaine qui se constitue en même temps que l'architecture computationnelle, font écho à ce besoin. La volonté de rationalisation méthodologique que l'on y

[1] def affordabce

observe à partir des années 1960 en est le reflet. L'ambition des auteurs du domaine relève bien plus de l'établissement d'un cadre de pratique commun que du souhait de se conformer aux contraintes industrielles. Lorsqu'ils parlent de leur pratique de la programmation, nombre de programmeurs indiquent aussi comme objectif principal le fait d'écrire du code lisible par les autres. Lorsqu'il s'agit de concevoir outils et interfaces, ils font les mêmes remarques. Les nombreux textes écrits par les praticiens du champ computationnel témoignent de la même ambition épistémologique. La pratique architecturale, comme tout autre champ, se caractérise par le fait d'être à cheval entre connaissances tacites et explicites. Ce qui caractérise les praticiens expérimentés d'une discipline est l'interaction entre les deux formes de savoir. Une épistémologie de la discipline doit donc pouvoir naviguer la ligne entre les deux. Nier le tacite au profit de l'explicite est source de renoncements, mais l'inverse aussi puisqu'il met en danger l'établissement de savoirs collectifs. Trop valoriser les savoirs tacites de la conception architecturale, au détriment du reste, pose le risque de négliger la communication que permettent les savoirs explicites. En effet, le versant explicite de la pratique a un rôle, indépendamment de la rationalisation à l'œuvre au travers de ces méthodes : créer des cadres communs de pratique. Or, un renoncement à ces cadres communs serait pour l'architecture tout aussi problématique qu'un renoncement aux savoir-faire propres aux pratiques architecturales et computationnelles.

Au-delà de la question du savoir-faire individuel de l'architecte se pose ainsi la question du devenir collectif des savoirs de la discipline, et donc de son épistémologie - un enjeu que le recours aux outils algorithmiques questionne inlassablement depuis son émergence dans les années 60. Pour se saisir véritablement des outils computationnels, les praticiens doivent se confronter à leurs ambitions vis-à-vis de la pratique de l'architecture. Plus exactement, pour appréhender pleinement les manières de se saisir des outils computationnels, puisque le panorama que nous avons brossé du champ computationnel révèle de multiples manières de le faire. Les savoirs tacites en jeu dans la conception architecturale relèvent du savoir-faire de la programmation, de la conception architecturale, de la rencontre de ces deux mondes. Mais ils relèvent aussi des convictions portées par les praticiens. La traduction tacite-explicite qui se joue dans le recours aux outils de programmation est une traduction de ces deux éléments. Les outils computationnels sont donc une occasion de questionner les pratiques architecturales, alors que la discipline est en crise depuis plusieurs décennies. Autant que d'automatiser l'architecture, c'est de programmer l'architecture dont il est question ici. Avant d'automatiser, c'est quoi programmer qu'il faut déterminer. Au fil de la genèse du champ computationnel, de nombreux praticiens ont proposé une réponse à cette question. Souvent, d'autres praticiens que des architectes. L'expansion des pratiques de programmation en architecture s'est construite sur une fragilité épistémologique. Les outils computationnels sont pourtant une invitation pour les architectes à se ressaisir des enjeux de définition de leur discipline.

## BIBLIOGRAPHIE


***Champ computationnel en architecture - analyse***

Abrams, Janet, “Muriel Cooper’s Visible Wisdom”, *ID Magazine*, Septembre-Octobre 1994.

Aish, Robert et Nathalie Bredella, “The evolution of architectural computing: from Building Modelling to Design Computation”, *Architectural Research Quarterly*, Vol. 21, n°1, 2017.

Ahlquist, Sean et Achim Menges, *Computational Design Thinking*, Wiley & Sons, 2011.

Ahlquist, Sean, “Sensorial Playscape: Advanced Structural, Material and Responsive Capacities of Textile Hybrid Architectures as Therapeutic Environments for Social Play”, dans Menges, Achim, Bob Sheil, Ruairi Glynn, et Marilena Skavara, *Fabricate 2017. Rethinking Design and Construction*, UCL Press, 2017.

Allen, Matthew, “Prehistory of the Digital: Architecture Becomes Programming, 1935-1990”, Thèse de doctorat, Harvard University, Graduate School of Arts & Sciences, 2019.

Allen, Matthew, “Architecture becomes Programming. Invisible technicians and local theories in the late 1960s”, dans Loosen, S., Heynickx, R. et Heynen, H. (éditeurs), *The Figure of Knowledge: Conditioning Architectural Theory, 1960s -1990,* Leuven university Press, 2020, p. 101-126.

André-Garguilo, Julie , “La fabrique de l'architecte extraordinaire : L'Architectural Association School, 1964-1983”, Thèse de doctorat, Université Paris-Est, 2020.

Ballantyne, Andrew, *Deleuze and Guattari for Architects*, Routledge, 2007.

Bird, Lawrence et Guillaume LaBelle, “Re-Animating Greg Lynn's Embryological House: A Case Study in Digital Design Preservation”, *Leonardo* Vol. 43, n°3, 2010.

Bourbonnais, Sébastien, “Sensibilités technologiques : expérimentations et explorations en architecture numérique 1987-2010”, Thèse de Doctorat, Université Paris-Est, 2014.

Carpo, Mario, *The Alphabet and the Algorithm*, The MIT Press, 2011.

Carpo, Mario, “Twenty Years of Digital Design”, dans Mario Carpo (éditeur), *1993-2012, The Digital Turn in Architecture*, Wiley & Sons, 2012.

Carpo, Mario, *The Second Digital Turn: Design Beyond Intelligence*, The MIT Press, 2017.

Cashen, Daniel et Neil Katz, “Skidmore, Owings & Merrill. Building on the legacy of technological and architectural innovation” dans Hensel, Michael et Fredrik Nilsson (éditeurs), *The Changing Shape of Architecture. Further Cases of Integrating Research and Design in Practice*, Routledge, 2019, p. 136-146.

Cristia, Emilien, Pierre-Paul Zalio et François Guéna, “Quand le BIM met la maquette à l’épreuve du

numérique", *Actes de la conférence SCAN'18 – 8e Séminaire de Conception Architecturale Numérique*, SHS Web Conf., Vol. 47, 2018.

Cogdell, Christina, *Toward a Living Architecture?: Complexism and Biology in Generative Design*, University Of Minnesota Press, 2019.

Alan Colquhoun, *Modern Architecture*, Oxford University Press, 2002.

Davis, Daniel, "Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture." Thèse de doctorat, RMIT University, 2013.

Davis, Daniel, "A History of Parametric", 6 Aout 2013,
https://www.danieldavis.com/a-history-of-parametric/,
consulté le 14 Novembre 2021.

Davis, Daniel, et Brady Peters, "Design Ecosystems: Customising the Architectural Design Environment with Software Plug-ins". *AD Architectural Design*, Vol. 83, n°2, 2013, p. 124-131.

Léda Dimitriadi, "Algorithmique du hasard et créativité : de la nature à la computation", intervention dans le cadre de la conférence *Projeter l'architecture, aux carrefours du numérique et du vivant*, 27 janvier 2020, ENSA Paris Val-de-Seine.

Dubberly, Hugh et Paul Pangaro, "How cybernetics connects computing, counterculture, and design", dans Blauvelt, Andrew (éditeur), *Hippie Modernism: The Struggle for Utopia*, Walker Art Center, 2015.

Eisenman, Peter, "The End of the Classical: The End of the Beginning, the End of the End", *Perspecta*, Vol. 21, 1984, p. 154-173.

Fankhänel, Teresa et Andres Lepik (éditeurs), *The Architecture Machine. The Role of Computers in Architecture,* Birkhäuser, 2020.

Fromonot, Françoise, *Jorn Utzon : The Sydney Opera House*, Phaidon Press, 2002.

Gaudillière, Nadja, "Towards an History of Computational Tools in Automated Architectural Design - The Seroussi Pavilion Competition as a Case Study", dans Haeusler, Matthias, MarcAurel Schnabel et Tomohiro Fukuda (éditeurs), *Intelligent & Informed – Proceedings of the 24th CAADRIA Conference - Volume 2, Victoria University of Wellington, Wellington, New Zealand, 15-18 April 2019*, Hong Kong : The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2019, p. 581-590.

Gaudillière, Nadja, "Computational Tools in Architecture and their Genesis : the Development of Agent-based Models in Spatial Design", dans Holzer, Dominic, Walaiporn Nakapan, Anastasia Globa et Immanuel Koh (éditeurs), *RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th CAADRIA Conference - Volume 2, Chulalongkorn University, Bangkok, Thailand, 5-6 August 2020*, Hong Kong : The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2020, p. 97-506.

Gaudillière, Nadja, “Evolutionary Tools and the Practice of Architecture: from Appropriated Typology to Becoming the Black Boxes of CAAD.” dans Estevez, Alberto T. (éditeur), *Proceedings of the 4th International Conference of Biodigital Architecture & Genetics*, Barcelona: iBAG-UIC Barcelona, 2020.

Gaudillière-Jami, Nadja, “AD Magazine. Mirroring the Development of the Computational Field in Architecture 1965-2020”, dans Slocum, Brian, Viola Ago et Adam Marcus (éditeurs), *ACADIA 2020 Distributed Proximities: Proceedings of the 40th Annual Conference of the Association for Computer Aided Design in Architecture, Volume I: Technical Papers, Keynote Conversations*, Acadia Publishing Company, 2021, p. 150-159.

Goodhouse, Andrew (éditeur), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, 2013.

Hardingham, Samantha, *Cedric Price Works 1952–2003: A Forward-minded Retrospective*, Architectural Association Publications, 2016.

Hernandez Vargas, José, “From the Fun Palace to the Generator. Cedric Price and the conception of the first intelligent building”, in *ARQ* (Santiago), n°90, p. 48-57, 2015.

Holzer, Dominik, “Design exploration supported by digital tool ecologies”, *Automation in Construction*, Vol. 72, n°1, p. 3-8, 2016.

Janssen, Patrick et Rudi Stouffs, “Types of parametric modelling”, dans Ikeda, Yasushi, Christiane M. Herr, Dominik Holzer, Sawako Kaijima, Mi Jeong Kim et Marc-Aurel Schnabel (éditeurs), *Emerging Experiences of the Past, Present and Future of Digital Architecture, Proceedings of the 20th International Conference of the Association of Computer-Aided Architectural Design Research in Asia CAADRIA 2015*, Hong Kong : The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2015, p.157–166.

Kuzmenko Kateryna, Nadja Gaudillière, Adélaide Feraille, Justin Dirrenberger et Olivier Baverel, “Assessing the Environmental Viability of 3D Concrete Printing Technology”, dans Gengnagel Christophe, Olivier Baverel, Jane Burry, Mette Ramsgaard Thomsen et Stefan Weinzierl (éditeurs), *Impact: Design With All Senses. Proceedings of the Design Modelling Symposium, Berlin 2019*, Springer, 2020, p. 517-528.

Kwinter, Sanford, *Far from Equilibrium: Essays on Technology and Design Culture*, Actar Press, 2008.

Leach, Neil, “Parametrics explained”, *Next Generation Building*, Vol. 1, p. 1-10, 2014.

Lindsey, Bruce, *Digital Gehry: Material Resistance, Digital Construction*, Birkhäuser, 2001.

Lynn, Greg (éditeur), *Archaeology of the Digital,* Sternberg Press, 2013.

Maudet, Nolwenn, “Muriel Cooper ‚Informations Landscapes”, *Back Office #1 - Faire avec*, Éditions B42, 2017.

Meister, Anna-Maria, “From Form to Norm : the Systematization of Values in German Design”, Thèse de doctorat, Princeton University, School of Architecture, 2013.

Mitchell, William J., “A Tale of Two Cities: Architecture and the Digital Revolution”, *Science*, Vol 285, n°5429, p. 839-841, 1999.

Pangaro, Paul, “An Idiosyncratic History of Conversation Theory in Software, and its Progenitor, Gordon Pask”, *Kybernetes*, Vol. 30 n° 5-6, 2001.

Pertigkiozoglou, Eliza, “1973. Nicholas Negroponte and Architecture Machine Group MIT”, 17 Février 2017, https://eliza-pert.medium.com/1973-a1b835e87d1c, consulté le 30 Novembre 2021.

Pertigkiozoglou, Eliza, “1976. Cedric Price & John Frazer”, 8 Avril 2017, https://eliza-pert.medium.com/1976-22121bb498c4, consulté le 15 Novembre 2021.

Pertigkiozoglou, Eliza, “1976. Architecture Machine Group MIT”, 14 Avril 2017, https://eliza-pert.medium.com/1976-852a377855fe, consulté le 15 Novembre 2021.

Picon, Antoine, “Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction”, *Revue d'histoire des sciences*, Vol. 42, n° 1-2, p. 155-172, 1989.

Picon, Antoine, *L'invention de l'ingénieur moderne: l'Ecole des ponts et chaussées, 1747-1851*, Presses de l'École Nationale des Ponts et Chaussées, 1993.

Picon, Antoine, *Culture numérique et architecture : une introduction*, Birkhäuser, 2010.

Picon, Antoine, “Histoires du numérique : information, ordinateur et communication”, dans Goodhouse, Andrew (éditeur), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, 2013, p. 79-98.

Port, Stanley, *The Management of CAD for Construction*, New York: Springer, 1989.

Schumacher, Patrik, “Parametricism - A New Global Style for Architecture and Urban Design”, *AD Architectural Design*, Vol. 79, n° 4, 2009.

Serraino, Pierluigi, *History of Form*Z*, Birkhäuser, 2002.

Simonnet, Cyrille, *Le Béton, histoire d’un matériau. Économie, technique, architecture*, Paris, éd. Parenthèses, 2005.

Stals, Adeline, Catherine Elsen et Sylvie Jancart, “L’immersion pour l’appréhension des outils de modélisation paramétrique en conception architecturale”, *Actes de la conférence SCAN’18 – 8e Séminaire de Conception Architecturale Numérique*, SHS Web Conf., Vol. 47, 2018.

Stals, Adeline, “Pratiques numériques émergentes en conception architecturale dans les bureaux de petite taille. Perceptions et usages de la modélisation paramétrique”, Thèse de doctorat, Université de Liège, 2019.

Stasiuk, David, “Design Modeling Terminology”, 13 Juin 2018, https://archinate.files.wordpress.com/2018/06/dstasiuk-design-modeling-terminology1.pdf, consulté le 14 Novembre 2021.

Wright Steenson, Molly, “Cedric Price’s Generator”, *Crit* n°69, p. 14-15, 2010.

Wright Steenson, Molly, “Archéologies de l’information”, dans Goodhouse, Andrew (éditeur), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, 2013, p. 9-22.

Wright Steenson, Molly, “Architectures of Information: Christopher Alexander, Cedric Price & Nicholas Negroponte”, thèse de doctorat, Princeton University, 2013,

Wright Steenson, Molly, “Microworld and Mesoscale”, *Interactions*, n° 22, p. 58-60, 2015.

Wright Steenson, Molly, *Architectural Intelligence: How Designers and Architects Created the Digital Landscape*, The MIT Press, 2017.

Sunwoo, Irene, “From the “Well-Laid Table” to the “Market-Place:” The Architectural Association Unit System”, *Journal of Architectural Education*, Vol. 65, n° 2, p. 24-41, 2012.

Roussel, Marion, “A la couture des mondes… Transarchitecture et hypersurfaces : une introduction” https://dnarchi.fr/analyses/a-la-couture-des-mondes-transarchitecture-et-hypersurfaces-une-introduction/, consulté le 30 Novembre 2021.

Ruffle, Simon, "Architectural design exposed: from computer-aided-drawing to computer-aided-design", *Environments and Planning B: Planning and Design*, Vol. 13, n°4, p. 385-389, 1986.

Terzidis, Kostas, *Expressive Form : A Conceptual Approach to Computational Design*, Routledge, 2003.

Teyssot, Georges et Samuel Bernier-Lavigne, « Forme et information. Chronique de l’architecture numérique, » dans Guiheux, Alain (éditeur), *Action Architecture*, Paris: Éditions de la Villette, 2011, p. 49-87.

Upitis, Alice, “Two or More Architectures. Computers and Design at MIT until 1963” dans Dutta, Arindam (éditeur), *A Second Modernism. MIT, Architecture and the “Techno-Social” Moment*, The MIT Press, 2013, p.516-535.

Vardouli, Theodora, “Architecture-by-yourself. Early studies in computer aided participatory design”, Mémoire de Master, M.Arch Design and Computation, MIT, 2010.

Weisberg, David, “The engineering design evolution”, https://www.cadhistory.net/toc.htm, consulté le 30 Novembre 2021.

Woodbury, Robert, *Elements of Parametric Design*, New York : Routledge, 2010.

Zardini, Mirko, “Huit millions d’histoires”, dans Goodhouse, Andrew (éditeur), *Quand le numérique marque-t-il l'architecture ?*, Sternberg Press, 2013, p. 9-22.

***Champ computationnel en architecture - cas d’études***

Adilenidou, Yota, Zeeshan Yunus Ahmed, Freek Bos et Marjan Colletti, “Unprintable Forms. Complexity as a Robustness Factor for Robotic Fabrication and 3DCP Constraints through Error Elimination and Reinsertion”, dans Bieg, Kory, Danelle Briscoe et Clay Odom (éditeurs), *ACADIA 19: Ubiquity and Autonomy. Proceedings of the 39th Annual Conference of the Association for Computer Aided Design in Architecture, The University of Texas at Austin School of Architecture, Austin, Texas 21-26 October, 2019*, ACADIA, 2019, p.168-177.

Agustí-Juan, Isolda, Florian Müller, Norman Hack, Timothy Wangler et Guillaume Habert, “Potential benefits of digital fabrication for complex structures: Environmental assessment of a robotically fabricated concrete wall”, *Journal of Cleaner Production*, Vol. 154, p. 330–340, 2017.

Aish, Robert, “DesignScript: Origins, Explanation, Illustration”, dans Gengnagel, Christoph, Axel Kilian, Norbert Palz et Fabian Scheurer (éditeurs), *Computational Design Modelling. Proceedings of the Design Modelling Symposium Berlin 2011*, Springer, 2012, p. 1-8.

Aish, Robert, “DesignScript: a learning environment for Design Computation”, dans Gengnagel, Christoph, Axel Kilian, Julien Nembrini et Fabian Scheurer (éditeurs), *Rethinking Prototyping. Proceedings of the Design Modelling Symposium Berlin 2013*, Berlin : Universität der Künste, 2013.

Aish, Robert, “DesignScript: Scalable Tools for Design Computation”, dans Stouffs, Rudi et Sevil Sariyildiz (éditeurs), *Computation and Performance – Proceedings of the 31st eCAADe Conference – Volume 2, Faculty of Architecture, Delft University of Technology, Delft, The Netherlands, 18-20 September 2013*, 2013, p. 87-96.

Aish, Robert, “First Build Your Tools”, dans Terri Peters et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013, p. 36-49.

Aish, Robert et Emmanuel Mendoza. “DesignScript: a domain specific language for architectural computing” dans Tolvanen, Juha-Pekka, Jeff Gray, Matti Rossi, et Jonathan Sprinkle (éditeurs), *Proceedings of the International Workshop on Domain-Specific Modeling (DSM 2016)*, New York : Association for Computing Machinery, 2016, p. 15–21.

Aish, Robert et Robert Woodbury, “Multi-level Interaction in Parametric Design”, dans Butz, Andreas, Brian Fisher, Antonio Krüger et Patrick Olivier (éditeurs), *Smart Graphics: 5th International Symposium, SG 2005, Frauenwörth Cloister, Germany, August 22-24, 2005, Proceedings*, Berlin, Heidelberg : Springer, 2005, p. 151-162.

Akizuki, Yuta, Mathias Bernhard, Reza Kakooee, Marirena Kladeftira et Benjamin Dillenburger, “Generative Modelling with Design Constraints - Reinforcement Learning for Object Generation”, dans Holzer, Dominic, Walaiporn Nakapan, Anastasia Globa et Immanuel Koh (éditeurs), *RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th CAADRIA Conference - Volume 2, Chulalongkorn University, Bangkok, Thailand, 5-6 August 2020*, Hong Kong : The

Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2020, p. 445-454.

Alexander, Christopher, Murray Silverstein, Shlomo Angel, Sara Ishikawa et Denny Abrams, *The Oregon Experiment*, Oxford University Press, 1975.

Alexander, Christopher, Murray Silverstein, et Sara Ishikawa, *A Pattern Language : Towns, Buildings, Construction*, Oxford University Press, 1977.

Alexander, Christopher, *The Timeless Way of Building*, Oxford University Press, 1979

Alexander, Christopher, *The Nature of Order: An Essay on the Art of Building and the Nature of the Universe*, Berkeley, Californie: Center for Environmental Structures, 2003.

Alison, Jane (éditrice), *Future city : experiment and utopia in architecture*, Londres : Thames & Hudson, 2006.

Anderson, Carl, Carlo Bailey, Andrew Heumann et Daniel Davis “Augmented space planning: Using procedural generation to automate desk layouts”, *International Journal of Architectural Computing*, Vol. 16, n°2, 2018, p. 64-177.

Andrasek, Alisa, *biothing*, HYX, 2009.

Ashour, Yassin et Branko Kolarevic, “Heuristic Optimization in Design”, dans Combs, Loon et Chris Perry (éditeurs), *ACADIA 2015: Computational Ecologies: Design in the Anthropocene. Proceedings of the 35th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA), Cincinnati 19-25 October, 2015)*, Association for Computer Aided Design in Architecture, 2015, p. 357-369.

Assem, Ayman, Sherif Abdelmohsen et Mohamed Ezzeldin, "A Fuzzy-Based Approach for Evaluating Existing Spatial Layout Configurations", dans Sousa, José Pedro, Gonçalo Castro Henriques et João Pedro Xavier (éditeurs), *Proceedings of the 37th eCAADe and XXIII SIGraDi Joint Conference, “Architecture in the Age of the 4Th Industrial Revolution”, Porto 2019*, São Paulo: Blucher, 2019 p. 35-44.

Baybars, Ilker et Charles M. Eastman,, “Enumerating architectural arrangements by generating their underlying graphs”, *Environment and Planning B*, Vol. 7, n°3, p. 289–310, 1980.

Benedikt, Michael, “To Take Hold of Space: Isovists and Isovist Fields”, *Environment and Planning B Planning and Design*, Vol.6, n°1, p. 47-65, 1979.

Beorkrem, Christopher, J. Ellinger, P. Bernstein et A. Hauck, « Multivariate Schematic Design Tooling », dans Chien, Shengfen, Seungyeon Choo et Marc-Aurel Schnabel (éditeurs), *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016) / Melbourne 30 March–2 April 2016*, Hong Kong : The Association for Computer-Aided Architectural Design Research in Asia, 2016, p. 383-394.

Bernholtz, Allen, “Some thoughts on computer role playing and design”, *Ekistics*, Vol. 26, n° 157, p. 522-524, 1968.

Besserud, Keith et Joshua Cotten, “Architectural Genomics”, intervention dans le cadre de la conférence *ACADIA 2008: Silicon + Skin: Biological Processes and Computation*, Université de Minneapolis, 16-19 Octobre 2008.

Bhooshan, Shajay, “Realizing Architecture’s Disruptive Potential”, *Oz*, Vol. 38, n° 1, 2016.

Bollinger, Klaus, Manfred Grohmann et Oliver Tessmann, “Form, Force, Performance Multi-Parametric Structural Design”, *AD Architectural Design*, Vol. 78, n° 2, p. 20-26.

Brayer, Marie-Ange (éditrice), *Architecture expérimentales 1950-2012*, Orléans : Éditions HYX : FRAC Centre, 2013.

Brayer, Marie-Ange (éditrice), *Imprimer le Monde*, Orléans, France : Éditions HYX, 2017.

Brayer, Marie-Ange et Frédéric Migayrou (éditeurs), *Naturaliser l’architecture : Archilab 9e édition*, Orléans : HYX, 2013.

Brayer, Marie-Ange et Olivier Zeitoun (éditeurs), *La Fabrique du Vivant*, Paris : Editions du Centre Pompidou, 2019.

Brown, Nathan et Caitlin Mueller, “Design for structural and energy performance of long span buildings using geometric multi-objective optimization”, *Energy and Buildings*, Vol. 127, p. 748-761, 2016.

Brown, Nathan et Caitlin Mueller, “Designing with data: moving beyond the design space catalog”, dans Nagakura, Takehiko, Caitlin Mueller, Skylar Tibbits et Mariana Ibanez (éditeurs), *Disciplines and Disruption - Proceedings Catalog of the 37th Annual Conference of the Association for Computer Aided Design in Architecture, ACADIA 2017,* ACADIA, 2017, p. 154-163

Brown, Nathan et Caitlin Mueller, “Quantifying diversity in parametric design: a comparison of possible metrics”, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 33, n° 1, 2018, p. 40-53.

Brown, Nathan et Caitlin Mueller, “Design variable analysis and generation for performance-based parametric modeling in architecture”, *International Journal of Architectural Computing*, Vol. 17, n°1, p. 36-52, 2019.

Bryant Greg, “Gatemaker: Christopher Alexander's dialogue with the computer industry”, dans Neis, Hans Joachim, Gabriel A. Brown et Greg Bryant (éditeurs), *Battle for the Life and Beauty of the Earth, PUARL Conference Proceedings*, Portland Urban Architecture Research Laboratory, 2013, pp.77-91.

Building Research Bord du National Research Council, “Report From the 1984 Workshop on Advanced Technology for Building Design and Engineering”, 1984.

Burry, Jane, Daniel Davis, Brady Peters, Phil Ayres, John Klein, Alexander Pena de Leon et Mark Burry, 'Modelling Hyperboloid Sound Scattering: The Challenge of Simulating, Fabricating and Measuring', dans Gengnagel, Christoph, Axel Kilian, Norbert Palz et Fabian Scheurer (éditeurs), *Computational Design Modelling. Proceedings of the Design Modelling Symposium Berlin 2011*, Springer, 2012, pp. 89–96.

Burry, Mark (éditeur), *Urban Futures : Designing the Digitalised City*, Wiley & Sons, 2020.

Burry, Mark, Jane Burry et Jordi Faulí, "Sagrada Família Rosassa: Global Computer Aided Dialogue between Designer and Craftsperson (Overcoming Differences in Age, Time and Distance)", dans Wassim Jabi, Wassim (éditeur), *Reinventing the discourse: how digital tools help bridge and transform research, education and practice in architecture : proceedings of the Twenty First Annual Conference of the Association for Computer-Aided Design in Architecture*, Buffalo, NY: Association for Computer-Aided Design in Architecture, 2001, p 76-85.

Cache, Bernard, *Terre Meuble*, Editions HYX, 1990.

Cache, Bernard et Patrick Beaucé, "Vers un Mode de Production Non-standard", dans Cache, Bernard et Patrick Beaucé, *Objectile. Fast-Wood : A Brouillon Project*, Vienna : Springer, Consequence Book Series on Fresh Architecture, vol 6., 2007.

Calixto, Victor et Gabriella Celani, "A literature review for space planning optimization using an evolutionary algorithm approach : 1992-2014", dans *Project Information for Interaction Proceedings of the 19th International Conference of the The Iberoamerican Society of Digital Graphic (SIGRADI) 2015, 23 -27 November 2015*, Universidade Federal de Santa Catarina, Florianópolis/SC, Brasil, 2015.

Castle, Helen, "Editorial", *AD Architectural Design*, Vol. 80, p. 5, 2010.

Chaillou, Stanislas, Conférence "Intelligence artificielle & Architecture" au Pavillon de l'Arsenal, le 27 février 2020. Visible sur Dailymotion. Consulté le 11 Juillet 2021. https://www.dailymotion.com/video/x7snxh1.

Chaillou, Stanislas, "Architecture & Style. A New Frontier for AI in Architecture", https://towardsdatascience.com/architecture-style-ded3a2c3998f, consulté le 12 Novembre 2021

Chéraud, Florian, "Beyond Design Freedom Providing a Set-Up For Material Modelling within Kangaroo Physics", dans Werner, Liss C. et Dietmar Koering (éditeurs), *Proceedings of 38th Education and research in computer aided architectural design in Europe (eCAADe) conference 2020: anthropologic: architecture and fabrication in the cognitive age*, eCAADe (Education and Research in Computer Aided Architectural Design in Europe), 2020, p. 459-468.

Choi, Yoon K., "The Morphology of Exploration and Encounter in Museum Layouts", *Environment and Planning B: Planning and Design*, Vol. 26, n°2, p. 241-250, 1999.

CNCCookbook, "2021 CAD Survey", https://www.cnccookbook.com/cnccookbook-2021-cad-survey-market-share-customer-satisfaction/, consulté le 13 Novembre 2021.

Coates, Paul, *Programming Architecture*, Routledge, 2011.

Coates, Paul, Tom Appels, Corinna Simon et Christian Derix, “Current work at CECA”, dans Soddu, Celestino (éditeur), *Proceedings of the 4th Generative Art Conference (GA2001)*, Milan: Generative Design Lab Milan Polytechnic University, Italy, 2001.

Coates, Paul et Claudia Schmid, “Agent Based Modelling”, dans Knight, Michael W. et André G. P. Brown (éditeurs) *Architectural Computing From Turing to 2000 : Proceedings of the 17th Conference on Education in Computer Aided Architectural Design in Europe, 15-17 September 1999, CAAD Research Unit, School of Architecture and Building Engineering, University of Liverpool, UK*, eCAADe (Education and Research in Computer Aided Architectural Design in Europe), 1999, p. 652-661.

Colletti, Marjan et Marcos Cruz, “Output 3 (Design) Invisible University”, University of Westminster, 2008.

Corke, Greg, “Rhino 3D”, 25 Juillet 2008, https://develop3d.com/product-design/rhino-4/, consulté le 02 Septembre 2021.

Coyne, Richard, D. et John S. Gero, “Design Knowledge and Sequential Plans”, *Environment and Planning B: Planning and Design*, Vol. 12, n°4, p. 401-418, 1985.

Coyne, Richard, D., Michael A. Rosenman, Antony D. Radford, M. Balachandran, et John S. Gero, *Knowledge-Based Design Systems*, Reading, Massachusetts : Addison Wesley, 1990.

Damski, José C. et John S. Gero, “An evolutionary approach to generating constraint-based space layout topologies”, dans Junge, Richard (éditeur), *CAAD Futures 1997*, Dordrecht : Kluwer, 1997, p. 855-864.

Das, Subhajit, Colin Day, Michael Dewberry, Varvara Toulkeridou et Anthony Hauck, “Automated Service Core Generator in Autodesk Dynamo - Embedded Design Intelligence aiding rapid generation of design options”, dans Herneoja, Aulikki, Toni Österlund et Piia Markkanen (éditeurs), *Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 2, University of Oulu, Oulu, Finland, 22-26 August 2016*, CUMINCAD, 2016, p. 217-226.

David, B.T., “Multi-Expert Systems for CAD”, dans ten Hagen Paul J.W. et Tetsuo Tomiyama (editeurs), *Intelligent CAD Systems I. Eurographic Seminars (Tutorials and Perspectives in Computer Graphics)*, Berlin, Heidelberg : Springer, 1987.

Davis, Daniel “DesignScript – Autodesk”, 27 Juin 2011, https://www.danieldavis.com/designscript-autodesk/, consulté le 13 Novembre 2021.

Day, Martyn, “Rhino Grasshopper”, AEC Magazine, 2 Juin 2009, https://aecmag.com/news/rhino-grasshopper/, consulté le 02 Septembre 2021

Day, Martyn, “Autodesk AEC customers demand better value”, 25 Juillet 2020, https://aecmag.com/bim/letter-to-autodesk-aec-customers-demand-better-value/, consulté le 13 Novembre 2021.

De Rycke, Klaas, Christoph Gengnagel, Olivier Baverel, Jane Burry, Caitlin Mueller, Minh Man Nguyen, Philippe Rahm et Mette Ramsgaard Thomsen (éditeurs), *Humanizing Digital Reality: Design Modelling Symposium Paris 2017*, Springer, 2018.

Deguine, Louise, Nadja Gaudillière, Laya Hermelin, Ines Rogriguez-Porcel, "Wikitecture, a collaborative and open-source platform for a decentralized architecture production", https://www.academia.edu/10391518/Wikitecture_a_collaborative_and_open-source_platform_for_a_decentralized_architecture_production, consulté le 12 Novembre 2021.

Diaz Alonso, Hernan (éditeur), *Exuberance*, Wiley & Sons, 2010

Din, Edward, & Athanassios Economou, (2007). "Rewind–Pause–Forward: The Wall Variations of the Smith House" dans Akleman, Ergun et Ranjith Perumalil, *Proceedings of the Sixth Conference of the International Society of Arts, Mathematics and Architecture-ISAMA Vol. 7*, Texas A&M University, College and Department of Architecture, 2007, p. 135-144.

Dincer, Ahmet Emre, Gülen Çağdaş et Hakan Tong, "A Digital Tool for Customized Mass Housing Design", dans Herneoja, Aulikki, Toni Österlund et Piia Markkanen (éditeurs), *Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 2, University of Oulu, Oulu, Finland, 22-26 August 2016*, CUMINCAD, 2016.

Dynamo Team, "Space planning in Dynamo with DynaSpace", https://dynamobim.org/space-planning-in-dynamo-with-dynaspace/, consulté le 12 Novembre 2021.

Eastman, Charles M., "Automated space planning", *Artificial Intelligence*, Volume 4, n°1, p. 41-64, 1973.

Ednie-Brown, Pia, "All-over, over-all: biothing and emergent composition", *AD Architectural Design*, Vol. 76, p. 72-81, 2006.

Ednie-Brown, Pia, Mark Burry et Andrew Burrow (éditeurs), *The Innovation Imperative : Architectures of Vitality*, Wiley & Sons, 2013

Ericson, Mark, "Review: Grasshopper Algorithmic Modeling for Rhinoceros 5", *Journal of the Society of Architectural Historians*, Vol. 76 n°4, p. 580–583, 2017.

EZCT Architecture and Design Research, "Studies on Optimization: Computational Chair Design using Genetic Algorithms (with Hatem Hamda and Marc Schoenhauer)", 2004. http://transnatural.org/wp-content/uploads/2011/09/EZCT_Booklet-Screen.pdf, consulté le 02 Septembre 2021.

Frazer, John, "Autotectonics : The choice of choice", Diplôme de fin d'études, Architectural Association, 1969.

Frazer, John, *An evolutionary architecture*, Architectural Association Publications, 1995.

Frazer, John, “The Cybernetics of Architecture: A Tribute to the Contribution of Gordon Pask”, *Kybernetes. The International Journal of Systems & Cybernetics*, Vol. 30, n°5-6, p. 641-651, 2001.

Freedman, Richard et George L. Heersey, *Possible Palladian Villas (Plus a Few Instructively Impossible Ones)*, The MIT Press, 1992.

Galle, Per, “An algorithm for exhaustive generation of building floor plans”, *Communications of the ACM*, Vol. 24, n°12, p.813-825, 1981.

Galle, Per, “Branch & Sample: Systematic combinatorial search without optimization”, rapport technique 87/5, Department of Computer Science, University of Copenhagen, Copenhagen, 1987.

Galle, Per, “Christopher Alexander's battle for beauty in a world turning ugly: The inception of a science of architecture?”, *She Ji: The Journal of Design, Economics, and Innovation*, Vol. 6, n°3, p. 345-385, 2020.

Galle, Per, “Christopher Alexander's battle for beauty: Any prospects of victory? (Author's response.)”, *She Ji: The Journal of Design, Economics, and Innovation*, Vol. 6, n°3, p. 380-385, 2020.

Gaspar-Cunha, António, Dirk Loyens et Ferrie van Hattum “Aesthetic design using multi-objective evolutionary algorithms”, dans Takahashi Ricardo H. C., Kalyanmoy Deb, Elizabeth F. Wanner et Salvatore Greco (éditeurs), *Evolutionary Multi-Criterion Optimization. 6th International Conference, EMO 2011, Ouro Preto, Brazil, April 5-8, 2011. Proceedings*, Berlin Heidelberg: Springer, 2011, p. 374-388.

Gengnagel, Christoph, Olivier Baverel, Jane Burry, Mette Ramsgaard Thomsen et Stefan Weinzierl (éditeurs), *Impact: Design With All Senses: Proceedings of the Design Modelling Symposium, Berlin 2019*, Springer, 2020.

Gengnagel, Christoph, Axel Kilian, Julien Nembrini et Fabian Scheurer (éditeurs), *Rethinking Protoyping. Proceedings of the Design Modelling Symposium Berlin 2013*, Berlin : Universität der Künste, 2013.

Gengnagel, Christoph, Axel Kilian, Norbert Palz et Fabian Scheurer (éditeurs), *Computational Design Modelling. Proceedings of the Design Modelling Symposium Berlin 2011*, Springer, 2012

Gero, John S. (éditeur), *Artificial Intelligence in Design '91*, Elsevier, 1991.

Getting Simple, épisode 9 : “Ian Keough - How to make Better Decisions Faster”, Aout 2018, https://open.spotify.com/episode/0Q5y5YWOdBW95XCZgtQ1xS?si=Z9YuDxY7QuSOcHGBn05pmg&nd=1, consulté le 13 Novembre 2021.

Gips, James, “Implementing Shape Grammars with Computers”, dans Gero, John, S. (éditeur), *Design Computing and Cognition '16*, Springer, 2017.

Gramazio, Fabio et Matthias Kohler (éditeurs), *Made by Robots: Challenging Architecture at a Larger Scale*, Wiley & Sons, 2014.

Guenoun, Elias (éditeur), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.

Harten, Arthur van der, "Pachyderm Acoustical Simulation: Towards Open-Source Sound Analysis", *AD Architectural Design*, Vol. 83, p. 138-139, 2013.

Hensel, Michael (éditeur), *Performance-Oriented Architecture: Rethinking Architectural Design and the Built Environment*, Wiley & Sons, 2013.

Hensel, Michael, Achim Menges et Michael Weinstock (éditeurs), *Emergence : Morphogenetic Design Strategies*, Wiley & Sons, 2004.

Hensel, Michael et Achim Menges, "Material and Digital Design Synthesis: Integrating Material Self-Organisation, Digital Morphogenesis, Associative Parametric Modelling and Computer-Aided Manufacturing", *AD Architectural Design*, Vol. 76, n° 2, p. 88-97, 2006.

Hensel, Michael et Achim Menges (éditeurs), *Versatility and Vicissitude Performance in Morpho-Ecological Design*, Wiley & Sons, 2008.

Hemberg, Martin, Una May O'Reilly, Achim Menges, Katrin Jonas, Michel da Costa Gonçalves et Steven R. Fuchs, "Genr8: Architects' Experience with an Emergent Design Tool", dans Romero Juan et Penousal Machado (éditeurs), *The Art of Artificial Evolution. Natural Computing Series*, Berlin, Heidelberg : Springer, 2008, p. 167-188.

Hillier, Bill, *Space is the Machine: A Configurational Theory of Architecture*, Cambridge University Press, 1996.

Hillier, Bill et Julienne Hanson, *The Social Logic of Space*, Cambridge University Press, 1984.

Holt, Steven et Mara Holt Skov (éditeurs), *Blobjects & Beyond: The New Fluidity in Design*, San Francisco : Chronicle Books, 2005.

Holzer, Dominik, Richard Hough et Mark Burry, "Parametric Design and Structural Optimisation for Early Design Exploration", *International Journal of Architectural Computing*, Vol. 5, n°4, p. 625-643, 2007.

Holzer, Dominik et Steven Downing, "Optioneering: A New Basis for Engagement Between Architects and Their Collaborators" *AD Architectural Design*, Vol. 80, p. 60-63, 2010.

Hwang, Irene (éditeur), *Verb Natures : What is Nature ? How Can we Modify Nature? What is the Nature of Nature*, ACTAR, 2006.

Jacob Grobman, Yasha et Eran Neuman (éditeurs), *Performalism: Form and Performance in Digital Architecture*, London ; New York : Routledge, 2012.

Jencks, Charles, *Architecture 2000: Predictions and Methods*, New York, Praeger, 1971.

Johnson, Philip et Mark Wigley, *Deconstructivist architecture*, MoMa, 1988.

Jon Peddie Research, “2020 CAD Report”, 2020.

Kara, Hanif, “On Design Engineering”, *AD Architectural Design*, Vol. 80, p. 46-51, 2010.

Karandikar, Swanandesh S., “Expert system applications in architecture”, Mémoire de master, Institut Polytechnique de Virginie, 1989.

Kilian Axel, “Design Innovation through Constraint Modeling”, *International Journal of Architectural Computing*, Vol. 4, n°1, p. 87-105, 2006.

Krause, Jeffrey, “Agent Generated Architecture”, dans Jordan, J. Peter, Bettina Mehnert et Anton C. Harfmann (éditeurs), *Design and Representation : Proceedings of the 1997 Association for Computer Aided Design in Architecture (ACADIA)*, Association for Computer Aided Design in Architecture, 1997, p. 63-70.

Lee, Jisun et Hyunsoo Lee, “Agent-driven accessibility and visibility analysis in nursing units”, dans Haeusler, Matthias, MarcAurel Schnabel et Tomohiro Fukuda (éditeurs), *Intelligent & Informed – Proceedings of the 24th CAADRIA Conference - Volume 2, Victoria University of Wellington, Wellington, New Zealand, 15-18 April 2019*, Hong Kong : The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2019, p. 351-360.

Legendre, George L., (éditeur), *Mathematics of Space*, Wiley & Sons, 2011

Li, Lezhi, Renyuan Hu, Meng Yao, Guangwei Huang et Ziyu Tong, “Sculpting the Space: A Circulation Based Approach to Generative Design in a Multi-Agent System”, dans Gu, Ning, Shun Watanabe, Halil Erhan, Matthias Hank Haeusler, Weixin Huang et Ricardo Sosa (éditeurs), *Rethinking Comprehensive Design: Speculative Counterculture, Proceedings of the 19th International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2014) / Kyoto 14-16 May 2014*, The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2014, p. 565–574.

Lynn, Greg, *Animate Form*, Princeton Architectural, 1997.

Lynn, Greg, Folds, *Bodies & Blobs. Collected Essays*, La lettre volée, 1998.

Mangelsdorf, Wolf, “Structuring Strategies for Complex Geometries”, *AD Architectural Design*, Vol. 80, p. 40-45, 2010.

March, Lionel, “Mathematics and Architecture since 1960”, dans Williams, Kim et Jose-Francisco Rodrigues (éditeurs), *Nexus IV : Architecture and Mathematics*, Kim Williams Books, 2002, p. 7-33.

Marsault, Xavier, “A multiobjective and interactive genetic algorithm to optimize the building form in early design stages”, Intervention dans le cadre de la conférence *Building Simulation 2013*, Août 2013, Chambéry, France.

Maxwell, Iain et Dave Pigram, “Inorganic Speciation: Matter, Behaviour and Formation in Architecture”, dans Krauel, Jacobo (éditeur), *Contemporary Digital Architecture Design & Techniques*, Links, 2010, p. 208-227.

Maxwell, Iain et Dave Pigram, “Algorithmic Typology : Towards in Operational Model”, dans Leach, Neil et Philippe F. Yuan (éditeurs), *Scripting the Future*, Tongji University Press, 2012, p. 107-112.

McCullough, Malcolm, William J. Mitchell, Patrick Purcell (éditeurs), *The Electronic Design Studio*, The MIT Press, 1990.

Menges, Achim (éditeur), *Material Synthesis : Fusing the Physical and the Computational*, Wiley & Sons, 2015.

Mesnil, Romain, “Explorations structurelles de domaines de formes constructibles pour l’architecture non-standard”, Thèse de Doctorat, ENPC, 2017.

Migayrou, Frédéric (éditeur), *Architectures Non-Standard*, Paris Centre Pompidou, 2003.

Migayrou, Frédéric (éditeur), *Coder le monde : mutations, créations*, Paris : Éditions du Centre Pompidou, 2018.

Mitchell, J. R. et Antony Radford. “EAVE, a Generative Expert System for Detailing”, *Environment and Planning B: Planning and Design*, Vol. 14, p. 281 - 292, 1987.

Mitchell, William, J., *Computer-Aided Architectural Design*, John Wiley & Sons, 1977.

Mitchell, William, J., *The Logic of Architecture. Design, Computation, and Cognition*, The MIT Press, 1990.

Mitchell, William, J., “Reviewed Work(s): Possible Palladian Villas (Plus a Few Instructively Impossible Ones) by George Hersey and Richard Freedman: The Program”, *AA Files* n° 26 (Autumn 1993), pp. 97-99.

Mueller, Caitlin et John A. Ochsendorf, “Combining structural performance and designer preferences in evolutionary design space exploration”, *Automation in Construction*, Vol. 52, p. 70-82, 2015.

Mueller, Caitlin et John A. Ochsendorf, “An integrated computational approach for creative conceptual structural design”, dans Obrębski, Jan B. et Romuald Tarczewski (éditeurs), *Beyond the limits of man : proceedings of the IASS 2013 Symposium, Wrocław, Poland 23-27 September 2013*, Wrocław : Oficyna Wydawnicza Politechniki Wrocławskiej, 2013.

Mukerjee, Amit, “Review of Knowledge-Based Design Systems”, *AI Magazine*, Vol. 12, n°3, p. 122, 1991.

Nagy, Danil, Damon Lau, John Locke, James Stoddart, Lorenzo Villaggi, Ray Wang, Dale Zhao, et David Benjamin, “Project Discover: An Application of Generative Design for Architectural Space Planning”, dans Turrin, Michela (éditrice), *Proceedings of the 2017 Symposium on Simulation for Architecture and Urban Design (SimAUD 2017)*, Red Hook, NY : Curran Associates, Inc., 2017.

Nagy, Danil, Lorenzo Villaggi, Dale Zhao et David Benjamin, “Beyond Heuristics: A Novel Design Space Model for Generative Space Planning in Architecture”, dans Nagakura, Takehiko, Skylar Tibbits, Mariana Ibañez et Caitlin Mueller (éditeurs), *ACADIA 2017 : Disciplines [and] Disruption : proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture : Massachusetts Institute of Technology, School of Architecture + Planning, Department of Architecture*, Association for Computer Aided Design in Architecture (ACADIA), 2017, p. 436-445.

Nagy, Danil, Lorenzo Villaggi et David Benjamin “Generative urban design: integrating financial and energy goals for automated neighborhood layout”, dans Rakha, Tarek, Michela Turrin, Siobhan Rockcastle, Daniel Macumber et Forrest Meggers (éditeurs), *Symposium on Simulation for Architecture and Urban Design (SimAUD 2018) : Delft, the Netherlands, 4-7, June 2018*, Vista, California : The Society for Modeling and Simulation International, 2018, p. 1-8.

Negroponte, Nicholas, “Towards a Theory of Architecture Machines”, *Journal of Architectural Education*, Vol. 23, n°2, p. 9-12, 1969.

Negroponte, Nicholas, *The Architecture Machine*, The MIT Press, 1970.

Noever Peter, Kimberli Meyer et Open Source Architecture (éditeurs), *The Gen(h)ome Project*, Los Angeles : MAK, 2006.

ONE TO ONE, “One Million Cells and Ten Thousand Panels: Digital Fabrication of Elbphilharmonie’s Acoustic Interior”, http://onetoone.net/wp-content/uploads/2017/01/161128_PR_Elbphilharmonie.pdf, consulté le 12 Novembre 2021.

Oosterhuis Kas, *HYPERBODY, First Decade of Interactive Architecture*, Jap Sam Books, 2012.

Oxman, Rivka, “Expert System for Generation and Evaluation in Architectural Design”, Thèse de doctorat, Technion – Israel Institute of Technology, 1988

Oxman, Rivka, “Performance-based Design: Current Practices and Research Issues”, I*nternational Journal of Architectural Computing*, Vol. 6, n°1, p. 1-17, 2008.

Oxman, Neri, “Structuring Materiality: Design Fabrication of Heterogeneous Materials”, *AD Architectural Design*, Vol. 80, p. 78-85, 2010.

Park, James et Athanassios Economou, “The Dirksen variations: Towards a generative description of Mies's courthouse language”, dans Martens, Bob (éditeur), *eCAADe 2015 : proceedings of the 33rd International Conference on Education and Research in Computer Aided Architectural Design in Europe, 16-18 September 2015, Vienna, Austria. Vol. 1, Real time : extending the reach of computation*, Vienna : eCAADe : Faculty of Architecture and Regional Planning, 2015.

Pantic, Igor et Someem Hahm, “Isomorphic Agency, Emerging Experience in Past, Present and Future of Digital Architecture”, dans Ikeda, Yasushi, Dominik Holzer, Sawako Kaijima, Mi Jeong Kim et Marc Aurel Schnabel (éditeurs), *Emerging experiences in the past, present and future of digital*

*architecture : proceedings (CD) of the 20th International Conference on Computer-Aided Architecture Design Research in Asia*, Hong Kong : The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), 2015, p. 178-188.

Petes, Terri et Brady Peters (éditeurs), *Inside Smart Geometry : Expanding the Architectural Possibilities of Computational Design*, Wiley & Sons, 2013.

Piacentino, Giulio, "Weaverbird: Topological Mesh Editing for Architects", *AD Architectural Design*, Vol. 83, p. 140-141, 2013.

Pierson, John, " Space Planning in Dynamo with DynaSpace – Using Generative Design in Revit", 24 Mai 2021, https://dynamobim.org/space-planning-in-dynamo-with-dynaspace-using-generative-design-in-revit/, consulté le 13 Novembre 2021.

Piker, Daniel, "Kangaroo: Form Finding with Computational Physics", *AD Architectural Design*, Vol. 83, p. 136-137, 2013.

Poinet, Paul, et Al Fisher. "Computational Extensibility and Mass Participation in Design" dans Sheil, Bob, Mette Ramsgaard Thomsen, Martin Tamke et Sean Hanna (éditeurs), *Design Transactions: Rethinking Information Modelling for a New Material Age*, UCL Press, 2020, p. 56–61.

Preisinger, Clemens, "Linking Structure and Parametric Geometry", *AD Architectural Design*, Vol. 83, p. 110-113, 2013.

Reas, Casey et Ben Fry, "A Modern Prometheus, The History of Processing by Casey Reas and Ben Fry", 29 Mai 2018, https://medium.com/processing-foundation/a-modern-prometheus-59aed94abe85, consulté le 02 Septembre 2021.

Ramsgaard Thomsen, Mette, Martin Tamke, Christoph Gengnagel, Billie Faircloth et Fabian Scheurer (éditeurs), *Modelling Behaviour. Design Modelling Symposium 2015*, Springer, 2015.

Reinhardt Dagmar, Densil Cabrera, Alexander Jung et Rod Watt, "Towards a Micro Design of Acoustic Surfaces", dans Reinhardt, Dagmar, Rob Saunders et Jane Burry (éditeurs), *Robotic Fabrication in Architecture, Art and Design 2016*, Basel : Springer, 2018, p. 136-149.

Retsin, Gilles (éditeur), *Discrete : Reappraising the Digital in Architecture*, Wiley & Sons, 2019.

Rolvink, Anke, Caitlin, Mueller et Jeroen Coenders, "State of the art of computational tools for conceptual structural design", dans *Proceedings of the IASS-SLTE 2014 Symposium : shells, membranes and spatial structures: footprints*, IASS-SLTE, 2014.

Rutten, David, "Discoverability", 25 Mai 2012, https://ieatbugsforbreakfast.wordpress.com/2012/05/25/discoverability/#more-431, consulté le 02 Septembre 2021.

Rutten, David, "Galapagos: On the Logic and Limitations of Generic Solvers", *AD Architectural Design*, Vol. 83, p. 132-135, 2013.

Rosenman, Michael A., John S. Gero et Richard Coyne. “SOLAREXPERT : A Prototype Expert System for Passive Solar Energy Design in Housing”, dans Topping, Barry H.V. (éditeur), *Artificial Intelligence Techniques and Applications for Civil and Structural Engineers*, Civil-Comp Press, Edinburgh, 1987, p. 131-139.

Sanguinetti, Paola, Marcelo Bernal, Maher El-Khaldi, et Matthew Erwin, “Real-time design feedback: coupling performance-knowledge with design iteration for decision-making”, dans McGraw, Robert, Eric Imsand et Michael J Chinni (éditeurs), *Proceedings of the 2010 Spring Simulation Multiconference (SpringSim '10)*, Society for Computer Simulation International, San Diego, CA, USA, 2010, p. 1–8.

Scheurer, Fabian, “Materialising Complexity”, *AD Architectural Design*, Vol. 80, p. 86-93, 2010.

Schumacher, Patrik, “Parametricism as Style - Parametricist Manifesto”, Transcription d’une intervention au Dark Side Club1 , 11th Architecture Biennale, Venice 2008. https://www.patrikschumacher.com/Texts/Parametricism%20as%20Style.htm, consulté le 12 Novembre 2021

Schumacher, Patrik, *The Autopoiesis of Architecture*, John Wiley & Sons Ltd., London 2010.

Schumacher, Patrik, “The Parametricist Epoch: Let the Style Wars Begin”, *AJ - The Architects’ Journal*, Vol 231, n°16, 2010.

Schumacher, Patrik, “Parametric Semiology – The Design of Information Rich Environments”, dans Pablo Lorenzo-Eiroa et Aaron Sprecher (éditeurs), *Architecture In Formation – On the Nature of Information in Digital Architecture*, Routledge, Taylor and Francis, New York, 2013.

Schumacher, Patrik, “Freedom via Soft Order : Architecture as A Foil for Social Self-Organisation”, dans Hopkins, Owen (éditeur), *AD Special Issue - Architecture and Freedom : Searching for Agency in a Changing World*, Wiley & Sons, 2018.

Schumacher, Patrik, « Social Performativity - Architecture’s Contribution to Societal Progress ». Consulté le 11 juillet 2021. https://www.patrikschumacher.com/Texts/AU%20Parametricism.html.

Schumacher, Patrik, « Advancing Social Functionality via Agent Based Parametric Semiology ». Consulté le 11 juillet 2021. https://www.patrikschumacher.com/Texts/Operationalising%20Architectures%20Core%20Competency.html.

Shea, Kristina, Robert Aish et Marina Gourtovaia, “Towards integrated performance-driven generative design tools”, *Automation in Construction*, Vol. 14, n° 2, p. 253-264, 2005.

SHoP/Sharples Holden Pasquarelli (éditeurs), *Versioning : Evolutionary Techniques in Architecture*, Wiley & Sons, 2002.

Smith, Rick, *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace, 2017.

Snooks, Roland, “Behavioral Formation: Multi-Agent Algorithmic Design Strategies”. Thèse de Doctorat, RMIT University, 2014.

Snoonian Glenn, Deborah, “Innovation: Technology Briefs. GenerativeComponents Software. GenerativeComponents Software gives "bending the rules" a whole new meaning. 01 Octobre 2003. https://www.architecturalrecord.com/articles/12260-generativecomponents-software, consulté le 02 Septembre 2021.

Stiny, George et James Dips, “Shape Grammars and the Generative Specification of Painting and Sculpture”, *IFIP Congress, Information Processing 71*, 1972, p. 1460-1465.

Stiny, George et William J. Mitchell, “The Palladian Grammar”, *Environment and Planning B*, Vol. 5, n°1, pp. 5-18, 1978.

Stiny, George et William J. Mitchell, “Counting Palladian Plans”, *Environment and Planning B*, Vol. 5, n°2, pp. 189-198, 1978.

Stuart-Smith, Robert, “Behavioural Matter”, Conférence á la faculté d’architecture, Université du Costa Rica, 7 Octobre 2011.

Stuart-Smith, Robert, “A Matter of Organization : Re-emphasizing Organization”, dans Neil Leach et Philippe F. Yuan (éditeurs), *Scripting the Future*, Tongji University Press, 2012 p. 157-162.

Sobek, Werner, “Radical Sources of Design Engineering”, *AD Architectural Design*, Vol. 80, p. 24-33, 2010.

Spatial Corp. VS Autodesk, Inc., https://www.wsgr.com/publications/PDFSearch/autodesk0306.pdf, consulté le 13 Novembre 2021.

ten Hagen, Paul J.W. et Tetsuo Tomiyama (éditeurs), *Intelligent CAD Systems I: Theoretical and Methodological Aspects*, Springer, 1987.

Tessmann, Oliver, “Collaborative Design Procedures for Architects and Engineers”, Thèse de doctorat, Université de Kassel, 2008.

Thompson, D’Arcy, *On Growth and Form*, Cambridge University Press, 1917.

Tibbits, Skylar, *Self-Assembly Lab: Experiments in Programming Matter*, Taylor & Francis, 2016

Tomlow, Jos, “Designing and Constructing the Olympic Roof (Munich 1972)” *International Journal of Space Structures*, Vol. 31, no. 1, p. 62–73, 2016.

Tseranidis, Stavros, “Approximation algorithms for rapid evaluation and optimization of architectural and civil structures”, Mémoire de master, Massachusetts Institute of Technology, 2015.

Tseranidis, Stavros, Nathan C. Brown et Caitlin Mueller, “Data-driven approximation algorithms for rapid performance evaluation and optimization of civil structures”, *Automation in Construction*, Vol. 72, p. 279-293, 2016.

Walker, John, *The Autodesk File: Bits of History, Words of Experience*, John Walker, 2017.

West, Martin, “Paul Coates 1945-2013 research emerging”, Mémoire de master, University of Cambridge, 2014.

Woodbury, Robert, Robert Aish et Axel Kilian, “Some Patterns for Parametric Modeling”, dans Lilley, Brian et Philip Beesley (éditeurs), *Expanding bodies : art, cities, environment ; proceedings of the ACADIA 2007 conference, Halifax, Nova Scotia, October 1-7, 2007*, Riverside Architectural Press, 2007.

Yessios, Christos, “Syntactic Structures and Procedures for Computable Site Planning”, Thèse de doctorat. Carnegie Mellon, 1973.

Youmans, Robert J. et Tomaz Arciszewski, “Design fixation: Classifications and modern methods of prevention”, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, Vol. 28, n°2, p. 129-137, 2014.

***Intelligence artificielle, cybernétique, informatique***

Autor, David, “Polanyi's Paradox and the Shape of Employment Growth”, NBER Working Paper Series, Cambridge, MA: National Bureau of Economic Research, pp. 1–48, 2014.

Barricelli, Nils Aall, “Esempi numerici di processi di evoluzione”, *Methodos* 1954, p. 45–68, 1954.

Berg, Marc, *Rationalizing Medical Work. Decision Support Techniques and Medical Practices*, The MIT Press, 1997

Bosqué, Camille, “La fabrication numérique personnelle, pratiques et discours d’un design diffus : enquête au coeur des FabLabs, hackerspaces et makerspaces de 2012 à 2015”, thèse de doctorat, Université Rennes 2, 2016.

Bringsjord, Selmer et Naveen Sundar Govindarajulu, "Artificial Intelligence", dans Edward N. Zalta (éditeur), *The Stanford Encyclopedia of Philosophy (Summer 2020 Edition)*, 2020.

Brand, Stuart, *The Medialab : Inventing the Future at MIT*, Penguin Books, 1987.

Cardon, Dominique, Jean-Philippe Cointet, et Antoine Mazières. « La revanche des neurones ». *Réseaux*, Vol. 211, n°5, p.173‑220, 2018.

Celani, Gabriela, et Carlos Eduardo Verzola Vaz, “CAD Scripting and Visual Programming Languages for Implementing Computational Design Concepts: A Comparison from a Pedagogical Point of View”, *International Journal of Architectural Computing*, Vol. 10, n° 1, p. 121–37, 2012.

Chiou, Stefanie, Craig Music, Kara Sprague et Rebekah Wahba, “A Marriage of Convenience : The Founding of the MIT Artificial Intelligence Laboratory” - http://web.mit.edu/6.933/www/Fall2001/AILab.pdf, consulté le 12 Novembre 2021.

Cole, David, "The Chinese Room Argument", dans Edward N. Zalta (éditeur), *The Stanford Encyclopedia of Philosophy (Winter 2020 Edition)*, 2020.

Crevier, Daniel, *AI: The Tumultuous Search for Artificial Intelligence*, New York, NY: BasicBooks, 1993.

Dawkins, Richard, *The Blind Watchmaker*, New York: W. W. Norton & Company, 1986.

De Mol, Liesbeth, "Turing Machines", dans Edward N. Zalta (éditeur), *The Stanford Encyclopedia of Philosophy (Fall 2021 Edition)*, 2021.

Do, Truong-Dong, Minh-Thien Duong, Quoc-Vu Dang et My-Ha Le, "Real-Time Self-Driving Car Navigation Using Deep Neural Network", dans *4th International Conference on Green Technology and Sustainable Development (GTSD)*, 2018, pp. 7-12.

Dreyfus, Hubert L., *Alchemy and Artificial Intelligence*, Santa Monica, CA: RAND Corporation, 1965.

Dreyfus, Hubert L., *What Computers Can’t do. The Limits of Artificial Intelligence*, The MIT Press, 1972.

Dreyfus, Hubert L., *What Computers Can’t do. A Critique of Artificial Reason*, The MIT Press, 1992.

Dreyfus, Hubert L. et Stuart E. Dreyfus, *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*, New York, NY: Free Press, 1987.

Duportail, Judith, *L’amour sous algorithme*, Paris: le Livre de poche, 2020.

Dyson, Freeman, “Science as a craft industry”, *Science*, Vol. 280, n°5366, p. 1014-1015, 1998.

Fraser, Alex, “Simulation of Genetic Systems by Automatic Digital Computers I. Introduction”, *Australian Journal of Biological Sciences*, Vol. 10, n°4, p. 484, 1957.

Elrad, Tzilla, Robert E. Filman et Atef Bader, “Aspect–Oriented Programming”, *Communications of the ACM*, Vol. 44 n°10, p. 29–32, 2001.

Erman, Lee D., Philip E. London et Stephen F. Fickas, “The design and an example use of hearsay-MI”, dans *Proceedings of IJCAI 81, 24-28 August 1981 University of British Columbia Vancouver, B.C., Canada*, 1981, p. 409-415.

Eubanks, Virginia, *Automating inequality: how high-tech tools profile, police, and punish the poor*, New York, NY: St. Martin’s Press, 2017.

Ferber, Jacques, *Les systèmes multi-agents*, InterEditions, 1995.

Gray, Colin M., Yubo Kou, Bryan Battles, Joseph Hoggatt, et Austin L. Toombs, “The Dark (Patterns) Side of UX Design”, dans *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 2018, p. 1–14.

Green, Thomas R. G., et M. Petre, “Usability Analysis of Visual Programming Environments: a Cognitive Dimensions Framework”, *Journal of Visual Languages and Computing*, Vol. 7, n°2, p. 131-174, 1996.

Gross, Benedikt, Claudius Lazzeroni, Hartmut Bohnacker et Julia Laub, *Generative Design : Visualize, Program, and Create with JavaScript in p5.js*, Princeton Architectural Press, 2018.

Hayes-Roth, Frederick, Donald Waterman et Douglas Lenat, *Building Expert Systems*, Addison-Wesley, 1983.

Hedlund, Gustav A., “Endomorphisms and automorphisms of the shift dynamical system”, *Math. Systems Theory*, Vol. 3, n°4, p. 320–3751, 1969.

Hodges, Andrew, “Uncomputability in the work of Alan Turing and Roger Penrose”, présentation donnée à Hambourg le 06 Octobre 2000 dans le cadre de la conférence Interface 5.

Hodges, Andrew, "Alan Turing", dans Edward N. Zalta (éditeur), *The Stanford Encyclopedia of Philosophy (Winter 2019 Edition)*, 2019.

Kahneman, Daniel, *Thinking, Fast and Slow*, New York: Farrar, Straus & Giroux, 2011.

Holland, John, *Adaptation in Natural and Artificial Systems*, The MIT Press, 1975.

LeCun, Yann, O. Matan, Bernard E. Boser, John S. Denker, Don Henderson, R.E. Howard, W.E. Hubbard, Larry D. Jackel, et Henri S. Baird, “Handwritten zip code recognition with multilayer networks”, dans *Proceedings 10th International Conference on Pattern Recognition June 16 1990 to June 21 1990, Atlantic City, NJ, USA, Vol. 2*, 1990, p. 35-40.

Leitao, Antonio et Luis Santos, “Programming Languages for Generative design.Visual or Textual?”, dans Zupančič-Strojan, Tadeja, Matevž Juvančič, Špela Verovšek et Anja Jutraž (éditeurs), *Respecting Fragile Places, 29th eCAADe Conference Proceedings*, University of Ljubljana, Faculty of Architecture (Slovenia), Ljubljana, 2011, 549-557.

Legrenzi, Christophe, “Informatique, numérique et système d’information : définitions, périmètres, enjeux économiques”, Vie & sciences de l'entreprise, Vol. 2015/2, n°200, p. 49-76, 2015.

Leviathan, Yaniv et Yossi Matias, “Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone”, 2018, consulté le 30 Novembre 2021. https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html.

Lohr, Steve. « Facial Recognition Is Accurate, If You’re a White Guy ». The New York Times, 9 février 2018, sect. Technology. Consulté le 11 juillet 2021. https://www.nytimes.com/2018/02/09/technology/facial-recognition-race-artificial-intelligence.html.

Macey, Jon, “A technical Introduction to Maya”, présentation au National Center for Computer Animation, Bournemouth University, 2020.

Mandelbrot, Benoît, *Les Objets fractals - Forme, hasard et dimension*, Flammarion, 1975.

Masure, Anthony, *Le design des programmes : des façons de faire du numérique*, Thèse de doctorat, Université Paris-1 Panthéon Sorbonne, 2014.

Matuszek, Cynthia, John Cabral, Michael Witbrock et John DeOliveira, “An Introduction to the Syntax and Content of Cyc”, dans “Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, Papers from the 2006 AAAI Spring Symposium”, Rapport technique SS-06-05, Stanford, California, USA, Mars 27-29, 2006. AAAI, 2006.

McBreen, Pete, *Software Craftsmanship : The New Imperative*, Boston : Addison Wesley, 2001.

McCarthy, John, “The inversion of functions defined by Turing machines”, Dans Shannon, Claude E. et John McCarthy, *Automata studies, Annals of Mathematics studies no. 34, Princeton University Press*, Princeton, 1956, p. 177–181.

McCarthy, John et Patrick J. Hayes, “Some philosophical problems from the standpoint of artificial intelligence”, dans B. Meltzer and D. Michie (éditeurs), *Machine Intelligence 4*, Edinburgh University Press, 1969, p. 463-502.

McCorduck, Pamela, *Machines Who Think* (2e édition), Natick, MA: A. K. Peters, Ltd., 2004.

McCulloch, Warren S. et Walter Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity”, *Bulletin of Mathematical Biophysics*, Vol. 5, p. 115-133, 1943.

Minsky, Marvin, "Steps toward Artificial Intelligence," , *Proceedings of the IRE*, Vol.49, n°1, p. 8-30, 1961.

Minsky, Marvin et Seymour Papert, “Proposal to ARPA for Research on Artificial Intelligence at MIT, 1970–1971”, MIT Artificial Intelligence Lab, Cambridge, MA, 1970.

Minsky, Marvin, “A Framework for Representing Knowledge”, Rapport technique, Massachusetts Institute of Technology, USA, 1974.

Minsky, Marvin, “Why People Think Computers Can’t”, *AI Magazine*, Vol. 3 n°4, p. 3-16, 1982.

Murata, Mikio, “Reaction–Diffusion Equations and Cellular Automata”, dans Anderssen, Robert S, Philip Broadbridge, Yasuhide Fukumoto, Kenji Kajiwara, Matthew Simpson et Ian Turner (éditeurs), *Agriculture as a Metaphor for Creativity in All Human Endeavors. FMfI 2016. Mathematics for Industry, Vol 28.*, Springer, 2018.

Novak, James et Jennifer Loy, “Recoding Product Design Education: Visual Coding for Human Machine Interfaces”, *KnE Engineering*, Vol. 2, n°2, p.227-234, 2017.

Open AI, “Dota 2 with Large Scale Deep Reinforcement Learning”, 2019, https://arxiv.org/abs/1912.06680, consulté le 30 Novembre 2021.

Oppy, Graham et David Dowe, "The Turing Test", dans Edward N. Zalta (éditeur), *The Stanford Encyclopedia of Philosophy (Fall 2020 Edition)*, 2020.

Papert, Seymour & Marvin Minsky, *Perceptrons*, The MIT Press, 1969.

Pask, Gordon, *Conversation, cognition and learning*, New York: Elsevier, 1975.

Pickering, Andrew, “Gordon Pask: From Chemical Computers to Adaptive Architecture”, dans Pickering, Andrew, *The Cybernetic Brain. Sketches of Another Future*, University of Chicago Press, 2009.

Prusinkiewicz, Przemysław et Aristid Lindenmayer, “Graphical modeling using L-Systems”, dans Prusinkiewicz, Przemysław et Aristid Lindenmayer, *The Algorithmic Beauty of Plants*, Springer, 1990, p. 1-50.

Reeves, Jack, “What is Software Design?”, *C++ Journal*, 1998.

Raymond, Eric et Bob Young, *The Cathedral & the Bazaar*, O'Reilly, 2001.

Rescorla, Michael, "The Computational Theory of Mind", dans Edward N. Zalta (éditeur), *The Stanford Encyclopedia of Philosophy (Fall 2020 Edition)*, 2020.

Reynolds, Craig, “Flocks, herds and schools: a distributed behavioral model”, dans Stone, Maureen C, (éditrice), *SIGGRAPH ’87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, p. 25-34, 1987.

Rogers, David, *An Introduction to NURBS with historical perspective*, Morgan Kaufmann Publishers, 2001.

Russell, Stuart et Peter Norvig, *Artificial Intelligence: A Modern Approach*, Saddle River, NJ: Prentice Hall, 1995.

Searle, John, “Minds, Brains and Programs”, *Behavioral and Brain Sciences*, Vol. 3, p. 417–57, 1980.

Seibel, Peter, *Coders at Work: Reflections on the Craft of Programming*, Apress, 2009.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel et Demis Hassabis, “Mastering the game of Go without human knowledge”, *Nature*, Vol. 550, p. 354–359, 2017.

Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, et

Demis Hassabis, “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”, 2017, https://arxiv.org/abs/1712.01815, consulté le 30 Novembre 2021.

Simon, Herbert A. et Stuart A. Eisenstadt, “A Chinese Room that Understands”, dans Preston, John M. et John Mark Bishop (éditeurs), *Views Into the Chinese Room: New Essays on Searle and Artificial Intelligence*, Oxford University Press, 2003.

Simonite, Tom, “What Really Happened When Google Ousted Timnit Gebru” *Wired*, 06 Août 2021, consulté le 11 juillet 2021, https://www.wired.com/story/google-timnit-gebru-ai-what-really-happened/.

Smith, Alvy Ray, “Cellular Automata Theory”, Thèse de doctorat, Stanford University, 1969.

Sviokla, John J., “An Examination of the Impact of Expert Systems on the Firm: The Case of XCON”, *MIS Quarterly*, Vol. 14, no. 2, p. 127–40, 1990.

Turing, Alan, “Systems of logic defined by ordinals”, *Proc. Lond. Math. Soc.*, Vol. 2, n°45, p. 161–228, 1939.

Turing, Alan, “Computing Machinery and Intelligence,” *Mind*, Vol. 59, n°236, p. 433–60, 1950.

Untersinger, Martin, “ Réussite contestée d'un ordinateur au légendaire test de Turing”, *Le Monde* du 09 Juin 2014.

Walia, Ravi K. et Y. S. Parmar, *Algorithm & Flowchart Manual*, Computer and Instrumentation Center, University of Horticulture & Forestry, Nauni Solan, Inde.

Ward, Colin D. et Peter I. Cowling, “Monte Carlo search applied to card selection in Magic: The Gathering”, dans Lanzi, Pier Luca (éditeur), *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009, Milano, Italy, 7-10 September,* IEEE, 2009.

Weimar, Jörg R., “Cellular automata for reaction-diffusion systems”, *Parallel Computing*, Vol. 23, n°11, p. 1699-1715, 1997.

Weizenbaum, Joseph, “ELIZA—a computer program for the study of natural language communication between man and machine”, *Communications of the ACM*, Vol. 9, n°1, p. 36-35, 1966.

Winsberg, Eric, *Science in the age of computer simulation*, University of Chicago Press, 2010

Wolfram, Stephen, *A New Kind of Science*, Wolfram Media, 2002.

Ye, Gan Zhen et Dae-Ki Kang, “Extended Evolutionary Algorithms with Stagnation-Based Extinction Protocol”, *Applied Sciences*, Vol. 11, n°8, p. 3461-3476, 2021.

Zuse, Konrad, *Rechnender Raum*, Braunschweig: Friedrich Vieweg & Sohn, 1969.

***Design Studies***

Alexander Christopher, *Notes on the Synthesis of Form*, Harvard University Press, 1964.

Archer, Bruce, “Design as a Discipline”, *Design Studies*, Vol. 1, n°1, p.17-20, 1979.

Buchanan, Richard, “Wicked Problems in Design thinking”, *Design Issues*, Vol. 8, n°2, p. 5-21, 1992.

Cross, Nigel, “Human and Machine Roles in Computer Aided Design”, Thèse de doctorat, Design Research Laboratory, Department of Building, University of Manchester, Institute of Science and Technology, 1974.

Cross, Nigel, “Designerly Ways of Knowing”, *Design Studies,* Vol. 3, n°4, p. 221-227, 1982.

Cross, Nigel,, “Can a machine design?”, *Design Issues*, Vol. 17, n°4, p. 44-50, 2001.

Eastman, Charles, “Cognitive processes and ill-defined problems: a case study from design”, *IJCAI'69: Proceedings of the 1st international joint conference on Artificial Intelligence*, Morgan Kaufmann Publishers, 1969, p. 669–690.

Friedman, Yona, *Pour l’architecture scientifique*, Pierre Belfond, 1971.

Lawson, Bryan, *How designers think*, Routledge, 2006.

Newell, Allen, “Heuristic programming: ill-structured problems”, dans Rosenbloom, Paul S., John Laird et Allen Newell (éditeurs), *The Soar papers Vol. 1: research on integrated intelligence*, Cambridge, MA, USA: MIT Press, 1993, p. 3–54.

Reitman, Walter, "Heuristic decision procedures, open constraints, and the structure of ill-defined problems", dans Bryan, Glenn, L. et Maynard W. Shelly (éditeurs) *Human Judgments and Optimality*, Wiley, 1964.

Rittel, Horst et Melvin Webber, “Dilemmas in a General Theory of Planning”, *Policy Sciences*, Vol. 4, p. 155-169, 1973.

Rudofsky, Bernard, *Architecture without Architects: A Short Introduction to Non-Pedigreed Architecture*, Doubleday & Company, 1964.

Schön, Donald, *The Reflective Practitioner. How professionals think in action*, Temple Smith, 1983.

Simon, Herbert A, “The Structure of Ill Structured Problems” *Artificial Intelligence* Vol. 4, n°3-4, p. 181-201, 1973.

***Structure des connaissances***

Bielaczyc, Katerine et Allan Collins, “Learning Communities in Classrooms: Advancing Knowledge For a Lifetime”, *NASSP Bulletin*, Vol. 83, n°604, p. 4-10, 1999.

Collins, Harry, *Tacit and Explicit Knowledge*, University of Chicago Press, 2010.

Gourlay, Stephen, "The SECI model of knowledge creation: some empirical shortcomings", 4th European Conference on Knowledge Management, Oxford, England, 18-19 Sep 2003.

Kruchten, Philippe, Patricia Lago, Hans van Vliet, et Timo Wolf. “Building up and Exploiting Architectural Knowledge” *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, 2005.

Nonaka, Ikujiro, Ryoko Toyama et Noboru Konno, “SECI, Ba and Leadership: a Unified Model of Dynamic Knowledge Creation”, *Long Range Planning*, Vol. 33, n°1, p. 5-34, 2000.

Pickstone, John, *Ways of Knowing : A New History of Science, Technology and Medicine*, University of Chicago Press, 2004.

Polanyi, Michael, *Personal Knowledge : Towards a post-critical philosophy,* University of Chicago Press, 1958.

Polanyi, Michael, *The Tacit Dimension*, Routledge, 1966.

Sennett, Richard, *The Craftsman*, Yale University Press, 2008.

Uluoglu, Belkıs, “Design knowledge communicated in studio critiques”, *Design Studies*, Volume 21, n°1, p. 33-58, 2000.

***Méthodologie***

Akrich, Madeleine, « La construction d'un système socio-technique. Esquisse pour une anthropologie des techniques », *Anthropologie et Sociétés*, Vol. 13, n°2, 1989.

Akrich, Madeleine, Michel Callon et Bruno Latour (éditeurs), *Sociologie de la traduction : textes fondateurs*, Paris : Mines ParisTech, les Presses, « Sciences sociales », 2006.

Coenen-Huther, Jacques, “Analyse de réseaux et sociologie générale”, *FLUX Cahiers scientifiques internationaux Réseaux et Territoires*, Vol. 13-14, p. 33-40, 1993.

Delpu Pierre-Marie, « La prosopographie, une ressource pour l’histoire sociale », *Hypothèses*, Vol. 18, n°1, p. 263-274, 2015.

Fors, Hjalmar, Lawrence M. Principe et H. Otto Sibum, “From the Library to the Laboratory and Back Again:Experiment as a Tool for Historians of Science”, *Ambix*, Vol. 63, n°2, p. 85-97, 2016.

Weber, Florence et Stéphane Beau, *Guide de l'enquête de terrain: produire et analyser des données ethnographiques*, La Découverte, 1997.

***Autres***

Deleuze,Gilles, *Le Pli. Leibniz et le Baroque*, Editions de Minuit, 1988.

Descartes, René, *Les principes de la philosophie*, Louis Elzevir, Amsterdam, 1644.

Feyerabend, Paul, *Contre la méthode, Esquisse d'une théorie anarchiste de la connaissance*, Paris, Le Seuil, 1979.

Feynman, Richard, « Cargo Cult Science », Intervention à Caltech, 1974, http://calteches.library.caltech.edu/51/2/CargoCult.pdf, consulté le 12 Novembre 2021.

Kuhn, Thomas (traduit de l'anglais par Laure Meyer), *La structure des révolutions scientifiques*, Paris : Flammarion, 2008 (première édition 1962).

Latour, Bruno, *La Science en action*, traduit de l'anglais par Michel Biezunski ; texte révisé par l'auteur, Paris : La Découverte, 1989.

Lynch, Kevin, *The image of a city*, The MIT Press, 1960.

Simondon, Gilbert, *Du mode d'existence des objets techniques*, Paris : Aubier, 1958.

Weber, Max, " La profession et la vocation de savant", dans Weber, Max, *Le savant et le politique*, Paris: Union Générale d'Éditions, 1963 (texte original rédigé en 1917).

## ANNEXE 1. PRATICIENS INTERROGÉS

*Felix Agid*
Architecte et enseignant-chercheur, Félix Agid a étudié à l'ENSA Paris-Belleville (licence d'architecture), à l'École d'architecture de la ville & des territoires Paris-Est (DPLG) et à l'EHESS ( (master en histoire et philosophie des sciences). Il a cofondé EZCT Architecture & Design Research en 2000 et XtreeE en 2015. Il enseigne à l'École supérieure d'art et de design - Tours Angers Le Mans, où il a cofondé en 2012 le programme de recherche Synthetic, et en 2018 le programme Design computationnel et mécatronique.

*Phil Ayres*
Architecte et enseignant-chercheur, Phil Ayres a étudié à l'Université de Greenwich (licence d'architecture), à la Bartlett School of Architecture (master d'architecture) et à l'école d'architecture d'Aarhus (doctorat en architecture). Cofondateur de l'agence d'architecture expérimentale Sixteen*(makers) en 1994, il a enseigné à la Bartlett School of Architecture entre 1998 et 2009, avant de rejoindre l'équipe de recherche du Centre for Information Technology and Architecture de l'École des Beaux-Arts de Copenhague.

*Joshua Bard*
Architecte et enseignant-chercheur, Joshua Bard a étudié au Taubman College of Architecture and Planning à l'Université du Michigan (licence d'arts, master d'architecture). Il a occupé différents postes au sein du même établissement entre 2008 et 2012, avant de devenir professeur associé à la Carnegie Mellon University. Il est également le cofondateur de l'agence d'architecture expérimentale Archolab, ouverte en 2008.

*Olivier Baverel (conversations)*
Ingénieur et enseignant-chercheur, Olivier Baverel est docteur en génie civil (Structural Engineering, Université du Surrey) et habilité à diriger des recherches (Institut Polytechnique de Grenoble). Membre de l'équipe Matériaux et Structures Architecturés du laboratoire Navier de l'École Nationale des Ponts et Chaussées, il enseigne également à l'École Nationale Supérieure d'Architecture de Grenoble.

*Jane Burry*
Architecte et enseignante-chercheuse, Jane Burry a d'abord été professeure à l'Institut royal de technologie de Melbourne, où elle a dirigé le Spatial Information Architecture Laboratory (SIAL). Elle a également fait partie pendant plusieurs années de l'équipe chargée de la poursuite de la construction de la Sagrada Familia d'Antoni Gaudi. Elle est actuellement doyenne de la faculté d'arts et de design de l'Université technique de Swinburne.

*Dana Cupkova*
Ingénieure-architecte et enseignante-chercheuse, Dana Cupkova est diplômée de l'Université technique de Bratislava (diplôme d'ingénieur-architecte), de l'Université de Californie à Los Angeles (master d'architecture) et de l'Académie des Beaux-Arts de Vienne (doctorat). Après avoir enseigné à l'Université de Cornell, elle est maintenant professeur associée à Carnegie Mellon. Elle a également

exercé dans plusieurs agences d'architecture aux Etats-Unis, en Europe et en Asie, et cofondé l'agence DCm-STUDIO ainsi que le collectif EPIPHYTE Lab.

*Klaas de Rycke*
Architecte-ingénieur, Klaas de Rycke a été formé à l'Université de Gand et à l'Escuela Tecnica Superior de Arquitectura de Madrid. Ingénieur au sein du bureau d'études Bollinger + Grohmann depuis 2003, il est le cofondateur et le gérant de la filiale parisienne de l'entreprise. Il est également maître de conférence à l'École Nationale Supérieure d'Architecture de Versailles.

*Jelle Feringa (conversations)*
Architecte, Jelle Feringa a étudié à l'école d'arts hollandaise ArtEZ ainsi qu'à l'Académie Gerrit Rietveld. Cofondateur de l'agence EZCT Architecture & Design Research et des entreprises de robotique architecturale Odico et Aectual, il a également enseigné à la Bartlett School of Architecture.

*Per Galle*
Architecte et enseignant-chercheur, Per Galle a étudié à l'Académie des Beaux-Arts de Copenhague et à l'Université de Copenhague (doctorat). Il a ensuite enseigné à l'Université de Copenhague et à l'Université technique du Danemark avant de revenir à l'Académie des Beaux-Arts de Copenhague, où il est actuellement professeur.

*Nicole Gardner*
Architecte et enseignante-chercheuse, Nicole Gardner a exercé au sein de plusieurs agences et bureaux d'études en Angleterre et en Australie. Après avoir enseigné à l'Université d'Adelaide, à l'Université d'Australie du Sud et à l'Université technique de Sydney, elle enseigne désormais à l'Université de Nouvelle Galle du Sud.

*Fabio Gramazio*
Architecte et enseignant-chercheur, Fabio Gramazio rencontre Matthias Kohler à l'ETH Zürich, dont ils sont diplômés en 1996. Ils y enseignent ensemble dès l'année qui suit. En 2000, ils fondent l'agence et groupe de recherche Gramazio & Kohler, spécialisée dans la robotique architecturale et ses applications. Fabio Gramazio est également assistant professeur à l'ETH Zürich.

*Matthias Hank Haeusler*
Architecte et enseignant-chercheur, Matthias Hank Haeusler a étudié à Stuttgart, Tokyo et Delft, avant d'exercer plusieurs années comme architecte en Allemagne, en Australie et aux États-Unis. Après sa thèse de doctorat, effectuée au laboratoire SIAL de l'Institut royal de technologie de Melbourne, il a été directeur du Media Architecture Institute Vienna / Sydney, professeur associé à l'Université de Nouvelle Galle du Sud en Australie et Professeur à l'Institut d'Innovation pour les Arts Visuels de Pékin.

*Someen Hahm*
Architecte et enseignante, Someen Hahm a étudié à l'Université Tsinghua à Pékin (licence d'architecture) et à l'Architectural Association à Londres (master d'architecture). Elle a ensuite exercé en tant qu'architecte au sein de l'agence Zaha Hadid Architects, avant de fonder sa propre agence, Soomeen Hahm Design Ltd. En parallèle, elle a enseigné à l'Architectural Association et à la Bartlett, et est maintenant en poste à Sci_Arc.

*Daniel Hambleton*
Mathématicien et développeur, Daniel Hambleton est diplômé de l'Université McMaster (licence de mathématiques) et de l'Université McGill (master de géométrie). Après avoir exercé chez Arup et Halcrow Yolles comme spécialiste des géométries complexes, il a fondé le cabinet de conseil MESH Consultants Inc.

*Gwyllim Jahn*
Architecte et développeur, Gwyllim Jahn a été formé à l'Université d'Australie Occidentale (licence de conception environnementale) et à l'Institut royal de technologie de Melbourne (master d'architecture). Après avoir brièvement exercé comme architecte au sein de l'agence LAB, il a créé en 2018 Fologram, un outil de réalité augmentée prévu pour l'aide à la construction. Il a également enseigné à l'Institut royal de technologie de Melbourne et à l'Université de Melbourne.

*George L. Legendre*
Architecte et enseignant-chercheur, George L. Legendre a étudié à la Graduate School of Design de l'Université de Harvard. Il a ensuite enseigné au sein du même établissement, puis à l'ETH Zürich, à l'Université de Princeton et à l'Architectural Association. Enfin, il est revenu à la Graduate School of Design de l'Université de Harvard, où il est désormais professeur associé. Il est également le fondateur de l'agence d'architecture expérimentale IJP Corporation.

*Peter Macapia*
Architecte et enseignant-chercheur, Peter Macapia a étudié à la Rhode Island School of Design (licence), à l'Université de Harvard (master) et à l'Université de Columbia (doctorat). D'abord enseignant au GSAPP de l'Université de Columbia, il est ensuite devenu professeur associé à l'Institut Pratt de New York. Il est également le fondateur de l'agence d'architecture expérimentale labDORA.

*Iain Maxwell*
Architecte et enseignant-chercheur, Iain Maxwell est titulaire d'une licence d'architecture de l'Université de Canberra et d'un master d'architecture de l'Architectural Association. Il a exercé pour le compte de plusieurs agences en Australie et en Grande-Bretagne, notamment Future Systems, Populous et Grimshaw Architect s. En 2006, il a cofondé l'agence d'architecture expérimentale supermanoeuvre. Il a enseigné à l'Université technique de Sydney entre 2012 et 2019, avant d'être nommé professeur associé à l'Université de Canberra.

*Philippe Morel (conversations)*
Architecte et enseignant-chercheur, Philippe Morel est cofondateur d'EZCT Architecture & Design Research et maître-assistant titulaire à l'École nationale supérieure d'architecture Paris-Malaquais, où il a créé avec Christian Girard le département Digital Knowledge. Il enseigne également à la Bartlett School of Architecture, après avoir enseigné à l'Architectural Association et au Berlage Institute de Rotterdam.

*Paul Nicholas (conversations)*
Architecte et enseignant-chercheur, Paul Nicholas a étudié à l'Institut royal de technologie de Melbourne (licence, master et doctorat d'architecture). Après avoir exercé comme architecte et comme spécialiste du recours au numérique au sein des bureaux d'études AECOM et Arup, il a cofondé l'agence d'architecture expérimentale Mesne. Il est désormais professeur associé au Centre for Information Technology and Architecture de l'École des Beaux-Arts de Copenhague.

*Igor Pantic*
Architecte et enseignant, Igor Pantic est diplômé de l'Université de Belgrade (master d'architecture) et du Design Research Lab de l'Architectural Association (master spécialisé). Il a d'abord été architecte au sein de l'agence Zaha Hadid Architecture avant de fonder l'agence qui porte son nom, et enseigne à la Bartlett School of Architecture.

*Dagmar Reinhardt*
Architecte et enseignante-chercheuse, Dagmar Reinhardt a étudié à l'Université technique de Hanovre (master d'architecture) et à l'Université de Sydney (doctorat en architecture). Elle a ensuite exercé au sein de plusieurs agences d'architecture, fondées par elle-même en équipe avec différents associés : frankfurterraum, twentytwentysix et reinhardt_jung. Elle a enseigné à l'Université des sciences appliquées ainsi qu'à la Städelschule de Francfort et à l'Université de Sydney.

*Andrea Rossi (conversations)*
Architecte, développeur et enseignant-chercheur, Andrea Rossi est diplômé de l'École Polytechnique de Milan (licence d'ingénierie) et de la Hochschule Anhalt (master d'architecture). Après avoir exercé comme spécialiste en robotique et programmation dans les agences d'architecture Coop Himmelb(l)au, Indexlab et Magnus Kaminiarz il a été assistant de recherche à l'ETH Zürich et à la Digital Design Unit de Darmstadt. Il est actuellement collaborateur scientifique au laboratoire EDEK de l'Université de Kassel.

*Jose Sanchez*
Architecte, développeur et enseignant, Jose Sanchez est diplômé de l'Architectural Association (master d'architecture). Il a longtemps exercé comme programmeur et architecte au sein de l'agence biothing. Il a ensuite cofondé l'entreprise Bloom et le collectif Plethora Project. Il a enseigné à l'Architectural Association et à la Bartlett School of Architecture, et est actuellement assistant-professeur à l'Université de Californie du Sud.

*Fabian Scheurer*
Informaticien et enseignant-chercheur, Fabian Scheurer est diplômé de l'Université technique de Munich (master en informatique). Il a d'abord travaillé en qualité d'assistant de recherche à l'Université technique de Munich, puis comme consultant informatique à Zürich, et enfin collaborateur scientifique au sein du groupe de recherche CAAD de l'ETH Zürich. En 2007, il a cofondé le bureau d'étude germano-suisse designtoproduction. Il a également enseigné à l'Architectural Association et à l'IAAC de Barcelone.

*Tim Schork*
Architecte et enseignant-chercheur, Tim Schork a effectué son doctorat au sein de l'Institut royal de technologie de Melbourne. Il a ensuite enseigné à l'Université de Kassel et à l'Université de Melbourne, et est désormais professeur associé à l'Université technique de Sydney, où il a cofondé le Transformative Technologies Lab.

*Bob Sheil*
Architecte et enseignant-chercheur, Bob Sheil a été formé à la Bartlett School of Architecture, ou il a ensuite enseigné à tous les niveaux, avant d'être nommé directeur de l'école en 2014. Il a également cofondé l'agence d'architecture expérimentale Sixteen*(makers) en 1994.

*Roland Snooks*
Architecte et enseignant-chercheur, Roland Snooks est diplômé de l'Université de Canberra (licence d'architecture), de l'Université de Columbia (master d'architecture) et de l'Institut royal de technologie de Melbourne (doctorat). Il a cofondé l'agence d'architecture expérimentale kokkugia et dirige le Studio Roland Snooks. Chargé de cours à l'Institut royal de technologie de Melbourne, il y dirige également le laboratoire de robotique architecturale.

## ANNEXE 2. CORPUS D'EXPOSITIONS EXAMINÉS

***L'architecture au-delà des formes***
Commissariat : Philippe Morel et Marie-Ange Brayer
Lieu : Maison de l'architecture et de la Ville, Marseille
Date : 2007

***architectures expérimentales***
Commissaire : Marie-Ange Brayer
Lieu : FRAC Centre, Orléans
Date : 2012
Référence du catalogue : Brayer, Marie-Ange (éditrice), *Architecture expérimentales 1950-2012*, Orléans : Éditions HYX : FRAC Centre, 2013.

***Architectures non-standard***
Commissaire : Mnam/Cci, Frédéric Migayrou
Lieu ; Centre George Pompidou, Paris
Date : 2003
Référence du catalogue : Migayrou, Frédéric (éditeur), *Architectures Non-Standard*, Paris Centre Pompidou, 2003.

***Blobjects & Beyond: The New Fluidity in Design***
Commissaire : Steven Holt et Mara Holt Skov
Lieu : San Jose Museum of Art
Date : 2005
Référence du catalogue : Holt, Steven et Mara Holt Skov (éditeurs), *Blobjects & Beyond: The New Fluidity in Design*, San Francisco : Chronicle Books, 2005.

***Coder le Monde***
Commissaire : Marie-Ange Brayer et Frédéric Migayrou
Lieu : Centre George Pompidou, Paris
Date : 2018
Référence du catalogue : Migayrou, Frédéric (éditeur), *Coder le monde : mutations, créations*, Paris : Éditions du Centre Pompidou, 2018.

***Future city : experiment and utopia in architecture***
Commissaire : Alison Jane
Lieu : Barbican Centre, Londres
Date : 2006
Référence du catalogue : Alison, Jane (éditrice), *Future city : experiment and utopia in architecture*, Londres : Thames & Hudson, 2006.

***The Gen[H]ome Project***
Commissaire : Kimberli Meyer, avec Open Source Architecture (Eran Neuman, Aaron Sprecher, Chandler Ahren)

Lieu : M.A.K, Los Angeles
Date : 2006-2007
Référence du catalogue : Noever Peter, Kimberli Meyer et Open Source Architecture (éditeurs), *The Gen(h)ome Project*, Los Angeles : MAK, 2006.

***Naturaliser l'architecture***
Commissaire : Marie-Ange Brayer, Frédéric Migayrou
Lieu : FRAC-Centre, Orléans
Date : 2013
Référence du catalogue : Brayer, Marie-Ange et Frédéric Migayrou (éditeurs), *Naturaliser l'architecture : Archilab 9e édition*, Orléans : HYX, 2013.

***Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch***
Commissaire : Elias Guenoun
Lieu : La Maison Rouge, Paris
Date : 2007
Référence du catalogue : Guenoun, Elias (éditeur), *Pavillon Seroussi: Biothing, DORA, EZCT Architecture and Design Research, IJP George Legendre, Gramazio and Kohler, Xefirotarch*, Editions HYX, 2007.

***Performalism: Form and Performance in Digital Architecture***
Commissaire : Eran Neumann, Yasha Grobman, Ahuva Israel
Lieu : Tel Aviv Museum of Art, Tel Aviv
Date : 2008
Référence du catalogue
Jacob Grobman, Yasha et Eran Neuman (éditeurs), *Performalism: Form and Performance in Digital Architecture*, Tel Aviv Museum of Art, 2008.

***scriptedbypurpose***
Commissaire : Marc Fornes
Lieu : FUEL Gallery, Philadelphia
Date : 2007
Référence du catalogue : catalogue en ligne https://scriptedbypurpose.wordpress.com/ (consulté le 14 Décembre 2021).

## ANNEXE 3. PROJETS CITÉS

100 Mount Street, Skidmore, Owings and Merrill, 2012-2019
1000 habitations à cour, Tianherenjia (Chine), Hubert & Roy, 2002-2003
Aalto University, kokkugia, 2012
Abribus pour l'exposition universelle 2000 à Hanovre, Gehry LLP, 1991-1994
Abu Dhabi Performing Art Center, Zaha Hadid Architects, 2007
Aéroport de Riga, kokkugia, 2010
AI & Architecture, Stanislas Chaillou, 2019
Al Hamra Tower, Skidmore, Owings and Merrill, 2006-2011
Alkmaar Housing, The Living / Autodesk, 2018
AlloBioExo, Marcos Novak, 2001
ARCHIMODOS, Christos Yessios, 1982-1985
ARCHIT, Architecture Machine Group
Architectural Genomics, Skidmore, Owings & Merrill, 2008
Armadillo Vault, Block Research Group (ETHZ), 2016
Artificial Zurichness, Media and Design Laboratory (EPFL), 2020
Aspen Movie Map, Architecture Machine Group, 1978
Associative Component Structure, EmTech (AA), 2006
Autodesk AU Las Vegas 2017 Exhibit Hall, The Living / Autodesk, 2017
Autotectonics : The choice of choice, John Frazer, 1969
Babi Yar Memorial, kokkugia, 2010-2012
Bibliothèque d'Helsinki, Robert Stuart-Smith, 2012
Bibliothèque Nationale Tchèque, OCEAN North, 2006
Bifid, biothing, 2005
Blob Wall, Greg Lynn FORM, 2005
Bloom House, Greg Lynn FORM, 2010
BMW Welt, Coop Himmelb(l)au, 2001-2007
Caserne Vitra, Zaha Hadid Architects, 1990-1993
Cayan Tower, Skidmore, Owings and Merrill, 2006-2013
CCTV Headquarters, OMA, 2002-2012
Centre Frans Masereel, Hideyuki Nakayama Architects, 2019
Centre Heydar Aliev, Zaha Hadid Architects, 2007-2012
Centre Pompidou Metz, Shigeru Ban Architects, 2003-2010
Chai Gantebein, Gramazio / Kohler, 2006
Cliff House, kokkugia, 2012

Combinatorial Tower, labDORA, 2012

Whitehall Reconstruction, Leslie Martin, 1963-1971

DFAB House, ETH Zürich et NCCR, 2018-2020

Dirksen variations, James Park et Athanassios Economou, 2015

Dynaspace Use Case 1, Shepley Bulfinch, 2017

Dynaspace Use Case 3, MT Højgaard, 2017

EAVE, Antony Radford et J.R. Mitchell, 1987

El Peix d'Or, Gehry LLP, 1992

ElbPhilharmonie, Herzog & De Meuron, 2001-2016

Embryological House, Greg Lynn FORM, 1997-2001

Flat Writer, Yona Friedman et Architecture Machine Group, 1967

Flight Assembled Architecture, Gramazio / Kohler, 2011

FLOP 1 et 2, Per Galle, 1981-1990

Fold Pavilion, Roland Snooks, 2017

Fondation Louis Vuitton, Gehry LLP, 2014

Fondation Luma, Gehry LLP, 2013-2021

Freshwater Pavilion, NOX, 1997

Fun Palace, Cedric Price et Gordon Pask, 1961

Gatemaker, Christopher Alexander et Greg Bryant, 1996

General Space Planner, Charles Eastman, 1971

Generator, Cedric Price, Gordon Pask, John Frazer et Julia Frazer, 1976-1979

GENR8, Emergent Design Group (MIT), 2008

Genware, biothing, 2001-2006

Grande Cour du British Museum, Foster + Partners, 1994-2000

Guoco Tower, Skidmore, Owings and Merrill, 2012-2018

Haesley Nine Bridges Golf Club House, Shigeru Ban Architects, 2010

HIDECS, Christopher Alexander et Allen Bernholtz, 1963

Hotel Morpheus, Zaha Hadid Architects, 2013-2018

HUNT, Architecture Machine Group, 1970s-1980s

ILCON, Christos Yessios, 1978

Isovists, Michael Benedikt et al., 1979

IMAGE, Architecture Machine Group, 1970s-1980s

Infobox Gare de Vienne, Michael Wallraff, 2011

Kartal Pendik Masterplan, Zaha Hadid Architects, 2006

Kazakhstan Symbol, Roland Snooks, 2013

Kilden Performing Arts Centre, ALA Architects et SMS Architekter, 2004-2012

Lakehouse Patagonia, Archiglobe, 2007

Layer Stalker, Paul Poinet, 2019

Learning Center EPFL, Sanaa, 2010

Living Morphologies, supermanoeuvre, 2006

Maison à Santa Monica, Gehry LLP, 1977

MARS, kokkugia, 2003-2004

Material Potency, biothing et Skidmore, Owings and Merrill, 2005

Media Room, Architecture Machine Group, 1976

Mesh Mould et In Situ Fabricator, Gramazio / Kohler, 2017

Mesonic Fabrics, biothing, 2007-2009

Morning Line, Aranda/Lasch, 2008-2013

Musée Guggenheim de Bilbao, Gehry LLP, 1997

Musée d'Ordrupgaard, Zaha Hadid Architects, 2001-2005

Mycellium Brick, The Living, 2014

National Art Museum of China, kokkugia, 2010-2012

Noeud rotoformé, Samim Mehdizadeh / Digital Design Unit, 2018

One World Trade Center, Skidmore, Owings and Merrill, 2007-2014

Opéra de Sydney, Jorn Utzon, 1955-1973

Orbita, biothing, 2009

Oregon Experiment, Christopher Alexander et al., 1975

Palladian Grammar, George Stiny et William Mitchell, 1978

Panneaux de bois, Objectile, 1991-1998

Parametric Semiology, Patrik Schumacher et al., 2013-2020

Pavillon Bloomberg, Akihisa Hirata, 2011

Pavillon CIAB, Zaha Hadid Architects, 2013

Pavillon Seroussi, biothing, 2006-2007

Pavillon Seroussi, EZCT Architecture & Design Research, 2006-2007

Pavillon Seroussi, Gramazio / Kohler, 2006-2007

Pavillon Seroussi, ijp corporation, 2006-2007

Pavillon Seroussi, labDORA, 2006-2007

Pavillon Seroussi, Xefirotarch, 2006-2007

Pavillon pour la Serpentine Gallery, Toyo Ito, 2002

Pavillon Turing, biothing, 2012

Philharmonie de Paris, Atelier Jean Nouvel, 2007-2015

Piazza Garibaldi, Dominique Perrault Architecture, 2004-2019

The Pinnacle, KPF, 2006-2020

Possible Palladian Villas, Richard Freedman et George L. Heersey, 1992

Really Universal Computer Aided Production System (RUCAPS), Gollins Melvin Ward Partnership, 1977

Recycled Toy Furniture, Greg Lynn FORM, 2008

Salle d'escalade, Robert Stuart-Smith, 2016

Saltwater Pavilion, ONL, 1997

School of Management and Design Essen, Sanaa, 2006

Scénographie de l'exposition Architectures Non Standard, EZCT Architecture & Design Research, 2003

Shape Grammars, George Stiny et James Dips, 1972

SOLAREXPERT, Michael A. Rosenman, John S. Gero et Richard Coyne, 1987

Space Syntax, Bill Hillier et al., 1970s-1980s

Spatial Data Management System, Architecture Machine Group, 1976

SQUINT, Architecture Machine Group, 1970s-1980s

Stade Olympique de Munich, Günter Behnisch et Frei Otto, 1972

Stranded Sear Tower, Greg Lynn FORM, 1992

StructureFIT, Digital Structures Group (MIT), 2014

Studies on Optimization: Computational Chair Design, EZCT Architecture & Design Research, 2004

Tailorcrete, designtoproduction et al., 2009-2013

TEKTON, Christos Yessios, 1982

Terminal multimodal d'Anaheim, HOK, 2014

The Tote, Serie Architects, 2011

Tour Leeza, Zaha Hadid Architects, 2015-2019

Trabeculae / Protosynthesis, supermanoeuvre, 2008

Tremplin de Bergisel, Zaha Hadid Architects, 1999-2002

Universal Constructor, John Frazer et Morphogenesis Unit, 1990

Unprintable Forms, Marjan Colletti et al., 2019

URBAN 2, Architecture Machine Group, 1969-1970

URBAN 5, Architecture Machine Group, 1973

Vagues d'Henderson, IJP George Legendre, 2004-2008

Wall Variations of the Smith House, Edward Din et Athanassios Economou, 2007

Walt Disney Concert Hall, Gehry LLP, 2003

Waterloo Station, YRM, 1994

Weisman Art Center, Gehry LLP, 1993

White Magnolia Plaza, Skidmore, Owings and Merrill, 2008-2018

YONA, Yona Friedman et Architecture Machine Group, 1975-1977

# TOWARDS AN HISTORY OF COMPUTATIONAL TOOLS IN AUTOMATED ARCHITECTURAL DESIGN

*The Seroussi Pavilion Competition as a Case Study*

NADJA GAUDILLIERE
[1]*Laboratoire GSA, Ecole Nationale Supérieure d'Architecture Paris-Malaquais, 75006 Paris, France*
[1]*nadja.gaudilliere@gmail.com*

**Abstract.** The present research proposes a method to analyse computational tools at the architect's disposal and the potential technical bias they induce in architectural design. Six case studies will be used as a demonstration of the method's ability to highlight those biases and how architects and designers manipulate those tools to translate their architectural expertise into algorithmic design. Those case studies are the six answers to the Seroussi Pavilion competition, organized in 2007 by Natalie Seroussi, a Parisian gallery owner. Having a keen interest into computational design, she invited six architectural practices specializing in this field. As the six case studies answer the same design brief, it represents a particularly suitable opportunity to analyse the intricate relationship between architectural constraints, their translation into computational data and instructions and the programming tools used to do so. Through the analysis of four different aspects of the project - algorithmic tools/method, computational set-up, organizational chart and architectural design - several issues of the computational turn in architecture are discussed.

**Keywords.** Digital heritage; computational design tools; architectural constraints; programming-based spatial design; Seroussi pavilion competition.

## 1. Introduction: toward an history of computational spatial design tools

### 1.1. ALGORITHMIC DESIGN AS A TOOL FOR SPACE DESIGN

Throughout the last 50 years or almost, architects, urbanists, structural engineers and designers have been experimenting with the computer's ability, given a proper set of instructions, to generate shapes and other contents on its own. But beyond the fascinating ability of computers to create geometries from nothing but a few written signs, the objective for architects has become generating a relevant spatial solution to a given architectural problem using computational design tools. To achieve this, the instrumental issue is the translation of architectural constraints into an operable set of data and instructions, or in other words, into an operable

*Intelligent & Informed, Proceedings of the 24th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2019*, Volume 2, 581-590. 

algorithm. But the structure of algorithms, alongside the coding languages used to create them, are fundamentally different of any other tool previously used by architects for space design. Therefore, architectural constraints such as program, context, structural loads and other are dealt with in a new approach, contributing to the renewal seen in architectural design as recourse to digital design methods spreads out. In light of these changes, understanding to what extent architects and designers resort to algorithms and computational design appears to be a major issue to tackle.

In order to tackle this issue, a key element to analyse is the computational design tool itself. If at the beginning of computational experimentation, architects shaped their tools for spatial design on their own, developing specific algorithms from scratch or almost for each project, in a more recent past the emergence of software such as Grasshopper or Processing has widely simplified the recourse to computational design methods. This has been followed by a blossoming of ready-made algorithms available to architects, granting access to elaborate mathematical and geometrical tools in an unprecedented way to designers untrained in these domains. This access has however been accompanied by a less sharp control of those tools, compared to those initially shaped by digital architects themselves. Lesser control of the programming tool induces more technical biases to be aware of in the translation of architectural constraints into data and algorithmic instructions. This situation calls for an analysis of what programming tools offer regarding architectural constraints translation and technical biases, a question scarcely discussed in the field of computational design.

### 1.2. THE COMPUTATIONAL FIELD IN ARCHITECTURE

Although the use of computational tools has in the last few years spread to a greater number of architects and designers, seminal research in the field has been conducted by a much smaller group of digital architects and designers. This group forms the core of a specific niche that has developed computational design through research projects, teaching and architectural practice for several decades. The variety of experimentations conducted within this group resulted over the years in very diverse architectural proposals, yet these designers were brought together by a common use of digital design tools, including programming tools, therefore earning the appellation of computational movement to the group (Menges & Alquist 2011).

The computational field in architecture is characterized not only by the common resort to specific design tools, but also by conditions that enabled research on this topic to blossom. Most of its architectural production consists in paper projects, prototypes, pavilions, produced in an academic environment or for specific exhibitions. The computational field therefore organizes itself around a series of individuals, of research units and architecture schools as well as around events such as exhibitions and conferences. This academic set-up has enabled the computational field to blossom away from most of the usual constraints of the construction industry. The consequence of this is an architectural production exploring various issues raised by the development of digital technologies, an exploration benefiting from much more freedom given this distance to industrial

constraints.

This freedom and explorations make the architectural production of this field a prefiguration of the global computational turn in architectural design currently happening. The computational field as it has developed in the last 50 years and its production forms therefore an ideal set of designers and projects to study programming-based spatial design and the translation of architectural constraints in computational tools.

### 1.3. COMPUTATIONAL SPATIAL DESIGN TOOLS AS A RESEARCH TOPIC

The present research proposes to take a closer look at the digital tools in use in those experimentations and to turn them into a full research topic, in order to analyse the phenomenon through a new angle. If numerous archaeologies of the digital in architecture already exist, including the eponymous book by Greg Lynn (Lynn 2013), they consist in their majority of an assembly of emblematic projects presentations and texts by the architects and designers of the computational field. These project reviews rarely contain more than scarce details on the technical computational aspects and only few designers develop in their written production comments on their use of technical tools. Furthermore, most of the written production focuses on theoretical aspects and does not consider sociological data on the computational field in architecture. Finally, almost none of these productions offers a detailed overview of the actual use of algorithmic tools in architectural design. Only a few studies of the development of specific digital tools exist, such as an history of Form*Z (Serraino 2002), as well as a few very recent researches seemingly starting to open this field of research (Laurent et al. 2018).

The present paper proposes a method to analyse computational design tools at the architect's disposal, their development throughout the years and their technical peculiarities, including possible bias they could induce in architectural constraint translation. Six case studies will be used as a demonstration of the method's ability to highlight the role played by these algorithmic tools. Those case studies are the six answers to the Seroussi Pavilion Competition (SPC), organized in 2006-2007 by Natalie Seroussi, a Parisian gallery owner. Having a keen interest into computational design, she invited the following architectural practices to submit a design for her pavilion: biothing, DORA, EZCT Architecture & Design Research, IJP George L. Legendre, Gramazio & Kohler and Xefirotarch. As the six case studies answer the same design brief, it represents a particularly suitable opportunity to analyse the intricate relationship between architectural constraints, their translation into computational data and instructions and the programming tools used to do so.

The main objective of this research is to propose an analysis method and to discuss algorithmic design tools and the notion of technical bias in the SPC, but also to question the results of this study as part of larger considerations on the use of algorithmic tools for architectural design in recent history.

## 2. The Seroussi Pavilion Competition

### 2.1. CONTEXT AND BRIEF

The Seroussi Pavilion Competition's aim was to build a pavilion that would be housing part of Natalie Seroussi's art collection in dedicated exhibition spaces as well as living spaces. In addition to this double architectural program, the other peculiarity of the competition was the project's site, a particularly strong and specific context: the pavilion was to be built in the garden of André Bloc's home, property of Natalie Seroussi. The villa, located in Meudon, was designed and built in 1949 by the architect and engineer for himself, and the garden hosts as well two of Bloc's *sculpture-habitacles*, added in 1964-1966. A specific area of the estate was selected for the pavilion and is mentioned in the competition brief, along with comments on the specificities of the site, to be considered by the participating architectural practices. The brief also includes a detailed program (Guenoun 2007):

*"The project will include an interior and an exterior space. The total inhabitable surface area of the project will be approximately 350 sqm. The interior will comprise 2 bedrooms, each with walk in closets; 1 living/dining room; 1 kitchen; 1 bathroom; 1 office (optional - can be included in one of the bedrooms); 1 toilet (outside the basement); 1 basement set up for purpose of video projections, 1 storage space for artworks between 20 and 30 sqm. Each of these spaces will, to the extent possible, include storage."*

Two proposals, the first by EZCT Design & Architecture Research and the second by IJP, were selected as winners by the jury, chaired by Claude Parent. The winning proposals were not built as it was initially planned, but shortly after the competition, from June to September 2007, an exhibit was organised at the Parisian exhibition space La Maison Rouge, showcasing the six proposals.

Taking place in 2006-2007 in France, the competition was organised during a specific momentum of the computational turn in architecture. It accompanied a series of instrumental exhibitions, such as the *Architectures Non Standard* exhibition at Centre Pompidou (Migayrou 2003), the *Architecture au-delà des formes* ("Architecture Beyond Form") exhibition at the Maison de l'Architecture et de la Ville in Marseilles (Morel 2007) or, the same year, the *Scripted by Purpose* exhibition at the FUEL gallery in Philadelphia (Fornes 2207). In the previous years, starting in 2000, numerous practices were founded that went on to be major designers in the computational field, including those participating in the SPC. Those elements all hint to the fact that the competition took place at a moment where the designers taking part in the computational field at the time were building a community around these issues.

### 2.2. AN IDEAL CASE STUDY

The interest of the Seroussi Pavilion Competition as a case study, beyond it happening in a key momentum, also lies in the fact that the six participating practices are emblematic to the computational turn in architecture. Moreover, they are emblematic of a specific generation. The term "generation" will be used in this paper to designate successive groups of individuals taking part in the

computational turn. Those generations are characterized by their use of tools, as the use of algorithmic design tools in architectural design can be described by characterising the relation of designers and practices to those tools throughout the years. Although most of the individuals of a generation are younger than those of the previous generation and although a relationship of knowledge transmission exists between most of the individuals from two successive generations, some overlapping still exists. Five specific groups or generations can be distinguished.

The first generation (G0) is formed by isolated projects and experimentations, on primitive computational set-ups, such as the Generator project by Frazer & Price (Landau 1985). The second (G1) regroups individuals that launched the first paperless studios and similar academic research groups. Therefore, this generation constitutes an embryo of network of people experimenting with existing algorithmic tools and methods as well as theorising a new paradigm based on possible production conditions. The third (G2) is formed by designers having been taught by individuals from G1 and simultaneously exposed to various computational tools developed in other fields, such as analysis tools for engineering purposes. Their use of computational tools is characterized by an earlier and further training on programming tools available at the time. The enlargement of the network as well as the technical progress of modelling, simulation and analysis tools at the time results in a fine-tuned capacity to build their own tools and to experiment with the possibilities offered by computational design. The fourth generation (G3) displays a similar use of tools. But since it benefitted from the experience of G2 as well as from new technological progress, this generation has had the possibility to develop tools exploring computational design further, including handling more data and data of various kinds. Finally, the fifth generation (G4) corresponds to the generalisation of the recourse to programming tools, using the tools developed by G2 and G3 providing a simplified interface and therefore easier access to sophisticated algorithms. Technical bias is stronger in this generation given that the recourse to pre-existing tools partly erases the need for mastering scripting.

The Seroussi Pavilion Competition regroups a series of practices all part of the G2 group and is therefore emblematic of this generation. Furthermore, the competition represents an ideal case study since its brief offered a particularly suitable working environment to develop research on computation architectural design. The practices invited were all young, with particularly homogeneous profiles, similar dates of founding, similarities in the previous production and in the means of integrating research on computational tools into the practice, as well as fitting the relationship to tools described for G2. The lead architects of each practice also have similar trajectories, with involvement in academic environments and in teaching - on some occasions together. It is also interesting to note the presence of younger architects in the teams such as Ezio Blasetti, Roland Snooks or Dave Pigram as it also hints to a strong insertion of the SPC into the network of computational architectural research, since these architects went on to be prominent players in G3. Furthermore, despite its strong adequation as case study, the SPC has never been used as such in digital tools overviews or digital heritage studies. The only publications regarding this competition are

the exhibition catalogue (Guenoun 2007) and a few brief articles in magazines such as CREE (n°332), D'Architectures (Scoffier 2007) or A+U (n°455), either reports of the exhibition or succinct presentations of a specific proposal in an article dedicated to the practice who authored it.

## 3. Analysis methodology

The methodology defined here aims to gather a common set of data on each of the proposals, on four different aspects of the project: algorithmic tools/method, computational set-up, organizational chart and architectural design. The specific information surveyed are detailed in Table 1, as well as the matching data collected for each proposal. The data was collected by consulting the published documents for each project - credits, plans, renders, diagrams, code excerpts - and by conducting interviews with the leader of each practice. The items available for each proposal are indicated in table 1 as well.

Table 1. Data collected for the six Seroussi Pavilion Competition proposals.

| | | Practices | | | | | |
|---|---|---|---|---|---|---|---|
| | | Ibathing | labDORA | EZCT Architecture & Design Research | UP Corporation | Grasmüll & Rohler | Xefirotarch |
| Algorithm | Family : | Formal Research | Performance-driven design | Performance-driven design | Formal Research | Fabrication-driven design | Formal Research |
| | Type | Attractors (agent-based); magnetic fields) | Finite Element Analysis, Particle System Optimization | Genetic Algorithm | Periodic Equation | Robotic Brick Placing | Growth |
| | Whole to split : | Whole | Split / Not related | Split / Related | Whole | Split / Related ?? | Whole ?? |
| | Main Technical Objective | Global Shape Generation | Structural Optimisation | Light Optimisation | Global Shape Generation | Detailed Shape Generation | Global Shape Generation |
| | Secondary Technical Objectives: | / | Global Shape Generation, Detailed Shape Generation | Structural Optimisation, Global Shape Generation, Detailed Shape Generation | Mathematical Sobriety | Fabrication files generation | / |
| | Output | "Flower" Shells (Roofing) | Shell Shape and Structural Elements | Main Shell Shape, Structural Pattern for Surface Subdivision | Modules | Wall Shape and Bricks Placement | ? |
| | Significant Variable: | Position of Attractors | Double Shell Structure | Shape of retrieved blocks | Periodic Equation / Cylinder | Robotic Placing Constraint | ? |
| Technical Set Up | Software: | Rhinoceros + FlowerPower | Rhinoceros, Finite Element Analysis Software | Rhinoceros, Mathematica, Radiance, Maya, Blender, VTK, Polytrans, Getfem++, 3Dmax, Vray, EO, Catia | MathCAD | Maya | ? |
| | Programmation Language | C#* | C#* | Python, C++, Wolfram | Mathematics | MEL | ? |
| | Interface type | Plug-in with slider parameters | / | Scripting | Scripting | Scripting | ? |
| | Graphic Representation Tool for Base Geometry Visualisation | Rhinoceros | Rhinoceros | Rhinoceros | AutoCAD | Maya | ? |
| | Final Output Type | 3D + 2D Drawings | 2D Drawings | 3D + 2D Drawings | 2D Drawings | 3D + 2D Drawings + Fabrication File | 3D + 2D Drawings |
| Chart | Date of Office Creation | 2001 | 2001 | 2000 | 2000 | 2003 | 2001 |
| | Number of people (Team; + Consultants) | 7 + 1 | 17 + 7 | 11 | 10 | 12 | 10 |
| | Background of principal | Architecture | Architecture | Architecture | Architecture | Architecture | Architecture |
| | Background of team | Architecture, Computer Science | Architecture | Architecture, Computer Science, Mathematics, Daylight Modelling, Structural Engineering, Visual Rendering | Architecture | Architecture | Architecture |
| | Background of Consultants: | Mathematics, Visual Rendering | Architecture, Computational Design, Structural and Environmental Design | / | Landscape Design | Structural Engineering | / |
| Architectural Design | Drawn Architectural Design independant of the Algorithmic Output | Yes (plan)* | Yes (structural principle) | Yes (plan)* | Yes (plan) | Yes (plan) | |
| | Site-Related items (verbal mention, written mention, site plan, sculptures present on drawings) | 1 | 2 | 4 | 1 | 1 | 4 |
| | Brief compliance (program) | Partial | No | Partial | Yes | Partial | Partial |
| Available documents | Site Plan | Y | Y | X | Y | Y | Y |
| | Ground Plan | Y | X | Y | Y | Y | Y |
| | Drawings (Sections, Renders) | Y | Y | Y | Y | Y | Y |
| | Code | X | X | X | Y | X | X |
| | Diagram/illustration of algorithmic method | Y | Y | Y | Y | X | Y |
| | Credits | Y | Y | Y | Y | Y | Y |
| | Interviews | Y | Y | Y (informal) | Y | Y | X |

## 4. Discussion

### 4.1. BEYOND TECHNICAL BIAS, THE ISSUE OF TECHNICAL LIMIT

One of the characteristics of this computational generation is high awareness of potential biases contained in the algorithmic methods and digital tools relied on. This is also apparent for the practices taking part in the SPC in the comments made on the use of tools and on the development of the algorithms for the competition. The interviews also highlight a common background and convergences in the discovery and learning of computational tools, often in a learning environment with strong intellectual emulation and exposure to disciplines other than architecture. Benefitting from this training and from the feedback of previous generations, they have a fine-tuned mastery of programmation tools and therefore an ability to fully explore their potential in shape generation, constraints assimilation or performance optimisation. But each of the proposals dealt mainly with one of these aspects, as the categories of design in line 1 of table 1 highlight.

The variety of software used (Table 1 Line 11) in is a reminder of the technical complexity in play at the time to implement the algorithms, ensure bridges between specific software and different file formats, plot geometries and so on. The documents published by EZCT include a chart of the different software and file format their algorithm for the competition relied on, including 12 software and 5 file formats. Although it is the only team that detailed the technical set-up this way in written documents, other interviews point to a same scale density.

Considering these two aspects, the issue at stakes in terms of technical tools and their use appears to be the technical limit existing at the time and how to handle it, rather than potential biases of specific algorithmic methods. The selection of one exploration direction at a time is a direct consequence of this. Each of the proposal illustrates a different strategy to manage this limit and produce an architectural answer to the brief.

For the Gramazio & Kohler proposal, the constitution of a set of specific fabrication constraints enabled an architectural design developed based on those, making it an example of fabrication-driven design.

EZCT and DORA both designed a performance-driven algorithm, but the extreme difficulty at the time to implement such algorithm in its entirety led them to explore it in a different way. If EZCT chose to complete the implementation of their light-optimization algorithm, thus handling the most complex technical set-up, DORA established the possibility of such an algorithm by defining the workflow and testing parts of it, but then moved on to a reflexion on architectural reference and how to integrate it in an algorithmic project design.

As expressed during the interview, what drove the design of IJP proposal appears to be an attempt both to remain faithful to a single mathematical idea - as has been seen in most of the other projects designed by the office, such as the Henderson Waves (Hwang 2006) - and to answer as closely as possible to the architectural brief. The idea of developing a project around a single mathematical idea pairs with keeping the problem of technical limits as well as potential technical bias afar and enabling to focus on the development of the architectural design.

Xefirotarch and biothing have in common the exploration of a predefined aesthetic and spatial potential, in the sense that the architecture of their proposal is a variation based on the shapes produced by the algorithm. In both cases, the developed algorithm offers a specific formal vocabulary, that is then adapted to the architectural proposal.

During the interview with biothing, the notion of technical limit was stressed through the description of a gap still existing in architectural algorithmic designed, in opposition to the often-described goal of uniting every step of the design by means of digital tools. This gap could be described as the intervention of architectural decision making in order to guide the use that is made of algorithmic tools. The articulation between those two elements is critical both in the description of how designers resort to computational tools and in the treatment of the technical limit.

### 4.2. THE PLACE OF ARCHITECTURAL EXPERTISE IN ALGORITHMIC DESIGN

The design process for the proposals all show a stage at which architectural elements of the proposal were not generated by the computer but conceived and drawn by the team. In several cases, it is the plan of the proposal that was drawn upstream (Table 1 Line 21). At that point, the algorithm already being in development, the formal results that were generated by it were evidently considered while drawing the plan. But the plan was drawn without recourse to computational tools. Afterwards, in a second phase, either the plan was fed as input to the algorithm, either a series of shapes was generated and then plugged onto the previously drawn plan, to create the architectural proposal itself.

Although the issue of translation of architectural constraints and decision-making into algorithmic language is at the core of computational design, a significant part of decision-making that is relevant to space design has in the case of the Seroussi Pavilion proposals taken place in part separately. The interlocking of algorithmic development in order to obtain specific geometries and of external architectural decision making therefore appears to be a characteristic of this generation of experimentations as well.

The architectural brief and specifically the program of the pavilion as highlighted in the competition brief was in general not fully respected, or even in the case of DORA, not respected at all (Table 1 Line 23).This disregard is explained by the at the time rare opportunity offered by the competition to experiment with computational design and each practice favoured the exploration of the formal and architectural possibilities that could come out of it. But despite this disregard, deep architectural concern shows in each proposal. Notably, the importance of the building site, hosting André Bloc's villa and more importantly his *sculptures-habitacles*, was mentioned several times. The visual and historical strength of these works were mentioned by most of the architects, and they appear on most drawings (see Table 1 Lines 22, 24, 25). The EZCT proposal is particularly interesting concerning this, as the shape of the retrieved blocks in the initial genetic algorithm, informing the shape of the final pavilion, seem a direct formal reference to the surrounding sculptures.

Beyond the tension between the exploration of computational tools and the design of an architectural proposal, the ongoing changes in the profession are also apparent in this analysis (Table 1 Lines 17 to 20). The different teams present both traditional players, such as a leading architect and engineering consultancies, but also other players specific to the development of computational design, such as software designers (biothing) or a daylight modelling specialist (EZCT). Furthermore, the leading architects all have developed skills specific to designing with programming tools. The professional background and set-up of the teams for the competition is therefore an interesting example of how transitioning to computational design might happen on the professional level.

## 5. Conclusion: Paradigmatic consistency & technical evolution

Several comments made by the architects on the competition during the interviews referred to their current practice - in opposition to their practice at the time - and highlight how the issues at play in architectural computational design have developed at the time and how they transformed until now. As computational tools develop and the technical limit retreats, both more and more data and new types of data can be considered. Tools that are being developed currently (Salim & Haque 2015) (Sanchez 2013) as well as the evolution of the general discourse on computational design (de Rycke et al. 2018) hint to the fact that computational tools currently develop towards data not only in the domain of performance analysis, but as well elements such as aesthetical choice (Brown & Mueller 2017) or data gathered through massive participation to online platforms. All these issues can although be summarized into on single question, which is the question of automated design. Ultimately, it is this same paradigm that leads all computational experimentation, with the same theoretical background for most designers in the computational field. The evolution of architectural proposals throughout the last 45 years is due in part to the evolution of the tools themselves, more than to the evolution of the issues architects and designers deal with, a relation that can be shown through a historical study of computational tools, such as this research has suggested.

In conclusion, the research presented in this paper attempts, through the elaboration of an analysis method for architectural design integrating programming tools and through the outline of a possible taxonomy for computational architecture, to document the existing digital heritage in a new approach. The application of this method to a specific case study demonstrates its ability to lay out new socio-historical data for a better understanding of the context of emergence of programming tools in space design and of their influence on the resulting architectural objects, in order to contribute to a history of computational tools in architecture.

## References

Anonymous, A.: 2007, Le pavillon d, *Architecture intérieure CREE*, **332**, 0294-8567.

Anonymous, A.: 2008, Xefirotarch / Hernan Diaz Alonso, Maison Seroussi – Paris, France, *A+U*, **455**, 0389-9160.

Brown, N. and Mueller, C.: 2017, Designing with data: moving beyond the design space catalog,

*Proceedings of ACADIA 2017*.
Fornes, M.: 2007, "Scriptedbypurpose : Explicit and encoded processes within design" . Available from <https://scriptedbypurpose.wordpress.com/> (accessed 17 December 2018).
Gaudillière, N.: Ongoing, *Digital Turn et projet architectural - Transmission des savoirs, croisement des disciplines et fabrication non-standard à l'ère numérique*, Ph.D. Thesis, Laboratoire GSA, Université Paris Est.
E. Guenoun (ed.): 2007, *Pavillon Seroussi : biothing, DORA (Design Office for Research and Architecture), EZCT Architecture & Design Research, Gramazio & Kohler, IJP – George L. Legendre, Xefirotarch*, HYX.
I. Hwang (ed.): 2006, *Verb Natures : What is Nature ? How Can we Modify Nature? What is the Nature of Nature*, ACTAR.
Landau, R.: 1985, A philosophy of enabling : the work of Cedric Price, *AA Files*, **8**, 3-7.
Laurent, C., Del, A., Morandi, C., Zreik, K. and Terracol, P.: 2018, De l'objectivité et du digital ; production des données et conceptions numériques – Retour sur le CIMA, *Séminaire Architecture et Culture Numérique, ENSAPVS*.
G. Lynn (ed.): 2013, *Archaeology of the Digital*, Sternberg Press.
A. Menges and S. Ahlquist (eds.): 2011, *AD Reader Computational Design Thinking*, Wiley.
F. Migayrou (ed.): 2003, *Architectures Non Standard*, Centre Pompidou.
Morel, P.h.: 2007, "Catalog of L'Architecture Au-Delà des Formes Exhibition" . Available from <https://petermacapia.com/assets/AuDelaDesFormes_Catalogue.pdf> (accessed 17 December 2018).
K. de Rycke, C. Gengnagel, O. Baverel, J. Burry, C. Mueller, M.M. Nguyen, P. Rahm and M.R. Thomsen (eds.): 2018, *Humanizing Digital Reality*, Springer.
Salim, F. and Haque, U.: 2015, Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and Internet of Things, *International journal of Human-Computer Studies*, **81**, 31-48.
Sanchez, J.: 2013, Gamescapes, *Proceedings of the 33rd Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*.
Scoffier, R.: 2007, Architecture, rayon passementerie, six propositions pour le pavillon Seroussi, *D*, **167**, 1145-0835.
Serraino, P.: 2002, *History of Form*Z*, Birkhauser.

# Evolutionary Tools and the Practice of Architecture: from Appropriated Typology to Becoming the Black Boxes of CAAD

Nadja Gaudillière[1]

[1] *Laboratoire GSA, ENSA Paris-Malaquais, 14 rue Bonaparte, 75006 Paris, France*

[1]*nadja.gaudilliere@paris-malaquais.archi.fr*

**Abstract.** The present research proposes a comprehensive methodology of investigation of algorithmic typologies currently in use in the computational field in architecture, focusing on three elements : practitioners, tools and projects. The methodology has been applied to evolutionary design tools in order to investigate the technical and epistemological optimization bias they entail. The research shows the existence of two approaches to handling the tension between formalization required by algorithms and tacit knowledge required by architectural practice. Whereas one of the approaches displays an intertwinement of explicit and tacit, of qualitative and quantitative, the other prioritizes a rational, performance-based approach. Given the conditions of development of evolutionary tools shown, this latter conception of architectural practice has spread while recent tools enabled a democratization of the resort to evolutionary techniques. By transforming evolutionary algorithms into ready-made tools, or black boxes of CAAD, this process of development prevents the implementation of tacit knowledge within them, potentially impoverishing architectural design.

**Keywords.** Evolutionary tools; tacit knowledge; socio-technical network; technical biases; optimization; architectural practice.

## Introduction

Since the rise of digital tools, and in particular of algorithmic design tools relying on programing skills, a tension has established between the formalization required by the use of programing languages and the expertise and mobilisation of tacit knowledge architectural practice relies on. In opposition to explicit knowledge, tacit forms of knowledge cannot be formalized : their transmission by written or oral communication entails a partial loss and they can only be acquired by the experience of practice (Collins 2010)[1]. Practices such as architecture rely on an equilibrium between the resort to tacit knowledge and the resort to explicit knowledge provided by a corpus of scientific methodologies. This specificity renders the automation of such practices in computational models a difficult but rich exercise. Studies on the computerization of medical practices have nevertheless shown that refusing to acknowledge the tacit dimension of such disciplines leads to conflictual relations to digital technologies for the practitioners (Berg 1997). The tension between computational formalization and tacit knowledge therefore constitutes an essential node of our understanding of the use of digital tools for architectural design. In the last six decades, several means of resolving this tension have been investigated in the digital experimentations led in architecture, both in theory and in practice.

Evolutionary tools in particular have been migrating during the history of their use in architecture from one mode of appropriation to another and therefore constitute an

[1] Sometimes referred to as "empirical decision making" in the field of computational, it can also be described in other fields as intuition or creativity - although each of these terms has specificities, they refer to the same general notion.

emblematic model of how practitioners tackle this tension. These typologies of algorithms enclose a variety of computational methods designed to mimic natural phenomena such as plant growth, fire spread or darwinian selection. As many of these natural phenomena involve reaching an optimal state in a given environment, they have been identified as potent models of optimization and are nowadays mainly used to that end. While scarcely observed in existing studies, biases inherent to such design tools are instrumental to be understood for us to grasp the influence of the digital paradigm on architecture. Evolutionary tools, in particular recent ones, hold an optimization bias in their very structure, a technical specificity that has been accompanied by an epistemological bias entertaining the predominance of a performance-based understanding of architectural practice. By studying the historical development throughout the past thirty years of evolutionary tools in architectural practice, as well as the structure of their interfaces, the present papers analyze the technical and epistemological optimization bias they encompass. It thus offers an illustration of the two main approaches to tackling the tension between formalization and tacit knowledge, through the investigation of the use of evolutionary tools for architectural design.

## Socio-technical networks of computation in architecture

### *The socio-technical network of evolutionary tools*

The computational field in architecture has been a small academic community for several decades, and before the recent mass democratization of digital tools, experimentations on their use for architectural design was conducted in a handful of research groups. These research groups and the tools and the projects they have developed constitute what Latour (1991) refers to as a socio-technical network. This core network further expanded over the years beyond the initial microcosm and towards various players of the AEC industry. This socio-technical network of computation in architecture yet remains to be described as such.

In existing literature, some aspects of this sociotechnical network have however been studied. In particular, the work of Mario Carpo (2011, 2018) must be mentioned as one of the major historical and global overview of the computational movements in architecture. It must nevertheless be underlined that in recent works, in particular on optimization tools, parts of the dynamics that have led to the almost systematic recourse to these tools are not examined, lacking somewhat critical approaches to these recent. If numerous archaeologies of the digital in architecture already exist as well, such as (Lynn 2013), they consist in their majority of an assembly of emblematic projects short presentations and texts by the architects and designers of the computational field. Only a few studies of the development of specific digital tools exist, such as a history of Form*Z (Serraino 2002) or a study of the development of digital tools used in Frank Gehry's practice (Smith 2017), focusing mainly on the period of the 1990s and early 2000s and on the development of 3D modelling.

While algorithmic design and the computational field form a global socio-technical network, tracking the development of major algorithmic typologies in use reveals local chains of connections. The present research focuses on the network formed by evolutionary tools, whose core is depicted in figure 1. The lines indicate connections at large between institutions, practitioners and evolutionary tools. Light blue circles and the constellation they are at the center of represent clusters of evolutionary algorithms development and use. These clusters include the early research at the AA, both between 1989-1996 at Diploma

Unit 11 while John Frazer led it and in the early 2000s with the work of Michael Hensel, Michael Weinstock and Achim Menges. It also includes the first applications for built projects by Bollinger + Grohmann, in association with a few prominent architecture practices from 2007 on, as well as major software clusters such as McNeel or Autodesk. Compared to other sub-networks such as the multi-agent systems network (Gaudillière 2020), constituted of academics or small-scale offices and open-source software, this network involves a greater number of larger players of the AEC industry as well as of the software industry. This greater variety of players in the clusters implies a greater diversity in users and in uses, to be assessed in the present study. Furthermore, while the multi-agent systems network features few tools travelling from one cluster to another, the evolutionary tools system features few clusters generating several tools each.

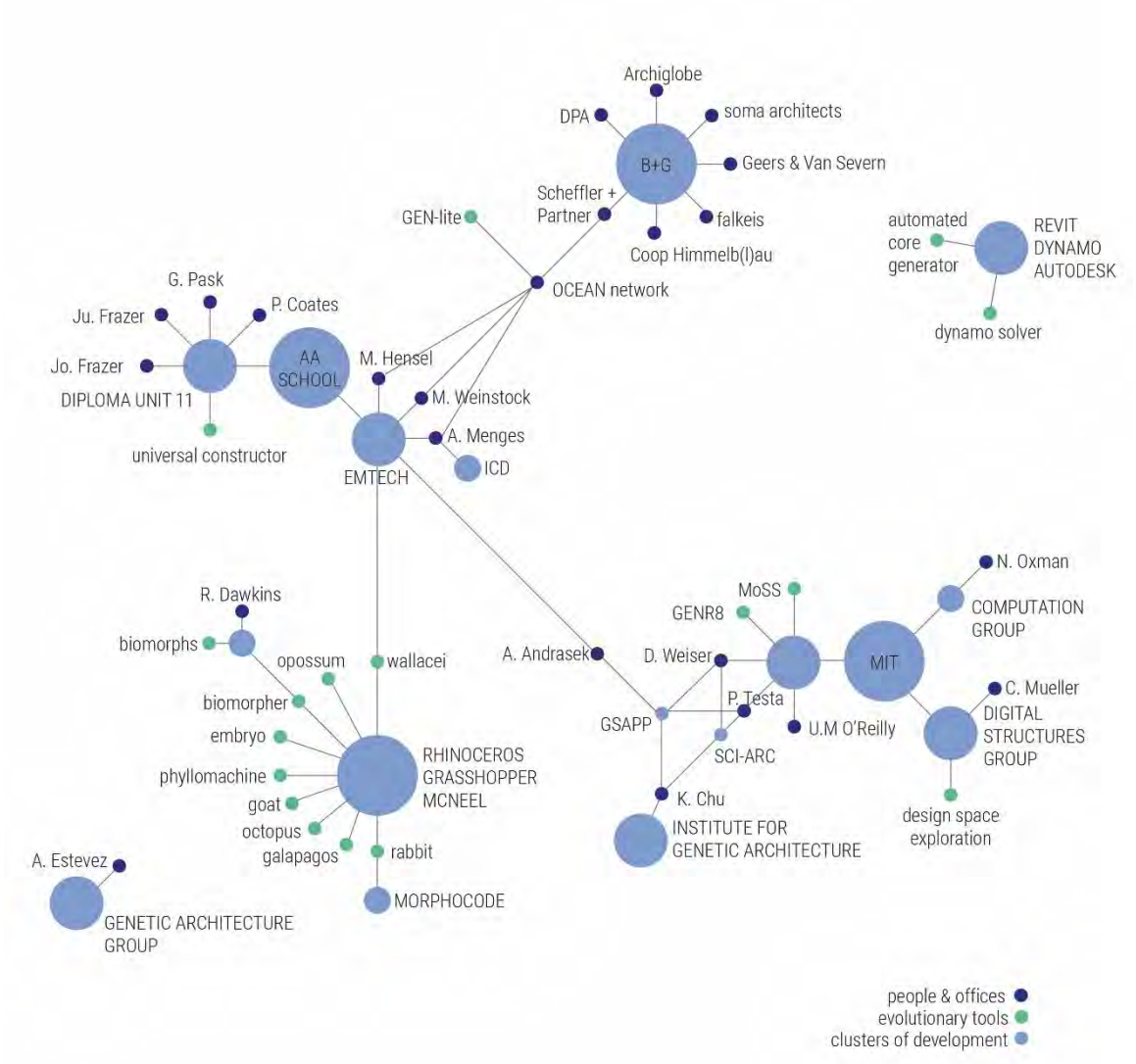



*Figure 1*
*Clusters of evolutionary tool development in computational architecture*

*Methodology*

The present research has been conducted on two algorithms families: growth systems such as L-systems and genetic algorithms, following four steps. First, in order to identify the clusters, AD issues and the Cumincad ACADIA database have been searched[2] to establish a list of projects and practitioners resorting to evolutionary tools, with a focus on the use of scripting to produce architectural objects or information relevant to the production of an architectural object. The second step focuses on the definition of a taxonomy of users, by looking at the practitioners identified and assessing their education (architecture, engineering, computer science or else), professional environment (academic, practice or both), programming skills (ability to use a ready-made algorithm, to develop a custom algorithm, to develop a tool). Third, projects[3] resorting to evolutionary tools have been assessed examining four areas: algorithmic tools/method, computational set-up, organizational chart and architectural design[4], in order to determine which technical aim was favored when using evolutionary algorithms (formal mimicry or optimization). Finally, released evolutionary design tools have been studied, focusing on the conditions of their development and on how their interface enables the manipulation of the algorithmic typologies they are designed for.

## Evolutionary tools in architecture

*Evolutionary tools: from biologists to architects*

First developed by biologists, geneticists and biophysicists in the 1950s and 1960s - Nils Aall Barricelli (1954), Alex Fraser (1957) or John Holland (1975) can be mentioned as pioneers - simulations of natural selection soon became a full-fledged field of study, meeting point of biology and computer science. Evolutionary tools, genetic algorithms in particular, were then adopted by engineers to solve complex optimization problems in a variety of applications such as encrypting data, designing motor parts, ranking players, etc. In many domains, genetic algorithms have become popular optimization tools, due to their ability to take multiple criteria into account and to introduce mutations in order to avoid local minima or maxima, thus enhancing their performance. Furthermore, their ability to explore a vast design space made genetic algorithms prized not only by engineers, but also by architects who would in the 1990s increasingly appropriate them, alongside morphogenetic growth systems also popular for their formal potential. Historically though, exploring the latter has first been predominant among architectural experimentation, while genetic algorithms would then gradually take a hegemonic position.

---

2 Following keywords were used when searching the database : “evolutionary tools”, “growth system”, “genetic algorithm”, “morphogenetic”.

3 Both paper / research projects and built projects indiscriminately

4 Data has been obtained by conducting interviews with practitioners and consulting projects drawings, programming files and publications.

More broadly, what many architects earlier on referred to as morphogenetic processes[5] enclose a large variety of projects. Despite the vocabulary used, not all of them resort to what is nowadays known as evolutionary tools. Those which resort to evolutionary tools employ various typologies, or families, of algorithms. Algorithmic typologies are identified by the rules of formalization they rely on, different typologies having different programming structures and therefore different types of mandatory inputs. In the case of evolutionary tools, observing the typologies' characteristics allows for an analysis of how the technical constraints linked to the optimization bias are implemented. Two are studied in the present paper : L-systems and genetic algorithms. L-systems, or branching systems, are mathematical structures dating back to the beginning of the XXth century, developed by several mathematician until their current form, proposed by Lindenmayer in 1968 (Prusinkiewicz & Lindenmayer 1990). Used in particular to simulate plant growth, their rule typology includes : a) the initiator shape, b) the generator shape, c) seed placement, d) the branching rules (angle, length, number), e) the number of iterations (figure 2.a). As for genetic algorithms, despite various kinds of computation methods exist, they are all based on the same principle of darwinian selection (Calixto & Celani 2014). Populations of individuals are generated and evaluated, and the better performing individuals' characteristics are retained to develop a new generation, a process repeated until a series of individuals as performing as possible are obtained. The rule typology for genetic algorithms includes a) selecting the computation method, b) defining the gene pool (characteristics varying in each individual), c) defining the fitness function (the criterias on which each individual is evaluated), d) selecting an individual amongst the last, most performing generation (figure 2.b).

### *Taxonomy of users*

The taxonomy of users developed in the present research results in three major categories, presented in table 1, also summing up the characteristics of each user category. First, two categories of skilled practitioners, characterized by their high programming competences, are identified : in architecture - with early adopters such as Karl Chu or John Frazer, but also a few contemporary users - and in engineering-based practice, with both academics such as the Digital Structures research group and professional practitioners such as Bollinger + Grohmann or Ramboll. A third group gathers mass users of the contemporary period, having accessed evolutionary algorithms after the release of tools with robust, appropriable interfaces. While the two first groups reflect the clusters of the socio-technical network shown previously, this third group embodies the turning point of evolutionary tools exiting the core network and expanding into common practice. Finally, a fourth group is also highlighted: while software developers only produce little research on the use of these programs for architectural design, they remain an important player given the key role their tools play in the democratization of evolutionary algorithms.

---

[5] As can be seen in several AD issues such as Emergence: Morphogenetic Design Strategies (2004) or Techniques and Technologies in Morphogenetic Design (2006).

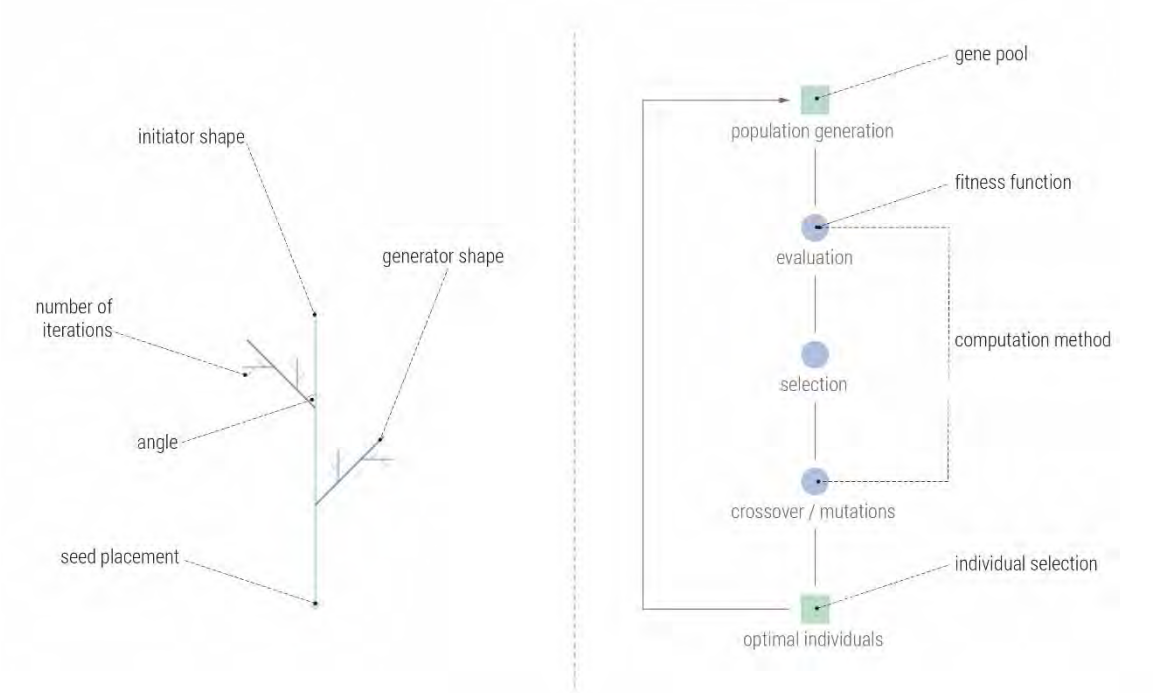



*Figure 2*
*a. L-system typology rules; b. Genetic algorithm typology rules.*

| | 01 - skilled practitioners (arch.) | 02 - skilled practitioners (eng.) | 03 - mass users | 04 - software developers |
|---|---|---|---|---|
| education | architecture *(for many also a technical background in engineeering / computer science)* | engineering | architecture | computer science architecture |
| professional environment | academic *(occasional projects as professional offices structures)* | academic professional | academic professional | professional |
| programming skills | custom algorithms *( also tool development but with little spread beyond their own institutional network)* | tool development custom algorithms | ready-made algorithms | tool development |
| use of evolutionary tools | formal research optimization *(mainly paper projects and academic research) (regular collaborations with computer scientists)* | optimization *(including for built projects in collaboration with large architecture offices)* | formal research optimization *(predominance of optimization)* | / |

*Table 1*
*Taxonomy of users*

## Appropriating evolutionary tools in the practice of architecture

### *Mastering programming tools, successfully implementing the tacit*

Among the established corpus of those mentioning morphogenetic processes at large, identifying the projects placing evolutionary tools at their core is necessary. Those drawing inspiration from natural growth phenomena (Roussel 2017) must be differentiated from those using programming tools simulating such phenomena - even though some practitioners have mobilized the two together. Whether the evolutionary tool resorted to is the typology at the forefront of the design process or not must also be determined, since evolutionary tools, in particular genetic algorithms, are often combined with other algorithmic typologies. Despite the large corpus of projects mentioning evolutionary tools, only few match those two criterias and attest of a complete architectural appropriation. Among those, some resort to evolutionary tools mainly for formal explorations, such as the Tote (Serie Architects, 2009) or the New Czech National Library (Scheffler + Partner, 2006), other push further the entanglement of formal, structural and computational requirements - such as the Seroussi Pavilion (EZCT, 2006) or the Jyvaskyla Art and Music Center (OCEAN, 2004). The practitioners, by relying either on collaboration with computer scientists or on a solid knowledge of programming languages, enable themselves to design complex, custom algorithms, and the resort to evolutionary tools provides in these case the evidence of the practice of scripting as a craft. Beyond the combination of requirements, it is the transformation of tacit knowledge on spatial design directly into the computational typologies rules rather than into visual representations that is at play, attesting of an appropriation of evolutionary tools as any other technical tool would be appropriated into a classical architectural practice.

### *Morphogenetic liberties and the heritage of cybernetics*

Discourse on the use of evolutionary tools in architecture is illustrated by a diversity of written explanations of the benefits of resorting to such techniques - a discourse that has been strongly criticized (Cogdell 2019). These critics underline the inaccuracy of the use made of evolution simulations, consequence of the absence of understanding in the architectural community of the details of biological phenomena studied and of the peculiarities of simulation models adopted. While the lack of precision on technical, computational and biological aspects in the vocabulary generally used by architects to discuss morphogenetic processes in their projects is indeed questionable, liberties taken with what evolutionary tools stand for in the biology and computer science communities is demonstrating precisely how the tension between tacit knowledge and formalization has been tackled through appropriation. Acknowledging this altogether classical dimension of part of the architectural production mobilizing digital tools means acknowledging the role played by tacit in architectural practice, which cannot - yet - be formalized and performed by a computer. In the early days of the computational field, practitioners such as Nicholas Negroponte (1969) or Nigel Cross (2006), and the cybernetic movement at large, have confronted this issue of automating the tacit. While a form of inheritance of these reflections is tangible in the practices of the contemporary computational field, the theoretical position assumed by these early researchers is nevertheless much less present

in discussions nowadays[6]. In particular for evolutionary algorithms, it is a line of thought that has disappeared while the use of tools progressed, making way for a preponderance of the notion of rationalisation.

## Becoming black boxes of CAAD

### *Interfaces for democratization*

In the past ten years, the computational field has grown through the democratization of its tools, including evolutionary algorithms, seeing both many new users and a multitude of dedicated software and plugins appear. Evolutionary tools have since the seminal work of Richard Dawkins (1986) and his Biomorphs evolved into both professionalized resources such as the Evolver software and open-source plugins such as the ones available on Grasshopper. Figure 3 displays a chronology of appearance of those various resources, all having in common the typological rules described previously. These various tools also feature another common trait: the presence of a user interface, contrariwise to custom algorithm developed in the first period of use of evolutionary processes in architecture. Those interfaces and their ease of use have been instrumental to the spread of those methods, despite the crucial simplification they entail. This simplification is stressed by many of the tool developers as necessary for a majority of architects to seize these potent design tools, sometimes at the expense of enabling a comprehensive apprehension of their structure, as can be seen in the description of some of those tools : "*the aim is for users (of all degrees of expertise) to better understand their evolutionary simulations, gain a thorough understanding of the outputted numeric values, and seamlessly (...); all within one user interface*." (Wallacei plugin description), "*helping to explore the wide combinatorial space of parametric models without always knowing where you are headed*." (Biomorpher plugin description). The result of this approach is, as expected, an increase in the number of users[7], but also a shift in user skills : while earlier users were skilled both in computer science and in their own domain (here architecture), mass users are untrained in scripting while maintaining their own field of competence. In parallel, a shift in the background of evolutionary tools developers is also witnessed, with a large predominance in professionals stemming from engineering, AEC and software industry (table 1).

The consequence of both the reduction of complexity entailed by a more easily grasped interface and of the growing distance between the developer and the user sets of skills results in a thickening of the interface of evolutionary design tools. The thickness of an interface is a notion developed by Anthony Masure (2019), transcribing the distance between what a user can perceive and manipulate of a program through its interface and the actual structure of the program. Figure 4 displays a comparison of this distance in the case of a custom evolutionary algorithm and in the case of a ready-made evolutionary tool[8],

---

6 Beyond scarce publications and the almost inevitable mention of "empirical decision-making" in paper introductions.

7 While the number of users of evolutionary tools has been in the present research be estimated to 35 000, significantly more than the few hundreds of the core network, the potential pool of users gathered by McNeel and Autodesk could reach up to 2 millions.

8 For the custom algorithm, the 2007 Seroussi Pavilion algorithm by EZCT (Guenoun 2007) was taken as example, and for the ready-made tool, Galapagos was selected.

by showing the layers of structure of each program. The specific rules of the typology and where to access them in this structure are also shown. A common goal was fixed for the two algorithms in order to compare the ease of access to each rule of the typology: performing an optimization in order to obtain a light repartition as uniform as possible in a given space. While the custom algorithm contains five straight layers, three programming languages and twelve software[9], with the selection of rules regrouped in two layers, the ready-made tool contains seven layers including three double-layers, six programming language and three software, with the selection of rules scattered across five layers. It is furthermore interesting to note that in most Grasshopper plugins, individual selection is a sub-layer of the first interface, as pictured here, whereas in most other, individual selection is the first layer, before accessing fitness function and individual selection.

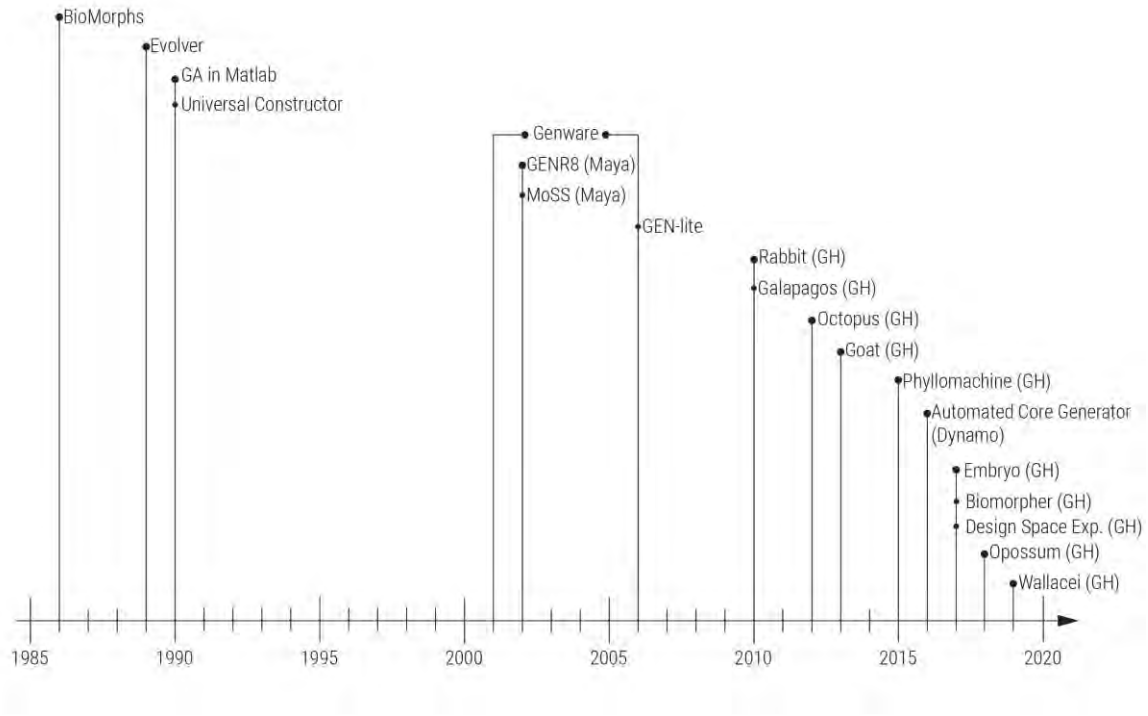



*Figure 3*
*Chronology of appearance of evolutionary tools for CAAD*

While crucial to democratization, these characteristics lead to a lack of taking into consideration the possibility to implement the tacit knowledge characterizing the skill set of architecture users and to a reduction of the flexibility of evolutionary tools. This also fuels an enduring lack of computational skills of untrained architects, weakening here again the possibility of appropriation through tacit knowledge implementation. Although it is also the consequence of a long-term diminishing foothold of architects into sciences for the built environment, from the apparition of the modern architect and engineer profession to

9 It is nevertheless an algorithm dating back to 2007, and a more recent custom algorithm might be able to reduce this large number of softwares.

contemporary starchitects (Picon 1989, Carpo 2011), the current structure of evolutionary tools further sustains it, as well as promoting a performance based approach of design.

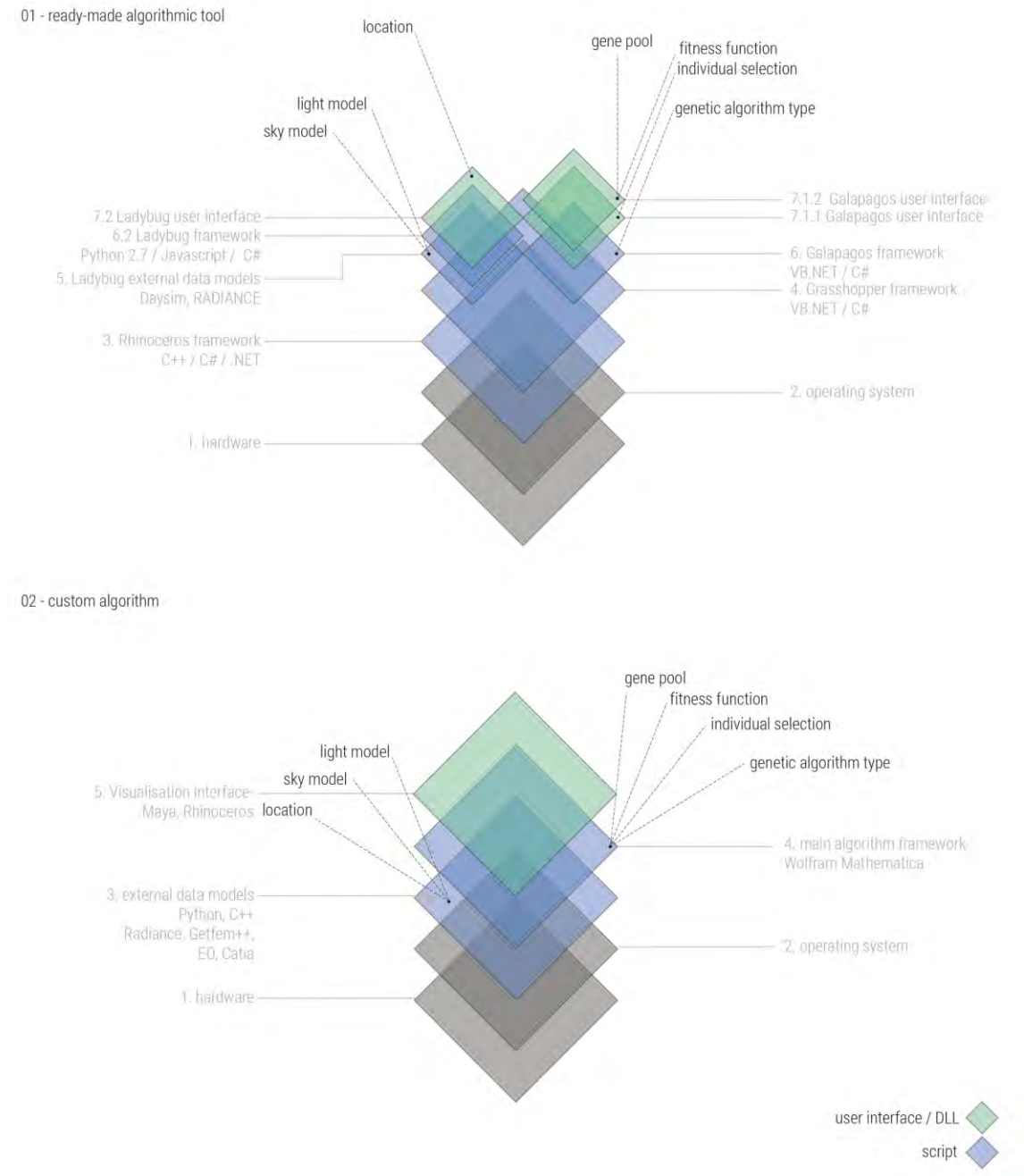



*Figure 4*
*Interface thickness for a ready-made tool and a custom algorithm*

*Promoting performance: rationalization at all costs?*

Performance-based design (Hensel 2013) is guided by the evaluation of rationally measurable performances, examining quantifiable physical phenomena affecting the built environment through verified mathematical models and assessing designs for them to withstand best those phenomena. Mechanical, thermal, acoustic, light performances, but also financial or environmental costs are thus taken into account during the design process. The interest in performance-based design in the computational field in architecture is not only found in link to evolutionary tools, as it is a global trend, but given that optimization is evidently at the core of it, links between both are strong.

The recent spike in the promotion of performance-based approaches is built on two major arguments. First, since the resulting designs have been designed resorting to optimization tools - often genetic algorithms - they can only be better. In recently released tools for Dynamo and Revit for example, Autodesk developers thus impose design decisions and fitness functions upon their users, arguing that those decisions ensure optimized solutions regarding building norms and therefore a rational approach to architectural design (Das & al 2016, Nagy & al. 2017). One could nevertheless ask what exactly an optimized architecture is, the very nature of the discipline being tirelessly interpreted and discussed by many in the profession. In the same spirit, Stanislas Chaillou (2020), in a recent talk, is going as far as claiming to know what any architect would ask from a computer, an impossible question to answer given the diversity of architectural practices.

The second main argument resorted to by proponents of performance-based design is the vastness of choice proposed to the architect, brought back to his tool user status, a vastness s.he is free to explore given that algorithms, developed and guided by other, are here to take care of tedious constraints such as mechanical or thermal characteristics. Evolutionary design tools such as those proposed by the Digital Structures group (Brown & Mueller 2017), where the user is expected to select his/her favorite individuals, are illustrations of this approach. Such rethoric and approach however replace the freedom given by mastery of tools by an illusion of choice and reduce the tacit dimension of architecture to aesthetic taste. As performance-based design is promoted by the same practitioners developing evolutionary tools, the epistemological and the technical optimization bias mutually strengthen, aligning with the general tendency to quantification and rationalization despite the many critics that have been made to this modern model (Latour 1991).

## Conclusion

The present research proposes a comprehensive methodology of investigation of algorithmic typologies currently in use in the computational field in architecture, focusing on three elements : practitioners, tools and projects. The methodology has been applied to evolutionary design tools in order to investigate the technical and epistemological optimization bias they entail. The research shows the existence of two approaches to handling the tension between formalization required by algorithms and tacit knowledge required by architectural practice. Whereas one of the approaches displays an intertwinement of explicit and tacit, of qualitative and quantitative, the other prioritizes a rational, performance-based approach. Given the conditions of development of

evolutionary tools shown, this latter conception of architectural practice has spread while recent tools enabled a democratization of the resort to evolutionary techniques. By transforming evolutionary algorithms into ready-made tools, or black boxes of CAAD, this process of development prevents the implementation of tacit knowledge within them, potentially impoverishing architectural design.

## References


Barricelli, N.A., "Esempi numerici di processi di evoluzione", *Methodos 1954*, p. 45–68, 1954.

Berg, M., *Rationalizing Medical Work : Decision-Support Techniques and Medical Practice*, The MIT Press, 1997.

Brown, N. & Mueller, C., "Designing with data: moving beyond the design space catalog", in Nagakura, T., Tibbits, S., Mueller, C. (eds.), *Disciplines & Disruption : ACADIA 2017 Conference Proceedings*, pp. 154-163, 2017.

Calixto, V. & Celani, G., « A literature review for space planning optimization using an evolutionary algorithm approach : 1992-2014 », in *Project Information for Interaction :Proceedings of the 19th SIGRADI Conference*, Universidade Federal de Santa Catarina, 2015.

Carpo, M., *The alphabet and the algorithm*, The MIT Press, 2011.

Carpo, M., *The Second Digital Turn : Design Beyond Intelligence*, The MIT Press, 2017.

Chaillou, S., "IA & Architecture", talk at the Pavillon de l'Arsenal, 27 February 2020.

Cogdell, C., *Toward a Living Architecture?: Complexism and Biology in Generative Design*, University of Minnesota Press, 2019.

Collins, H., *Tacit and Explicit Knowledge*, University of Chicago Press, 2010.

Cross, N., *Designerly Ways of Knowing*, Springer, 2006.

Das, S., Day, C., Dewberry, M.J., Toulkeridou, V., Hauck, A., "Automated Service Core Generator in Autodesk Dynamo : Embedded design intelligence aiding rapid generation of design options", in Herneoja, A., Österlund, T., Markkanen, P. (eds.), *Complexity & Simplicity : eCAADe 2016 Conference Proceedings*, University of Oulu, pp. 217-226, 2016.

Daston, L. & Galison, P., *Objectivity*, The MIT Press, 2007.

Dawkins, R., *The Blind Watchmaker*, Norton, 1986

Fraser, A., "Simulation of Genetic Systems by Automatic Digital Computers I. Introduction", *Australian Journal of Biological Sciences*, Vol. 10, num. 4, p484 1957

Frazer, J., *An Evolutionary Architecture*, Architectural Association, 1995.

Gaudillière, N., "Towards an History of Computational Tools in Automated Architectural Design - The Seroussi Pavilion Competition as a Case Study", in M. Haeusler, M. A. Schnabel, T. Fukuda (eds.), *Intelligent & Informed - Proceedings of the 24th CAADRIA Conference - Volume 2*, Victoria University of Wellington, Wellington, New Zealand, 15-18 April 2019, pp. 581-590, 2019.

Gaudillière, N., "Computational Tools in Architecture and Their Genesis : The Development of Agent-Based Models in Spatial Design", in *CAADRIA 2020 Conference Proceedings*, to be published.

Guenoun, E. (ed.), *Pavillon Seroussi : biothing, DORA (Design Office for Research and Architecture), EZCT Architecture & Design Research, Gramazio & Kohler, IJP – George L. Legendre, Xefirotarch*, HYX, 2007.

Hensel, M. (ed.,), *Performance-Oriented Architecture: Rethinking Architectural Design and the Built Environment*, AD Special Issue, John Wiley & Sons, 2013.
Holland, J.H., *Adaptation in Natural and Artificial Systems*, The MIT Press, 1975.
Kuhn, T., *The Structure of Scientific Revolutions*, University of Chicago Press, 1962.
Latour, B., *Science in Action*, Harvard University Press, 1987.
Latour, B., *Nous n'avons jamais été modernes*, La Découverte, 1991.
Lynn, G. (ed.), *Archaeology of the Digital*, Sternberg Press, 2013.
Masure, A., *De l'influence des interfaces*, Emilieu, 2019
Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., Zhao, D., Benjamin, D., "Project Discover : An application of generative design for architectural space planning", in Turrin, M., Peters, B., O'Brien, W., Stouffs, R., Dogan, T. (eds.), *2017 Proceedings of the Symposium on Simulation for Architecture and Urban Design*, University of Toronto, 2017.
Negroponte, N., "Towards a theory of architecture machine", *Journal of Architectural Education*, Vol. 23, num. 2, pp. 9-12, 1969.
Picon, A., "Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction", *Revue d'histoire des sciences*, vol. 42, num. 1-2, pp. 155-172, 1989.
Prusinkiewicz, P. & Lindenmayer, A., "Graphical modeling using L-Systems", in Prusinkiewicz, P. & Lindenmayer, A., *The Algorithmic Beauty of Plants*, pp. 1-50, Springer, 1990.
Roussel, M., "De la cybernétique à l'architecture numérique : retour sur un demi-siècle de théories, pratiques et projets expérimentaux", *Canadian Journal of Communication*, Vol. 42, pp. 515–533, 2017.
Serraino, P., *History of Form*Z*, Birkhauser, 2002.
Smith, R., *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace, 2017.

# COMPUTATIONAL TOOLS IN ARCHITECTURE AND THEIR GENESIS: THE DEVELOPMENT OF AGENT-BASED MODELS IN SPATIAL DESIGN

NADJA GAUDILLIERE
[1]*Laboratoire GSA, Ecole Nationale Supérieure d'Architecture Paris-Malaquais, 75006 Paris, France*
[1]*nadja.gaudilliere@gmail.com*

**Abstract.** Based on the assumption that socio-technical networks of computation in architecture exist and must be analyzed deeper in order to understand the impact of algorithmic tools on the design process, the present paper offers a foray into it, drawing on science studies methodologies. The research explores in what regard multi-agent systems (MAS) are representative as much from the existence of these socio-technical networks as of how their development influences the tension between tacit and explicit knowledge at play in procedural design processes and of the strategies architectural designers develop to resolve this tension. A methodology of analysis of these phenomena is provided as well as results of the application of this method to MAS, leading to a better understanding of their development and impact in CAAD in the past two decades. Tactics of resolution shaped by early MAS users enable, through a double appropriation, a skillful implementation of architectural practice. Furthermore, their approach partially circumvents the establishment of technical biases tied to this algorithmic typology, at the cost of a lesser massive democratization of the algorithmic tools developed in relation to it.

**Keywords.** Computational tools; multi-agent system; architectural practice; tacit knowledge; digital heritage.

## 1. Introduction

The Digital Turn in architecture is a widely accepted notion since the famous AD special issue *The Digital Turn 1992-2012*, edited by Mario Carpo and stating the existence of experimentations around the idea of the digital in architecture as well as popularizing the term (Carpo 2012). The Digital Turn as currently defined encompasses the use of a variety of digital tools. The computational movements in architecture are nevertheless also characterized by a specific socio-historical context of emergence (Gaudillière 2019). Most of its architectural production consists in paper projects, prototypes, pavilions, produced in an academic environment. The computational movements therefore organise around a series of practitioners, of research units and institutions, shaping a network of knowledge transmission. This academic set-up has enabled the field to blossom

*RE: Anthropocene, Proceedings of the 25th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2020*, Volume 2, 497-506. 

away from most of the usual constraints of the construction industry. This freedom and diversity of explorations make its production a prefiguration of the global computational turn in architectural design currently happening. The computational field as it has developed in the last 50 years forms therefore an ideal set of designers and projects to study programming-based spatial design and the translation of architectural constraints into computational design tools. Furthermore, as computational movements in architecture are characterized as much by the technical aspects of the practice as by the structure of its community of practitioners, they can be identified as a socio-technical network (STN) (Latour 1991), still to be described as such.

The present researches focuses on the analysis of STN of computation in architecture for a specific algorithmic typology, multi-agent systems (MAS), and studies the trade-off happening between the mobilization of tacit knowledge and the explicitation of formal instructions when resorting to it for architectural design purposes, as well as how the structure of programming interfaces of MAS has influenced this trade-off. The study highlights three major dynamics influencing the development of MAS in CAAD: appropriation, democratization, rationalization. The paper will first address the characteristics of the algorithmic typology studied, before presenting the methodology on which the research is based and discussing the findings regarding the three dynamics identified, analyzing to what extent the resort to algorithmic design tools impacts architectural practice.

## 2. Multi-Agent Systems as a typology

MAS, also referred to as swarms, are object-oriented algorithmic systems modeling the behavior and interactions of sets of agents. Each agent is obeying a series of given laws, and their interactions generate what is known as emergent behavior: a conduct of the set that differs from the specific behavior instructed to each agent. The components of a MAS are the following : an *environment* - usually a measurable space -, *objects* populating this environment - that can be modified by the *agents*, *relations* tying objects and agents together, and *operations* enabling the interactions between objects and agents (Ferber 1995). Amongst the early developments of MAS are the Boids, by Craig Reynolds, in 1986. Craig Reynolds, a computer scientist and graphic designer working at the time on the animation of large numbers of elements for the movie industry, programmed the Boids to simulate the behavior of a bird flock. While being one of the first MAS developed, the flocking algorithm devised by Reynolds also constitutes an example of the type of rules that are applied to agents in swarm algorithms. The Boids obey three rules: separation - respecting a minimal distance from other agents - alignment - head towards the average displacement direction of the flock - and cohesion - respecting a maximal distance from other agents(Reynolds 1987). Following this breakthrough, a large number of frameworks were developed to enable programming with MAS from 1995 on, and those have since been regularly used, mainly in the film and video games industries. In particular, the pioneering development of MASSIVE in 1996 for the animation of crowds of thousands or more in the *Lord of the Rings* films, and the Golaem Crowd Maya plugin,

developed in 2011, are nowadays still in use in the industry.

Since MAS can be identified as a specific typology of algorithm, they imply the resort to a predefined typology of instructions, identical for all algorithms (figure 1). The MAS typology as used in CAAD entails three levels of instructions, in the form of rules. First, the rules pertaining to the initial state of the system, such as the position or entry point of the agents and the position of the objects. Secondly, behavior rules for the agents are implemented regarding their movement - direction rules, such as the boids' separation, alignment and cohesion - and their interactions with the objects - attraction or repulsion for example. Finally, rules relating to the geometrical exploitation of the MAS output, in order to convert it in data relevant to the architectural object conceived, by using the final position of the agents, the traces of their displacement or of the operations, and potentially by adding novel geometrical elements.

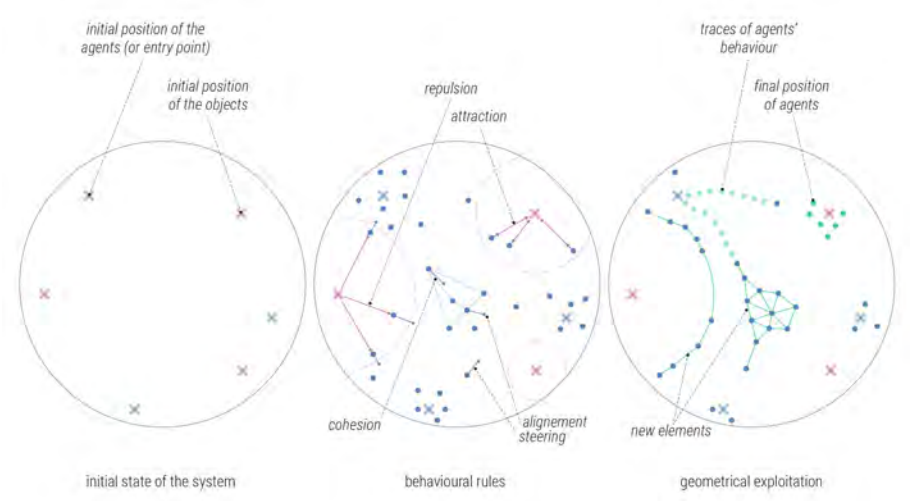



Figure 1. Rule typology for multi-agent systems.

## 3. Methodology

The research is based on the analysis of a series of case studies by practitioners key to the development of the MAS typology, its related tools and its use for architectural design. In order to outline the computational field as described in the introduction, AD issues have been searched to establish a list of projects and practitioners resorting to digital tools, with a focus on the use of scripting to produce architectural objects or information relevant to the production of an architectural object. The compiled list highlights practices, institutions, projects and tools of the computational movements in architecture, and identifies each project with one or more typologies of algorithms. From this list, the projects associated with MAS were extracted and practices and practitioners appearing

more than three times in the list were identified as having made a significant contribution to the development of their use in CAAD. In complement, a literature review of published projects resorting to MAS in the Cumincad database was conducted on 115 papers to assess the contemporary use of MAS, after the creation of a number of algorithmic tools enabling an easier manipulation. The chronological development of major algorithmic tools for MAS modelling has also been examined, for standard frameworks as well as the specific tools involved in the case studies - such as Toxiclibs for Processing and the Flower Power Rhinoceros plugin - as well as for grasshopper plugins - Quelea, Zebra, Nursery, Boid, Physarealm, Culebra - offering an overview of the democratization of this typology through tool-making.

Three aspects are at stake in this study: the constitution of the STN and the chronologic development of MAS use in the computational movement in architecture, the development of ready-made tools and the trade-off between tacit knowledge and programming instructions formalisation. A grid of analysis for the projects themselves has been developed, and each case study has been analyzed following this same grid of criteria, containing four parts. First, the algorithmic specifications and the choices made for the MAS rules are examined. The articulation with other typologies of algorithms is also assessed, in the cases where the MAS is only a part of the complete architectural design algorithm. Secondly, the architectural instantiation of the algorithm is examined. The goals targeted while using the swarm are sorted in four categories: generating an initial state, a space hierarchisation, structural elements and shapes. Architectural specifications such as the ones originating in the brief or the context and their implementation are also recorded. Finally, information pertaining to the technical set-up - type of programming interface, visualization software and programming software - and the organization chart - composition of team, role, background, programming skills - are collected. The sources of information used in the analysis are interviews with practitioners, public documents such as finalized drawings for the projects - including diagrams explaining the algorithmic structure - and documents found in public and private archives, in particular programming scripts, but also complementary sketches and documents throughout the genesis of the project.

## 4. Multi-Agent Systems as tailored algorithms : tacit-explicit tension and appropriation

As MAS blossomed in the 1990s, architects started resorting to them as well (Krause 1997, Coates & Schmid 1999), and these algorithms gained further momentum in CAAD during the 2000s. The emergent behavior of swarms and their ability to self-organise turned them into popular devices among architects of the computational field, as well as their capacity to handle complexity and the possibility they encapsulate to program matter (Pantic & Hahm, 2015). Users devised various applications based on swarms behaving according to diverse rules, from simple flock algorithms to simulations of magnetic fields (Andrasek 2009) or behavioral data (Schumacher 2018). More complex uses were also developed, such as algorithms combining several swarms together, or with other algorithmic

typologies. A few examples of the resort to MAS for architectural design can be given. The project Mesonic Fabrics, by biothing, associates a MAS with cellular automata in order to generate roofing structures. The Cliff House project, by kokkugia, is an experimentation tying MAS to the use of composite fiber architecture to develop a structural shell for a house on a extreme cliff topology. A third example is the Trabeculae / Protosynthesis project by supermanoeuvre, reimagining an office tower and its atrium by implementing two MAS, one shaped by light transmission requirements and the other creating a structural truss network. Finally, Living Morphologies (supermanoeuvre) revisits Le Corbusier's Unité d'Habitation with a MAS retro-engineering the circulation logic devised by the architect. These explorations of the potential of MAS for CAAD where accompanied by the development of tailored algorithms, as well as libraries developed by emblematic practices of the field, in particular supermanoeuvre, Kokkugia and biothing. This first period of appropriation of MAS by architects is characterized by the exploration of multiple architectural issues - structure, light, form, users' movements -, and a small-scale STN, depicted in figure 2. This phase is also marked by the high programming skills of MAS users, resulting in a skillfull negotiation of the tacit-explicit tension.

Practitioners generally draw on a form of subjectiveness and on their expertise to produce a relevant answer, in the form of an architectural object, to a given spatio-temporal context. The expertise built by architects throughout their years of training and practice relies on intuition as well as on tacit knowledge. The latter is a form of knowledge that cannot be formalised or rendered explicit by the person holding it and is therefore hardly transferable (Collins 2010). It cannot be put in writing neither given the shape of a set of instructions. Whereas tacit knowledge is instrumental to the practice of a discipline, given its inability to be formalised, it also renders specific disciplines very hard to automate and specific sets of knowledge very hard to transfer to computer-supported programs. This points out a key issue for architecture: it is first and foremost a practice, and should be analyzed as such, including in the understanding we have of how architects resort to algorithmic design tools. The fact that a computer relies on sets of explicit knowledge, in the form of instructions transmitted through programming languages, to execute series of calculations, not only makes it difficult to automate some practices, it also requires from architects using algorithmic tools to embrace a procedural understanding of the design process. This renewed understanding of the design process is nevertheless at the core of the extensive potential seen in algorithmic design tools and of the renewal and enrichment of architectural production in the last decades. While the superimposition of the practical dimension of architecture and the formal dimension of computation creates a negotiation between tacit and explicit, practitioners of the computational movements have since their dawn developed strategies to tackle it. The core STN associated with the development of MAS is the illustration of one strategy in particular, resulting in the appropriation of the typology.

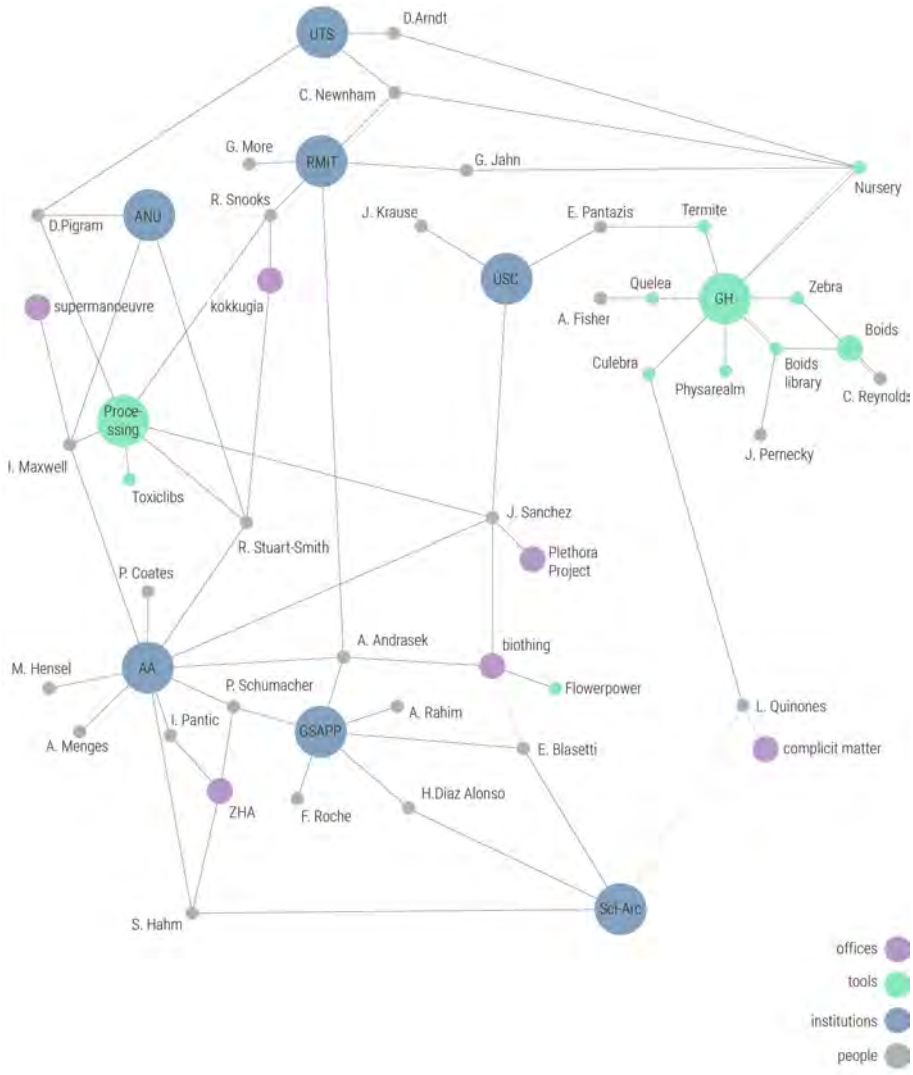



Figure 2. Core socio-technical network for multi-agent systems.

The assessment of the organization chart of each project, as well as the skills and areas of specialization of the participants and the role they played in the project, has highlighted an especially high level of mastery of programming skills for architectural practitioners. Furthermore, the practitioners also belong mostly in tools developing and algorithms scripting categories, rather than in ready-made algorithms using. In other typologies, such as evolutionary tools, the discrepancy existing between the knowledge profile of the tool developers and of the tool user results, because not accounted for, in the appearance of technical biases

(Gaudillière 2020). This can be the result of a resort to complex algorithmic typologies originating in other disciplines - such as biology for evolutionary tools - with a rapid democratization and the lack of an appropriation period through the development of tailored algorithmic design tools by and for users in the field of architecture. In the case of MAS, however, the practitioners of this first phase represent precisely this appropriation period, and despite the typology originating in other disciplines, algorithmic tools currently in use have been mostly devised by the practitioners themselves, resulting in the overlapping of developers and users competences.

The programming skills observed also ensue in a strong bias awareness. The combination of programming mastery and bias awareness results in a fully grasped negotiation between tacit knowledge and explicit instructions. Both the interviews and the analysis of the scripts attest to the mastery and understanding of technical set-ups and typological rules, enabling a conscient mobilization both of tacit knowledge and of these technical aspects in architectural implementation. Nevertheless, while the mobilization of MAS through both the genesis of libraries and the scripting of tailored algorithms relying on these libraries for each project confirms the appropriation of this algorithmic typology, the segmentation of project specific algorithms hints to a delineation of the process. The early version of the Flower Power plugin, relying on the previous and separated drawing of the plan, is an example. Another example is the description of the kokkugia libraries and their two-steps use in the practice : first devising MAS behavior libraries by expliciting instructions and secondly, choosing the library based on known - formal - results of the behavior and on a intuition regarding a specific project, to build a tailored script based on it. The practitioners observed thus recreate a classic mobilization of tacit knowledge, therefore demonstrating an appropriation of this algorithmic typology for architectural design.

## 5. Multi-Agent Systems as ready-made tools : interfaces for democratization

More recently, a second phase of MAS development in spatial design has begun, as this typology, alongside many algorithmic design tools, has initiated a massive democratization, with the appearance of numerous new tools designed for an easy manipulation of the typology (figure 3). The resort to MAS currently addresses three major fields: behavioral simulation for urban planning, self-organizing workflows and decision steering, and complex shape generation for architectural objects. While the first phase is characterized by one major category of users, the second phase, given its democratization dynamic, shows new types of users, with distinctive ways of mobilizing this algorithmic typology. The six Grasshopper plugins allowing for the use of swarms show a cumulated number of 56 636 downloads, an average of 8090 per plugin - Kangaroo Physics, the most downloaded Grasshopper plugin, displays a total of 443 415 downloads. While this already hints to a small community of users, active members of corresponding Grasshopper groups only represent less than 2% of this number, as well as published papers on the topic. Despite the difficulty to assess properly the number of regular users of such algorithmic typologies, these figures outline a key element regarding the field of MAS for CAAD in academia: the differentiation

between superficial and extensive uses. These elements suggest that, while the presence of Grasshopper plugins hints to attempts at democratising MAS through easier interfaces, the pool of both types of users remains small, in particular when compared to other typologies such as evolutionary tools, currently much more widespread (Gaudillière 2020).

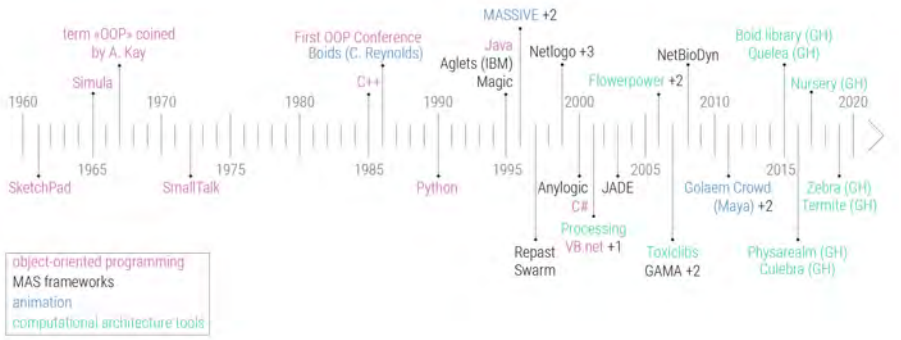



Figure 3. Chronology of multi-agent systems programming principles, frameworks, and tools.

To understand the mechanism of democratization for algorithmic design tools in architecture, a key element to study is the interface of tools, and its thickness. The notion of thickness of the interface acknowledges the varying distance existing between the initial architectural intention and the final representation that is the output of the algorithm, and is similar to the multiples *layers* of an interface that UX designers refer to (Masure 2014). The variation of thickness of the interface is intricately connected to the ability of the user to access rules and details of the mathematical model on which the algorithm is based and to the similarity of knowledge profile of users and developers. Therefore, in the first phase, as users are in their majority capable of scripting a tailored algorithm and to devise their own libraries and interfaces, the knowledge profile overlaps, as previously highlighted, thus providing almost zero-thickness interfaces of MAS. Interfaces and the layers forming them for both tailored algorithms and ready-made algorithmic tools, each typical from one phase, are depicted in figure 4. While tailored algorithms enable an access to rules in upper layers, and are composed of a small number of layers, ready-made tools are composed of many layers and the access to the programming of specific rules is located in lower layers, harder to access for users with little programming skills and leaving upper layers with only predetermined behaviors. Thus, the second phase of development of MAS is characterized by interfaces with high thickness, hardly enabling a detailed manipulation of the typology and hindering appropriation through tacit implementation, a step crucial to the use of MAS given the complexity of this typology, and therefore to their democratization.

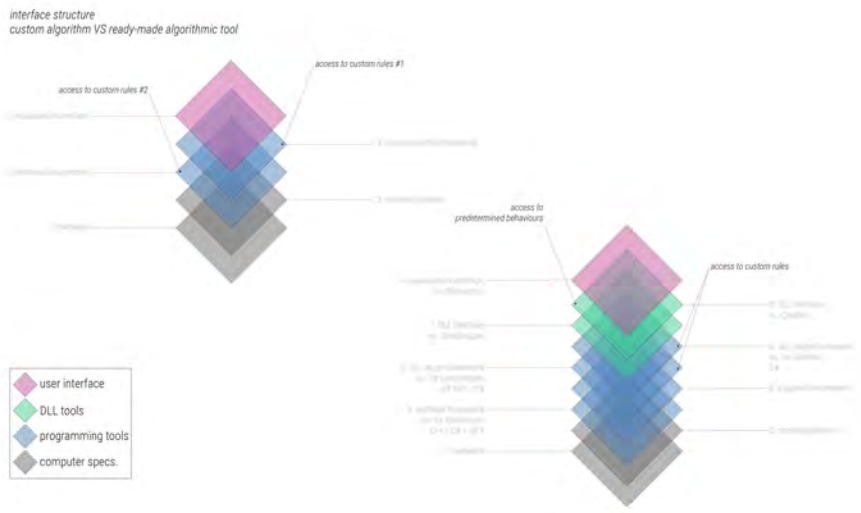



Figure 4. Interface layers for ready-made and tailored algorithms.

Of aforementioned strategies of negotiation between tacit and explicit depends the becoming or not black boxes of algorithmic design tools. By providing an easier resort to algorithmic design tools, interfaces display a black box mechanism (Latour 1987) : they become standard tools of architectural design, and users do not need any longer to question in detail their functioning and relevance, no need to scratch beyond the interface to fully grasp the complex mathematical, physical and informational models they are based on. While the simplification through an interface easier to manipulate is key to the democratization of algorithmic design tools in architecture, tactics of negotiation depend in a large part of the understanding of tools by their users, and on biases enabled by the structuration of algorithmic typologies, of tools and of interfaces.

## 6. Conclusion

Based on the assumption that socio-technical networks of computation in architecture exist and must be analyzed deeper in order to understand the impact of algorithmic tools on the design process, the present paper offers a foray into it, drawing on science studies methodologies. The research explores in what regard MAS are representative as much from the existence of these STN as from how their development influences the negotiation between tacit and explicit knowledge at play in procedural design processes. A methodology of analysis of these phenomena is provided as well as results of the application of this method to MAS, leading to a better understanding of their development and impact in CAAD in the past two decades. Tactics of negotiation shaped by early MAS users enable a double appropriation - borrowing an algorithmic typology from other fields to create tailored tools from it and managing to implement a classical architectural

practice by mobilizing tacit knowledge. Furthermore, their approach partially circumvents the establishment of technical biases tied to this algorithmic typology, at the cost of a lesser massive democratisation of the algorithmic tools developed in relation to this typology, displaying especially thick interfaces. MAS are an algorithmic typology that necessitates technical mastery, but in returns pushes a sensible use of computation in architectural design, as well as a diminution of epistemological and technical biases. These characteristics clash with the global contemporary trends of architects pulling away from technical issues (Picon 1989, Carpo 2011) and of computation tools as vectors of rationalization of architectural practice for the industry (Gaudillière 2020), further explaining the difficult democratization of this typology, despite what it has to offer.

## References

"Kokkugia" : 2004. Available from <https://kokkugia.com> (accessed 21/01/2019).

"supermanoeuvre" : 2007. Available from <https://www.supermanoeuvre.com> (accessed 21/01/2019).

"Plethora Project" : 2011. Available from <https://www.plethora-project.com> (accessed 21/01/2019).

"Food4Rhino" : 2019. Available from <https://www.food4rhino.com> (accessed 12/12/2019).

Andrasek, A.: 2009, *biothing*, HYX.

Carpo, M.: 2011, *The Alphabet and the Algorithm*, The MIT Press.

M. Carpo (ed.): 2012, *The Digital Turn in Architecture 1992-2012*, Wiley & Sons.

Coates, P. and Schmid, C.: 1999, Agent Based Modelling, *Architectural Computing from Turing to 2000 : Proceedings of the 1999 eCAADe*, 652-661.

Collins, H.: 2010, *Tacit and Explicit Knowledge*, University of Chicago Press.

Ferber, J.: 1995, *Les systèmes multi-agents*, InterEditions.

Gaudillière, N.: 2019, Towards an History of Computational Tools in Automated Architectural Design - The Seroussi Pavilion Competition as a Case Study, *Intelligent & Informed - Proceedings of the 24th CAADRIA Conference - Volume 2*, 581-590.

Gaudillière, N.: 2020, Biomimétisme & Optimisation : algorithmes génétiques, influence de l'interface & biais techniques dans la conception architecturale assistée par ordinateur, *Projeter l'architecture, aux carrefours du numérique et du vivant, ENSAPVS*.

Krause, J.: 1997, Agent Generated Architecture, *Design and Representation : Proceedings of the 1997 Association for Computer Aided Design in Architecture (ACADIA)*, 63-70.

Latour, B.: 1989, *La science en action*, La Découverte.

Latour, B.: 1991, *Nous n'avons jamais été modernes*, La Découverte.

Masure, A.: 2014, *Le design des programmes : des façons de faire du numérique*, Ph.D. Thesis, Université Paris-1 Panthéon Sorbonne.

Pantic, I. and Hahm, S.: 2015, Isomorphic Agency, *Emerging Experience in Past, Present and Future of Digital Architecture, Proceedings of the 20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia CAADRIA*, 178-188.

Picon, A.: 1989, Les ingénieurs et la mathématisation. L'exemple du génie civil et de la construction, *Revue d'histoire des sciences*, **42**(1-2), 155-172.

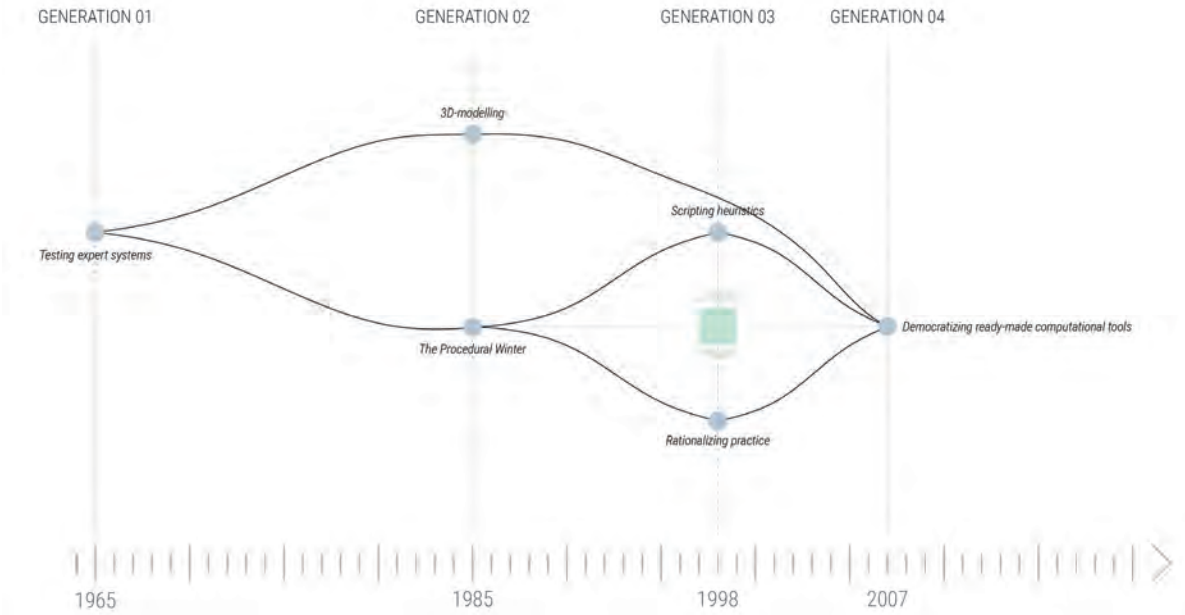
Reynolds, C.: 1987, Flocks, herds and schools: a distributed behavioral model, *SIGGRAPH '87 Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 25-34.

Schumacher, P. 2018, Freedom via Soft Order : Architecture as A Foil for Social Self-Organisation, *in* O. Hopkins (ed.), *AD Special Issue - Architecture and Freedom : Searching for Agency in a Changing World*, Wiley & Sons.

LEAVE THIS PAGE BLANK
DO NOT DELETE THIS PAGE

# AD Magazine : Mirroring the Development of the Computational Field in Architecture 1965-2020

**Nadja Gaudillière-Jami**
Technische Universität Darmstadt, DDU (Digital Design Unit), Department of Architecture.
GSA Laboratory, ENSA Paris-Malaquais.




1

## ABSTRACT

The present papers aims at contributing to an history of computational design and to an historiography of the field, by proposing a study of the development of sociotechnical networks of computation in architecture between 1965 and 2020 as shown in AD magazine. The research focuses on two aspects : a methodological approach for the constitution of a comprehensive history of the field and the application of that methodology to a corpus of items published in AD, and questions the relevance of the outlook into computational design as given by the magazine in comparison to a more comprehensive history taking into account other sources. First, the paper presents the history and the editorial line of AD magazine, as well as its pertinence as a primary source. Secondly, a brief account of the history emerging from this research is made, with a focus on four different periods : pioneering research of the 1960s-1970s, emergence of 3D modelling tools and *procedural winter* in the 1980s-1990s, constitution of a large-scale academic and professional network in the 2000s and democratisation of algorithmic design tools in the 2010s. Third, observations are made on editorial choices of the magazine and the biases of its account of computational research, with a special focus on the period 2000-2020, during which many issues have been dedicated to computational design themes, therefore making potential biases more visible. Despite the preponderance of specific topics, editors and contributors, AD magazine provides an outlook into key concerns of the community at given times. The main biases identified, including a strong focus on the themes of biodesign and rationalisation of practices, mirror the biases of the computational field itself, demonstrating the value of AD as an archive for the history of the field.

1 Chronology of the computational field in architecture 1965-2020.

## INTRODUCTION

The computational movements in architecture are characterized not only by the common resort to specific design tools - programming tools, but also by the conditions that enabled research on this topic to blossom : a specific socio-historical context of emergence. The computational movements organize around a series of practitioners, of research units and classes hosted by specific architecture schools and institutions, as well as around events such as exhibitions and conferences. Furthermore, computational movements in architecture are characterized as much by the technical aspects of the practice as by the structuration of the community of its practitioners, identifying it as a sociotechnical network (Latour 1987) still to be described.

In existing literature, several aspects of this sociotechnical network have been studied. In particular, the work of Mario Carpo must be mentioned as a recent historical and global overview of the computational movements in architecture (Carpo 2011, Carpo 2018). If numerous archaeologies of the digital in architecture already exist as well, including the eponymous book by Greg Lynn (Lynn 2013), they consist in their majority of an assembly of emblematic projects presentations and texts by the architects and designers of the computational field. Only a few studies of the development of specific digital tools exist (Serraino 2002, Smith 2017), focusing mainly on the period of the 1990s and early 2000s and on the development of 3D modelling. While these works all contribute to an understanding of past and present use of computational design tools in architecture, none provide a global study of the associated sociotechnical network as such.

In the sociological study of science and technics, a sociotechnical network is a system of social, cultural and economical links tying a given series of stakeholders together. In order to highlight the topology of a given network, one can follow specific technical objects, as is done in the present research with algorithmic design tools. Sociotechnical networks exist at different scales, from large state or societal-wide sociotechnical landscapes to smaller niche systems, such as the computational field, a microcosm also part of larger contemporary architecture related sociotechnical systems. The effect of sociotechnical networks on the trajectory of innovations in contemporary societies (1) and on our relationship to technical objects has been studied by several major thinkers of our time (Simondon 1958, Callon 1986, Latour 1987, Edgerton 2006). The study of such networks in the computational field can be of interest for two reasons. First, the ability of such study to unveil the articulations between communities of practices and algorithmic design tools as well as the reasons for which given tools are favored rather than others, be it technical, economical or cultural reasons. Secondly, while the amount and diversity of experimentations in the computational field makes it difficult to build a coherent historical account encompassing all of it, focusing on practices themselves by following algorithms through their sociotechnical networks enables the emergence of common threads out of this diversity, making a comprehensive history of computational movements in architecture possible. The first has been documented in (Gaudillière 2020a) and (Gaudillière 2020b), and the current research focuses on the second.

The present papers aims at contributing to a practical and epistemological history of computational design and to an historiography of the field, by proposing a study of the development of sociotechnical networks of computation in architecture between 1965 and 2020 as shown in AD magazine. The research focuses on two aspects : a methodological approach for the constitution of a comprehensive history of the field and the application of that methodology to a corpus of items published in AD in the past fifty-five years, and questions the relevance of the outlook into computational design as given by the magazine in comparison to a more comprehensive history taking into account other sources. First, the paper presents the history and the editorial line of AD magazine, as well as its pertinence as a primary source. Secondly, a brief account of the history emerging from this research is made, with a focus on four different periods : pioneering research of the 1960s-1970s, emergence of 3D modelling tools and *procedural winter* in the 1980s-1990s, constitution of a large-scale academic and professional network in the 2000s and democratisation of algorithmic design tools in the 2010s. Third, observations are made on editorial choices of the magazine and the biases of its account of computational research, with a special focus on the period 2000-2020, during which many issues have been dedicated to computational design themes, therefore making potential biases more visible. Despite the preponderance of specific topics, editors and contributors, AD magazine provides an outlook into key concerns of the community at given times. The main biases identified, including a strong focus on the themes of biodesign (2) and rationalisation of practices, mirror the biases of the computational field itself, demonstrating the value of AD as an archive for the history of the field.

## I. AD MAGAZINE, A COMPANION TO THE COMPUTATIONAL FIELD SINCE THE 1960S

### AD Magazine

In order to contribute to the mapping of sociotechnical networks of computation, the choice of sources for the

constitution of the corpus of practitioners, institutions and firms, projects, typologies and tools, conceptual approaches and references constituting the network is instrumental. AD magazine, given its history and editorial line, appears to be a particularly suitable source for the constitution of a primary corpus. The Architectural Design (AD) magazine was founded in 1930 and has since continuously been edited, first by Academy and later by Wiley. 12 numbers a year were released until the magazine switched to 6 numbers a year in 1987, with each number being the occasion for AD to invite a guest-editor focusing on a specific concern for architectural design of the time. AD magazine has represented for decades a key reference for many practitioners of the field, in particular for technological developments of the discipline, and since the dawn of the computational turn in architecture, it has been recognized at large as a reference for this specific area.

Compared to other print titles in architectural press, AD magazine has the advantage of having more in-depth contents than Architectural Digest, Dezeen or Archdaily, with a more rigorous editorial selection process for the contents than the two latter. It is an intermediary between titles focused on written theoretical essays such as Log and others focused on practical issues of construction such as Detail. Furthermore, AD magazine's editorial line presents an especially open mindset towards innovation and in particular towards computational design than many magazines, and this since early on in the digital age. As an example, reviews of the Seroussi Pavilion Competition and the exhibition that followed in the French press have been studied and show that as late into the computational turn as 2007, generalist architectural press in France was still very critical of anything that might involve experimenting with algorithmic design (Gaudillière 2019).

Furthermore, to the author's knowledge, no study of AD magazine as a resource on the history of computational design exists, despite the very complete archive of the development of the field it represents and the valuable inputs it could provide in several domains.

### Methodology

The methodology outlined here is the complete approach proposed for the constitution of a database then used as the basis for a comprehensive history of the field. As explained above, this methodology has been used for a global survey of the computational field. As part of this survey, AD magazine archives have been studied, The present paper describes the observations made based on the AD archives, and balances them with a comparison to the observations using the entire database and methodology. First, information has been gathered from various sources. AD magazine issues were the first consulted, thus creating a database listing every project mentioned in it that referred to computational design. This database was then completed using exhibition catalogs (3) and conference proceedings (4). Interviews were then conducted with a series of practitioners of the field, some of them also having given access to their private archives. Three types of objects were considered for the research : practitioners trajectories, design projects and design tools, all followed through the sociotechnical network in becoming and for each of which a series of information was gathered in the sources.

The corpus studied in order to establish the database consists of 456 AD issues, from the n°1 of 1965 to the n°6 of 2020 and 8 issues from the AD Reader series, accessed in the KADK (Copenhagen), ENSAPB (Paris) and ENSAMV (Paris) libraries. Of these 464 issues, 134 contained items relevant to the proposed study of the computational field [6] in architecture. 1398 projects cited in the articles and 139 written essays were identified and integrated to the database for further analysis. Four categories of items have been established, with specific sets of information gathered on each. First, the editorial team of AD magazine in place at the time of the release as well as the guest-editors have been extracted for each issue. Second, for projects, the practitioners and firms participating, the computational framework, the type of algorithm and the area of focus have been listed. Third, for studio classes featured, students, teachers, institutions, computational framework, type of algorithm and area of focus have been listed. Fourth, for written essays, author, institution, references, area of focus and theoretical position have been listed. Finally, the network is mapped out by establishing ties between practitioners and firms, practitioners and institutions - where each one studied and taught - as well as by documenting which project was created by whom using which type of algorithm, thus mapping generations and clusters of development throughout time.

## II. A BRIEF HISTORY OF THE COMPUTATIONAL FIELD IN ARCHITECTURE

Clusters of development forming the network throughout four successive generations have been observed, with each generation marked by the transmission of theoretical and practical knowledge from the previous and to the next, and by the extension of the network to further clusters. The starting point of this history is the constitution of the first clusters of research in computational architecture in the 1960s, and unfolds itself following the chronology depicted on fig. 1, with four periods and six major approaches to

the role played by algorithmic tools. While the history of the computational field in architecture as uncovered in the research described in this paper would deserve a much longer and detailed text, the brief description that follows is intended as a reference point regarding a few key issues - programming knowledge of practitioners, level of development of technical set-ups and algorithms, and network constitution - in order to further comment on the outlook into this history provided by AD magazine.

### Pioneers of computation

The first generation encompasses the work of pioneers such as Nicholas Negroponte, Cedric Price or the Frazers, with limited technical set-ups such as the one used at M.I.T. by the Architecture Machine Group (AMT) for the URBAN 5 project in 1973, resorting to room-sized processors, tiny cathodic screens and light pens (Steenson 2014). Those enquiries took place in only a few academic institutions (M.I.T., Bartlett School of Architecture, AA, UCLA, UTS), bringing together a handful of practitioners that relied on collaborations with computer scientists to develop not only programs in early languages such as FORTRAN, but also pioneering research in the field of computer science and artificial intelligence. In particular, alongside other research groups that would soon together become the M.I.T Medialab, the AMT contributed significantly to the development of verbal communication between a computer and a human in natural languages (Brand 1987). The primitive technical set-ups also entailed the traditional limits of computerized logic systems of the time, in particular the combinatorial explosion and the problem of commonsense knowledge (McCorduck 2004), on which architectural design heavily relies. At the time, practitioners of the computational field pursued their work in very close proximity to the field of artificial intelligence, in its early developments at the time. As a consequence, this period is also characterized by many enquiries on the nature of the design process - while Nigel Cross' texts are especially known to our field, he is not the only one to have been drawn from attempts at coding scripts enabling to automate design decision-making to a long-term reflection on the nature of the discipline - a characteristic of pioneering experimentations that would then fade with time.

### .The Procedural Winter

The second period is marked by a split in two parallel fields of development. On one hand, 3D-modelling tools appeared, first in the animation industry and then in architectural practices who started resorting to them to handle complex geometries - developing into the well known curvy shapes that would in part characterize the digital turn in architecture for many in later years. This is the most documented period and set-up, with both historical pieces on 3D-modelling softwares developed in specific firms and analysis of the impact of such tools on the practice (Serraino 2002) (Smith 2017). Following the appropriation of existing 3D-modelling tools such as Maya, it is the moment where the making of their own tools by architects has been established as a key marker of the digital field in architecture - with architects embracing 3D modelling and creating their own softwares, such as form*Z in Chris Yessios' company auto.des.sys or Digital Project in what would become Gehry Technologies. It is also the moment of appearance of a representation bias : as 3D modelling went on to develop in everyday architectural practice, digital tools would be for many be reduced to representation tools with affordances equivalent to pen and paper. The algorithms on which modelling is based were concealed behind the complex geometrical explorations the software enabled, and the procedural processes at large those algorithms enable were also shielded away.

And while 3D-modelling boomed, the research on algorithmic design processes entered a *procedural winter*, a term coined for the present research in reference to what computer scientists and historians of artificial intelligence call the *Winter of AI* (McCorduck 2004). Research in computational design slowed down, with the network of practitioners hardly expanding and little experimentations with scripting programs being conducted. This stagnation echoes the situation of the field of AI, where research also slowed and fundings considerably diminished, as researchers struggled to make significant progress in tackling major issues : limited computer power, intractability, Moravec's paradox. A few of the new practitioners working in the field of computational architecture kept trying to solve the main issues of decision-making automation and made a few unsuccessful attempts at producing industrialised expert system solutions for standard architectural practice. Most practitioners new to the field at the time, such as Bernard Cache and Patrik Beaucé, rather than looking into on these unloved and unfunded problems, focused on experimentations with digital fabrication and the associated theory of non standard fabrication, producing major texts of the theoretical framework of the following years (Cache 1995).

### The Golden Years

While research in computational architecture withered during the procedural winter, it did not disappear, and later years were marked by an exponential development of the network, in two different directions again. On one hand, the computational field developed into a more generalist and construction oriented practice. Engineering offices

such as Arup or Bollinger + Grohmann, in association with classic architectural practices such as Dominique Perrault Architecture or SANAA, started mobilizing algorithms for post-rationalisation of designs in order to build them, but also looked more and more into developing design tools enabling pre-rationalisation in order to save time and money. Large architecture offices such as Foster & Partners, Skidmore Owings & Merrill or Zaha Hadid Architects created small in-house research groups to look into the potential of computational design tools. The software industry also began to take growing interest in the field, first with Bentley Systems conducting research and proposing software packages such as Generative Components, followed by McNeel and Autodesk whose interest would ultimately lead to the integration of Grasshopper into Rhinoceros and of Dynamo into Revit.

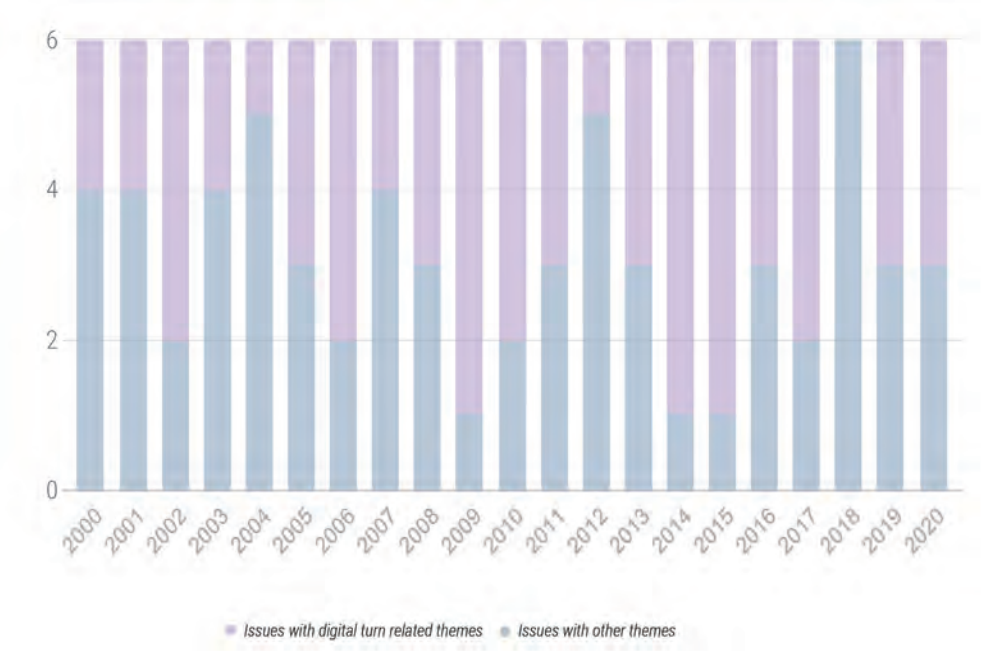
The network is at the time becoming denser not only through the appearance of these various private firms as new stakeholders, but also through the multiplication of university groups and master programmes dedicated to computational practices in architecture (such as CITA, Mediated Matter group or EmTech). Within these groups, researchers also maintained a professional practice, giving rise to a multitude of small experimental offices (such as biothing, EZCT, Xefirotarch, supermanoeuvre, kokkugia, R&Sie(n), OCEAN or theverymany). The activity of these practitioners is characterised by a fine-tuned mastery of programming, acquired for some of them in the course of complementary training to their architecture studies and always honed by regular practice. While these practitioners tend to use the same software and programming languages - in particular Maya/MEL and Rhinoceros/Python, very popular for the first before and for the second after 2007 - they create a custom algorithm for each new project. These algorithms are assembled from a patchwork of many specialised computer tools (sometimes up to 10 or 15) and are the marker of a genuine craft of programming. Moreover, these algorithms are often inspired by algorithmic structures developed in other fields (biology, animation), which are hijacked to be used for the production of architectural objects.

Much more than the first split, this one is happening by separating academy and industry in their goals and practices. At this point nevertheless, a lot of links remain between the two branches, constituting a dense network of exploration of the potential of computational tools, both in a heuristic and in a rational approach.

### Democratizing Computational Tools

The different groups of practitioners that have emerged over the generations all share an interest in computational tools and their potential for architectural design. While they have different visions, they are all involved in the transmission of the knowledge they have helped to develop and are committed to ensuring that their knowledge and vision is shared. Thus, in the most recent period of development of the computational field, it is the democratization of algorithmic design tools to a great number of practitioners that has become a key issue. However, the combination of the hijacking of algorithmic structures originating from other disciplines, of the search for a certain efficiency in professional computational practices and of the need for practitioners to be able to rapidly transmit their knowledge leads to the promotion of algorithmic typologies and the constitution of associated software, rather than to the transmission of an in-depth knowledge of a scripting craft for architectural design. These ready-made tools have in the latest period of development taken over from custom algorithms in the field (Gaudillière 2020a).

If some softwares enable the development by practitioners of custom libraries and extensions, ready-made software solutions enabling an easy use of algorithmic typologies popularized by previous generations are spreading rapidly, and allow computational practices to exit the initial network, with the appearance of many new users all over the world. The interfaces of these software are decisive components - they are what makes democratisation possible, by allowing amateur users to access and manipulate complex algorithmic models. As these models are increasingly based on algorithmic typologies, they make use of predefined rules that users can access to set up the model. Over the decades, from tailor-made algorithms to ready-made software solutions, the thickness of interfaces is increasing more and more, and the various parameters on which an algorithmic model is based are not always accessible easily : algorithmic design tools then become black box software. The users who are less familiar with programming techniques are therefore constrained in their use of the tools and guided by the tool developer and the structure he or she put in place. In the case of computational architecture, this also means being pushed to adopt the programmer's vision of the discipline and the role to be played by algorithms (Gaudillière 2020a). Neophyte users therefore tend to build a biased practice, influenced by the structure of ready-made software solutions and with little or no opportunity to question or modify these structures. Some typologies and the tools associated with them also predominate due to their ease of adaptation for amateur interfaces. Others, on the contrary, more difficult to adapt, see their use diminish (Gaudilliere 2020b). Some approaches to computational design are promoted through these black




2 Number of AD issues dedicated to computational design issues, 2000-2020.

2

box tools, in particular a rationalisation bias through which attempts are made of systematically anchoring architectural decision-making in scientific knowledge and scientific methodologies.

Several factors influence this rationalisation bias : first, the inheritance of a mechanistic bias, with a recent resurgence ; second, the split between industry and academy and the related structure of socio-technical networks in the field, leading to the promotion mainly of computational tools spreading a capitalistic approach ; third, the current democratisation dynamic is at work with little questioning of those black box tools and with little epistemological reflection on the nature of the discipline. This contributes to strengthening the rationalisation bias in the field, to the detriment of the diversity of practices that characterizes not only the computational field since its early developments, but architecture as a discipline itself.

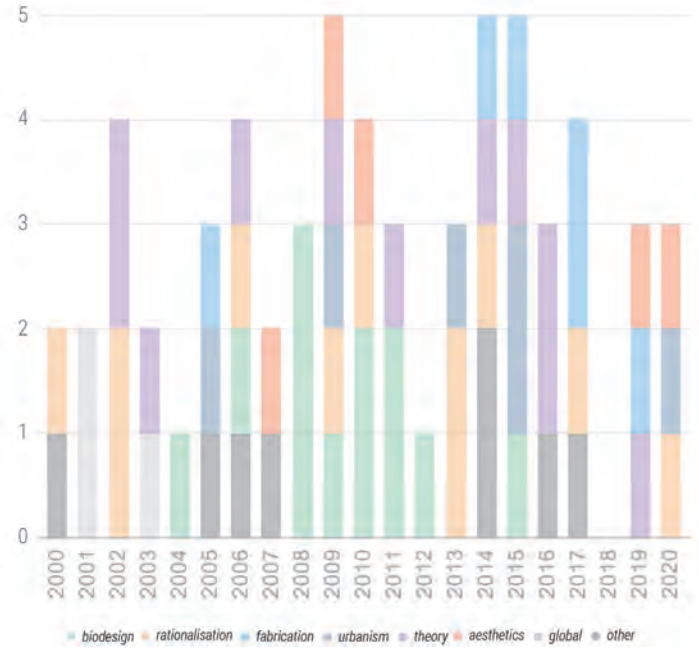
## III. AD MAGAZINE : A BIASED HISTORY?

Using AD magazine as a source has enabled us an overview of the history of the computational field in architecture, by contributing to the identification of key practitioners and projects over fifty-five years. But as a magazine with a given editorial line, AD has made choices of curation over the years, and in order to assess the accountability of the development of the field as based on AD issues as sources, the biases that might have been triggered by such choices must be evaluated. To assess this potential curation bias, in order to highlight nuances of AD magazine's editorial line, the AD corpus has been cross-referenced with the general study. And while the contents displayed in AD overtime overall fit the cross-referenced account made in this paper of the development of the computational field, editorial choices tend to highlight some practitioners and themes much more often than other. Examples of editorial choices and potential biases in the account of the experimentations in the computational field have therefore been studied for the 2000-2020 period. As this is the period in which a greater number of AD issues have been dedicated to topics relevant to the computational field, editorial choices and potential biases thus become more apparent.

During that period, AD magazine's outline has slightly evolved, but remains constituted of a selection of articles and projects presentations by the guest-editor, related to the issue's theme as well as a series of short fixed sections with regular writers, such as Craig Kellogg, Will McLean, Valentina Croci or Neil Spiller. The editorial team, lead by Helen Castle since 1999, has evolved as well over the years, and now includes several key figures of the computational field (5). Between 2000 and 2020, 126 issues of AD magazine have been issued, as well as 8 issues from the AD Reader collection. Of those, 62 issues of the magazine and 4 readers were dedicated to themes relating to the computational turn, as is shown on figure 2, almost half, with a slight increase over the years - this pointing out to the importance granted by the magazine to the stakes of computational practice in architecture.

On figure 3, themes of the computational-related issues throughout the years are examined. Eight categories have

3 Themes of AD issues dedicated to computational design issues, 2000-2020.




3

been identified :

- Biodesign themed-issues look at morphogenetic design processes, biomimicry, evolutionary algorithms or biomaterials. 12 have been released in the past twenty years, for example the 2004 *Emergence : Morphogenetic Design Strategies* issue or the 2015 *Material Synthesis: Fusing the Physical and the Computational* issue.

- Rationalisation themed-issues look at construction oriented practices. 11 have been released, for example the 2002 *Versioning : Evolutionary Techniques in Architecture* issue or the 2013 *The Innovation Imperative: Architectures of Vitality* issue.

- Theory-themed issues look at possible global keys for the understanding of the computational paradigm in architecture. 2 readers and 11 issues have been released, for example the 2011 *Mathematics of Space* issue or the 2019 *Discrete: Reappraising the Digital in Architecture* issue.

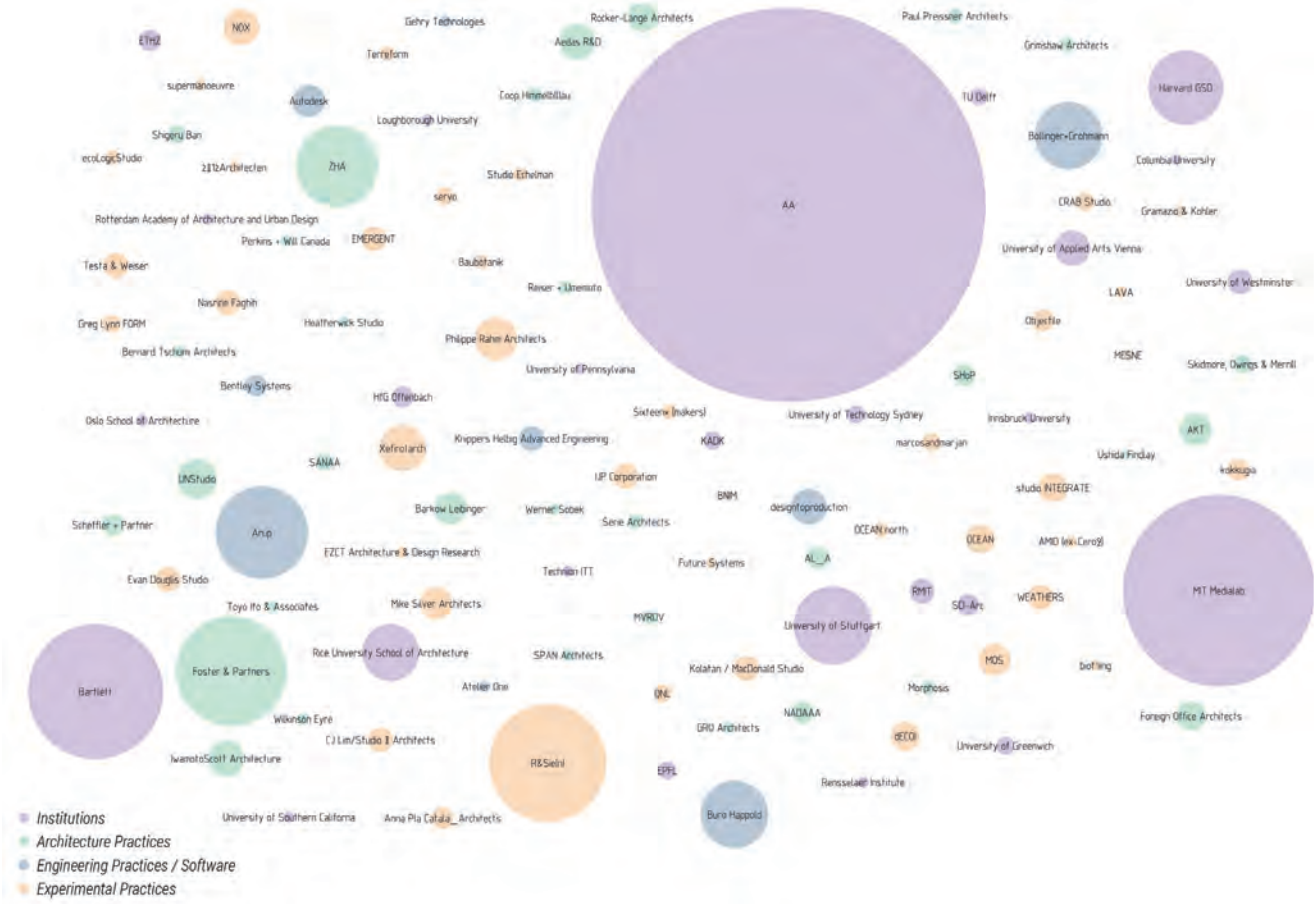
- Fabrication-themed issues look into novel fabrication methods such as robotic processes and into non-standard fabrication workflows and frameworks. 1 reader and 6 issues have been released, for example the 2014 *Made by Robots* issue.

- Urbanism-themed issues look into questions regarding smart cities and analysis and design tools for the city. 6 issues have been released, for example the 2020 *Urban Futures: Designing the Digitalised City* issue.

- Aesthetics-themed issues look into the potential of computational design techniques for a renewal of the formal complexity of architecture. 5 issues have been released, for example the 2010 *Exuberance* issue.

- Global issues (3 and a reader) offer an overview of the computational field as a whole at the time of publication, and other-themed issues (8 and a reader) regroup topics that are unrelated to the major themes identified.

As can be seen in the figure, some themes are represented twice as much as others. While it is understandable that theoretical issues would often be edited, as the computational field is in constant evolution and still in need of a global understanding of the diversity of experiments of the past decades, the large number of issues dedicated to biodesign and to rationalisation indicates an editorial choice of focus. The biodesign theme has had a particularly prolific period (2008-2013) and although the history laid out by AD does not fundamentally differ from the history documented with cross-referenced sources, the importance given to the subject as an all-encaspulating narrative outweights by far the other narratives that have played a role in the field at some moment in time. This can be explained by the fact that the morphogenetic narrative, as an inheritance of cybernetics and of the paskian architecture paradigm (Pickering 2009) is one of the historically most ancient and therefore well-accepted understanding




4 Most frequently mentioned firms and institutions in AD issues dedicated to computational design issues, 2000-2020.

prisms for the computational in architecture. The presence of numerous rationalisation-themed issues on the other hand accompanies the rise of the bias of the same name in the field we have described earlier.

On figure 4, the most often cited institutions and firms in computational design-themed issues of that period are shown - from 3 mentions for the smallest circles to 111 for the most cited (the AA School). On figure 5, the first tab shows the most frequently invited contributors, the number of papers they have authored in issues they have not edited and their background, and the second tab shows the repartition of all the contributors in computational-themed issues. The third tab shows the most frequently invited guest editors, as well as the number of mentions of projects they have led in all of the computational issues - including the ones they have themselves edited. It shows that some guest-editors have been invited for several issues, while other practitioners instrumental to the constitution of the field have never been invited. If taking into account both the issues edited and the contributions, out of 905 contributors and editors, 15 have been invited for a major feature in one issue out of 10, and 3 have been invited for a major feature in one issue out of 5. Furthermore, some guest-editors tend to feature much of their own work or works by close collaborators in issues they edit, also distorting the overview given by AD magazine, as can be seen in the number of mentions column. This highlights the existence of an "AD network" of practitioners and curators, hinting to the fact that being featured in AD magazine or invited as a guest editor is guided by a system of co-optation - a phenomenon not rare in the world of edition and curation, but that still weights on the accuracy of the narrative of the computational field as offered by AD.

Overall, despite a tendency to always put the same people forward, AD nevertheless remains a mirror of key concerns of the time in the computational field. At some periods, it has taken the pulse of the field very accurately - in particular two issues looking into open-source match very closely the concerns expressed by practitioners interviewed about the period of release of these issues - and at some other periods it has been a more loose account. On the topic

| Contributor | Number of contributions*** | Background |
|---|---|---|
| Mark Burry | 9 | Architect |
| Patrik Schumacher | 9 | Architect |
| Michael Weinstock | 8 | Architect / Academic |
| Greg Lynn | 7 | Architect / Academic |
| Neri Oxman | 7 | Academic |
| Ben van Berkel | 6 | Architect |
| Mario Carpo | 6 | Academic |
| Peter Cook | 6 | Architect |
| Francois Roche | 6 | Architect |
| Dennis Shelden | 6 | Architect / Academic |
| Benjamin H. Bratton | 5 | Academic |
| Manuel DeLanda | 5 | Academic |
| Hernan Diaz Alonso | 5 | Architect / Academic |
| Mark Garcia | 5 | Academic |
| Achim Menges | 5 | Architect / Academic |
| Antoine Picon | 5 | Academic |
| Philippe Morel | 5 | Architect / Academic |
| Leon van Schaik | 5 | Academic |
| Philip F Yuan | 5 | Architect / Academic |

| Number of contributions | Number of contributors |
|---|---|
| 1 | 735 |
| 2 | 97 |
| 3 | 31 |
| 4 | 11 |
| 5 | 10 |
| 6 | 7 |
| 7 | 3 |
| 8 | 1 |
| 9 | 2 |
| 10 | 3* |
| 11 | 1* |
| 16 | 2* |
| 19 | 1* |
| **Total contributors**** | 905 |

* Contributors for a fixed section

** Including contributors co-authoring papers

*** In issues other than the ones edited by the contributor

| Editor | Computational issues edited | Total number of issues edited | Number of mentions | Background | Part of the editorial board |
|---|---|---|---|---|---|
| Achim Menges | 7 | 7 | 51 | Architect / Academic | No |
| Michael Hensel | 5 | 7 | 36 | Architect / Academic | Yes |
| Neil Leach | 4 | 4 | 7 | Curator | No |
| Bob Sheil | 4 | 4 | 7 | Academic | No |
| Ali Rahim | 4 | 4 | 17 | Architect | No |
| Lucy Bullivant | 3 | 4 | 0 | Curator | No |
| Mark Garcia | 3 | 4 | 0 | Academic | No |
| Neil Spiller | 3 | 5 | 4 | Academic | No |
| Michael Weinstock | 3 | 3 | 28 | Architect / Academic | No |
| Mark Burry | 2 | 2 | 13 | Architect | Yes |
| Richard Garber | 2 | 2 | 3 | Architect | No |
| Christopher Hight | 2 | 2 | 6 | Designer / Academic | No |
| Hina Jamelle | 2 | 2 | 4 | Architect | No |
| Leon van Schaik | 2 | 2 | 3 | Academic | Yes |

5 Most frequently invided contributors and guest-editors of AD issues dedicated to computational design issues, 2000-2020.

of computational design tools and their control as well, AD mirrors the vision of the field, encouraging black box tools by providing little technical detail about the projects featured and accompanying a wave of democratization sustaining a rationalist approach.

## CONCLUSION

The study has allowed for the creation of a database listing all the items referenced in AD as well as the information gathered on them. General observations on the genesis of the field have been extracted, as well as information on instrumental contributions by practitioners and institutions, on the first appearance of the various algorithmic typologies and on their frequency of use throughout time, showing the historical development of the resort to those. An account of the development of the computational field is provided based on this information, showing the four stages of development of the network and the impact of the modalities of transmission of knowledge throughout time and throughout the network. Furthermore, the AD history is compared to the global, cross-referenced network history, enabling a study of the curation bias entailed by AD's editorial line and choices. Finally, this database could be used in various research projects either as an overview of the existing contributions to specific issues of the field, or by further exploiting the referenced data. As an example, the present research has focused on mapping the development of the network through its projects and tools, but much remains to be said on the areas of focus, the writings and sets of references used by the practitioners in their essays and the epistemological ramifications of the computational field they render visible, thus contributing to a theoretical history of the computational turn.

## ACKNOWLEDGEMENTS

The author would like to thank the Architecture of Order Research Cluster and the Hessian State Ministry of Higher Education, Research and the Arts: State Offensive for the Development of Scientific and Economic Excellence (LOEWE) for supporting this research.

## NOTES

1. In particular, the stakeholders involved in a given sociotechnical network influence, depending on their interests, the adoption and promotion of given technical artefacts and the disregard of others.
2. With changes of names to call this field across the period, from "morphogenetic design" to "biodesign", but a common interest in natural phenomenon and materials..
3. *The Gen(h)ome Project, Naturaliser l'architecture, Architectures Non-Standard, Archilab, scriptedbypurpose, L'architecture au-delà des formes*, as well as archives of the Canadian Center for Architecture and of the FRAC Centre.
4. RobArch, Design Modelling Symposium, ACADIA, eCAADe, CAADRIA.
5. Members of the editorial board are currently Will Aslop, Denise Bratton, Paul Brislin, Mark Burry, André Chaszar, Nigel Coates, Peter Cook, Teddy Cruz, Max Fordham, Massimiliano Fuksas, Edwin Heathcote, Michael Hensel, Anthony hunt, Charles Jencks, Helen Castle, Jayne Merkel, Mark Robbins, Deborah Saunt, Leon van Schaik, Patrik Schumacher, Ken Yeang, and Alejandro Zaera-Polo.

## REFERENCES


Brand, Stewart. 1989. *The Medialab : Inventing the Future at M.I.T.*, Penguin Books.

Cache, Bernard. 1997. *Terre Meuble*, HYX.

Callon, Michel. 1986. "Éléments pour une Sociologie de la Traduction: La Domestication Des Coquilles Saint-Jacques Et Des Marins-Pêcheurs Dans La Baie De Saint-Brieuc." *L'Année Sociologique* , vol. 36, pp. 169–208.

Carpo, Mario. 2011. *The alphabet and the algorithm*, The MIT Press.

Carpo, Mario. 2017. *The Second Digital Turn : Design Beyond Intelligence*, The MIT Press.

Cross, Nigel. 1977. *The Automated Architect*, Pion.

Cross, Nigel. 2006. *Designerly Ways of Knowing*, Springer,.

Edgerton, David. 2006. *The Shock of the Old: Technology and Global History Since 1900*, Oxford University Press.

Eisenman, Peter. 1984. "The End of the Classical: The End of the Beginning, the End of the End", *Perspecta*, Vol. 21, pp. 154-173.

Gaudillière, Nadja. 2019. "Towards an History of Computational Tools in Automated Architectural Design - The Seroussi Pavilion Competition as a Case Study", in M.Haeusler, M. A. Schnabel, T. Fukuda (eds.), *Intelligent & Informed – Proceedings of the 24th CAADRIA Conference - Volume 2*, Victoria University of Wellington, Wellington, New Zealand, pp. 581-590.

Gaudillière, Nadja. 2020a. "Evolutionary Tools and the Practice of Architecture: from Appropriated Typology to Becoming the Black Boxes of CAAD", in Estevez, A. (ed.), *Proceedings of the 4th International Conference of Biodigital Architecture & Genetics*, iBAG-UIC Barcelona.

Gaudillière, Nadja. 2020b. "Computational Tools in Architecture and their Genesis : the Development of Agent-based Models in Spatial Design", in D. Holzer, W. Nakapan, A. Globa, I. Koh (eds.), *RE: Anthropocene, Design in the Age of Humans - Proceedings of the 25th CAADRIA Conference - Volume 2*, Chulalongkorn University, Bangkok, Thailand, 5-6 August 2020, pp. 497-506.

Latour, Bruno. 1987. *Science in Action*, Harvard University Press.

Lynn, Greg. (Ed.). 2013. *Archaeology of the Digital*, Sternberg Press.

Masure, Anthony. 2014. *Le design des programmes : des façons de faire du numérique*, Ph.D. Thesis, Université Paris-1 Panthéon Sorbonne.

McCorduck, Patricia. 2004. *Machines Who Think: A Personal Inquiry Into the History and Prospects of Artificial Intelligence*, Taylor & Francis.

Pickering, A., "Gordon Pask: From Chemical Computers to Adaptive Architecture", in Pickering, A., *The Cybernetic Brain. Sketches of Another Future*, University of Chicago Press, 2009.

Serraino, Pierluigi. 2002. *History of Form*Z*, Birkhauser.

Simondon, Gilbert. 1958. *Du mode d'existence des objets techniques*, Editions Aubier-Montaigne.

Smith, Robert. 2017. *Fabricating the Frank Gehry Legacy: The Story of the Evolution of Digital Practice in Frank Gehry's office*, CreateSpace.

Steenson, Molly W. 2014. *Architectural Intelligence : How Designers and Architects Created the Digital Landscape*, The MIT Press.


## IMAGE CREDITS

All drawings and images by the authors.


**Nadja Gaudillière-Jami** is a Ph.D. researcher at the GSA Laboratory (ENSAPM), a researcher at the Digital Design Unit (TU Darmstadt), co-founder of XtreeE and Vice-President of the NGO thr34d5.

# The Optimisation Game
# Computational Architecture and AI: from Symbolist Case Study to Machine Learning Data

Nadja Gaudillière-Jami, Technische Universität Darmstadt, DDU (Digital Design Unit), Department of Architecture

**Keywords.**

**A**rtificial Intelligence, History of the Computational Field, Rationalisation, Optimisation, Symbolism, Connectionism

**Abstract**

In the 1960s, the first experiments in computational architecture emerged, allowing for novel issues on the automation of the design process to blossom. The first nodes of the network that appeared at this time were part of the field of artificial intelligence (AI), a field with which computational architecture thus forged very strong ties as they both developed. While research in computational architecture was then largely concerned with the creation of new computer systems, it also forged institutional and intellectual links with the field of AI in addition to these technical collaborations, sharing the same campuses, the same sources of funding and the same research questions. The 1960s were a pivotal moment for these two fields, which theorized and popularized a symbolist approach to the modeling of the design process and of the mind. The following decades, however, marked a rupture: research in AI was struck by two winters (McCorduck 2004), periods during which the symbolist approach was strongly criticized. At the same time, the field of computational architecture emancipated itself from the field of AI, and extended beyond the academic domain to other professional spaces: engineering offices, architectural firms and software publishers became increasingly interested in algorithmic tools. This independence was accompanied by a change in the practices of the computational field, which moved from experimental heuristic programming practices to a new design rationality. Finally, more recently, new practices in AI, in particular machine learning techniques, have initiated a renewed strengthening of the links between the two fields, with tools as vectors. These new practices are also overturning theories: the use of machine learning marks an abandon of the symbolist approach and a return to the connectionist paradigm (Cardon et al. 2018), the repercussions of which have yet to be understood in the field of computational architecture if the resort to machine learning techniques continues to develop. The historical overview of the ties between the fields of AI and of computational architecture offered in the present research aims at highlighting some of the key theoretical principles computational architecture has inherited from its parent field, and the consequences of this inheritance, from the promotion of a new design rationality through optimization tools to the interrogations that the necessary data fragmentation in machine learning tools triggers.

## I. Introduction

In the middle of the 1960s, while the computer revolution raged across American campuses, the windowless rooms full of electronic equipment also hosted a handful of researchers diving into computerizing architecture. Not digitizing it, as we do these days when we scan entire buildings or forests to use this data as basis for designs, but computerizing it: rendering it intelligible to a computer, enabling the automatization of the design process. This was not an easy task, for the design process holds intangible parts, that despite being identified by many practitioners, from Christopher Alexander (1964) to Nigel Cross (1982), resist formalization. But it is precisely the difficulty of the task that put pioneers of the field up to the challenge: if they found a way to solve architecture, then certainly the mighty all-encompassing algorithms making artificial intelligence possible would not be far out of reach anymore. Logic theorems, chess and word processing were indeed not the only things AI researchers were looking into - architecture and design were prized case studies in their work, leading them early on to team up with architects within the pioneering institutions developing cutting-edge computer science - from MIT to Berkeley and Cambridge (UK).

As they collaborated to break down the intricacies of designing a gate, a house or a city into written instructions, relationship matrices or organizational diagrams, practitioners led the foundations of both fields while devising a common technical and theoretical project: formalizing the intangible process of thought at work in design and in the human mind. The computational field in architecture as we know it today is rooted in this project, and in shared research with the field of artificial intelligence (AI), the first to spark the network that would eventually

evolve into our many academic groups, education structures and professional practices. The technical, institutional and philosophical ties between the two fields thus played a strong role in defining how we currently envision the role to be played by computational tools in architecture.

Several decades after these early endeavors into the automatization of design, a larger circle of practitioners and researchers have adopted algorithmic design techniques, from multi-agent systems to genetic algorithms. The promotion of these techniques for optimization purposes and the development of ready-made tools enabling their spread in mainstream architectural practices has led computational practices from heuristic programming to a new design rationality for the built environment. The consequences of this change are still unfolding but echo a similar shift in the field of artificial intelligence, influencing the conditions of development of contemporary AI techniques and in turn their applications in the field of computational architecture. Beyond documenting the early ties between the two fields, we must thus also interrogate their contemporary status and dive into both their promises and shortcomings in regard to the nature of the task as defined by early AI and computational architecture researchers, as well as in regards to the political and social ambitions of architecture at large.

## II. Artificial intelligence, computational architecture: a joint birth

*A blossoming research network*

Before the rise of computer graphics in the 1980s and the ensuing growing resort to computers in professional environments, the computational architecture bubble, initially mainly academic research, started with only a handful of pioneering practitioners. Mostly in the English-speaking world, a small number of researchers devoted themselves to the topic, gathering around them small groups of colleagues and students. At the Bartlett School of Architecture, Ben Hillier spent most of the 1970s and 1980s developing space syntax with his team. From the end of the 1970s, the space syntax system was emulated by many in the small world of computational architecture, in particular Michael Benedikt, who developed isovists at the University of Texas. At UCLA, William Mitchell, director of the architecture faculty, contributed through his teaching and publications to promoting computer-aided design in the broadest sense and to establishing an early habit of using computer tools in the Californian architectural ecosystem. He later became the director of the MIT School of Architecture, and his presence and advocacy work helped establish it as a leading authority on computational trends in architecture. At Harvard and then at Berkeley, Christopher Alexander played a major role in establishing the computational field, particularly through his writings. Allen Bernholtz, who had founded the Laboratory for Computer Graphics and Spatial Analysis during his stay at Harvard between 1965 and 1972, used several of Alexander's programs in housing studies to test them, and continued research there after his colleague left for Berkeley. At the University of Technology in Sydney, John Gero founded the Computer Applications Research Unit in 1968, which he directed until 2007.

Among these seeds to our contemporary computational architecture field, the notorious Architecture Machine Group (AMG) founded in 1967 by Nicholas Negroponte and Leon Groissier had for initial ambition to provide architects with computer tools to simplify their design work, something they believed could only be achieved through a strong interactivity between machine and human (Negroponte 1969). At this period, given the early state of the art, research in the field of AI dealt mainly with designing computers themselves, and the projects of the AMG were no exception. The URBAN 5 project for example became one of the pioneering programs for verbal interaction with a computer: more similarities can be found with ELIZA, a famous conversational program of the same period, than with other architectural projects of the time (Weizenbaum 1966). In 1985, the AMG was integrated into what became the MIT Medialab, bringing together all the pre-existing AI research groups, and at the time of this merger, its work no longer had much to do with architecture, developing instead various media systems such as the Media Room or the Aspen Video Map (Brand 1987). Between 1967 and 1985, however, a core group of practitioners emerged within and around the AMG, joining the interest expressed in his texts by Nicholas Negroponte for the development of what he called architecture machines - a materialization of the automation of architectural design.

The work that emerged in England in and around the Architecture Association (AA) in London was to play an equally important role. In the early 1960s, the interest of Cedric Price in cybernetics and of Gordon Pask in art and architecture led to a famous collaboration: the Fun Palace. Between 1962 and 1969, a few students took a particularly strong interest in the themes promoted by Pask and Price in their teachings at the AA: Julia Frazer, John Frazer and Paul Coates. A few years later, in 1976, the Generator project would lead the four first to collaborate together. At the same time, John Frazer, Julia Frazer and Paul Coates took up teaching positions at

the AA and at UEL, and they were to maintain regular exchanges and collaborations in the decades that followed, notably for the creation of Autographics. And if these London experiments are not based around a research group as such, and therefore went on without an official operating and funding structure, they nevertheless brought together a group of recurring practitioners around a set of projects as founding for computational architecture as those of the AMG. Among those, the long-term *Morphogenesis* project, developed by John Frazer between 1989 and 1996 at the AA with the help of Gordon Pask, did promote some of the ideas that still live on in the computational community nowadays - buildings growing, evolving as biological organisms. But it also looked into novel ways of communicating with computers, relying not only on the development of new software, but also on interfaces that were in development since early in the 1980s at the school, in particular a system of cubes with embedded processors that could be used to communicate patterns to the computers, patterns that could then assume a variety of roles, from structural system to relations between different rooms.

Conversational programs, experimental interfaces and media systems, heavy program scripting, pioneering computer graphics software - the need for technical progress in computer science had the early researchers of computational architecture contribute a great deal to the development of computer technology itself, as much as they did contribute to a novel paradigm for architectural design. These contributions were made possible by the long-term involvement of these practitioners, and by the strong institutional ties and collaborations with computer science departments of the time they were able to foster.

*From institutional to intellectual proximity*

Not only was the AMG part of the M.I.T research initiative on artificial intelligence, it differed only little from other groups dealing with computer science. It was multidisciplinary, but had more computer scientists than architects among its permanent members and students. Many of the pioneers cited worked in collaboration with computer scientists and technicians (Allen 2020), sometimes even with the same programmers for years, such as Christopher Alexander's long-term association with Greg Bryant (Bryant 2013), leading without doubt to numerous conversations. The geographical proximity with the great names in AI research also favored exchanges: Allen Newell and Herbert Simon shared their time between Carnegie Mellon and RAND in Boston, the latter also hosting Hubert and Stuart Dreyfus and also having offices in California; Marvin Minsky, Seymour Papert and John McCarthy worked at MIT - all research centers that also hosted some of the pioneers of computational architecture. Many of the publications written about computational architecture at the period were published in journals and conferences devoted to computer science research, in particular the conferences of the Association for Computer Machinery (ACM), where John Gero, Michael Benedikt or Charles Eastman and their students regularly contributed. The institutional proximity of the time is also visible in sources of funding, which are identical to those for research on artificial intelligence: DARPA, RAND, and private sources on the model of corporate funding adopted by MIT. The projects funded ranged from the early attempts of the AMG to develop Information and Communication Technologies (ICT) and programs to facilitate architectural design, to later research by John Gero on the automation and optimization of design processes and to systematic spatial description systems developed by Michael Benedikt and Ben Hillier.

The early computational experiments in architecture appeared at a pivotal moment in the history of artificial intelligence : the shift from cybernetics, that had birthed ideas aiming at the construction of an electrical brain based on the first neurological observations of electrical communication between neurons in the brain between the 1930s and the 1950s, to the symbolist approach in artificial intelligence, that had its golden years between 1956 and 1974 and amounted intelligence to logic reasoning (McCorduck 2004). Geographical proximity and shared institutional spaces existed with the field of artificial intelligence, but also with its prior field of cybernetics. The graph presented in Figure 1 not only shows the various practitioners and the main research clusters they are part of, but also the connections between the field of computational architecture and those of cybernetics, of artificial intelligence and of design research[1], all emerging at the same period and interrogating the same issue of modelling society and the human mind. By mapping these interactions, the proximity between them can be observed, in particular the shared institutional network in the 1960s to 1980s. And beyond this institutional proximity, an ideological proximity is also built at the time, with many cyberneticians and researchers in artificial intelligence (in purple) strongly influencing the pioneers of computational architecture (in green) and acting as links between the various research clusters.

[1] Given the theoretical issues at hand at the period the ties to the design theory community are mapped on the figure. However, as for the artificial intelligence field, the entire community is far from being represented, as the graph mainly focuses on the field of computational architecture itself, looking into how it reaches out to other fields sustaining its institutional and intellectual development.

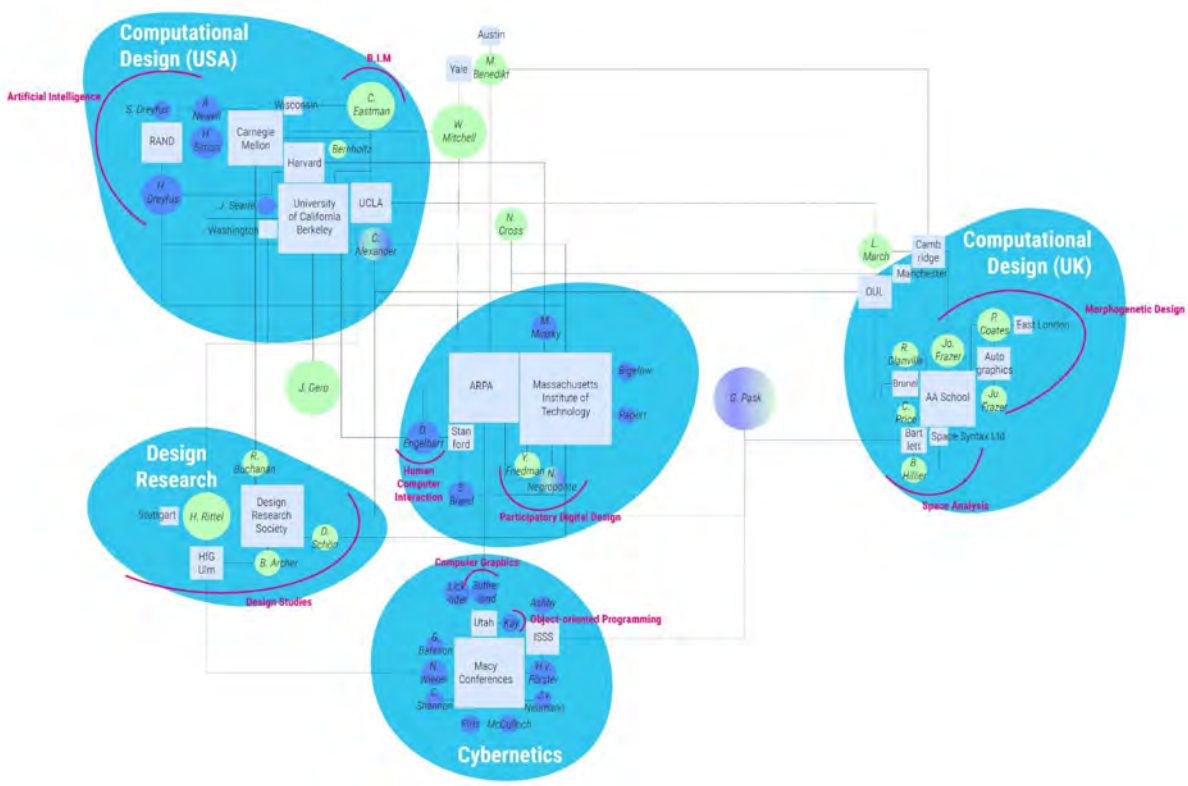



*Figure 1 - Network of researchers and institutions at the period of emergence of the fields of artificial intelligence and computational architecture*

The influence of Gordon Pask, but also of several other cyberneticists of the time, first provided a systemic vision offering many advantages for the establishment of architectural procedural processes, including the social dimension that cybernetics aimed to consider in its models. Pask's influence is thus felt at the AA and at the AMG, where Nicholas Negroponte called on him to discuss several projects (Pickering 2009). Paul Coates drew heavily on the vision of the American anthropologist and psychologist Gregory Bateson, who specialized in questions of communication. Ben Hillier drew on the seminal work of Norbert Wiener, exploring the applications of the social dimension of cybernetics to the procedural transformation of architecture, and Christopher Alexander was guided in the development of his patterns by the work of the English psychiatrist and engineer William R. Ashby, a pioneer of automation. The paradigm theorized by their cybernetic counterparts was then seen by pioneers of the computational field as particularly appropriate for the development of automated architecture, leading them to draw on it for their research. It fostered a vision of architecture as a system, paving the way for it to become a relevant case study of artificial intelligence research. The links between the field of artificial intelligence and computational architecture that emerged in the 1960s were thus built around a common fascination for the human mind and around a collective effort to try to understand and model it.

The first practitioners of the computational field in architecture found a strong echo to their research among their computer science colleagues. Christopher Alexander's work, in particular, was deemed of interest. His *Notes on the Synthesis of Form* (1964) were required reading for many years after their publication in American computer science curricula. His next book, *A Timeless Way of Building* (1979), was also widely read by the American computer science community, not only in academic circles, but also in the entrepreneurial networks of the then emerging Silicon Valley - all of whom promoted the book as a reflection of their own influence (Bryant 2013). However, Alexander also had influence over the development of computer science, since the patterns he developed throughout his career played an important role in the emergence of object-oriented programming, which in turn had a lasting influence on the programming structures adopted in the computational field of architecture, contributing here too to a ping-pong like back-and-forth between the two disciplines during their development periods.

At their birth and during their avant-garde period, artificial intelligence and the computational field in architecture shared a common theoretical and technical project, supported by strong technical, institutional and philosophical ties. Sustained by early successes in their research, pioneers of the field of AI quickly claimed they would be able to produce truly intelligent programs imminently. But these ambitions were soon contradicted by repeated technical failures, and the identification of major obstacles. Early AI research of the 1960s indeed relied on the computational theory of the mind, a symbolist approach that amounts intelligence to the manipulation of symbols through logic reasoning, a process that can be shaped into computer programs. But such theory implies the description of every single element and relation involved in the making of a decision and in the performing of an action. In order for a program to dictate to a computer how to behave like a human, it would need to describe an enormously vast amount of information. Finding all-encompassing algorithms describing how to adapt to any situation as humans are able to do proved much more difficult to overcome than anticipated by the supporters of the symbolist approach. The failure to deliver on their promises resulted in the loss of institutional support and the withdrawal of funding for their research by the military and the DARPA. The symbolist approach to AI of these pioneers was further put in question by philosophers such as John Searle (1980) or Hubert Dreyfus (1972), that argued there is no proof that logical reasoning might truly be the sign of intelligence given its functioning in isolation from the world and vigorously criticizing the failure to consider the role played by perception and situatedness in human thinking of such theory[2]. This wave of technical, institutional and philosophical distrust towards the field of artificial intelligence triggered what is now called the first winter of AI, between 1974 and 1980, a period of great weakening of the field that profoundly changed not only AI itself and its theoretical and technical project, but also the links entertained by the world of computational architecture to this associated field, links that had so far been foundational.

## III. Optimization, fragmentation: the foundations of contemporary practices

*Institutional ties gone by*

As time went by, the computational field in architecture went through several phases of development. After the blossoming of the first years, computational experimentations went quiet for a while, as funding underwent the same difficulties as for the rest of the field of AI during its winters. But the digital in architecture kept developing, as computer graphics and later associated software boomed and spread across professional practices in the 1980s and 1990s. In parallel, the computerization of architecture kept being researched by smaller groups of practitioners, confronted to the same technical issues as their AI colleagues. But this period of practical difficulties was also a time of theorization, during which key writings were produced. And despite the obstacles, the field kept developing its network in the decades following its initiation in the 1960s (Gaudillière-Jami 2021). By the 2000s, the field had become a key area of development for architectural practice at large. Not only were CAD-software standard equipment for any architectural practice, but computational research also structured itself as a full-blown topic, with many subfields of specialization, from biodesign to architectural robotics. By then it featured all the characteristics of a traditional academic field, from practitioners' associations such as ACADIA and regular conferences such as RobArch or CAADRIA to academic journals such as the IJAC, and the issues tackled frequently echoed in more mainstream publications such as AD magazine.

Players of such an academic network evidently include research units such as the Block Research Group at the ETHZ, the ICD at the University of Stuttgart or CITA at the Royal Danish Academy and dedicated courses such as the ones provided by the AA DRL or the IAAC. But for the computational field to become the network it is today, it took complementary players - for architecture is not only an academic discipline and for the popularization of algorithmic tools in design practices would take more than academic researchers. Engineering practices such as Arup or Bollinger + Grohmann developed an interest in computational design techniques early on, developing tools for post and pre-rationalization. Large architecture practices such as Zaha Hadid Architects or Foster+Partners also started hosting internal research groups diving into experimental digital and computational practices. Software editors such as McNeel or Autodesk, looking for new potential tools and markets, also took an interest in the field and worked towards expanding available software from computer-aided drawing to algorithmic design. Conferences and workshops provided places of meetings for these diverse players, in traditional academic formats hosted by universities but also in hybrid associations such as Smart Geometry. Thus, not only did the academic network of computational architecture broaden in the decades following the winters of AI, it also merged with a professional network looking into the same issues of automatizing parts of the design process.

[2] For a detailed account of the history of artificial intelligence, see (McCorduck 2004).

As the computational field in architecture underwent its structuration, its two first phases of development paralleled the field of artificial intelligence - during its pioneering decades in the 1960s and early 1970s as well as during the winter of AI. The second bloom of AI, in the 1980s, was echoed with similar expert system research in the field of computational architecture, but to a lesser extent, as the latter had already started following its own, independent development path, branching out as an autonomous field and loosening its institutional links with AI. This newly established field developed their own practices with algorithmic techniques, hijacking methods from other fields - from biology to animation -, and crafting their own tools, from Processing libraries to Rhinoceros and Grasshopper plugins. This institutional and technical estrangement between the field of AI and of computational architecture thus resulted in an appropriation of issues and tools and enabled the development of heuristic practices of programming specific to computational architecture, fostering renewed architectural designs[3].

*Displaced technical ties*

If the empowerment of the computational field has allowed the emergence of promising themes, sub-fields of research and experimentations for architecture, the recent democratization of algorithmic design tools is changing practices again. The resort to computational design tools by a larger number of users, often less specialized and less familiar with the subtleties of programming, implies the integration of the various algorithmic typologies favored by computational architecture practitioners into ready-made tools, with interfaces that facilitate access to complex algorithmic models for novice users. Although allowing democratization, these new tools tend to make some of the parameters on which these models are based more difficult to access: judged too complex to manipulate by novice users, they are relegated to the background. For example, genetic algorithms put the gene pool and the final selection of individuals first, immediately accessible to users, then the fitness function, with all other parameters being relegated to the background and made harder to update (Gaudillière 2020a). In general, the shift from custom scripting by computational practitioners to the integration of algorithmic tools into mainstream software making them accessible to the general architectural public gives these tools the status of black boxes. This status makes users dependent on the design intentions of the tools' programmers, since the latter structure the tools according to these intentions, and users no longer always have the knowledge to circumvent the initial programming structure and readapt the script to their own intentions. This new relationship to tools orients practices towards a new design rationality for the built environment (Gaudillière 2020a). If design tools from the field of artificial intelligence are for the moment less widespread than genetic algorithms and other typologies favored by the computational field, their mode of development and integration is nevertheless similar to the latter, and thus tends to have the same effect. We are therefore witnessing a reconnection to the AI field through tools, which have become intermediaries between the two fields.

The most widespread practices of the computational field, as well as those currently emerging using AI design tools, are largely based on optimization principles. Architectural productions are evaluated according to material and financial constraints: we seek to optimize cost or space occupancy, but also material quantity and structural efficiency or adequacy to norms, instrumental to the construction of a project. More broadly, we seek to optimize according to criteria of production and promotion of architecture in a market economy, which have become the essential evaluation criteria for most contemporary production. The most used tools are used because they easily allow the optimization of these criteria: the coupling of form-finding algorithms with genetic algorithms allowing the exploration of a large design space in search of optimal solutions to a design problem is today the most widespread workflow in practices of the field (Gaudillière 2020a). Optimization has also taken a prominent place in the theorization of computational practices in architecture, with performance-based design (Hensel 2013) now being a widespread and widely promoted mode of practice. More broadly, our theorization of the design process itself, through the term *wicked* or *ill-structured problem* - widely used to describe the tasks faced by designers in the literature - is formulated within the semantic field of optimization and originating in the field of AI. This terminology is indeed the heir to a chronology starting in 1956 with a text by John McCarthy and pursued in 1961 with a text by Marvin Minsky. The two researchers put forward the notion of *well-defined problems* - problems formalized so as to be processed and solved by a computer. In 1964, Walter Reitman theorized the opposite notion of *ill-defined problems*, which was further developed in 1967 by Alan Newell and in 1969 by Charles Eastman, the latter making the link between ill-defined problems and design. This articulation was later taken up by Herbert Simon in 1973 in *The Structure of Ill-structured Problems*, a long section of which is dedicated to the examination of the architectural design process. In the same year, Horst Rittel and Melvin Webber adapted the notion by coining the term *wicked problems*, and the concept spread throughout design research over the

[3] For a more detailed description of these heuristic programming practices, see (Gaudillière 2020b).

following decades. Rooted in the early relationships between the computational movement and the world of AI, and promoted by the renewed links between the fields crafted by contemporary tools, this ensemble contributes to a new epistemology advocated by many practitioners - the becoming science of architecture, with optimization practices as a foundational principle.

However, architectural practices are not limited to optimization, or at least not to the optimization of easily quantifiable factors. Questioning what an optimized architecture can be implies formulating an optimization function for criteria that are not easily quantifiable: an optimized apartment certainly requires structural and thermal characteristics that can be modeled based on systematic mathematical models, but it also requires spatial hierarchization and relations based on different models depending on the architectural choices. An optimized architecture also requires identifying the context in which the project is situated and adjusting the features to reflect it, a task for which clear methods are particularly difficult to formalize, as pointed out by Christopher Alexander in the first pages of *Notes on the Synthesis of Form* (1964). Programming for an optimized architecture thus requires, as in any design process, making choices and adapting the scripts used accordingly. The heuristic programming practices of the 2000s clearly show the necessary adaptation work (Gaudillière 2020a), the play on the fitness functions set up, which allows a subtler mobilization of the optimization mechanisms, almost hijacking the notion to suit architectural purposes. Automating architecture according to fast-tracked optimization principles, as the most recent tools and practices of the computational field push for, tends however to make this heuristic, artisanal dimension of the use of programming in architecture disappear. If AI tools currently tend to follow the same trajectory, the recent paradigm shift in the field implied by the shift to machine learning could nevertheless allow us to question and to rethink the automation mechanisms of the design process at work today.

*Philosophies of AI*

Recent history of artificial intelligence is marked by the turn towards machine learning and big data at the beginning of the XXI$^{st}$ century. The emergence of new methods of data processing, in particular neural networks, with impressive results in the field of image analysis at first and then progressively in all fields of information processing, triggered a radical change of paradigm in the artificial intelligence community. If the connectionist approach, popularized by cyberneticians in the 1930s to 1950s, was later buried by the symbolists in the 1960s the turn of machine learning marks a return to this paradigm. In an article published in 2018, Dominique Cardon, Jean-Philippe Cointet, and Antoine Mazières summarize this shift in focus in the world of AI, describing the symbolist and connectionist paradigms by defining what the machine processes in each case and the relationship to the world it fosters. In the symbolist approach, the aim is to give the machine abstract and logical reasoning capabilities, by providing an input and a program, and obtaining the output. The rules provided to the machine through the program create an internal reasoning space, a “toy world” isolated from the real world and within which the machine evolves. In the connectionist approach, on the other hand, the machine is provided with an input and the corresponding output. It is then up to the machine to find the program that allows it to match best the one to the other. The aim of this mechanism is to take the machine out of its toy world and re-establish a link with the real world, via massive quantities of raw data. The shift in the 2000s to deep learning techniques thus marks a renewal of ambitions in the field of artificial intelligence, through a profound technical change, that in turn triggers a paradigmatic change, both having repercussions at several levels in the field of computational architecture.

Cardon, Cointet and Mazières also argue in their article that despite this change in the techniques used in the field of artificial intelligence, researchers still tend to apply theories inherited from the period of domination of the symbolic approach to contemporary practices. Although the latter are anchored in a renewed connectionist paradigm, the field of AI itself lacks theoretical explorations that acknowledge this change and provide appropriate theories for these new practices of deep learning, a lack that then tends to be reflected in the field of computational architecture. Furthermore, AI practices are more and more oriented towards commercial exploitation, which promotes a rationalization and the propagation of tools in which this rational approach is embedded. Thus, these new AI technologies are used without ever really requiring the underlying theoretical principles to correspond to the promoted practices, or even by encouraging the diffusion of techniques without any discussion of the theoretical stakes, a more favorable scenario for development within a market economy. Because of its long-standing proximity to the field of AI, as well as its similar trajectory in the contemporary market economy, computational architecture replicates this focus on the technical, sometimes even amplifying them given the black-box status of the tools when the practitioners are simply users.

Finally, if we examine how deep learning techniques are applied by researchers in computational architecture, some approaches raise questions in view of the remarks made by Cardon, Cointet and Mazières. In addition to their comments on the nature of the relationship to the world fostered by symbolic and connectionist machines, they also point out the need to fragment the data supplied to machine learning systems. Not only must the data be big, it must also be the most elementary units possible, in order to give the machine the ability to describe the world according to systems defined by itself rather than according to relations pre-determined by a human programmer. The development of the use of AI tools for architectural design thus involves not only optimization as a key principle, but also the fragmentation of processed data. Architectural robotics research using deep learning tools aims at making machines acquire a more direct link to the world around them - giving them the same five senses we have -, and uses standard machine learning data types for standard purposes: recognizing a brick on an image to aim at it for example. Architectural and urban design works mobilizing machine learning techniques, on the other hand, are often satisfied with providing GANs with numerous images of facades, renders or satellite views, hoping to bring out innovative and relevant systemic features. However, if to make connectionist systems work at their best they need to be provided with as much and as fragmented data as possible as input, this might not be the most relevant approach. The new practices of machine learning in the field of computational architecture, in addition to renewing the links to the communities of artificial intelligence, thus question what the elementary numerical data of architecture could be, what to provide to machine learning systems, what to expect from those novel tools, and, more importantly, the practice of ordering that architecture is itself.

## IV. Conclusion

The present paper offers an historical account of the relationship to AI entertained by the field of computational architecture, a relationship that has been constant albeit of shifting form. Two main periods have been identified: while the ties between the two fields had been very tight in early years, with institutional, technical and philosophical links leading practitioners to shared perspectives, the latter period is characterized by more diffuse links. Strong institutional ties have disappeared, technical ties are displaced and now fostered through tools and philosophical ties are essentially an inheritance of past times yet to be reinvented. The analysis of these past and present relations enables us to put in perspective current explorations in architecture using AI tools. In particular, the resort to optimization and fragmentation at the core of those practices question the possibility of transcribing these principles in architecture. However, while fragmentation operations are specific to AI, with their integration in architectural design practices and workflow still ongoing and being interrogated by practitioners mobilizing AI tools, optimization practices have far more become standard, aligning with design theories developed in the past decade and transforming the practice by imprinting on it a new design rationality, with architecture tending to become an optimization game. The historical perspective that the present work brings to the role artificial intelligence in computational architecture thus essentially seeks to trigger a questioning of the epistemological weight that such tools bring with them, and the relevance as well as the possibility to recreate a common technical and theoretical project between architecture and artificial intelligence.

**Bibliography**


- Alexander, Christopher. *Notes on the Synthesis of Form*. Cambridge: Harvard University Press, 1964.
- Alexander, Christopher. *The Timeless Way of Building.* Oxford: Oxford University Press, 1979.
- Allen. Matthew. “Architecture becomes Programming. Invisible technicians and local theories in the late 1960s.” In *The Figure of Knowledge: Conditioning Architectural Theory, 1960s -1990s*, edited by Sebastiaan Loosen, Rajesh Heynickx and Hilde Heynen. Leuven: Leuven University Press, 2020.
- Brand, Stewart. *The Medialab : Inventing the Future at M.I.T*. New York: Viking Press, 1987.
- Bryant, Greg. “Gatemaker: Christopher Alexander's dialogue with the computer industry.” In *Battle for the Life and Beauty of the Earth. PUARL Conference Proceedings*, edited by Hans Joachim Neis, Gabriel A. Brown and Greg Bryant, 77-91, Portland: Portland Urban Architecture Research Laboratory, 2013.
- Cardon, Dominique, Jean-Philippe Cointet and Antoine Mazières. “La revanche des neurones. L’invention des machines inductives et la controverse de l’intelligence artificielle.” *Réseaux* 5, n°211 (2018) 173-220.
- Cross, Nigel. “Designerly ways of knowing.” *Design Studies* 3, n°4 (1982): 221–227.
- Dreyfus, Hubert. *What Computers Can’t Do: The Limits of Artificial Intelligence*. New York: Harper and Row, 1972

- Eastman, Charles. “Cognitive processes and ill-defined problems: a case study from design.” In *IJCAI'69: Proceedings of the 1st international joint conference on Artificial intelligence*, 669–690. San Francisco: Morgan Kaufmann Publishers, 1969.
- Gaudillière, Nadja. “Evolutionary Tools and the Practice of Architecture: from Appropriated Typology to Becoming the Black Boxes of CAAD.” In *Proceedings of the 4th International Conference of Biodigital Architecture & Genetics*, edited by Alberto T. Estevez. Barcelona: iBAG-UIC Barcelona, 2020a.
- Gaudillière, Nadja. “Computational Tools in Architecture and their Genesis : the Development of Agent-based Models in Spatial Design.” In *RE:Anthropocene, Design in the Age of Humans - Proceedings of the 25th CAADRIA Conference - Volume 2*, edited by Dominic Holzer, Walaiporn Nakapan, Anastasia Globa and Immanuel Koh, 497-506. Bangkok: Chulalongkorn University, 2020b.
- Gaudillière-Jami, Nadja. “AD Magazine. Mirroring the Development of the Computational Field in Architecture 1965-2020.” In *Distributed Proximities: ACADIA 2020 Conference Proceedings*. 2021.
- Hensel, Michael (ed.) *Performance-Oriented Architecture: Rethinking Architectural Design and the Built Environment*. Hoboken: John Wiley & Sons, 2013.
- McCarthy, John. “The Inversion of Functions Defined by Turing Machines.” In *Automata studies, Annals of Mathematics studies no. 34*, edited by Claude E. Shannon et John McCarthy, 177-181. Princeton: Princeton University Press, 1956.
- McCorduck, Pamela. *Machines Who Think. A Personal Inquiry into the History and Prospects of Artificial Intelligence*. Natick: AK Peters, 2004.
- Minsky, Marvin. “Steps toward Artificial Intelligence.” Proceedings of the IRE 49, n°1 (1961): 8-30.
- Negroponte, Nicholas. “Towards a Theory of Architecture Machines.” *Journal of Architectural Education* 23, n° 2 (1969): 9-12.
- Newell, Alan. “Heuristic Programming: Ill Structured Problems.” In *Progress in Operations Research Vol.3*, edited by Julius Aronofsky, 360-414. New York: Wiley, 1969.
- Pickering, Andrew. *The Cybernetic Brain. Sketches of Another Future*. Chicago: University of Chicago Press, 2009.
- Reitman, Walter. ‘Heuristic Decision Procedures, Open Constraints, and the Structure of Ill-Defined Problems.’ In *Human Judgments and Optimality*, edited by Maynard W. Shelly and Glenn. L. Bryan, 282-315. New York: Wiley, 1964.
- Rittel, Horst W.J.and Melvin W. Webber. “Dilemmas in a General Theory of Planning.” *Policy Science* 4 (1973): 155-169.
- Searle, John. “Minds, Brains and Programs.” *Behavioral and Brain Sciences* 3, (1980): 417–57.
- Simon, Herbert. “The Structure of Ill-structured Problems.” *Artificial Intelligenc*e 4, (1973): 181–201.
- Steenson, Molly Wright. *Architectural Intelligence : How Designers and Architects Created the Digital Landscape*. Cambridge: The MIT Press, 2014.
- Weizenbaum, Joseph. “ELIZA—a computer program for the study of natural language communication between man and machine.” *Communications of the ACM* 9, n°1, (1966): 36-35.