Algorithmes entre objets et médiums : étude des traces d'un matériau numérique

Pierre Depaz, Nadja Gaudillière-Jami

Algorithmes entre objets et médiums : étude des traces d'un matériau numérique

Algorithms between objects and media: study on the traces of a numérique material

Pierre Depaz
THALIM, Paris-3 Sorbonne Nouvelle
Paris, France
pierre.depaz@sorbonne-nouvelle.fr
https://orcid.org/0009-0009-1489-247X

Nadja Gaudillière-Jami
Digital Design Unit, Université Technique de Darmstadt
Darmstadt, Allemagne
gaudilliere@dg.tu-darmstadt.de
https://orcid.org/0009-0001-9773-710X

Mots-clés

Algorithme
Relation
Récursion
Médium

Keywords

Algorithm
Relation
Recursion
Médium

Résumé

Cette contribution s'intéresse à la relation entre médium et objet à travers le prisme des algorithmes. Partant du constat de leur existence en tant qu'objets du design de la part des informaticiennes et programmeuses, avant de devenir médiums du design pour d'autres professions, nous présentons dans un premier temps les traces concrètes laissées par les algorithmes à différentes étapes du processus de conception, entre extraits de codes décrits sur des forums, représentés lors d'expositions, ou encore utilisés sous forme de bibliothèques. L'existence de ces traces nous permet alors de développer une compréhension de la conception d'algorithmes en tant qu'objets médiateurs. À travers une série d'exemples de logiciels où la limite entre moyen du design et produit du design s'estompe, nous mettons à jour une certaine matérialité de l'algorithme en tant qu'objet natif du numérique, avant de nous concentrer sur deux spécificités de la matière algorithmique qui facilitent son existence en tant que médium : récursion et relation. En conclusion, nous montrons que ces deux propriétés de récursion et de relation contribuent à faire disparaître la distinction entre objet et médium.

Abstract

This contribution investigates the relationship between media and objects via a study of the nature of algorithms. Beginning with the observation that for computer scientists and

programmers, algorithms exist as design objects before they become a design medium for other professions, we first examine the tangible traces left by algorithms at different stages of the design process. Source code snippets described on forums, diagrammatic representations of algorithmic logics at exhibitions or libraries in programs are some of the materials we examine. The existence of these traces then leads us to envision algorithms as mediating objects; through a series of examples where the boundary between product and means of design fades away, we uncover a specific materiality of the algorithm as a native digital object. Finally we focus on two specificities of algorithmic matter that facilitate its existence as a medium: recursion and relation. In conclusion, we show that these two properties contribute to blurring the distinction between object and medium.

Introduction

Le logiciel est généralement considéré comme un outil du design, un moyen de création doté de limites spécifiques et existant le long d'une chaîne de production dont le résultat consiste en un objet matériellement concrétisé. En cela, il est un médium, un intermédiaire qui fournit un cadre d'interaction entre humains et non-humains (Latour, 1994) et à travers lequel une tâche est accomplie. Nous nous intéressons ici en particulier à l'interface du logiciel, la partie rendue visible à l'utilisatrice et par laquelle elle interagit avec un espace de conception. Celle-ci tend paradoxalement à s'invisibiliser, tout comme les procédés auxquels elle donne accès (Galloway, 2012 ; Krug, 2014 ; Bryant 2013). En considérant avant tout l'objet — au sens premier d'une chose maniable, extérieure à soi — produit à travers l'interface, il y a une tendance pour la designer à ne pas considérer pleinement le logiciel comme un objet. En effet, si nous avons tendance à comprendre l'objet comme résultat distinct du processus de conception et de création, il faut aussi prendre en compte des objets sans propriétés physiques (Turner, 2018 ; Hui *et al.*, 2016) : ceux de l'écologie des interfaces que constituent les outils numériques du design.

En faisant pivoter notre perspective d'utilisatrice à conceptrice, du design à l'informatique, le logiciel se révèle comme un objet du design à part entière. Alan Kay, dans sa conception du système SmallTalk, pose le principe fondateur que « tout [y] est un objet » (Kay, 1993, p. 78) ; pour Erich Gamma *et al.*, un logiciel est constitué d'objets, et un objet « comporte des données et les procédures qui opèrent sur ces données » (Gamma *et al.*, 1995, p. 11) ; pour Raymond Turner, toute entité dans un ordinateur consiste en un « artefact computationnel » (Turner, 2018, p. 25). À travers différents champs de l'informatique, la conception du logiciel comme moyen plutôt que comme résultat s'avère n'être que partielle.

SmalltTalk est aussi connu pour compter parmi les pionniers de la programmation orientée objet (POO) (Kay, 1993). Ce paradigme de programmation informatique consiste en la définition de composants logiciels dits objets et représentant des idées ou des entités du monde physique, et de leurs relations. La conception des objets qui composent le programme est donc cruciale dans ce paradigme, et les objets de la POO posent d'autant plus de questions de design que celle-là fait partie de développements informatiques construits sur d'importantes interactions avec le monde du design. Les ouvrages de Christopher Alexander (1964 ; 1979) ont été très lus dans la communauté informatique américaine de l'époque. Les motifs (*patterns*) conceptualisés par Alexander dans ses textes sont des objets architecturaux modulables, adaptés et recombinés à l'infini en fonction des projets, et sont aujourd'hui compris comme ayant largement contribué à la conceptualisation de la POO (Bryant, 2013).

Si, traditionnellement, designer équivaut à matérialiser, qu'en est-il de ces médiums qui ne semblent pas posséder de propriétés physiques ? Comment le design peut-il appréhender ces objets sans matière ? Comment façonner des objets sans point de départ physique ? Comment ces objets opèrent-ils ensuite en tant que médiums ? Cet article propose d'analyser la relation que le logiciel, comme médium programmé composé d'algorithmes, pose entre matérialité et environnement computationnel. Plus particulièrement, nous envisageons ce rapport à la matière par l'utilisation, la création et la distribution de l'algorithme-en-tant-qu'objet, afin de montrer que, dans l'environnement computationnel, la distinction entre objet et médium est des plus ténues.

Spécifiquement, nous considérons cet algorithme-en-tant-qu'objet au-delà de l'objet

virtuel (représentation numérique d'un étant analogue), c'est-à-dire en tant qu'objet numériquement natif. Mais, que sont exactement les algorithmes ? Compris comme une liste d'instructions à exécuter pour atteindre un objectif spécifique, les algorithmes sont antérieurs à l'ordinateur. Les instructions qui composent un algorithme doivent être calculables, formalisées selon des principes mathématiques spécifiques (Hodges, 2000). Notre point de départ de ce texte : les algorithmes constituent la plus petite unité fonctionnelle de la programmation. Mais, ils peuvent être combinés pour former des objets informatiques plus larges : les programmes sont des ensembles délimités et instanciés de plusieurs algorithmes. Au quotidien, les designers interagissent avec les algorithmes par le biais de programmes informatiques, le plus souvent réunis en logiciels et commercialisés par des entreprises comme Adobe ou Autodesk.

Pour effectuer cette étude, nous nous inscrivons dans une approche méthodologique matérialiste du numérique (Reichert et Richterich, 2015). Il s’agit ici de dépasser la doxa selon laquelle un médium (transitif) sert à créer des objets (définitifs) à partir de matériaux, cette relation étant unidirectionnelle, et de partir plutôt de l’hypothèse que cette disparition des limites entre objet et médium est due à la spécificité du matériau numérique. Dès lors, comment peut-on qualifier la relation entre médium et objet dans un environnement numérique ? Que peut-on apprendre des traces laissées par des artefacts digitaux, et comment ces traces peuvent-elles illustrer cette relation ? L’étude de la distinction entre médium et objet nous permet d’interroger la matérialité du numérique, c’est-à-dire ce qui le rend manifeste (Leonardi, 2010). Cette interrogation s’inscrit dans la lignée des travaux de William J. Rapaport sur la relation entre software et hardware, et la nature de chacun (Rapaport, 2023), et de Friedrich Kittler sur la matérialité de ces éléments (Kittler, 2014).

Nous tracerons d’abord les manifestations concrètes des productions faites à l’aide de logiciels jusqu'à la manifestation tridimensionnelle des objets que ces productions décrivent, en prêtant notamment attention aux réseaux d’actantes et d'intermédiaires permettant de révéler l'algorithme comme produit du design.

Dans un second temps, nous allons adopter le point de vue des designers d'algorithmes pour observer la place qu'occupe la matérialité dans leurs discours et leurs pratiques. Nous verrons alors que les constructions computationnelles peuvent être considérées comme étant dotées de propriétés matérielles, particulièrement dans des dimensions spatiales, sous forme d'effets de grandeur, mais aussi temporelles, sous forme d'effets de décomposition.

Finalement, il s'agira de conjuguer ces deux approches — le design comme producteur d'objets par des assemblages d’algorithmes et l'algorithme comme objet désigné — pour réexaminer la notion de médium dans l'espace numérique, en tant qu'objet récursif, infiniment médiatisant.

1. — Instances de matériaux

1.1. — De la formalisation des concepts à la matière du code

1.1.1 — Objets produits

L’algorithme est une formalisation. Mais, définir ce que l’algorithme formalise

varie en fonction de l'utilisatrice. Les mathématiciennes y voient les fonctions récursives qui délimitent l'univers mathématique calculable, tandis que les logiciennes y voient des ensembles de relations, cette double perspective engendrant des débats poussés sur la nature des algorithmes (Naibo et Seiller, 2023). Les programmeuses manifestent des fonctions récursives et des relations entre objets à travers la manipulation de langages informatiques de plus ou moins haut niveau. Les designers manipulent ensuite les résultats de l'exécution de la production des programmeuses.

En miroir de l'influence d'Alexander sur le paradigme de la POO, la variabilité des objets produits à travers le logiciel se retrouve conceptualisée également dans le monde du design. Les objets conçus par Bernard Cache et Patrick Beaucé, sculptés à l'aide d'une fraiseuse numérique, tous différents et néanmoins obéissant aux mêmes principes géométriques et à la même logique de fabrication, sont baptisés *objectiles* par Gilles Deleuze, dans le but de saisir en un vocable cette malléabilité mathématique et géométrique (Deleuze, 1988 ; Cache, 1998). Robert Woodbury, d'autre part, définit ces objets par leur qualité paramétrique : ils relèvent de règles logiques définies par la conceptrice, ils sont modifiables en permanence et permettent donc le développement d'alternatives de conception parallèles au cours du processus (Woodbury, 2010). La confrontation répétée à la définition des propriétés des objets virtuels reflète la difficulté que représente le saut de l'objet conçu à l'objet physique dans le champ du design (Evans, 1995), et la navigation de l'un à l'autre qui est le propre de cette activité.

La difficulté de cette confrontation est renforcée par la variabilité des objets produits virtuels et la dissonance qu'elle crée non seulement avec l'objet physique figé ensuite matérialisé, mais aussi avec les traces matérielles conventionnelles des processus de conception algorithmiques : dessins, rendus, plans d'exécution. Cette variabilité se matérialise pourtant dans d'autres traces matérielles. Les modèles 3D paramétrables, à la géométrie en constante évolution possible, la capturent tout en la laissant difficilement visible. Logigrammes et pseudo-codes saisissent la logique de ces modifications, tout en demeurant peu visibles aux novices. Une trace matérielle se détache cependant, reflétant avec exactitude cette propriété nouvelle des objets produits algorithmiquement : le catalogue (Witt, 2021). Collection sans fin d'alternatives produites par les systèmes paramétriques, le catalogue se dévoile à travers les grilles d'objets diffusées en ligne et dans les présentations des projets. Matérialisation de la difficulté à figer en un seul objet produit final le processus de design, il constitue aussi une trace caractéristique des objets algorithmiques.

1.1.2 — Environnements techniques

Se pencher de plus près sur les conditions de la programmation révèle les très nombreux objets matériels qui encadrent cette pratique.

Un vaste ensemble d'infrastructures techniques permet d'interagir avec le monde numérique : souris, claviers, stylets, écrans, cartes-mères. Cet environnement technique comprend aussi des dispositifs physiques de dessin ou de construction de maquettes propres au design. C'est le cas du crayon optique et de l'écran du célèbre programme de dessin précurseur Sketchpad. C'est aussi le cas du projet Seek, développé entre 1969 et 1970 au sein de l'*Architecture Machine Group* du Massachusetts Institute of Technology. Seek est un programme qui 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 (Brand, 1987). Un prototype du système est construit, où des gerbilles hébergées dans une grande cage en verre font office d'usagers. Celles-ci

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, ce qui manifeste matériellement une cybernétique logicielle.

Autre projet mobilisant un système de cubes en guise d'interface physique, mais cette fois-ci pour des utilisatrices humaines : l'Universal Constructor, conçu par John et Julia Frazer avec leurs étudiants de la AA School of Architecture au début des années 1990 (Frazer, 2001). Le projet propose un système de cubes à manipuler à la place des traditionnels claviers et souris. Le motif créé par les cubes est prévu 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 cubes constituent ainsi une représentation physique des relations logiques entre les éléments, quels que soient les éléments considérés. Là encore, des manifestations matérielles de l'abstraction logicielle.

1.1.3 — Objets textuels

Le code constitue lui aussi une matérialisation des algorithmes à part entière. Le code désigne, 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 univoque sous une forme discrète, en vue de faciliter leur traitement automatique ou leur transmission » (Abelson et Sussman, 1979, pp. 40-41). 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. Les scripts, eux, désignent un morceau de code transmis d'une programmeuse à l'autre ou d'un programme à l'autre. Ce morceau de code n'est pas nécessairement fonctionnel, ne produit pas forcément de résultats s'il est exécuté seul. Mais, comme le code complet, il est constitué d'un morceau de texte qui peut être copié, collé, envoyé, réusiné (Fowler *et al.*, 1999).

Lorsque les algorithmes sur mesure s'allongent, le travail de programmation peut être très long. Les bibliothèques permettent de contourner cet écueil et de gagner du temps : il s'agit de collections de blocs de code prédéfinis, développés par des programmeuses comme des raccourcis vers des fonctions déjà écrites. Les bibliothèques sont 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, lorsqu'écrit dans le code, fait appel à cette fonction directement. 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 un important soutien aux pratiques de programmation. En effet, une programmeuse peut faire appel à des bibliothèques codées par elle-même, mais aussi à celles d'une autre programmeuse. Les bibliothèques de code peuvent être comparées à des bibliothèques physiques : des collections de ressources collectives à utiliser. Il existe des bibliothèques publiques, privées, ou partagées seulement par quelques individus. Néanmoins, une différence significative est le degré d'abstraction que permet une bibliothèque de programmation : par exemple, le travail de décennies de recherches en implémentations mathématiques peut être et résumé, et utilisé par le biais d'un simple mot, tel que *numpy*, bibliothèque de code qui permet alors à la programmeuse d'utiliser ces opérations mathématiques dans son code.

Cette écologie d’objets textuels est complétée par un autre ensemble, plus éloigné de la manipulation directe du code, mais tout aussi crucial dans l’appréhension des pratiques de programmation et de leurs manifestations matérielles. La modification des codes, comme l’apprentissage de la programmation, repose sur de fréquents échanges écrits entre programmeuses. Nombre de ces échanges sont publics puisqu’ils ont lieu sur les forums en ligne qui soutiennent ces pratiques : par exemple, GitHub ou Reddit pour la programmation en général, et les forums de Processing ou de Grasshopper pour les pratiques de programmation plus directement associées au design. Reflets des interactions avec la matière du code, ces échanges en sont une matérialisation à part entière.

1.2. — Enjeux de mise en scène, d’archivage et d’analyse

S’intéresser à la matière des algorithmes révèle sa fragilité. Le code peut devenir techniquement obsolète très vite, en raison de la volatilité des objets numériques et de la déliquescence des fichiers vieillissants. Le recours à d’anciens langages de programmation, logiciels ou ordinateurs peut rendre compliquées la consultation et l'exécution d’un vieux programme. Les praticiennes ne conservent pas toujours les anciennes machines, même lorsqu'elles conservent les données relatives au projet, y compris les fichiers de code. Cette volatilité force à s’intéresser aux matières annexes des algorithmes : diagrammes, paroles, logigrammes. Les modes d’émergence de ces matières annexes mettent en évidence différents enjeux autour de la matière des algorithmes, au-delà de l’exécution du code lui-même : mise en scène par les programmeuses, archivage par les conservatrices, analyse par les historiennes et les sociologues. Si le travail d'archivage des données numériques pose des enjeux de conservation classiques, le travail de mise en scène effectué par les programmeuses et les designers pour donner à voir leur processus de travail avec les algorithmes constitue une forme particulière d’archivage. Si elle est sélective, celle-ci donne néanmoins à voir d’autres interactions avec la matière des algorithmes.

1.2.1 — Mise en scène

Le code devient d’abord, au fil des expositions et des magazines, un élément de mise en scène dans la présentation des projets, un étendard visuel des pratiques de design computationnel. Cela est particulièrement visible dans le domaine très codifié de la représentation architecturale : 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 que d’en comprendre la logique de conception. Font donc irruption, dans la présentation des projets, des diagrammes permettant de saisir cette logique. Ces diagrammes cherchent à saisir des ensembles d’opérations mathématiques et à les donner à comprendre au spectateur. En complément, le code s’invite lui aussi 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, minuscule extrait de code suivant chaque forme générée. L’exposition *scriptedbypurpose*, organisée à la FUEL Gallery de Philadelphie en 2007, en est un

exemple emblématique, du fait que 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.(Fornes, 2007).

Le titre de l'exposition — scriptedbypurpose[1] — et l'usage du mot « entrée » — qui fait référence à une entrée de code — pour désigner les projets dans le manifeste complètent ici cette esthétique avec un vocabulaire ancré dans la pratique de la programmation. La retranscription partielle du code dans les expositions et les magazines ne permet pourtant pas de comprendre véritablement les algorithmes présentés ou de les reproduire. Sur les panneaux d'exposition s'affichent souvent à peine quelques lignes, ou tout juste la retranscription complète d'une ou de deux des fonctions clés. Mais, ni la relation entre ces fonctions ni les quelques centaines, voire les quelques milliers de lignes du code complet, ne sont montrées. Il s'agit donc bien d'une esthétique, d'un marqueur qui permet aux praticiennes de s'identifier au sein d'un territoire commun.

1.2.2 — Archivage et analyse

La volatilité des supports met aussi en question les méthodes d'archivage et d'analyse par les conservatrices et les historiennes. Les sources premières offrant des matériaux à consulter pour contribuer à des histoires du logiciel sont assez conventionnelles : archives privées et publiques, en particulier celles des architectes informaticiennes et des éditrices de logiciels de CAO. En observant le travail des historiennes, plusieurs techniques se dévoilent, en association avec de nouvelles traces matérielles du travail algorithmique.

Le développement d'espaces de préservation appropriés, qui permet de sauvegarder la matière algorithmique sur le long terme, fait partie des stratégies de travail. Le musée des *malware* (Hyponen, 2016) ou encore la base de données FID Baudigital, dédiée aux modèles 3D produits par les architectes au cours du travail de conception (Arndt *et al.*, 2022), sont des exemples de ces espaces et des traces qu'ils génèrent à leur tour. Autre approche, l'étude des métadonnées des fichiers pour reconstituer le processus de conception. C'est le cas du travail mené par le Centre Canadien d'Architecture (CCA) sur les modèles de l'*Embryological House* de Greg Lynn. Les archives numériques données par l'architecte, composées d'une multitude de fichiers natifs pour différents logiciels, n'étaient en grande partie plus lisibles. Pour comprendre l'évolution du processus de modélisation comme de la forme du projet, les équipes du CCA sont parvenues à reconstituer l'histoire numérique de cet objet architectural seulement à partir d'une petite quantité de métadonnées directement accessibles pour chacun de ces fichiers (Bird et LaBelle, 2010). Cette reconstruction interroge néanmoins la nature des sources pour le travail des historiennes : quel est le statut de ces nouveaux objets algorithmiques ?

En cas de déliquescence des matériaux initiaux, d'autres peuvent également aider à étudier la structure et les résultats des algorithmes : la documentation de travail telle que les commissions ou les directives, les organigrammes, les notes préparatoires et les diagrammes, ainsi que les publications sur les forums, les tutoriels et les entretiens avec les programmeuses. Extraits de codes, pseudo-codes, environnements de programmation

[1] codé avec intention.

et logiciels utilisés, diagrammes explicatifs, descriptifs textuels et audios, flux de travail permettent de reconstituer les détails du processus de conception des algorithmes. Un document en particulier retient l'attention dans les pratiques de design computationnel, à la frontière entre les deux mondes. De nombreuses praticiennes travaillent avec des esquisses de projet algorithmiques. Mélanges de logigrammes et de croquis éclairant les étapes précédant les sorties que les praticiennes cherchent à obtenir, ces esquisses sont précieuses pour saisir la logique d'aller-retour entre intentions de conception et écriture de l'algorithme. Ensemble, ces usages du code produisent de nouvelles traces matérielles tout en mettant en question l'interaction de professions non formées à la programmation avec les algorithmes-en-tant-qu'objets : designers, architectes, historiennes, sociologues.

Il y a donc bien des existences matérielles des algorithmes en tant que médiums du design dont nous pouvons observer les traces. Ces derniers, et les programmes qu'ils composent, se révèlent au travers d'assemblages robotisés qui incarnent physiquement la possibilité de configurer des algorithmes, de représentations privées et textuelles par le biais de bibliothèques de code, ou encore de mises en scène lors d'expositions publiques. À chaque fois, ce sont différents domaines et différentes professions qui l'envisagent selon différentes modalités : l'environnement physique, les lignes de codes ou encore les impressions affichées.

Envisager l'algorithme au premier abord en tant qu'interface permettant de désigner des produits dits computationnels fait émerger est un faisceau d'indices pointant vers l'algorithme comme objet du design : conception, modification, présentation d'un résultat plutôt qu'une simple interface qui se limiterait alors à l'invisibilité précédemment évoquée. Il s'agit néanmoins bien de remédiations, de la présentation d'un nouveau médium par le biais de médiums précédents. Si le code est un objet qui n'est ni physique, ni textuel, ni visuel, quel genre de matériau est-il ?

2. — Perceptions et sensualités des algorithmes

Nous nous tournons à présent vers la matérialité intrinsèque des algorithmes-en-tant-qu'objets. Pour ce faire, nous commençons par une présentation des formes expérientielles des objets algorithmiques, avant de nous pencher sur certaines propriétés spatiotemporelles propres à ce genre d'objet. Il s'agit dans cette section de montrer la spécificité des objets qui vont ensuite constituer le médium de travail des designers, lors d'étapes subséquentes de la chaîne de création.

2.1. — Références discursives

Une première approche de la matérialité des objets digitaux est de s'intéresser aux vocabulaires et métaphores révélant un positionnement clair de création d'objets constitués d'attributs perceptibles, sensuels et manipulables à travers une pratique de design.

Les termes utilisés par les programmeuses se cristallisent autour de références à

leur code en matière de propreté[2], de simplicité[3] ou d'élégance[4]. L'aspect ineffable du processus de conception rejoint, depuis le champ de l'informatique, les tendances qualitatives d'autres champs du design.

Chacun de ces adjectifs se retrouve à développer un certain sens d'une matière designée : plutôt que de considérer le logiciel comme un donné, cela suggère que le logiciel soit le résultat d'une fabrication et d'une organisation particulière, tant sur le plan de l'idée que sur le plan du code source. Notons l'usage que font les programmeuses d'une citation d'Antoine de Saint-Exupéry au sujet du processus de création : « Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher. » (Saint-Exupéry, 1972, p.41) Cette remarque, que nous retrouvons dans des publications de blogues, des signatures de messages ou encore des manuels techniques, met bien en avant l'équivalence entre l'objet concret (le moteur auquel Saint-Exupéry fait référence) et l'objet présumé abstrait (les algorithmes).

De plus, nous voyons à travers ces différentes qualifications une certaine cohérence de standards : réduction de la matière superflue (propreté), rapport direct avec l'intentionnalité de l'artefact (clarté) ou encore optimisation de la matière programmable (les lignes de code) par rapport à sa fonction (élégance).

Il est également intéressant de considérer les références discursives négatives, telles que la qualification des programmes comme des tas de spaghetti (Steele, 1977) ou comme des amas de boue (Foote et Yoder, 1997). Ces deux métaphores permettent de saisir une sensation matérielle du travail avec le code et certaines des propriétés de ce matériau qu'est le code. La métaphore du spaghetti dénote, entre autres, l'enchevêtrement, la difficulté à suivre un fil conducteur particulier au milieu de nombreux semblables ; de manière similaire, la référence à la boue traduit l'absence complète de design délibéré, et donc d'évolution entropique incontrôlée et indéfinie (Foote et Yoder, 1997).

Tant les qualificatifs positifs que négatifs opèrent d'une manière métaphorique. Au-delà d'une approche poétique, nous les considérons ici comme des dispositifs cognitifs (Lakoff, Johnson, 1980) qui nous permettent de mettre au clair des aspects du domaine cible (c.-à-d. l'implémentation de l'algorithme), notamment leur existence en matière d'interface (clarté, transparence), de spatialité (spaghetti) et de temporalité (élégance éternelle, amas de boue).

2.2. — Propriétés spatiotemporelles

Si un programme est un objet mental, c'est aussi un objet sensuel, dont la matérialité est évoquée par celles qui le conçoivent et qui le manipulent. Il s'agit également d'un objet qui comporte des spécificités sur le plan des propriétés spatiales et

---

[2] « Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control » (Martin, 2008, p. 23). « Le code propre n'obscurcit jamais l'intention de la designer, mais est plutôt rempli d'abstractions bien tracées et de lignes de contrôle évidentes. »

[3] « Don't spoil a perfectly good program by over-embellishment and over-refinement » (Hunt, 1999) « Ne gâchez pas un programme parfaitement convenable par un sur-embellissement et trop de décorations. »

[4] « elegant : adj. [common ; from mathematical usage] Combining simplicity, power, and a certain ineffable grace of design » (Jargon File, 2003) « élégant [adj., d'usage mathématique] : Combine simplicité, pouvoir et une certaine grâce ineffable du design. »

évolutives, lesquelles, sans être strictement physiques et euclidiennes, peuvent néanmoins être saisies par l'apparence, afin de montrer une tendance à la représentation de relations et à la facilitation de navigation.

2.2.1 — Les relations

L'utilisation de techniques de visualisation permet également de faire sens de la structure de l'objet développé (Diehl et Görg, 2006), au-delà des références discursives que nous venons de voir. Une telle visualisation dépend notamment du fait que la structure d'un programme puisse être considérée comme celle d'un graphe (Bernhardt, 2021) et qu'un algorithme puisse se définir sur le plan de la géométrie (Naibo et Seiller, 2022).

Le programme-objet peut donc être pensé sous la forme d'un réseau de relations dynamiques, telles que celles de fonctions entre elles (*call graph*), de classes entre elles (*inheritance graph*) ou de modules de code entre eux (*dependency graph*) ; ces relations peuvent être intrinsèques au programme, dans les deux premiers cas, ou extrinsèques, dans le second cas. Cet aspect relationnel est explicité par la notion de milieu associé (Simondon, 2012), qui consiste en l'ensemble des éléments mobilisés par un objet technique et résultant en une individuation de cet objet technique. Cette individuation montre bien que l'objet technique se définit en tant qu'entité distincte, mais se révèle aussi en tant qu'interface pour son milieu associé. Par exemple, un algorithme tel que les T-Splines, qui permet de dessiner des courbes d'une certaine qualité au sein du logiciel de conception industrielle Fusion 360, existe en tant qu'entité individuelle (qui peut d'ailleurs être soumise aux lois du marché, puisqu'elle a été acquise par Autodesk en 2011), mais aussi en tant que mise en relation avec d'autres entités, telles que les entités de surface et les entités solides (Sohi, 2020). L'algorithme en tant qu'objet qu'est le T-Spline peut donc être considéré comme un objet et aussi comme une interface.

2.2.2 — La navigation

La structure relationnelle d'un programme peut donc être considérée par les programmeuses comme un objet, un artefact concrétisé, mais peut ensuite devenir médium pour les utilisatrices par cette même vertu de mise en relation. Un autre aspect de la matérialité du programme, c'est-à-dire de sa possibilité d'être saisi par l'utilisatrice, est donc celui de la navigation.

Par exemple, la conception et l'utilisation du logiciel de conception pour applications multimédia en temps réel TouchDesigner nous montrent la traduction de cette structure en termes visuels. Chaque programme de TouchDesigner est composé de nœuds, lesquels maintiennent des données et effectuent des opérations sur les données procurées par d'autres nœuds. La clarté et la propreté d'un programme TouchDesigner peuvent être visuellement représentées par la minimisation des interconnexions entre nœuds, avec des lignes visibles dans l'interface graphique établissant les relations programmatiques entre chaque composant. La qualité de l'objet peut être également mise en avant par l'utilisation d'une navigation en réseau permettant le déplacement de l'utilisatrice à l'intérieur ou à l'extérieur d'un nœud, lequel va alors être considéré comme objet ou interface, respectivement.

Cette navigation fait référence enfin à un autre aspect de l'existence d'un programme : celle des ordres de grandeur. Un programme existe bien à différents niveaux, niveaux

d’abstraction qui parcourent un dégradé depuis la concrétisation du hardware jusqu’à l’abstraction de l’objet mental (Détienne, 2002). Suivant que la designer tout comme l’utilisatrice du programme se déplacent le long de ce dégradé, les propriétés d’un programme vont présenter des natures différentes (Hui, 2016).

2.2.3 — Le temps

Il est également intéressant de noter certaines propriétés temporelles des programmes résultant de cette mise en relation. Par exemple, le terme de *software rot*[5] fait référence au phénomène de dégradation d’un programme qui, sans se faire de manière purement matérielle (les bits du programme ne se dégradent pas), est tout de même une réalité de l’existence et de l’utilisation d’un programme. Il s’agit de la désynchronisation d’un programme avec l’environnement (le milieu associé) dans lequel il opère. Certains programmes peuvent être plus fragiles et plus sensibles que d’autres au changement de leur environnement : un programme dépendant d’une interface concrète telle qu’une carte graphique sera plus facilement sujet à des évolutions externes qu’un programme opérant sur lui-même, tel un programme de calcul. L’évolution dans le temps est une évolution dans l’écosystème et demande donc aux concepteurs de programmes de repenser l’aspect temporel de l’objet créé.

D’autre part, nous pouvons aussi saisir la temporalité des programmes non pas par la manière dont ils changent dans le temps, mais par la manière dont ils conçoivent une certaine approche du temps : Baptiste Mélès (2017) montre que la notion du temps dans les systèmes d’exploitation UNIX est conçue par les créateurs du programme, des temps internes et externes, tandis que Gabriel Alcaras (2022) montre comment le programme Git permet une conception du temps qui se conjugue à l’organisation productive. La conception d’un programme n’est donc pas strictement celle de relations à un temps donné, mais également de relations dans le passé et le futur, relations plus complexes que celles d’une conception classique, mais néanmoins pas limitées à des objets digitaux, comme Lewis Mumford l’a montré dans son analyse de l’horloge comme objet produisant des secondes (Mumford, 1934).

En abordant la sensibilité des designers d’objets algorithmiques par une analyse discursive, nous avons abordé ces objets algorithmiques à travers une double perspective spatiotemporelle. La perspective spatiale a révélé les affordances d’échelles et de navigation propres aux objets digitaux, tandis que la contrepartie temporelle a révélé des possibilités parallèles d’historicité, mais aussi de déliquescence de l’abstraction à long terme, dévoilant une nouvelle/ancienne matérialité de l’infrastructure numérique.

La notion d’ordre de grandeur, adaptée aux objets digitaux, nous permet donc de voir le rapport objet-médium, tel qu’il existe au sein d’un environnement numérique, de manière plus relative. Après avoir montré que les objets digitaux circulent dans les médiums du design, nous avons vu que ceux-ci sont eux-mêmes le résultat d’un processus de design. C’est vers cette imbrication que nous nous tournons à présent, en considérant les algorithmes comme des objets médiatisants.

3.— Instances de matériaux

[5] Pourriture logicielle.

Il s’agit maintenant de s’intéresser plus fondamentalement aux aspects uniques de la matière numérique. Pour cela, nous proposons de nous appuyer sur l’analyse du mode d’existence des objets digitaux par Yuk Hui (2006) et de nous pencher sur les deux aspects essentiels qu’il identifie pour les objets digitaux : la récursivité et la relation.

3.1. — Récursivité

La récursivité consiste en la possibilité pour un concept d’être défini en des termes qui l’incluent. Par exemple, l’acronyme GNU, *GNU’s Not UNIX*, est un acronyme récursif. Cette référence au système d’exploitation montre l’affinité qu’ont les programmeuses avec ce concept de récursion. Ici, nous définissons la récursion des médiums comme la capacité d’un objet à devenir médium, pour ensuite créer des objets qui eux-mêmes deviendront médiums.

Des années 1960 aux années 1990 s’est développée une grande quantité de modeleurs et d’outils algorithmiques pour l’animation, le design, l’architecture, en parallèle les uns des autres, mais toujours par les designers eux-mêmes. Form*Z, Digital Project ou Processing, parmi de nombreux autres, en sont des exemples, tous reflétant l’appropriation du médium par des praticiennes et son adaptation pour coller au plus près de leurs besoins dans le processus de conception. L’instant partagé de l’émergence de ces outils impose une idée : les designers peuvent et doivent créer leurs propres outils informatiques. Cette injonction ne perdure pas dans les pratiques conventionnelles, mais devient pourtant un marqueur clé des pratiques de design computationnelles, inspirées par ce moment de réappropriation et de réinvention du médium. Développer ses propres outils conserve un sens très large : algorithmes, plug-ins, logiciels, robots, outils de fabrication robotique — la notion d’outils recouvre toutes ces possibilités sans distinction, mais l’impératif demeure fort, porté par une ambition de reprise de contrôle face à l’industrialisation et à la standardisation croissante des métiers du design.

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 enfin sont réalisés par l’intermédiaire de programmes déjà codés et simplement exécutés une nouvelle fois avec de nouveaux paramètres. Chaque praticienne ou agence se construit ainsi un univers algorithmique qui évolue au fil de la conception architecturale des différents projets menés. Le point commun des différents outils utilisés, c’est d’être sur-mesure, par opposition aux logiciels prêts à l’emploi.

Les différents formats d’interaction avec l’algorithme peuvent être compris en regard de cet enjeu comme des marches d’apprentissage, des niveaux de progression, car ils nécessitent différents niveaux de compétence. Les environnements comme Grasshopper sont des boîtes à outils qui offrent la possibilité d’assembler des groupes de fonctions spécifiques pour former des algorithmes. Ainsi, même les formats d’interaction les plus accessibles permettent de construire son propre code, et donc son propre outil, de façonner un objet qui, à son tour, devient outil. La programmation visuelle à laquelle les plug-ins sont souvent associés forme à ce titre un intermédiaire intéressant entre logiciels complets et un champ des possibles parfois trop vaste pour des novices de la programmation.

Au-delà de la possibilité de recréer des objets-médiums à partir des médiums objets, la tendance à considérer les objets digitaux uniquement comme interface peut trouver son fondement dans la deuxième propriété de tels outils digitaux : la mise en

relation.

3.2. — Relation

La multiplicité d'approches des algorithmes comme médiums du design évoquée en première partie tend enfin à une seconde propriété de ces algorithmes. Plutôt qu'un objet aux limites clairement identifiées par des propriétés physiques, ces algorithmes ont un rapport particulier à la mise en relation. Cette relation, ce n'est pas seulement la relation entre objets, mais aussi la relation comme objet. C'est ce « comme » qui permet de révéler ce qui fait de cet objet un médium (Hui, 2006).

Comme nous l'avons vu, le milieu associé d'un objet technique est l'ensemble des entités qui sont mises en relation par cet objet technique : les utilisatrices, les environnements sur et au sein desquels l'objet technique opère et les autres objets techniques qui y sont reliés ; et les objets digitaux, en tant que techniques de manipulation de l'information, font de la mise en relation, de l'encodage et du décodage de l'information le fondement de leur opération.

Il s'agit alors de mettre en exergue ce genre de relations en tant qu'objet : la disparition temporaire des palettes de couleurs Pantone de la suite Adobe en 2022, à la suite d'une évolution du programme de distribution de Pantone. Plutôt qu'exister au sein d'un fichier Adobe, les couleurs de Pantone seront désormais accessibles commercialement à travers une relation technique à un nouvel objet numérique, Pantone Connect. Cette relation technique, appelée Application Programming Interface[6] (API), consiste en un objet permettant l'accès selon des règles conçues par des impératifs non seulement techniques, mais aussi économiques et légaux. Ce genre d'objet est une nouvelle donnée dans le champ du design et atteint le statut de médium invisible lorsqu'il s'agit de la dernière vague d'outils basés sur des techniques d'intelligence artificielle, tels que ChatGPT ou Midjourney : c'est bien l'accès à l'API qui permet le travail de conception, et non l'accès à l'outil lui-même, situé à des milliers de kilomètres de l'utilisatrice dans des serveurs de l'entreprise qui a conçu cet objet numérique et les règles d'accès, dynamiques et modulaires, qui l'accompagnent.

Ce genre d'hybridité d'objet-médium permet, pour les designers traditionnels, de considérer le produit non pas uniquement comme un objet, mais aussi comme un médium pour l'utilisatrice de l'objet. Un dernier exemple, pour penser cette relation dynamique sur le plan de la manifestation physique, est celui du RepRap. RepRap est un objet produit assemblé en imprimante 3D, qui est lui-même composé de différentes entités[7]. Enfin, cette relation entre entités est elle aussi récursive : RepRap a pour but principal de pouvoir se recréer en imprimant chacune de ses pièces 3D, tentant de reconstituer dans le physique un analogue de la récursion numérique. Le médium de l'impression 3D, avec l'objet de l'imprimante, permet de recréer des objets qui ensuite redeviennent instances de médium.

Conclusion

[6] Interface de Programmation d'Application.

[7] On peut voir un graphe représentant les composants du RepRap ici : https://reprap.org/wiki/File:RepRap_Component_Structure.svg

La chaîne de production du design traditionnellement considéré ne peut plus être envisagée comme exclusivement linéaire : les concepts d'objets produits, de médiums et d'objets intermédiaires s'entremêlent. Leur matérialité suscite des questions : le médium, initialement considéré comme un objet intermédiaire voué à s'éclipser au profit d'un résultat qu'il a pour but de faciliter, sans matérialité propre, par contraste avec les produits du design, à la matérialité intentionnelle et identifiée, détient pourtant bien lui aussi une matérialité. Les pratiques de design, distinguant médium et objet, se retrouvent face à des nouveaux objets digitaux ; au lieu de remédier ces derniers afin de les adapter aux pratiques, il faudrait donc considérer adapter ces dernières pour prendre en compte les spécificités numériques de récursivité et de relation.

À partir du constat que la distinction entre le médium et l'objet est extrêmement floue, voire inexistante dans un environnement numérique, nous notons donc un nouveau statut pour les algorithmes-en-tant-qu'objets : des objets médiatisants. L'algorithme-en-tant-qu'objet médiatisant est donc abordé en fonction des usages, des émanations physiques, des expertises. À ce titre, l'algorithme-en-tant-qu'objet médiatisant redéfinit les frontières entre professions en faisant bouger les lignes des expertises associées.

De manière plus pratique, cette redéfinition des expertises pose la question de la maîtrise de la programmation, ou plus précisément de la *software literacy* nécessaire (Vee, 2017). Cela nous permet, à partir de l'algorithme-en-tant-qu'objet médiatisant, de suggérer un échange productif de pratiques entre programmeuses et designers, et de souligner l'existence de rôles intermédiaires ou, plus exactement, d'expertises intermédiaires, entre design et programmation.

L'algorithme-en-tant-qu'objet médiatisant ouvre donc la voie à de multiples solutions de remplacement à l'acquisition traditionnelle de la *software literacy* par l'apprentissage du code. Le code textuel n'est qu'un niveau du médium. La programmation visuelle, le travail par logigrammes ou par esquisses algorithmiques sont autant de manières de saisir les implications de l'algorithme-en-tant-qu'objet médiatisant pour le design. Il s'agit alors de mieux maîtriser le médium en le comprenant à un niveau plus fondamental, en tant que matière numérique.

RÉFÉRENCES

Abelson, H., Sussman, G. J. et Sussman, J. (1979). ***Structure and Interpretation of Computer Programs*** **(2e éd.). Justin Kelly.**

Alcaras, G. (2022). ***Des logiciels libres au contrôle du code : L'industrialisation de l'écriture informatique*** **[Thèse de doctorat, Paris, EHESS]. https://www.theses.fr/2022EHES0120**

Alexander, C. (1964). ***Notes on the Synthesis of Form*****. Harvard University Press.**

Alexander, C. (1979). ***The Timeless Way of Building*****. Oxford University Press.**

Arndt, S., Beer, A., Blümel, I., Elsner, C., Hauschke, C., Holste, D., Kampe, B., Lindlar, M., Mofakhamsanie, G., Noback, A., Saemann, H., Tittel, S., Voormann, F., Wermbter, K. et Winkler, R. (2022). FID Civil Engineering, Architecture and Urbanism digital - A platform for science (BAUdigital). ***Research Ideas and Outcomes, 8*** **: e82563.**

Bernhardt, G. (2021). Code Is Mostly Graphs. Destroy All Software. https://www.destroyallsoftware.com/compendium/code-is-mostly-graphs, consulté le 10 mai 2023.

Bird, L. et LaBelle, G. (2010). Re-Animating Greg Lynn's Embryological House : A Case Study in Digital Design Preservation. *Leonardo*, *43*(3).

Brand, S. (1987). The Medialab : Inventing the Future at MIT. Penguin Books.

Bryant, G. (2013). Gatemaker: Christopher Alexander's dialogue with the computer industry. Dans N. H. Joachim, G. A. Brown et G. Bryant (dir.), *Battle for the Life and Beauty of the Earth* (p. 77-91). PUARL Conference Proceedings, Portland Urban Architecture Research Laboratory.

Cache, B. (1998). Objectile : Poursuite de la philosophie par d'autres moyens ? *Rue Descartes*, (20), p. 149-57.

Deleuze, G. (1988). *Le Pli*. Éditions de Minuit.

Détienne, F. (2002). *Software Design – Cognitive Aspect*. Springer Science & Business Media.

Diehl, S. et Görg, C. (2006). Aesthetics and the Visualization and Quality of Software. Dans *Aesthetic Computing* (p. 229-237). MIT Press. https://ieeexplore.ieee.org/document/6285455

Evans, R., (1995). *The Projective Cast*. M.I.T. Press.

Foote, B. et Yoder, J. (1997). Big Ball of Mud. Laputan.Org. http://www.laputan.org/mud/mud.html#BigBallOfMud, consulté le 14 juillet 2023.

Fornes, M. (2007). Scriptedbypurpose. Consulté le 22 octobre 2023. https://scriptedbypurpose.wordpress.com/

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. et Gamma, E. (1999). *Refactoring: Improving the Design of Existing Code* (1re édition). Addison-Wesley Professional.

Frazer, John (2001). The Cybernetics of Architecture: A Tribute to the Contribution of Gordon Pask. *Kybernetes. The International Journal of Systems & Cybernetics*, *30*(5-6), p. 641-651.

Galloway, A. R. (2012). *The Interface Effect*. Polity.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. et Booch, G. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software* (1re édition). Addison-Wesley Professional.

Hodges, A. (2000). "Uncomputability in the work of Alan Turing and Roger Penrose", présentation donnée à Hambourg le 6 octobre 2000 dans le cadre de la conférence Interface 5. https://www.calculemus.org/MathUniversalis/NS/10/hodges1.html, consulté le 22 octobre 2023.

Hui, Y., (2016). *On the Existence of Digital Objects*. University of Minnesota Press. http://www.jstor.org/stable/10.5749/j.ctt1bh49tt

Hunt, A. et Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master* (1re éd.). Addison-Wesley Professional.

Hyponen, M., The Malware Museum, 2016. https://archive.org/details/malwaremuseum&tab=collection, consulté le 22 octobre 2023.

Jargon File 4.4.7. (2003). Elegant. Catb.Org. http://www.catb.org/~esr/jargon/html/E/elegant.html, consulté le 10 juillet 2023.

Kay, A. C. (1993). The early history of Smalltalk. *ACM SIGPLAN Notices*, *28*(3), 69-95. https://doi.org/10.1145/155360.155364

Kittler, F. A. (2014) *There Is No Software. The Truth of the Technological World: Essays on the Genealogy of Presence*. Redwood City : Stanford University Press, p. 219-229.

Krug, S. (2014). *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*. New Riders.

Lakoff, G., & Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press. https://press.uchicago.edu/ucp/books/book/chicago/M/bo3637992.html

Latour, B. (1994). Une sociologie sans objet ? Note théorique sur l'interobjectivité. Dans *Sociologie du Travail* (vol. 4, p. 587-607). MSH-Presses de l'Université Laval. http://www.bruno-latour.fr/fr/node/231

Leonardi, P. M. (2010). Digital materiality? How artifacts without matter, matter. *First Monday*, *15*(6-7). https://firstmonday.org/ojs/index.php/fm/article/download/3036/2567

Martin, R. C. (2008). *Clean Code : A Handbook of Agile Software Craftsmanship*. Pearson Education.

Mélès, B. (2017). Time and activity in Unix. *Réseaux*, *206*(6), 125-153.

Mumford, L. (1934). *Technics And Civilization*. Harcourt Brace Jovanovich. http://archive.org/details/in.ernet.dli.2015.49974

Naibo, A. et Seiller, T. (2022). *Formalizing The Notion of Algorithm, History and Philosophy of Computing*. Warsaw.

Naur, P. (1985). Programming as theory building. *Microprocessing and Microprogramming*, *15*(5), 253-261. https://doi.org/10.1016/0165-6074(85)90032-8

Rapaport, W. J. (2023). *Philosophy of Computer Science*. Wiley-Blackwell.

Reichert, R. et Richterich, A. (2015). Introduction. Digital Materialism. Dans *Digital Culture & Society*, *1*(1), 5-17. Jg.

Saint-Exupéry, A. de (1972). *Terre des Hommes*. Gallimard.

Simondon, G. (2012). *Du mode d'existence des objets techniques*. Aubier.

Sohi, P. (2020, 29 mai). From Solids to Organics: Solids, Surfaces, and T-Splines. Fusion 360 Blog. https://www.autodesk.com/products/fusion-360/blog/from-solids-to-organics-when-and-how-to-use-solids-surfaces-and-t-splines/

Steele, G. L. (1977). Macaroni is better than spaghetti. *ACM SIGPLAN Notices*, *12*(8), 60-66. https://doi.org/10.1145/872734.806933

Turner, R. (2018). Computational Artifacts. Dans R. Turner (éd.), *Computational Artifacts : Towards a Philosophy of*

Computer Science **(p. 25-29). Springer. https://doi.org/10.1007/978-3-662-55565-1_3**

Vee, A. (2017). ***Coding Literacy: How Computer Programming Is Changing Writing*****. The MIT Press. https://doi.org/10.7551/mitpress/10655.001.0001**

Witt, A. (2021). ***Formulations : Architecture, Mathematics, Culture*****. M.I.T. Press.**

Woodbury, R. (2010). ***Elements of Parametric Design*****. Routledge, New York.**