Développement d'un modèle logiciel de cellule sur processeurs multi-cœurs pour la simulation de morphogenèse de tissus

Anne Jeannin-Girardon

THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE
Mention : Informatique
École Doctorale SICMA

présentée par
Anne JEANNIN-GIRARDON
Préparée au département informatique de l'université de Brest
Laboratoire Lab-STICC UMR 6285

Développement d'un modèle logiciel de cellule sur processeurs multi-cœurs pour la simulation de morphogenèse de tissus

Thèse soutenue le 21 novembre 2014
devant le jury composé de :

Angélique STEPHANOU
Chargée de recherche CNRS (HDR), Institut de l'Ingénierie et de l'Information de Santé / *rapportrice*

Yves DUTHEN
Professeur, Université de Toulouse / *rapporteur*

Marie BEURTON-AIMAR
Maître de conférences (HDR), Université de Bordeaux / *examinatrice*

Dominique LAVENIER
Directeur de recherche CNRS, Institut de Recherche en Informatique et Systèmes Aléatoires / *examinateur*

Pascal BALLET
Maître de conférences (HDR), Université de Brest / *examinateur*

Vincent RODIN
Professeur, Université de Brest / *examinateur*

Ces travaux de thèse ont été financés par la région Bretagne

The saddest aspect of life right now is that science gathers knowledge faster than society gathers wisdom.
Isaac Asimov

The usefulness of an opinion is itself matter of opinion.
John Stuart Mill

Remerciements

La réalisation de la thèse est constituée d'une succession d'étapes aux niveaux desquelles différentes personnes interviennent. Ce mémoire, sa rédaction et son évaluation, marquent la fin d'une étape importante qui doit permettre de formaliser de manière rigoureuse et détaillée les réalisations de la thèse et de situer précisément ces réalisations dans un domaine déjà bien fourni. Je remercie donc Mme Angélique Stéphanou et M Yves Duthen d'avoir accepté de rapporter mon travail de thèse à travers l'étude de ce mémoire et pour les judicieux conseils et remarques qu'ils m'ont prodigués. L'étape suivante de la thèse, qui est d'une certaine manière son aboutissement, est la soutenance. Moment important s'il en est, puisqu'il est l'occasion de montrer de manière didactique les réalisations de la thèse : je remercie M Dominique Lavenier d'avoir examiné mon travail et d'avoir présidé mon jury de thèse, ainsi que Mme Marie Beurton-Aimar pour avoir également examiné mon travail. De manière plus générale, merci à tous les membres du jury pour les échanges inspirants qui ont suivi la présentation de mes travaux de thèse.

Mais je n'ai évoqué là que les dernières étapes d'un travail de thèse. D'une manière générale, on trouve là une succession d'étapes diverses et variées, tant par leur nature que par leur nombre. Certaines sont plus évidentes que d'autres ; certaines sont décourageantes tandis que d'autres sont de véritables tremplins vers la suite. À mon sens, c'est précisément cette diversité qui rend ce travail aussi passionnant et fascinant. Mais ce chemin sinueux ne peut être abordé en solitaire ; à tout le moins, il ne devrait pas l'être. Il ne faut pas oublier que la thèse reste une formation à la recherche — voire au métier d'enseignant-chercheur lorsque l'on a la chance, comme je l'ai eu durant ma thèse, de pouvoir dispenser des enseignements — et qu'il est donc important de se sentir guidé et soutenu afin de traverser ces différentes étapes, les bonnes comme les moins bonnes, sans trop laisser de plumes au passage. Je remercie donc Pascal, dont la liberté d'esprit et l'inventivité ont beaucoup apporté à ces travaux. Lorsque j'ai débuté ma thèse, j'avais dans l'idée (qui ne m'a en fait jamais vraiment quittée) que le directeur d'une thèse n'était qu'une créature que l'on pouvait apercevoir, avec un peu de chance, les soirs de pleine lune uniquement (ou pour signer des papiers). Il est difficile de juger pour les autres, mais en ce qui me concerne, je n'aurais pas pu me tromper davantage. J'adresse donc mes remerciements, s'il en est encore besoin, à Vincent, pour sa présence, sa patience, ses conseils et ses idées.

D'autres étapes ont encore précédées la thèse. Ces étapes ont vu passer de nombreuses personnes qui ont, à des degrés divers, apporté leur contribution à cet aboutissement. Je ne peux toutes les remercier ici mais si elles et ils lisent ces quelques lignes, je sais qu'elles et ils sauront se reconnaître ; leur apport a été apprécié. L'entourage peut jouer un rôle important dans le déroulement d'une thèse, même si je pense que cela reste une aventure très personnelle tant dans son déroulement que dans la façon dont on peut le vivre. Merci à Pierre (docteur !), qui s'est aussi lancé dans une telle entreprise, pour son soutien ces dernières années. Enfin, merci également à ma famille, mon grand frère ainsi que mes (incroyables) parents, qui ont toujours été d'un soutien inconditionnel.

À mes parents

Table des matières

Table des figures

Liste des tableaux

Introduction

Les travaux présentés dans ce mémoire de thèse se situent dans le contexte interdisciplinaire de l'informatique et de la biologie (figure 1). L'ordinateur est ici un moyen pour reproduire en simulation des mécanismes fondamentaux intervenant lors de la morphogenèse de tissus sains ou pathologiques.

L'ordinateur est depuis ses débuts, dans les années 1950, utilisé pour tenter de reproduire le vivant à différents niveaux. Par exemple, John Von Neumann a développé le principe de l'automate cellulaire, où une matrice contient différentes valeurs pouvant représenter l'état des cellules de l'automate. En ajoutant des règles de voisinage et de transition, il a défini des comportements d'évolution de ces cellules virtuelles, reproduisant par là une notion forte du vivant : à partir de relations locales naissent des comportements globaux plus complexes. Dans la même pensée, Alan Turing développa un système de réaction-diffusion qui posa les bases de l'étude de la morphogenèse par simples réactions chimiques. Ces deux travaux fondateurs sont à la base de nombreuses branches dans l'informatique où le vivant se mêle à l'informatique. Citons par exemple la vie artificielle, l'intelligence artificielle, les réseaux de neurones, les L-systèmes ou les systèmes multi-agents.

Un problème que l'on retrouve dans toutes ces branches est la nécessité de faire appel à d'importantes capacités de calcul. Or, depuis les années 2000 nous sommes à la croisée des chemins et nous ne pouvons plus compter sur l'augmentation des fréquences d'horloges (nombre de calculs réalisés par seconde) des unités de traitement de nos ordinateurs pour gagner en puissance de calcul. En effet, à cause de la consommation d'énergie engendrée par des fréquences importantes (supérieures à 5 GHz), la production de chaleur est telle que sa dissipation ne peut plus être effectuée par les systèmes de refroidissement classiques à base de ventilateurs (IBM a néanmoins réalisé une puce fonctionnant à 500 GHz, mais nécessitant d'être refroidie à -268^{o}C). Les fréquences stagnant, depuis quelques années les fondeurs multiplient les unités de calculs dans leurs processeurs (CPU et GPU) et donc dans nos machines (ordinateurs personnels, smartphones, tablettes, etc). La programmation de matériels parallèles demande toutefois un changement de paradigme, certes initié depuis

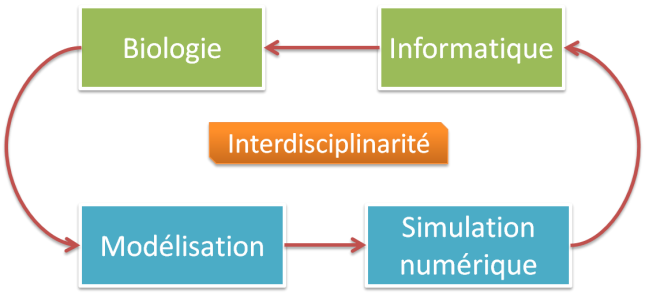


Figure 1 – Contexte de la thèse et interdisciplinarité. Les travaux de thèse présentés dans ce mémoire sont au croisement de deux disciplines : l'informatique et la biologie. Ces deux disciplines sont reliées, dans le présent contexte, par la notion de *modélisation* et de *simulation numérique*.

déjà de nombreuses années, mais restant souvent cantonné à des domaines spécifiques non liés à notre problématique. Fort heureusement, des plateformes de développement voient le jour et permettent de faciliter, du moins en partie, la programmation de ces matériels. Certaines plateformes, comme OpenCL, permettent même de programmer de manière unifiée ces matériels hétérogènes. Il devient alors possible, au prix d'une adaptation de la programmation, de progresser en terme de puissance de calcul utilisable pour simuler toutes sortes de processus biologiques et notamment la morphogenèse de tissus impliquant plusieurs millions de cellules en interaction. La simulation numérique d'un système biologique virtuel devant contenir un grand nombre d'entités en interaction ne peut se faire qu'en tirant partie des technologies adaptées. Les technologies comme les cartes graphiques sont aujourd'hui devenues très abordables du fait de leur développement massif grâce aux jeux vidéo. Les cartes graphiques sont des matériels hautement parallèles et leur modèle de programmation (consistant à traiter des flux de données) est bien adapté à l'idée de système multi-cellulaires. Il en va de même pour les processeurs multi-cœurs qui, s'ils ne comportent pas autant d'unités de calcul que les cartes graphiques, permettent de traiter rapidement du code complexe tout en offrant un certain degré de parallélisme. Cette idée est d'autant plus intéressante lorsque l'on se place dans un cadre de modélisation discrète, où l'on désire modéliser des entités autonomes en interaction. Comme on le verra dans ce mémoire, à travers des états de l'art et la présentation de nos travaux, ces résultats sont atteignables aujourd'hui.

La seconde discipline qui concerne nos travaux est une branche de la biologie qui se focalise sur l'étude du développement de tissus dont les composants fondamentaux sont les cellules. Il s'agit de la morphogenèse de tissus cellulaires, d'organes ou d'organismes entiers, sains ou défaillants. De manière générale, l'étude de systèmes multi-cellulaires implique l'étude et l'utilisation de gros volumes de données provenant d'observations *in vivo* et *in vitro*. Plusieurs problèmes se posent sur ces données, comme par exemple leur fiabilité numérique ou bien leur interprétation dans des modèles explicatifs. Nous nous intéressons plus particulièrement aux problèmes de l'interprétation de ces données au travers de modèles capables de les utiliser pour expliquer les mécanismes élémentaires qui régissent la morphogenèse de tissus. L'idée principale est de fournir aux biologistes des outils de simulation suffisamment réalistes et expressifs pour qu'ils puissent être épaulés dans cette longue et difficile étape de l'interprétation. En effet, l'intégration de ces données à des modèles peut aider les biologistes à acquérir une connaissance plus approfondie du système réel. La modélisation peut en outre permettre aux biologistes de formaliser leurs connaissances sur un système donné.

La notion de modèle constitue le premier lien entre les deux disciplines présentes dans le cadre de ces travaux de thèse. Bien que ce terme soit utilisé dans les deux domaines, de fortes différences existent. Il convient de bien les définir pour éviter les sources d'incompréhension entre biologistes et informaticiens. Du côté de la biologie, un modèle est une représentation d'un système biologique réel. Il tente d'expliquer ou de faire des prévisions sur un système réel. Les modèles en biologie sont construits de diverses manières : cela va, par exemple, du texte en langage courant en passant par les diagrammes d'interactions en allant jusqu'aux systèmes d'équations. En ce qui concerne l'informatique, un modèle peut représenter une façon d'architecturer un logiciel. Ces modèles utilisent différents langages dont les plus usités sont les algorithmes et le langage UML (*Unified Modeling Language*). Un modèle sert en premier lieu à construire un programme. Dans un deuxième temps, un modèle sert aussi à représenter un logiciel pour en assurer la lisibilité et la maintenance. Malgré ces différences, notons qu'un modèle joue un rôle important dans la compréhension et la maîtrise d'un système, qu'il soit biologique ou informatique.

Nous proposons dans ce travail de thèse de bâtir, à partir des connaissances actuelles en biologie, des briques logicielles capables de reproduire en simulation certains mécanismes exprimés par les cellules vivantes lors la morphogenèse de tissus. Ces briques logicielles servent alors à bâtir des modèles en biologie pour comprendre les systèmes biologiques réels. Les briques que nous réalisons ont la particularité d'être :

1. réalistes par rapport aux observations afin de contraindre les simulations à correspondre à la physique, la biologie et la chimie ;
2. combinables entre-elles pour exprimer toutes sortes d'idées ;
3. efficaces en matière de calcul pour s'approcher des importantes quantités de cellules en interaction dans les tissus vivants (cette partie à souvent tendance à s'opposer au point 1 car des simplifications sont nécessaires).

Outre la notion de modèle, nous devons aussi comparer la notion d'expérience en biologie et en informatique. Pour un biologiste, une expérience va consister à réaliser des tests *in vitro* ou *in vivo*. Pour un informaticien, une expérience sera faite par le biais de simulations, d'expériences *in silico*.

Dans ces deux cas, il s'agit ici d'observer des systèmes dynamiques et d'en extraire de la connaissance. Malgré cette apparente similarité, les différences fondamentales entre ces deux approches doivent être bien comprises :

4. en biologie, lors d'une expérience, toutes les données ne sont pas connues (possiblement très peu), ce qui complique la tâche de compréhension du système étudié. En informatique, tous les éléments sont connus et maîtrisables, ce qui permet une étude plus sûre sur le système (attention toutefois car deux modèles différents peuvent donner des résultats similaires ou parfois, par manque de puissance de calcul, il n'est pas possible d'atteindre le niveau de détails souhaité) ;
5. en biologie, il est possible de découvrir de nouveaux éléments jusque là inconnus (une nouvelle molécule, un nouveau comportement cellulaire, etc) alors qu'en informatique il n'y a dans la simulation que ce que l'informaticien à placé (cependant une simulation peut permettre de détecter des dynamiques inattendues, en positif si cela reproduit effectivement ce qui est réellement observé ou en négatif si la simulation ne correspond pas à la réalité) ;
6. en biologie le système se comporte comme il doit réagir alors qu'en informatique, à cause des simplifications ou des erreurs d'implémentation, il peut avoir des comportements irréalistes rendant son utilisation au mieux inutile ou au pire trompeuse.

En gardant à l'esprit les 6 points exprimés précédemment, nous proposons d'une part un modèle individu-centré de cellule virtuelle avec son environnement mécanique et chimique. D'autre part, nous proposons un simulateur parallèle, pour simuler de nombreuses cellules virtuelles en interaction dans l'optique d'étudier la morphogenèse de tissus sains ou pathologiques ; il s'agit là de nos deux propositions principales.

Nous avons mis en place, de manière simplifiée, de nombreux processus impliqués dans la morphogenèse de tissus sains ou pathologiques tels que les tissus cancéreux : migration, division, déformation, adhésion différentielle, prise en compte de contraintes physiques, production et consommation de molécules, cycle cellulaire, apoptose, lyse, interactions cellulaires via des signaux biochimiques et interactions des cellules avec leur environnement. Nous avons focalisé notre travail sur l'implémentation de ces comportements en sachant qu'ils peuvent intervenir dans tout type de développement de tissus. Dans ce mémoire, nous

montrerons divers exemples présentant des éléments de validation de ces différents comportements. Parmi ces exemples, le dernier prend en compte tous ces comportements sur un exemple de cancer où nous avons reproduit, de manière très simplifiée, un bourgeonnement de vaisseau sanguin dans le cadre du phénomène d'angiogenèse lié au cancer. Notons que cet exemple a vocation à être fortement amélioré pour le rendre plus réaliste par rapport à des expérimentations réelles.

Organisation du mémoire

Ce mémoire est constitué de deux parties, reprenant nos deux principaux axes de travail : la première partie, « Modélisation de systèmes biologiques » comporte trois chapitres :
« Contexte biologique et positionnement » : ce chapitre nous permet de préciser plus avant le contexte de la thèse, tant d'un point de vue biologique qu'informatique. La présentation de la cellule biologique nous permet de dégager les points que nous retiendrons dans notre proposition de modèle de cellule virtuelle. Nous abordons également les notions de modélisation et de simulation numérique en apportant des précisions sur ces deux notions et en présentant les objectifs liés à la modélisation ainsi que ses limites.
« Approches pour la modélisation de systèmes biologiques » : l'objet de ce chapitre est de présenter en détails les deux grandes approches pour la modélisation de systèmes biologiques : l'approche continue et l'approche discrète. Ces approches sont étudiées à travers des cas d'étude tirés de la littérature. Nous dressons par la suite une synthèse complète des deux approches nous permettant de nous positionner pour la suite de ces travaux.
« Proposition : modèle de cellule virtuelle » : dans ce chapitre, nous introduisons notre modèle biomécanique de cellule virtuelle. Après avoir présenté le cœur du formalisme de modélisation que nous utilisons, à savoir un système multi-agents, nous présentons notre modèle de cellule à travers différents éléments : sa structure physique ; ses comportements (cycle cellulaire, croissance des cellules, division cellulaire, interactions et prise en compte des contraintes physiques issues de l'environnement) ; une chimie artificielle puis le couplage entre le modèle conceptuel et un système multi-agents, abordant ainsi les notions de granularité du système, d'ordonnancement des agents, de mémoire des agents, d'environnement et d'interactions entre des agents.

La seconde partie de ce mémoire, « Simulateurs parallèles, applications et résultats » comporte quant à elle deux chapitres :
« Intérêts de la programmation parallèle » : ce chapitre est une introduction à la programmation parallèle. En partant de la notion de parallélisme au sens large (abordant les motivations de la programmation parallèle et les contraintes liées à ce type de programmation), nous resserons ensuite notre étude pour présenter différents environnements permettant de programmer en parallèle à la fois des processeurs « classiques » (CPU) et des processeurs graphiques (GPU). Nous nous sommes en particulier intéressés à la plateforme de développement OpenCL car elle permet de programmer des matériels variés. Pour notre problématique, la présentation de l'architecture des GPU et de la plateforme OpenCL nous permet de proposer un simulateur parallèle capable d'être utilisé sur matériels hétérogènes.
« Proposition d'un simulateur parallèle et exemples d'application » : ce dernier chapitre est dédié d'une part à la présentation de notre simulateur parallèle et d'autre part à la présentation de quelques cas d'étude que nous avons réalisé avec nos outils. Une revue de différents simulateurs présentés dans la littérature nous permet de nous positionner par rapport à l'existant. Nous présentons ensuite notre simulateur : architecture logicielle, structures de données adaptées à des matériels parallèles, mécanismes de gestion mémoire et para-

métrage du modèle. Après avoir présenté l'architecture et l'implémentation de notre modèle, nous introduisons diverses simulations de systèmes biologiques comportant des cellules en interaction. Ceux-ci nous permettent d'apporter des éléments de validation tant du point de vue du modèle que de son implémentation. Une étude de performances du simulateur est également présentée dans ce chapitre.

Ce mémoire est terminé par un chapitre de conclusions permettant de dresser un bilan des travaux que nous avons réalisés et introduits dans le présent document. Nous y décrivons également les différentes perspectives de travail que nous envisageons de poursuivre.

Première partie

Modélisation de systèmes biologiques

Introduction à la partie I

Dans la première partie de ce document, nous abordons les notions de modèles appliqués à la simulation de systèmes biologiques. Le premier chapitre nous permet d'introduire des notions de biologie cellulaire que nous retrouverons au fil du mémoire. Nous évoquons également les intérêts de la simulation numérique, en montrant qu'elle est aujourd'hui présente dans tout type de domaines et qu'elle sied particulièrement à la biologie. Le second chapitre est consacré à la présentation de l'état de l'art des approches existantes pour modéliser les systèmes biologiques. A travers une étude de travaux variés issus de la littérature, nous établissons une comparaison approfondie des deux approches majeures de la modélisation : l'approche continue et l'approche discrète. Ces éléments nous permettent de positionner une partie de nos travaux de thèse à savoir la proposition d'un modèle biomécanique de cellule virtuelle. Le chapitre trois est consacré à la présentation de ce modèle de cellule virtuelle : structure de la cellule, comportements implémentés, chimie artificielle et couplage au formalisme de modélisation choisi : un système multi-agents.

CHAPITRE 1

Contexte biologique et positionnement

Ce chapitre présente des éléments de biologie et d'informatique permettant de positionner nos travaux de thèse. En premier lieu, nous introduisons la cellule biologique (structure, régulation et interactions) puis nous donnons un exemple de cas de développement de tissus pathologiques avec le cas du cancer. La section 3 nous permet, du point de vue de l'informatique, de préciser les notions de modélisation *et de* simulation numérique *et de présenter leurs avantages et leurs limites. Une dernière partie sera consacrée à une synthèse exprimant notre positionnement par rapport aux notions présentées dans ce chapitre.*

Sommaire

1 La cellule biologique

1.1 Structure d'une cellule biologique

La cellule est souvent considérée comme étant la brique de base du vivant. Ce principe constitue un des piliers de la théorie cellulaire, issue d'une part des progrès réalisés en microscopie au XIX^{e} siècle ainsi que des observations réalisées, entre autres, par le botaniste Matthias Schleiden et le zoologiste Theodor Schwann en 1839 (Schwann, 1839). La cellule est un élément commun à tout organisme vivant. Il existe deux types majeurs de cellules : les cellules eucaryotes, qui contiennent un noyau, et les cellules procaryotes qui en sont dépourvues. Les bactéries représentent des exemples d'organismes procaryotes. Ces cellules sont structurellement plus simples que les eucaryotes et ont une taille moyenne variant de 0.5 à $1.5\mu m$ (Klein et Satre, 2008). Le matériel génétique des cellules procaryotes est libre dans le cytoplasme. Les cellules eucaryotes composent quant à elles des organismes unicellulaires ou pluricellulaires, allant des amibes aux plantes ou aux animaux. Ces cellules ont un diamètre moyen variant de 20 à $25\mu m$ (Klein et Satre, 2008). Leur matériel génétique est, lui, contenu dans un noyau.

Les cellules eucaryotes Les cellules eucaryotes contiennent un cytosquelette. Le cytosquelette d'une cellule est une structure présente dans le cytoplasme et permettant le maintien de la cellule, la division cellulaire ou encore la *motilité* cellulaire[1]. Les trois éléments que l'on retrouve dans le cytosquelette d'une cellule sont :

– les filaments d'actines, jouant un rôle important dans la motilité cellulaire ;

1. La capacité des cellules à se déplacer

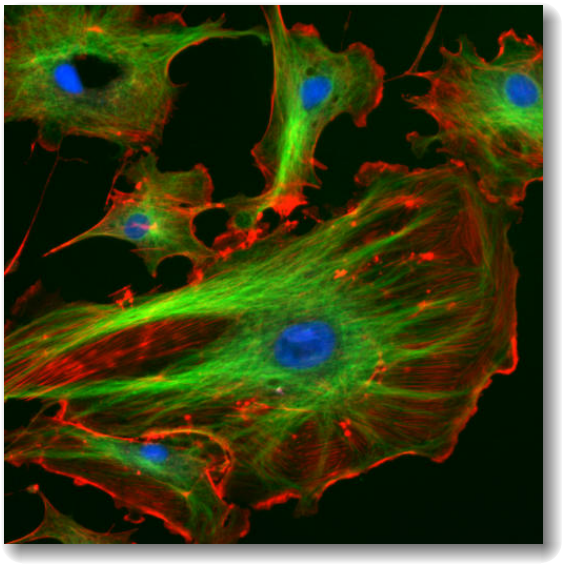


FIGURE 1.1 – Visualisation du cytosquelette d'une cellule eucaryote en microscopie à fluorescence. Les filaments d'actine sont visibles en rouge, les microtubules en vert et des filaments intermédiaires en bleu. Source de l'image : `http://rsb.info.nih.gov/ij/images/` (en ligne, consulté le 1er septembre 2014).

- les microtubules, intervenant eux lors de la mitose, notamment pour la séparation des chromosomes ;
- les filaments intermédiaires qui interviennent, de manière générale, au niveau du maintien de la structure de la cellule.

La figure 1.1 est une visualisation du cytosquelette d'une cellule eucaryote en microscopie à fluorescence. Les différentes structures du cytosquelette sont visibles dans différentes couleurs.

Les cellules eucaryotes contiennent différents compartiments assurant diverses fonctions. Parmi ces compartiments, on trouve un noyau (contenant le matériel génétique de la cellule), les mitochondries (qui permettent la respiration cellulaire, une réaction permettant de produire de l'ATP [2] qui est un carburant utilisé par les cellules), le réticulum endoplasmique (qui permet la synthèse de protéines), l'appareil de Golgi (permettant d'amener les protéines à maturité) et les lysosomes (qui dégradent des molécules complexes). La figure 1.2 illustre ces différents compartiments au sein de la cellule.

Une cellule biologique eucaryote a donc une structure complexe, grâce à son cytosquelette composé de différents types de filaments ; ces filaments assurent des fonctions spécifiques. Mais une cellule biologique eucaryote est également pourvue de divers compartiments, ou organites, assurant des fonctions variées. Nous verrons, lors du chapitre 2, que la prise en compte explicite de ces compartiments n'est pas forcément indispensable dans notre approche car les fonctions assurées par ces organites peuvent être prises en compte à plus haut niveau. Le cytosquelette de la cellule nous a, par contre, beaucoup plus intéressé : comme nous le verrons dans les chapitres suivants (2 et 3), la prise en compte, même simplifiée, du cytosquelette de la cellule permet de modéliser une cellule ayant une forme bien définie, ainsi que ses déformations.

2. Adénosine Triphosphate

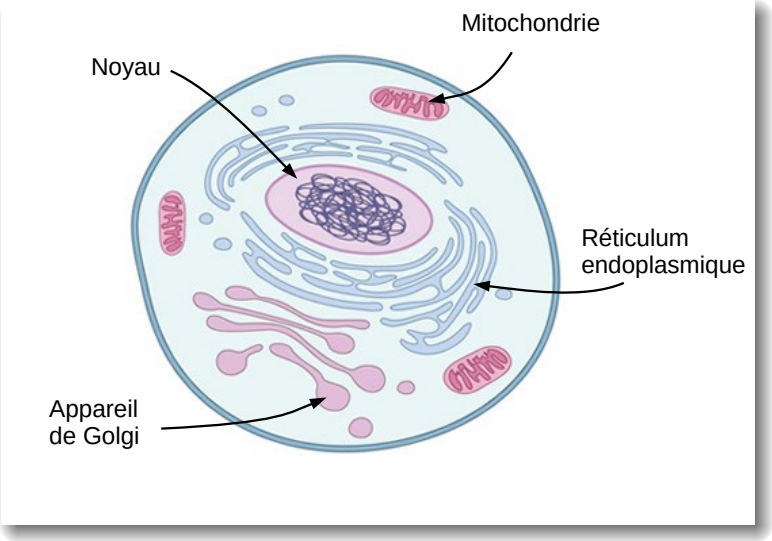


Figure 1.2 – Compartiments composant une cellule eucaryote. Les différents compartiments présents dans la cellule assurent des fonctions variées (production d'énergie, synthèse de protéines ou encore dégradation de molécules). Image adaptée de (O'Connor et Adams, 2010a).

1.2 Cycle cellulaire

La multiplication cellulaire, ou division, est un acte d'auto-réplication réalisé par les cellules. La division cellulaire est composée de deux étapes : la mitose, qui est la division des chromosomes, et la cytodiérèse, qui divise le cytoplasme et partage les organites présents dans le cytoplasme entre les deux cellules résultantes de la division. Avant que la division ne se produise, la cellule passe par différentes phases que l'on appelle le *cycle cellulaire*. Ce cycle est illustré sur la figure 1.3. Les différentes phases du cycles sont les suivantes : les phases G1 et G2 sont des phases pendant lesquelles des contrôles sont effectués, notamment afin de garantir que le matériel génétique est intègre. La phase S est la phase de synthèse de l'ADN et M est la phase de mitose. Pendant la phase G1, la cellule peut entrer en quiescence (elle ne se divise plus) dans le cas où l'environnement de la cellule contiendrait une quantité insuffisante de signaux de croissance. En matière de durée, il faut compter 10 à 20 heures pour qu'une division de cellule eucaryote ne se produise, contre une vingtaine de minutes pour une cellule procaryote.

Il faut noter que la multiplication n'a pas pour unique but la multiplication cellulaire en elle-même, mais elle permet aussi l'adaptation des organismes suites aux interactions des cellules avec leur environnement : nous sommes ici dans le registre de la *différentiation cellulaire*, permettant aux cellules de spécialiser leurs fonctions. Les cellules disposent toujours du même matériel génétique mais celui-ci est alors exprimé différemment.

Lors du cycle cellulaire, il faut à tout prix préserver l'intégrité du matériel génétique de la cellule. Les points de contrôle permettent de vérifier ce matériel. En cas de problème, le matériel génétique peut être réparé ou, dans le pire des cas, la cellule finit par entrer en apoptose (la mort cellulaire programmée). Si l'altération du matériel génétique concerne des gènes permettant de réguler le cycle cellulaire, alors cela peut avoir comme conséquence de donner lieu à des lignées de cellules anormales : nous reviendrons sur cet aspect dans la section 2 présentant des mécanismes intervenant lors du développement de cancer.

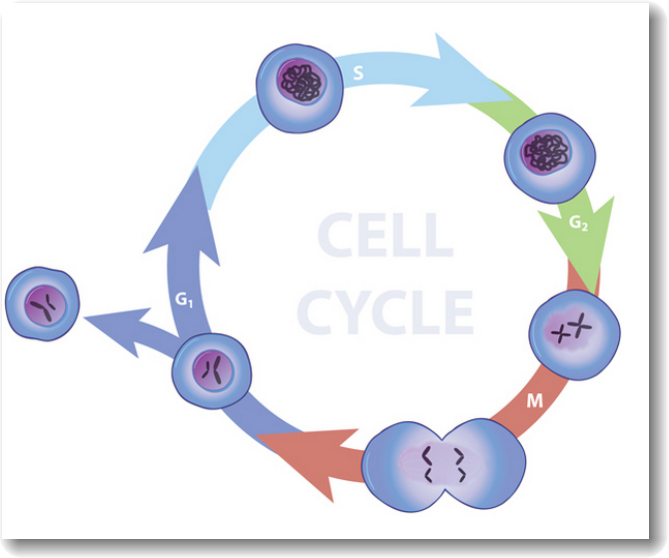


FIGURE 1.3 – Phases du cycle cellulaire. Image extraite de (O'Connor et Adams, 2010b).

1.3 Interactions cellulaire

Les interactions cellules / cellules ou cellules / matrice extra-cellulaire (le milieu dans lequel évoluent les cellules) sont indispensables au maintien des tissus, à la migration cellulaire ou encore à la réaction inflammatoire (Duperray, 2008). L'adhésion cellulaire est réalisée grâce à des protéines présentes sur la membrane des cellules. Ces protéines peuvent se lier avec d'autres protéines présentes sur d'autres cellules ou dans la matrice extra-cellulaire.

Il existe différentes molécules d'adhésion qui sont regroupées en familles. Les intégrines constituent la famille principale intervenant dans les liaisons cellules / matrice extra-cellulaire. La famille des cadhérines intervient principalement dans les adhésions cellules / cellules. Enfin, la famille des immunoglobulines intervient dans la reconnaissance cellules / cellules et joue en particulier un rôle dans le fonctionnement du système immunitaire. Le cytosquelette des cellules intervient de manière importante dans les adhésions cellulaires : les molécules de cadhérine sont par exemple ancrées aux filaments d'actine du cytosquelette.

Il existe différents types d'adhésion entre cellules, appelées jonctions. Les jonctions étanches bloquent le passage des molécules entre les cellules jointes ; les jonctions adhérentes permettent l'adhésion d'une cellule à une autre ou à la matrice extra-cellulaire ; les jonctions communicantes permettent quant à elles le passage de signaux (chimiques, électriques) entre cellules adjacentes.

L'adhésion cellulaire n'est pas le seul moyen d'interaction à la disposition des cellules. Celles-ci peuvent interagir à distance, via des gradients de molécules. Les contraintes physiques issues de l'environnement sont également des signaux pouvant être interprétés et traduits en signaux chimiques et électriques par les cellules. Les cellules disposent de quatre moyens principaux de communication :

- par contact direct : il s'agit des jonctions communicantes que nous avons évoquées ci-dessus ;
- par l'émission de signaux d'une cellule qui seront reçus par les cellules proches de celle-ci (communication paracrine) ;
- par l'émission de signaux propagés via le système circulatoire (communication endocrine) ;

– par l'émission de neurotransmetteurs (communication synaptique).

Il est à noter que les cellules peuvent également percevoir des signaux qu'elles ont elles-mêmes émis : il s'agit de communication autocrine.

Pour communiquer par signaux chimiques, les cellules sont pourvues de récepteurs pouvant être situés sur leurs membranes ou à l'intérieur des cellules. Ces récepteurs peuvent être de nature variée : canaux ioniques, récepteurs enzymatiques, synapses, etc.

Tous ces mécanismes sont très robustes et garantissent l'intégrité des organismes vivants. Cependant ils ne sont pas infaillibles et leurs dysfonctionnements peuvent entraîner de graves maladies, telles que les cancers, qui sont des exemples de développement de tissus pathologiques, auxquels nous avons porté un intérêt particulier dans le cadre de cette thèse.

2 Exemple de développement de tissus pathologiques : cancérogenèse

2.1 Généralités

En 2011, on a estimé qu'en France le nombre de nouveaux cas de cancers était de 365 500 [3]. Si la mortalité liée aux cancers diminue d'années en années, on estime tout de même pour 2011 que 147 500 décès sont liés au cancer.

Un cancer peut avoir des causes variées : génétiques, environnementales ou encore virales et résulte d'anomalies dans les mécanismes régulant la prolifération cellulaire. En temps normal, la prolifération des tissus dans l'organisme est régulée ; c'est ce que l'on appelle *l'homéostasie cellulaire* : le nombre de morts cellulaires par apoptose et le nombre de divisions cellulaires d'un organisme sont équilibrés. La prolifération cellulaire est régulée par deux familles de gènes : les proto-oncogènes, favorisant la multiplication cellulaire, et les gènes suppresseurs de tumeur, permettant de ralentir voire de stopper la multiplication cellulaire.

Les cellules, selon leur type, ont des rythmes de division variés : alors que les cellules de l'intestin sont renouvelées en quelques jours, certaines cellules telles que les cellules du foie sont renouvelées après plusieurs centaines de jours. La prolifération cellulaire seule n'est donc qu'un élément de la survenue de cancer. Cependant, l'altération d'un proto-oncogène peut permettre à ce gène de s'exprimer de manière anormale et de favoriser une multiplication cellulaire dérégulée en devenant plus actif. Le proto-oncogène devient alors un oncogène.

Les points de contrôle lors du cycle cellulaire permettent de vérifier l'intégrité du matériel génétique de la cellule se divisant et de réparer celle-ci ou de déclencher son apoptose si ce matériel génétique est altéré. L'altération d'un gène suppresseur de tumeur va empêcher les contrôles intervenant dans le cycle cellulaire et donner libre cours à la multiplication non contrôlée de cellules endommagées.

En plus de ces deux mécanismes principaux, les cellules cancéreuses disposent également d'autres capacités, comme de pouvoir échapper à des signaux qui, dans des cas normaux, déclencheraient par exemple leur apoptose. Finalement, les cellules cancéreuses ont deux propriétés majeures : elles ont des capacités de division très largement supérieures aux cellules saines et elles peuvent, via différents mécanismes, échapper à la mort cellulaire.

De manière générale, les mécanismes à l'œuvre lors du développement de cancers sont les mêmes que ceux intervenant dans le développement de tissus. La différence se situe

3. Données : ONG « Ligue contre le cancer », `http://www.ligue-cancer.net/article/6397_les-chiffres-cles-des-cancers`, consulté le 1er septembre 2014.

dans la manière dont ces mécanismes sont exprimés : ils peuvent par exemple être exacerbés (capacité de division) ou encore dysfonctionnels (capacité d'échapper à des signaux apoptotiques).

2.2 Migration cellulaire et facteurs biomécaniques lors du développement de tumeurs

La capacité des cellules tumorales à migrer peut faciliter l'expansion d'une tumeur. La migration peut être dirigée par *chimiotaxie* (migration directionnelle selon un gradient de molécules en solution dans l'environnement), par *haptotaxie* (migration directionnelle selon un gradient de molécules liées au substrat sur lequel évoluent les cellules) mais aussi sous l'influence de *contraintes physiques* (Kass et coll., 2007).

Ces contraintes physiques sont en partie transmises par la matrice extra-cellulaire via des sites d'adhésion. L'adhésion des cellules à la matrice extra-cellulaire cause, dans une certaine mesure, une réorganisation du cytosquelette de la cellule. La raideur de la matrice extra-cellulaire va donc influencer la manière dont est organisé le cytosquelette des cellules lors de leur adhésion (Tracqui, 2009). Les tensions générées par la matrice extra-cellulaire peuvent également participer à la modification du phénotype des cellules et en particulier contribuer à l'expression de gènes mécanosensibles (gènes dont l'expression est dépendante de la présence de stimuli mécaniques).

2.3 Vascularisation de tumeurs

Lorsque la tumeur atteint une certaine taille (de l'ordre de quelques millimètres), elle doit trouver des sources de nutriments et d'oxygène supplémentaires. La tumeur va pouvoir déclencher la formation de nouveaux vaisseaux sanguins à partir du réseau existant : c'est le phénomène d'*angiogenèse*.

Pour déclencher l'angiogenèse de vaisseaux sanguins, la tumeur émet un facteur de croissance nommé VEGF (*Vascular Endothelial Growth Factor*). Ce facteur de croissance va stimuler la prolifération des cellules endothéliales, composant, parmi d'autres types cellulaires, les parois des vaisseaux sanguins. Les cellules endothéliales vont dégrader la matrice extra-cellulaire grâce à la production d'une protéase : ce phénomène permet de faciliter leur migration par chimiotaxie en direction de la tumeur. La prolifération des cellules endothéliales, induite par le VEGF, va donner lieu à un phénomène de bourgeonnement du vaisseau sanguin. Par la suite, ce bourgeon donnera naissance à un vaisseau sanguin via la poursuite de la prolifération des cellules endothéliales.

Il est à noter que le mécanisme d'angiogenèse déclenché par une tumeur donne lieu à un réseau constitué de nombreux vaisseaux (donc la tumeur est très vascularisée) même si ces vaisseaux sont « mal construits », par opposition à ceux issus de l'angiogenèse se produisant dans des cas sains, comme la réparation d'une blessure.

Le développement de ce nouveau réseau sanguin permet également de favoriser le développement de métastases, puisque les cellules cancéreuses vont pouvoir se déplacer à travers le réseau sanguin pour s'implanter dans d'autres tissus.

Dans le cas de l'angiogenèse, l'influence des contraintes mécaniques est encore assez peu connu. Certains travaux permettent d'émettre l'hypothèse que les propriétés mécaniques de la matrice extra-cellulaire et du stroma (l'arrangement des cellules et du tissu conjonctif où se situent blessures ou encore tumeurs), en particulier leurs raideurs, peuvent influencer le développement des cellules épithéliales ; il s'agit de développements encore récents (Edgar et coll., 2014).

2.4 Thérapies appliquées au cancer

Différentes approches existent pour traiter les cancers. La *chimiothérapie* est un traitement médicamenteux visant à détruire les cellules qui se multiplient trop rapidement : c'est le cas des cellules cancéreuses, mais également de certaines cellules de notre organisme, telles que les cellules du tube digestif ou encore les cellules de cheveux. Ce traitement est souvent utilisé en parallèle d'un autre traitement (chirurgie, radiothérapie). La *chirurgie* consiste en l'ablation des cellules cancéreuses. Le plus souvent, la chirurgie est précédée d'autres formes de traitements afin de faire diminuer la taille de la tumeur à opérer. La *radiothérapie* est l'utilisation de rayons ionisants pour détruire les cellules cancéreuses. Dans le cas de certains cancers, les leucémies par exemple, il est également possible de procéder à des *greffes de cellules souches* provenant d'un donneur ou du patient lui même, avec un risque plus important de récidive s'il reste des cellules cancéreuses dans le greffon, malgré les traitements subis par le patient au préalable. L'*immunothérapie* est un traitement visant à stimuler le système immunitaire du patient afin qu'il détecte les tumeurs : malgré l'altération des cellules cancéreuses, celles-ci sont peu détectées par le système immunitaire ; en outre, les cellules cancéreuses auraient des moyens de « tromper » le système immunitaire (Hanahan et Weinberg, 2011). Les *traitements ciblés* sont des traitements médicamenteux visant à entraver, si ce n'est empêcher, le déroulement de certains mécanismes liés aux cancers. Par exemple, il existe des traitements ciblés contre l'angiogenèse. La *virothérapie* consiste à utiliser des virus génétiquement modifiés pour traiter des pathologies. La thérapie génique a été le premier exemple de virothérapie et ce type de traitement commence à être utilisé dans le cadre de cancers. Dans le cadre des travaux de Breitbach et coll. (2011) par exemple, l'utilisation d'un virus permet à la fois de déclencher la mort des cellules par lyse[4] ainsi que de stimuler le système immunitaire. Il est plus que probable que de nouvelles formes de thérapies voient le jour afin de traiter les cancers. On peut par exemple citer les travaux de Lee et coll. (2012), qui étudient les effets de jeûnes de courtes durées sur l'évolution des tumeurs et leurs sensibilisation accrue à la chimiothérapie.

Ces deux premières sections nous ont permis de situer le contexte biologique dans lequel se situent ces travaux. La présentation de la cellule biologique et des conséquences que des dérégulations de son fonctionnement nous conduisent maintenant à présenter plus avant les notions de modélisation et de simulation numérique, ceci dans l'optique d'utiliser de tels outils afin d'étudier, sous un angle différent que celui offert par les expérimentation *in vitro* et *in vivo*, le développement de tissus sains ou pathologiques.

3 Modélisation et simulation numérique

Les notions de modèle biologique et de simulation numérique sont fortement liées. Pour commencer, nous proposons de définir le terme « modèle » comme suit :

> Un modèle est une représentation simplifiée et néanmoins réaliste d'un objet ou d'un phénomène.

Cette définition répond à deux questions : *qu'est-ce qu'un modèle ?* et *que représente un modèle ?* Un modèle peut représenter des objets, comme des cellules biologiques ou encore des organes, mais aussi des phénomènes comme des interactions entre des cellules biologiques. La notion de simplification que l'on retrouve dans le modèle est importante puisqu'elle implique que tous les éléments d'un système ne sont pas présents dans un modèle : le modèle

4. La membrane de la cellule est détruite sous l'action d'un agent extérieur, conduisant à la mort de la cellule.

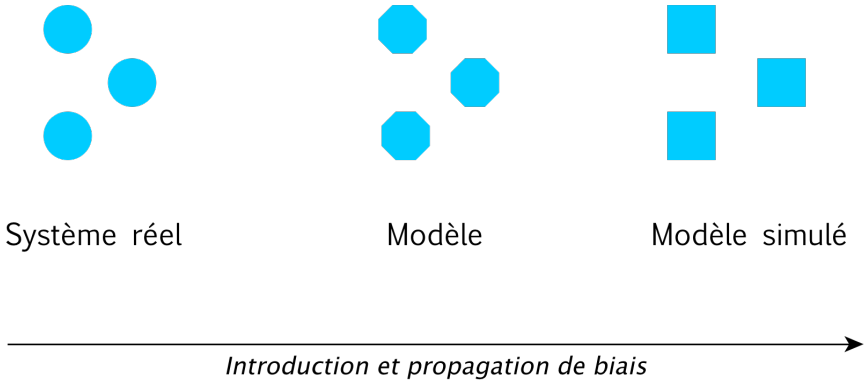


FIGURE 1.4 – Processus de modélisation.

ne peut être aussi complexe que le système réel, c'est un fait inhérent à la notion de modèle (sinon les recherches sont réalisées directement sur le système réel). Cela implique que le modèle comporte des *biais* et des *approximations*. La construction d'un modèle demande donc à priori de bonnes connaissances du système que l'on souhaite représenter. Mais un modèle peut aussi être utilisé dans l'optique de mieux comprendre un système complexe. Ici un processus itératif est à l'œuvre : le modèle est construit et consolidé tout en étudiant le système réel, lui-même pouvant être mieux compris grâce au travail réalisé sur le modèle.

Le modèle conçu est ensuite *simulé*. En plus du biais introduit lors de la modélisation, il faut savoir que la simulation du modèle, en particulier la simulation *numérique*, introduit un biais supplémentaire lié aux limites des ordinateurs (comme la précision des calculs par exemple). Il faut garder à l'esprit qu'une simulation n'est pas forcément numérique : il est par exemple possible de résoudre sans machine un système d'équations modélisant un système. Aujourd'hui, les ordinateurs nous permettent de prendre en compte des modèles toujours plus complexes ou de considérer un nombre d'entités plus important (dans le cadre de simulation de tissus cellulaire comportant des milliers de cellules par exemple). Dans le cadre de cette thèse, nous nous plaçons dans le contexte de la *simulation numérique*. Finalement, le processus de modélisation et de simulation peut être vu comme le processus illustré sur la figure 1.4 : lorsque l'on cherche à modéliser un système, quel qu'il soit, le modèle correspondant comporte des simplifications ou des biais. Ces derniers sont propagés à l'étape de simulation numérique.

Quels sont les objectifs de la modélisation et, à fortiori, de la simulation numérique ? En premier lieu, on peut y trouver un avantage en matière de coût : en effet, l'utilisation d'un modèle peut permettre de multiplier les expérimentations pour un coût réduit, ce qui n'est généralement pas le cas lors d'expérimentations réelles. Par exemple, dans le domaine de l'aéronautique, la conception d'ailes d'avions peut être grandement facilitée par la simulation en utilisant les principes de la mécanique des fluides afin d'étudier différentes conceptions possibles. La modélisation et la simulation ont aussi leur utilité dans le cas d'applications pouvant présenter des dangers : l'armement, le nucléaire ou encore la synthèse de nouveaux médicaments. Dans d'autres cas, on souhaite simplifier des processus afin de les étudier pour mieux les comprendre. La biologie dans son ensemble bénéficie également de l'avènement de la simulation numérique. On trouve par exemple des applications permettant d'étudier la structure spatiale des protéines dont les fonctions sont liées à cette structure spatiale. Ce type d'application date des années 1970 (Levitt et Warshel, 1975). La modélisation et la simulation sont très diversifiées dans le domaine de la biologie : simulation de systèmes

multi-cellulaires (au cœur des travaux de thèse présentés dans le présent document), de réseaux de régulation génétique, évolution de pandémies, etc : la biologie est un domaine vaste et les possibilités en matière de simulation le sont aussi. En outre, la modélisation et la simulation offrent une expressivité que l'on ne peut retrouver (ou difficilement) lors d'expérimentations réelles : la modélisation permet de prendre des libertés sur le système réel et la simulation offre un aspect exploratoire peu commun, permettant de tester toutes sortes d'hypothèses, aussi hardies soient-elles. Évidemment, derrière cela se pose la question de la pertinence : les modèles et leurs simulations numériques sont donc à manipuler avec précaution si l'on souhaite rester en adéquation avec le système réel.

Pour résumer, quatre éléments principaux se dégagent quant à l'utilisation de la simulation : les coûts sont réduits par rapport à des expérimentation réelles ; la simulation peut limiter le danger lié à certains domaines d'étude ; les modèles et leurs simulations numériques peuvent contribuer à l'acquisition d'une meilleure compréhension des systèmes étudiés ; le caractère exploratoire et la liberté inhérentes aux modèles et à leurs simulations permettent au modélisateur de laisser libre court à son expressivité. Selon nous, l'élément se trouvant au cœur des avantages liés à la modélisation et à la simulation numérique est la possibilité de mieux comprendre les systèmes étudiés, les autres avantages pouvant découler de celui-ci.

Nous nous plaçons dans le cadre de l'étude de systèmes biologiques par leurs modélisations et leurs simulations numériques. Différentes applications peuvent émerger dans ce contexte, situées dans deux aspects distincts mais pouvant se chevaucher : les aspects *applicatifs* et les aspects *théoriques*. Les aspects applicatifs peuvent par exemple concerner l'ingénierie des tissus, discipline émergeante en biologique depuis quelques années, où le numérique peut aider à la conception de tissus synthétiques (que l'on imprime avec des imprimantes 3D aujourd'hui (Bertassoni et coll., 2014)). Les aspects applicatifs sont fortement liés aux aspects théoriques car ces derniers permettent de mieux appréhender le système et son fonctionnement. Ces aspects sont illustrés sur la figure 1.5. Dans le domaine de la biologie, la modélisation et la simulation vont de pair avec la notion d'interdisciplinarité. En effet, la conception de modèles est souvent réalisée par des biologistes théoriciens, des mathématiciens, des physiciens ou des informaticiens. Les aspects applicatifs et théoriques se retrouvent possiblement eux-mêmes dans d'autres disciplines : biologie, médecine régénérative, etc., ne faisant qu'accroître la complexité et les besoins en interdisciplinarité.

4 Synthèse du chapitre et positionnement

L'objet de ce chapitre était de situer le contexte de la thèse à la fois dans les aspects « biologie » ainsi que dans les aspects « modélisation et simulation ». Plus précisément, nous avons mis en avant des notions élémentaires que l'on retrouve au long de ce mémoire.

Dans ce chapitre, nous avons présenté la structure générale des cellules (en particulier des cellules eucaryotes). Selon la théorie cellulaire, la cellule peut être considérée comme la brique de base du vivant. C'est alors naturellement que la cellule a été le point de départ de notre modélisation (nous appuyons cette approche dans le chapitre 3, section 1.1). La cellule biologique est structurée et organisée en différents compartiments assurant des fonctions variées : nous avons décidé de considérer ces comportements à haut niveau dans notre modèle. Il s'agit là du problème d'échelles de modélisation : en effet, les systèmes biologiques sont constitués de niveaux différents qu'on ne peut tous prendre en compte dans un modèle : par exemple un modèle de cellule détaillé, dont tous les compartiments et leurs fonctions respectives sont modélisés, sera peu adapté à la simulation de tissus car

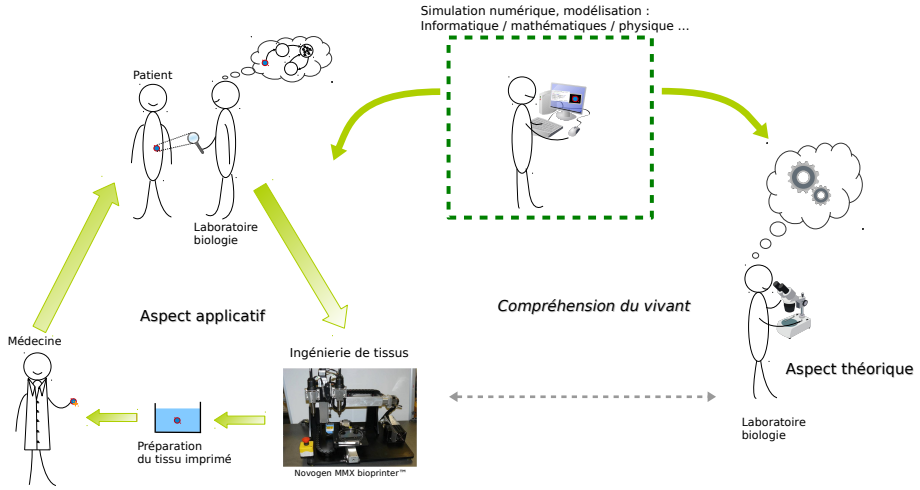


FIGURE 1.5 – Exemples d'objectifs de la modélisation et de la simulation de systèmes biologiques. La modélisation et la simulation sont liées à la notion d'interdisciplinarité : biologie théorique, mathématiques, physique, informatique. Les applications se retrouvent également dans différentes disciplines, par exemple la biologique ou la médecine régénérative.

trop complexe et impossible à simuler lorsque des milliers de ces cellules sont mises en interaction. Nous revenons sur ces éléments dans le chapitre 2, section 1.

Nous avons donc également expliqué, dans ce chapitre, les avantages et les limites concernant la modélisation et la simulation.

Notre modèle est bâti autour de trois concepts biologiques principaux :

- la structure physique de la cellule, le cytosquelette, qui est adapté à une modélisation biomécanique. À notre connaissance, il existe peu de modèle biomécaniques de cellules virtuelles pouvant prendre en compte les effets de la mécanotransduction (voir l'état de l'art que nous avons réalisé dans le chapitre 2).
- la vie des cellules est régulée via le cycle cellulaire. Si en pratique c'est un élément complexe d'une cellule biologique, il peut être simplifié et intégré à un modèle ;
- les interactions des cellules sont indispensables au développement de tissus, quelle que soit leur nature. Ces interactions seront, dans notre modèle, abstraites (problème des échelles de modélisation abordé au chapitre 2, section 1).

Un intérêt de notre modèle réside dans la prise en compte des contraintes physiques issues de l'environnement. Actuellement, la plupart des modèles prenant en compte ce type de contrainte se focalise sur l'étude de la morphogenèse de plante ou sur l'arrangement des cellules dans les tissus épithéliaux [5].

Différents éléments interviennent dans le développement de tissus pathologique tels que les cancer : altération du matériel génétique, dérégulation de cycle cellulaire donnant lieu à des proliférations cellulaires non contrôlées, capacité des cellules à échapper aux signaux de mort cellulaire. La migration des cellules est également un aspect important du développement de tumeurs. Ces migrations se font sous l'influence de facteurs chimiques et biomécaniques. Enfin, la vascularisation est une étape importante pour qu'une tumeur poursuive son développement. Ces différents éléments entrent en jeu dans le cas de la morphogenèse de tissus, qu'ils soient sains ou non. Nous souhaitons donc les intégrer à notre modèle.

5. Essentiellement des tissus de revêtement

Dans le cadre de morphogenèse de tissus, la simulation peut permettre d'étudier des mécanismes menant au développement de tissus pathologiques ou encore de réaliser des études préliminaires de thérapies. Nous retrouvons les avantages cités ci-dessus : coût, baisse de la dangerosité, expressivité. Nous gardons évidemment à l'esprit les limites de la modélisation et de la simulation numérique : biais et approximations, modèles incomplets ou non valides, c'est-à-dire qui ne présentent pas — ou pas suffisamment — de corrélation avec les observations ou les mesures réalisées lors d'expérimentations réelles. Il est à noter que rendre un modèle computationel valide passera le plus souvent par des échanges interdisciplinaires entre informaticiens et biologistes.

Le prochain chapitre a pour objet de présenter les différents formalismes que l'on peut utiliser lorsque l'on souhaite concevoir un modèle pour simuler la morphogenèse de tissus sains ou pathologiques. En particulier, nous montrons que les deux grandes approches pour la modélisation, à savoir l'approche continue et l'approche discrète, comportent chacune leurs avantages et inconvénients et que la tendance devient donc de se tourner vers des approches hybrides mêlant les formalismes discret et continu. Une synthèse détaillée nous permet de mettre en avant ces caractéristiques afin de nous positionner et de choisir un formalisme.

CHAPITRE 2

Approches pour la modélisation de systèmes biologiques

L'objet de ce chapitre est de présenter les différents formalismes de modélisation que l'on peut utiliser lorsque l'on souhaite modéliser et simuler la morphogenèse de tissus. Une première section permet de donner quelques éléments concernant les échelles de modélisation dans les systèmes biologiques. Ensuite, les sections 2 et 3 décrivent respectivement les approches continues et les approches discrètes au travers d'exemples de la littérature. La section 4 montre comment ces deux approches peuvent finalement être couplées dans des modèles hybrides. Enfin, la synthèse du chapitre (section 5) permet de faire un comparatif entre les différentes approches présentées pour la modélisation de systèmes biologiques et d'apporter les conclusions du chapitre.

Sommaire

1 Echelles de modélisation

Une difficulté majeure de la modélisation de systèmes biologiques est la complexité du phénomène que l'on souhaite étudier : en particulier celui-ci nécessite le plus souvent l'étude de différentes échelles spatiales ou temporelles. Par exemple, la biologie humaine est constituée de vastes échelles : du gène à la protéine en passant par les cellules, puis les tissus qui forment les organes et enfin l'organisme. Cela représente des échelles spatiales de l'ordre de 10^{-9}m (gène) à 10^{0}m (organisme). Temporellement, les échelles sont de l'ordre de 10^{-12}s (interactions moléculaires) à 10^{9}s (vie d'un être humain).

Le premier point à considérer lors de la mise au point d'un modèle est le niveau d'échelle spatiale et temporelle auquel il faut se situer en fonction du système que l'on souhaite étudier. Selon le système, il faut tout d'abord décider par quel angle on souhaite adresser le problème : en effet, deux modes principaux de modélisation existent : 1) on peut observer un phénomène et élaborer une ou des théories permettant de l'expliquer : c'est l'approche descendante, ou « top-down » ; 2) on peut étudier les composants du système individuellement, les modéliser et les mettre en interaction afin d'observer le produit de ces interactions : c'est l'approche ascendante, ou « bottom-up ». En ce qui concerne la modélisation de systèmes biologiques, et en particulier la modélisation de la morphogenèse de tissus, la cellule est souvent considérée comme un point de départ adéquat à la construction d'un modèle (Walker et Southgate, 2008; Merks et Glazier, 2005) : en premier lieu, la cellule biologique a été l'objet de beaucoup

d'études et certains aspects de son fonctionnement sont bien connus, même si d'autres restent plus difficiles à appréhender : rôle de certains éléments du code génétique ou encore complexité des interactions de cellules de types variés, parfois sur diverses échelles. En outre, la cellule biologique apparaît comme un point charnière du système biologique : située entre les échelles microscopique et macroscopique, la cellule biologique peut moduler des processus sous-cellulaires comme l'expression des gènes au niveau microscopique, ainsi qu'influencer les niveaux supérieurs : les dérégulations au niveau cellulaire peuvent causer de graves perturbations au niveau des tissus (niveau macroscopique), comme dans le cas des cancers.

Les sections suivantes décrivent les principaux formalismes de modélisation utilisés pour modéliser et simuler numériquement des systèmes biologiques. Les cas d'études présentés permettent de dégager les avantages et inconvénients des approches de modélisation que l'on peut utiliser et permettent de cerner les enjeux liés à la modélisation et à la simulation au travers d'exemples concrets issus de travaux de recherche.

2 Modélisation continue

Un modèle continu est une description mathématique du phénomène que l'on souhaite étudier. Une telle description peut être réalisée à l'aide d'équations différentielles. En fonction de ce que l'on souhaite représenter, il est possible d'utiliser des équations différentielles ordinaires (EDO) ou des équations aux dérivées partielles (EDP).

L'utilisation en simulation de ces deux types d'approches nécessite de pouvoir résoudre les équations présentes dans les modèles. Il est souvent difficile de trouver des solutions analytiques à des systèmes qui peuvent devenir très complexes. Au lieu de cela, on utilise des méthodes de discrétisation permettant d'approcher les solutions par pas de temps. Différentes méthodes de résolution numérique existent, telle que la méthode d'Euler ou les méthodes de Runge-Kutta. Il faut garder à l'esprit que la méthode utilisée peut conduire à des instabilités ou demander des temps de calculs importants.

2.1 Équations differentielles ordinaires

Les équations différentielles ordinaires permettent de modéliser l'évolution d'un système au cours du temps. On peut illustrer l'utilisation de ce type d'équation dans le cas d'étude de la dynamique de populations, par exemple avec un système proies / prédateurs. Les règles de ce système sont les suivantes :

- le taux de reproduction des proies est constant en l'absence de prédateur (on ne considère pas de compétition entre les proies pour l'accès à des ressources) ;
- l'absence de proie entraîne la baisse du nombre de prédateurs ;
- le taux de prédation est proportionnel au nombre de prédateurs ;
- la variation du nombre de prédateurs dépend du nombre de proies.

La mise en équation de ces règles donne un couple d'équations différentielles appelées équations de Lotka-Volterra (Volterra, 1926) :

$$\begin{cases} \dfrac{dx(t)}{dt} = \alpha x(t) - \beta x(t) y(t) \\ \dfrac{dy(t)}{dt} = -\gamma y(t) + \delta x(t) y(t) \end{cases} \tag{2.1}$$

Où $x(t)$ et $y(t)$ représentent respectivement les populations de proies et de prédateurs en fonction du temps t ; $\frac{dx(t)}{dt}$ et $\frac{dy(t)}{dt}$ représentent l'évolution des populations de proies et

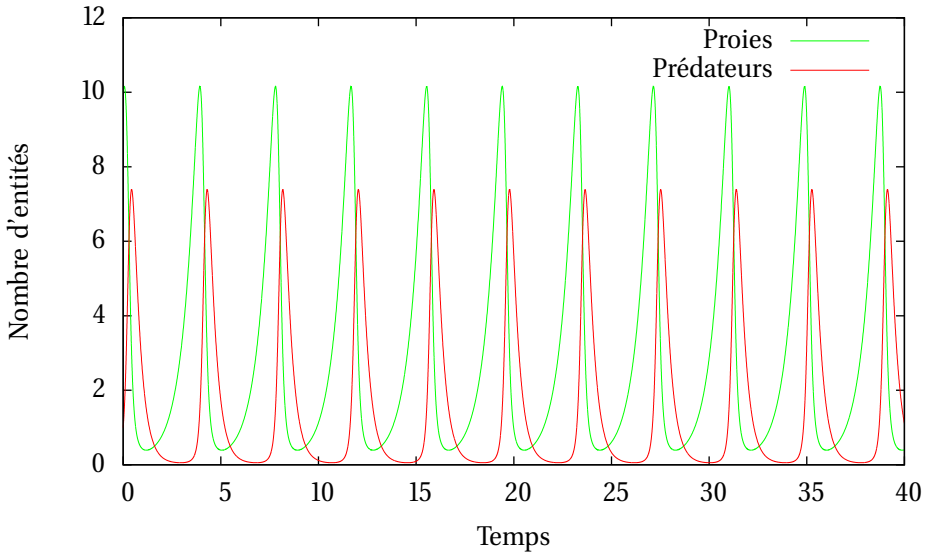


Figure 2.1 – Résolution numérique du système d'équations 2.1 modélisant l'évolutions de populations de proies et de prédateurs. Les valeurs des paramètres sont ici les suivantes : $\alpha = 1.5$, $\beta = 1.0$, $\gamma = 3.0$, $\delta = 1.0$

de prédateurs au cours du temps. Les paramètres du système expriment les règles listées ci-dessus : α est le taux de reproduction des proies en l'absence de prédateur et γ est le taux de mortalité des prédateurs en l'absence de proie. Les paramètres β et δ représentent les interactions entre les deux espèces : β est le taux de prédation et δ représente le taux de reproduction des prédateurs selon le nombre de proies à leur disposition. La figure 2.1 montre le résultat de la résolution numérique du système avec la bibliothèque GSL[1] en utilisant la méthode de Runge-Kutta d'ordre 2. On observe sur la figure que les deux populations en présence évoluent de manière oscillatoire. Cette dynamique peut être intuitivement anticipée : tant que le nombre de proies est suffisant, les prédateurs peuvent proliférer. La prédation fait cependant diminuer la population de proies : quand celle-ci est trop basse, la population de prédateurs décline. La prédation étant moins importante, la population de proies augmente. Ces étapes se produisent de manière cyclique.

Les EDO sont largement utilisées pour modéliser des réseaux de régulation génétiques (Polynikis et coll., 2009). Un exemple très complet de modèle mettant en œuvre ce type de modèle est E-Cell (Tomita et coll., 1999), qui est avant tout une plateforme pour la modélisation de processus cellulaire. En particulier, cet environnement inclut un modèle de cellule « complet » contenant 127 gènes permettant de disposer d'une cellule virtuelle avec un métabolisme cellulaire minimal. La cellule peut synthétiser des protéines (et contient donc des mécanismes de transcription d'ADN en ARN et de traduction de cet ARN) ou encore générer de l'ATP. Les réactions métaboliques de ce modèle sont modélisées par des équations différentielles.

De manière plus spécifique, la modélisation avec des EDO peut, par exemple, permettre d'étudier les échanges réalisés par les canaux ioniques présents sur la membrane des cellules : Fridlyand et coll. (2013) ont proposé d'étudier le rôle des ions Ca^{2+} dans la sécrétion d'insuline par les cellules β du pancréas.

1. Gnu Scientific Library : http ://www.gnu.org/software/gsl/

Un autre exemple d'application de cette approche est l'étude de cancers : croissance de tumeur, chimiothérapie ou encore immunothérapie. Dans ce domaine d'application en particulier, il existe un nombre important de modèles basés sur des EDO. Dans certains cas, les modèles vont même jusqu'à inclure du contrôle optimal afin, par exemple, de déterminer des dosages optimaux lors des traitements (Engelhart et coll., 2011; Hadjiandreou et Mitsis, 2014).

Nous présentons ici deux exemples de modèle permettant de manière assez générale de prendre en compte la croissance des tumeurs et l'association de thérapies de natures diverses. Ils permettent dans un premier temps de voir comment les systèmes d'équations sont construits selon les comportements attendus du modèle et dans un second temps, nous verrons comment leur expressivité permet aux concepteurs de ces modèles d'émettre des hypothèses par rapport au système qu'ils modélisent et étudient.

Modélisation de l'évolution de tumeurs sous traitements conjoints par chimiothérapie et immunothérapie. de Pillis et coll. (2006) proposent un modèle basé sur des EDO pour la modélisation de l'évolution de tumeurs. En particulier, le but de ces travaux est d'étudier l'influence de l'immunothérapie conjointement utilisée à la chimiothérapie sur l'évolution de tumeurs.

Les auteurs ont raffiné un modèle qu'ils avaient déjà proposé en ajoutant à ce modèle des termes permettant de prendre en compte la chimiothérapie et l'immunothérapie. Le modèle est un ensemble d'EDO couplées permettant la description de l'évolution de quatre populations de cellules et de deux concentrations de médicaments dans le flux sanguin. Les types cellulaires sont des cellules cancéreuses et trois types de lymphocytes. Les concentrations de médicaments sont les concentrations de chimiothérapie d'une part et les concentrations d'immunothérapie d'autre part.

La construction de différents termes permet de modéliser des comportements propres à chaque population. Par exemple le terme exprimant la croissance de la population de cellules cancéreuses est $G_T = aT(1 - bT)$. Tous les termes permettent ensuite de proposer le système d'EDO utilisé pour modéliser le système dans son ensemble. En tout, ce système d'EDO comporte huit équations différentielles. À titre d'exemple, voici l'équation différentielle permettant de modéliser l'évolution de la population de cellules tumorales au cours du temps :

$$\frac{dT}{dt} = \underbrace{aT(1-bT)}_{(1)} - \underbrace{cNT}_{(2)} - \underbrace{DT}_{(3)} - \underbrace{K_T(1-e^{-M})T}_{(4)} \tag{2.2}$$

Les différents termes de l'équation permettent de modéliser 1) la croissance de la population de cellules tumorales, comme on l'a déjà évoqué ; 2) les interactions entre cellules tumorales et cellules lymphocytes NK ; 3) les interactions entre cellules tumorales et cellules CD8+T et 4) l'interaction des cellules tumorales et de la chimiothérapie.

Les paramètres du système sont tirés de données expérimentales. Différents ensembles de simulations numériques ont conduit les auteurs à émettre l'hypothèse que la meilleure stratégie de soin est l'utilisation conjointe de chimiothérapie et d'immunothérapie plutôt que d'une de ces deux thérapies seule. En outre, ce type de modèle devrait permettre de prévoir des protocoles de traitements spécifiques aux patients en prenant en compte, dans le modèle, des caractéristiques de ces patients.

Modélisation mathématique de radiovirothérapie de cancers. La virothérapie consiste à utiliser des virus reprogrammés (manipulé génétiquement) afin, dans le cas de cancers,

que ces derniers détruisent les cellules tumorales. Dingli et coll. (2006) proposent un modèle mathématique pour l'étude de thérapie combinée : radiothérapie et virothérapie. Sans entrer dans les détails, leur modèle est basé sur des EDO utilisées pour modéliser la dynamique de population de cellules. En particulier, il contient des termes pour modéliser les effets des thérapies (radiothérapie et virothérapie) sur la croissance de la population de cellules tumorales. Les simulations réalisées tendent à montrer que l'efficacité de la radiovirothérapie conduit à une guérison complète quand la virothérapie seule ne permet que de réduire la taille de la tumeur. Sans modifier les paramètres du modèle, les auteurs ont joué sur les dates de début de radiothérapie ainsi que ses doses, et sur la dose virale initiale lors de différentes simulations. Les auteurs ont pu émettre les hypothèses suivantes : la radiothérapie est plus efficace si elle est débutée après l'administration des virus ; une dose virale initiale trop élevée n'apporte pas d'amélioration substantielle et peut s'avérer dangereuse et contre productive ; cette dernière observation est réalisée pour les doses d'iode utilisées dans la radiothérapie.

Ce type de modèle permet, comme nous l'avons vu, de réaliser des simulations numériques réalistes des systèmes qu'ils modélisent. En particulier, nous avons vu dans le premier exemple comment des comportements observés expérimentalement sont l'objet de propositions pour être inclus dans le modèle (comme l'action d'un traitement sur les cellules de la tumeur). Dans la section suivante, l'approche mathématique décrite permet, en plus de simuler l'évolution de population dans le temps, de prendre en compte — dans une certaine mesure — la notion de spatialité dans le système ; toutefois ce sont toujours des populations dans leur globalité qui sont prises en compte, non pas des individus.

2.2 Équations aux dérivées partielles

Les équations aux dérivées partielles (EDP) sont utilisées pour décrire l'évolution d'un système à la fois dans le temps et dans l'espace. Les équations de réaction-diffusion représentent un exemple bien connu d'EDP : ce type de système permet de décrire comment sont modifiées des concentrations de substances — comme des molécules — par des processus chimiques locaux et la diffusion de ses substances dans l'espace. La forme générale d'une équation de réaction-diffusion pour une espèce chimique i est la suivante :

$$\frac{\partial i(x,t)}{\partial t} = D_i \Delta i(x,t) - R_{i(x,t)} \tag{2.3}$$

Où D_i est le coefficient de diffusion de l'espèce chimique i ; Δ est l'opérateur laplacien et $R_{i(x,t)}$ sont les réactions locales (réaction, production, etc.) de l'espèce chimique i.

Proposé par Turing (1952), ce type de modèle permet entre autres de simuler la formation de motifs, sur la base de deux substances interagissant (deux morphogènes : un activateur et un inhibiteur). Des adaptations du modèle proposé par Turing ont notamment permis de modéliser la formation de motifs par exemple chez le léopard (figure 2.2 (Liu et coll., 2006a)) ou encore chez le poisson zèbre (Kondo et Miura, 2010).

L'utilisation d'EDP pour la modélisation de systèmes ne s'arrête pas à la seule étude de la formation de motifs. Impliquant la diffusion et la réaction de morphogènes, l'étude de la formation de membres chez les vertébrés est un exemple d'application des équations aux dérivées partielles pour la modélisation. Une revue des principaux modèles utilisés pour modéliser la croissance de membres avec des EDP a été proposée par Zhang et coll. (2013).

Des modèles mathématiques basés sur l'utilisation d'EDP ont également été proposés pour l'étude des cancers. Les deux paragraphes suivants décrivent deux de ces modèles

(a) **(b)**

FIGURE 2.2 – Utilisation du modèle de Turing pour simuler les motifs observables sur la robe des léopards : (a) motifs observés sur un léopard adulte ; (b) motifs obtenus en simulation. Figures extraites de (Liu et coll., 2006a).

qui permettent, grâce à l'utilisation du formalisme des équations aux dérivées partielles, de prendre en compte des aspects locaux dans l'évolution des différentes populations modélisées.

Modèle pour la croissance de tumeur avasculaire. Les premiers stades du développement d'une tumeur ne sont que rarement observés *in vivo* à cause de la taille trop peu importante de la tumeur. On peut utiliser des sphéroïdes, *in vitro*, pour observer la croissance d'une tumeur : une cellule tumorale, placée dans un médium contenant des nutriments, va proliférer jusqu'à ce que la sphéroïde atteigne quelques millimètres. Cette tumeur va être constituée d'un centre de cellules nécrotiques, entouré d'un ensemble de cellules quiescentes. L'extérieur de la tumeur est constitué de cellules prolifératives. À ce stade, la tumeur n'est pas vascularisée : autrement dit, le processus déclenchant l'angiogenèse n'est pas encore mis en place par la tumeur. La figure 2.3 montre la structure d'une sphéroïde : le cœur nécrotique de la tumeur est entouré d'une couche de cellules quiescentes, elle même entourée par une couche de cellules prolifératives.

Sherratt et Chaplain (2001) proposent un modèle de tumeur avasculaire qui prend en compte la motilité des cellules quiescentes et prolifératives (dans une sphéroïde, les cellules nécrotiques ne sont pas mobiles), et en particulier la motilité des cellules dues aux interactions entre cellules : la présence d'un type cellulaire peut contraindre la motilité d'un autre type. Ce comportement peut avoir une influence significative sur la croissance de la tumeur. Le modèle prend également en compte la présence de nutriments et facteurs de croissance ; ces deux éléments étant modélisés par la même variable.

Le modèle est un système de quatre équations aux dérivées partielles, décrivant chacune l'évolution temporelle et spatiale des trois types cellulaires et des nutriments / facteurs de croissance. Les densités des types cellulaires respectifs sont dénotées par : $p(x,t)$ les cellules prolifératives ; $q(x,t)$ les cellules quiescentes ; $n(x,t)$ les cellules nécrotiques. Les densités du nutriment et du facteur de croissance sont regroupées dans une même variable $c(x,t)$. Voici, par exemple, l'équation modélisant l'évolution de la population de cellules quiescentes :

$$\frac{\partial q}{\partial t} = \underbrace{\frac{\partial}{\partial x}\left[\frac{q}{p+q}\frac{\partial}{\partial x}(p+q)\right]}_{(1)} + \underbrace{f(c)p}_{(2)} - \underbrace{h(c)q}_{(3)} \tag{2.4}$$

où le terme (1) modélise l'influence des interactions sur la motilité cellulaire : $-\frac{\partial}{\partial x}(p+q)$

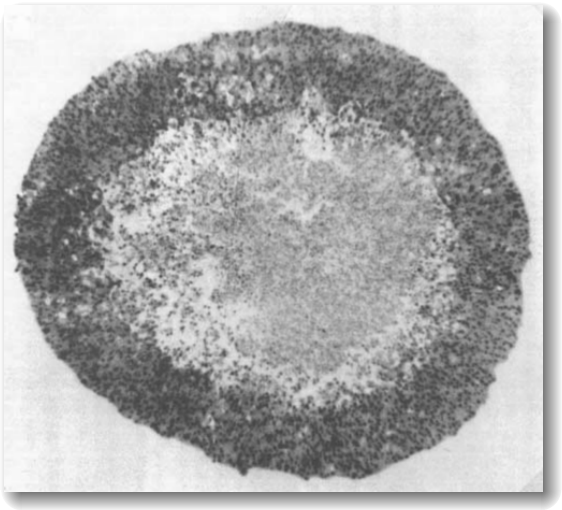


Figure 2.3 – Image *in vitro* d'une coupe de sphéroïde illustrant la structure en couches de celle-ci : le cœur de la tumeur est composé de cellules nécrotiques. Le cœur est entouré d'une couche de cellules quiescentes, elle même entourée d'une couche de cellules prolifératives. Figure extraite de (Sherratt et Chaplain, 2001).

représente le flux de cellules total et $q/(p+q)$ est la proportion de cellules quiescentes du flux total ; le terme (2) modélise le passage à l'état quiescent des cellules prolifératives et le terme (3) modélise le passage à l'état nécrotique des cellules quiescentes. Ces deux derniers termes dépendent tous deux de la concentration du nutriment / facteur de croissance c.

Le modèle a permis de simuler la croissance d'une tumeur exhibant les trois couches présentées ci-dessus. Le modèle permet en outre la prise en compte des mouvements des cellules selon leurs interactions. Le terme décrivant la motilité des cellules a en particulier montré son utilité lors de la simulation de croissance de tumeur dans un tissu épithélial : la croissance de la tumeur est alors fortement ralentie par rapport à la croissance *in vitro* d'une sphéroïde. L'influence du tissu environnant sur la croissance de tumeurs est effectivement un comportement attendu *in vivo*.

Protéolyse et invasion de tissus par des cellules cancéreuses. L'invasion par une tumeur des tissus qui l'environnent (et à fortiori sa capacité à produire des métastases) passe par la capacité des cellules tumorales à dégrader la matrice extra-cellulaire qui environne la tumeur. La dégradation de la matrice extra-cellulaire se produit par des enzymes, et plus précisément des protéases, réalisant un processus de protéolyse sur les molécules composant la matrice extra-cellulaire. Cette dégradation permet aux cellules cancéreuses de migrer dans les environs de la tumeur. Ce processus de protéolyse n'intervient pas que dans des processus anormaux tels que le développement d'une tumeur mais également lors de processus tel que la réparation de lésions cutanées. C'est un processus à considérer dans le cadre de traitement de cancers.

Lolas (2006) propose un modèle mathématique, pour modéliser l'invasion de tissus par une tumeur, basé sur la protéolyse de la matrice extra-cellulaire environnante. Les entités prises en compte dans le système sont donc les cellules tumorales, deux protéases, un inhibiteur et le substrat de la matrice extra-cellulaire. La densité des cellules tumorales prend en compte les mouvements aléatoires, la chimiotaxie due à une protéase et à l'inhibiteur, l'haptotaxie due au substrat de la matrice extra-cellulaire et la prolifération cellulaire. Les

densités des autres composants du système sont décrites de la même façon, permettant de construire un système de cinq EDP permettant de simuler l'invasion de la tumeur dans les tissus environnants.

La modélisation au moyen de systèmes d'équations permet donc la simulation de phénomènes aussi complexes que l'évolution de cancers ou leurs traitements possibles. L'approche continue a en outre l'avantage de mettre à disposition des modélisateurs des outils mathématiques permettant de montrer la stabilité des solutions du système ou de déterminer les comportements stationnaires. L'approche continue n'est toutefois pas la seule qui soit à notre disposition lorsque l'on s'intéresse à la modélisation de tissus biologiques : la section suivante présente des approches discrètes et leurs applications à travers différents exemples issus de la littérature.

3 Modélisation discrète

Alors que la modélisation continue est davantage tournée vers l'étude de populations de manière globale, les approches discrètes permettent de modéliser les composants d'un système de manière individuelle, de même que leurs interactions. Dans le cas de modélisation de tissus, le composant de base est le plus souvent la cellule. Différents formalismes discrets existent : dans la suite nous nous intéresserons d'une part aux automates cellulaires et deux de ces variantes les plus connues — les automates cellulaires de type gaz sur réseau et le modèle de Potts cellulaire — et d'autre part aux approches individus-centrées, et en particulier aux systèmes multi-agents. Nous verrons que ces approches sont d'une grande expressivité et permettent de prendre en compte à la fois des caractères qualitatifs et quantitatifs des systèmes.

3.1 Automates cellulaires

Un automate cellulaire est une grille composée de cellules : ici il s'agit de cases de la grille — ou sites — et pas de cellules au sens biologique du terme. Ces cellules peuvent être dans un nombre fini d'états, par exemple « vivant » ou « mort ». Les cellules de l'automate ont un état initial au temps $t = 0$ et l'évolution du système est réalisée en étudiant les voisinages de chaque cellule : au temps $t + 1$, l'état d'une cellule dépend de l'état de ses voisines au temps t. Un ensemble de règles permet de définir comment les cellules évoluent.

Le concept d'automate cellulaire a été proposé par Stanislaw Ulam et John von Neumann dans les années 1940 dans le but de créer des systèmes auto-reproductifs, mais le concept a pris son essor lorsque John Conway a proposé le Jeu de la Vie en 1970 (Gardner, 1970). Cet automate cellulaire est composé d'une grille en deux dimensions et l'évolution des cellules est réalisée en considérant un voisinage de Moore, illustré sur la figure 2.4a.

Les cellules de la grille peuvent avoir un des deux états suivants : vivant ou mort. Les règles ci-après permettent de faire évoluer le système :

- si une cellule a moins de deux voisines vivantes, elle sera morte à la génération suivante (sous-population) ;
- si une cellule a pour voisines deux ou trois cellules vivantes, alors elle sera vivante à la génération suivante (survie) ;
- si une cellule a plus de trois voisines vivantes, elle sera morte à la génération suivante (sur-population) ;
- si une cellule morte a pour voisines exactement trois cellules vivantes, alors elle sera vivante à la génération suivante (reproduction).

Figure 2.4 – Différents types de voisinage dans un automate cellulaire : (a) voisinage de Moore : le voisinage de la cellule bleue est constitué des huit cellules vertes adjacentes : horizontalement, verticalement et dans les diagonales ; (b) voisinage de von Neumann : le voisinage de la cellule bleue est constitué des quatre cellules vertes adjacentes : horizontalement et verticalement.

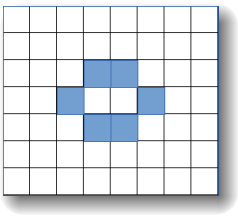


Figure 2.5 – Jeu de la Vie de Conway : motif stable (la « ruche »).

Ces règles sont appliquées de manière simultanée à l'ensemble des cellules de la grille. L'évolution de l'automate peut amener à faire apparaître des structures très variées, qui peuvent être stables (figure 2.5), périodiques (figure 2.6), croissantes ou finissant par disparaître.

Modèle d'automate cellulaire pour l'étude de tumeurs invasives. Le caractère invasif d'un cancer est à la fois dû à des interactions complexes entre les cellules tumorales ainsi qu'entre les cellules tumorales et leur environnement. L'invasion des tissus environnants par la tumeur met en jeu différents processus, comme la séparation d'agrégats de cellules homotypiques, la dégradation de la matrice extra-cellulaire, l'altération de la motilité cellulaire ou encore la mise en place d'adhésions hétérotypiques. Il a été observé, expérimentalement, la formation de branches invasives depuis la tumeur. Jiao et Torquato (2011) proposent un modèle d'automate cellulaire afin d'étudier les mécanismes permettant à une tumeur de devenir invasive. La topologie de l'automate est basée sur un diagramme de Voronoï, une décomposition particulière de l'espace en régions. Les règles d'évolution de l'automate sont les suivantes : 1) une cellule proliférative trop éloignée du bord de la tumeur pour recevoir un apport suffisant en nutriments devient quiescente. Sinon elle peut se diviser et donner lieu à une nouvelle cellule proliférative ou à une cellule invasive ; 2) une cellule quiescente trop éloignée du bord de la tumeur devient nécrotique ; 3) une cellule invasive peut dégrader

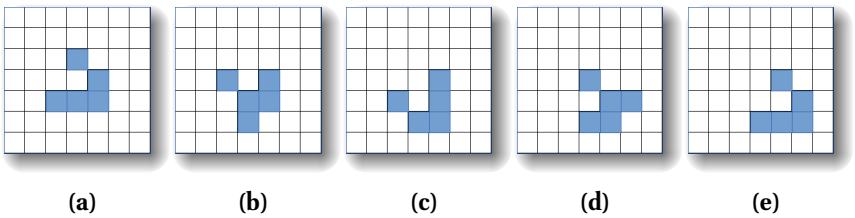


Figure 2.6 – Jeu de la Vie de Conway : motif périodique de période 4 (le « planeur »).

la matrice extra-cellulaire environnante, lui permettant de se déplacer dans une région de l'automate voisine de la sienne.

Après une première validation de leur modèle, illustrée sur la figure 2.7, les auteurs ont testé différents paramètres afin de réaliser des prédictions sur le caractère invasif de la tumeur. En particulier, ils ont mis en évidence l'importance de la motilité cellulaire et l'influence de la rigidité et de la densité de la matrice extra-cellulaire dans le développement de la tumeur.

3.2 Variantes d'automates cellulaires

Les paragraphes suivants introduisent deux variantes majeures d'automates cellulaires : les gaz sur réseau et le modèle de Potts cellulaire.

a) Automates cellulaires de type gaz sur réseau

Les gaz sur réseau (LGCA, pour *Lattice Gaz Cellular Automata*) sont des variantes d'automates cellulaires utilisées en mécanique des fluides, en particulier pour déterminer des solutions numériques aux équations de Navier-Stockes. Le premier modèle s'apparentant à un automate de gaz sur réseau a été proposé par Hardy et coll. (1973).

Cet automate est constitué d'une grille dont les nœuds sont connectés par des canaux, les mailles du réseau, qui représentent des directions. Un nœud peut être occupé par une ou plusieurs particules mais une direction ne peut être empruntée que par une particule (principe d'exclusion). L'évolution de l'automate est réalisé 1) par des collisions locales (entre particules sur un nœud donné) qui, selon les règles définies dans le système, permettent de déterminer les nouvelles directions des particules et 2) par des propagations de particules vers les nœuds voisins.

La figure 2.8 montre un exemple d'évolution d'un automate de gaz sur réseau tel que celui proposé par Hardy et coll.. Il existe différentes variations d'automates de gaz sur réseau, par exemple dans la topologie ou encore les règles d'évolution du système ; une description de plusieurs de ces automates est proposée par Wolf-Gladrow (2000).

Étant donnée que l'utilisation première des automates de gaz sur réseau concerne la mécaniques des fluides, les modèles de systèmes biologiques les utilisant concernent généralement le sang (écoulement dans le cœur, coagulation) (Golbert et coll., 2012).

Modélisation de l'invasion de gliome avec un automate de type gaz sur réseau. Les gliomes sont des tumeurs cérébrales issues des cellules gliales. Ces dernières forment l'environnement des neurones : elles produisent de la myéline, fournissent nutriments et oxygène au réseau nerveux, éliminent les cellules mortes, etc. Un gliome est composé de deux zones : un cœur ainsi qu'une zone d'invasion sur le bord de la tumeur. Tektonidis et coll. (2011) proposent un modèle de LGCA, construit sur la base d'observations *in vitro* de cellules de gliomes (les données expérimentales), afin d'étudier et de déduire les mécanismes d'invasion des gliomes : dans ce cas, les auteurs se concentrent sur les mécanismes des cellules tumorales, résultant de processus intra-cellulaires, et sur les interactions des cellules de la tumeur. L'objectif est de pouvoir réaliser des prédictions au niveau macroscopique de l'évolution du tissu. Dans cet automate, les particules du réseau représentent des cellules. La phase de collision correspond, de façon plus générale, à une phase d'interaction permettant de modéliser les comportements de mort et de prolifération cellulaire. La phase de propagation correspond à une phase de migration des cellules.

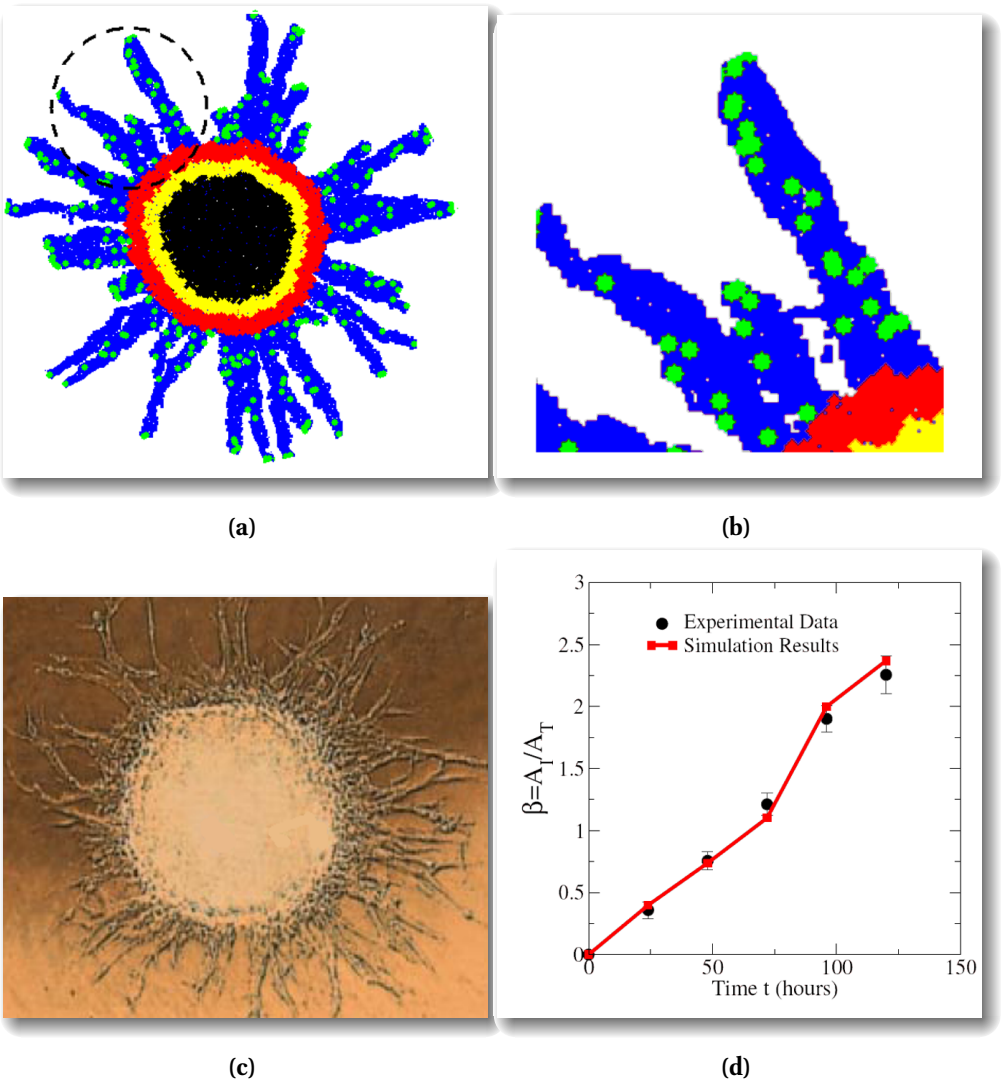
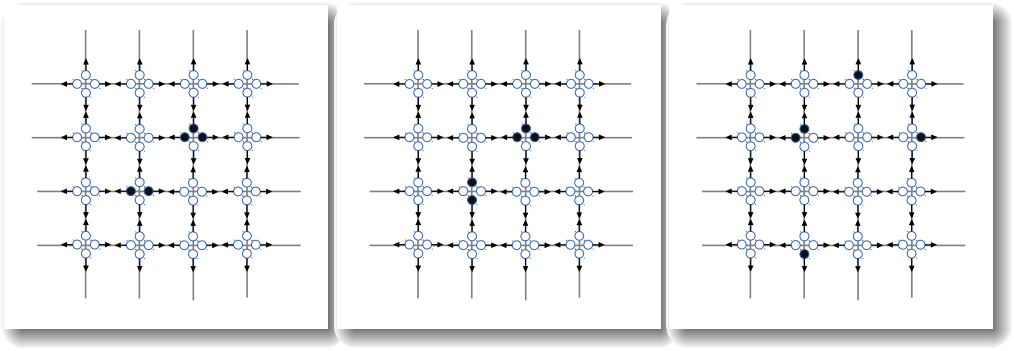


Figure 2.7 – Modélisation et simulation par un automate cellulaire de la formation de branches invasives lors du développement de tumeurs (Jiao et Torquato, 2011). (a) simulation de l'invasion de tumeur avec l'automate cellulaire : la zone noire est composée de cellules nécrotiques, la zone jaune est constituée de cellules quiescentes, la zone rouge est constituée de cellules prolifératives, les zones vertes sont les cellules invasives, les zones bleues correspondent à la matrice extra-cellulaire dégradée ; (b) zoom sur des branches invasives de la tumeur (partie cerclée de la figure a) ; (c) observation expérimentale de la formation de branches invasives ; (d) ratios *zone invasive sur tumeur primaire* : comparaison entre des données expérimentales (en noir) et les données issues de la simulation (en rouge). Figures extraites de (Jiao et Torquato, 2011).



(a) **(b)** **(c)**

FIGURE 2.8 – Exemple de réseau d'un automate de gaz sur réseau tel que défini par Hardy et coll.. Les disques pleins représentent les canaux occupés par des particules ; sinon les canaux sont vides. (a) Étape avant la collision des deux particules en bas à gauche. (b) Après collision des deux particules : dans le cas du modèle de Hardy et coll., deux particules peuvent entrer en collision si elles sont sur un même site dans des directions opposées et que les deux autres canaux sont inoccupés. (c) Étape de propagation : toutes les particules de la grille évoluent dans la direction donnée par leur vitesse.

TABLE 2.1 – Dynamique de l'automate : les mécanismes de prolifération, motilité et changement de phénotype ont fait l'objet de raffinements jusqu'à obtention de simulations dont les résultats sont similaires aux données expérimentales. Les règles présentées sont les règles finales ayant permis d'obtenir les résultats de simulation présentés sur la figure 2.9.

Mécanisme	*Règle dans l'automate*
Prolifération	Cellules non migrantes
Motilité	Répulsion cellules/cellules
Changement de phénotype	Selon la densité de cellules dans l'environnement local

Ce modèle est un LGCA composé de deux grilles, chacune permettant de modéliser un phénotype de cellule : cellules en mouvement, cellules immobiles. La dynamique du LGCA est construite avec des règles, que les auteurs vont raffiner jusqu'à observer des corrélations fortes entre leurs résultats simulés et les résultats expérimentaux dont ils disposent. Ces résultats sont visibles sur la figure 2.9.

Les règles ayant permis d'obtenir ces résultats — ou, pour formuler d'une autre manière, les mécanismes cellulaires mis en jeu — sont présentées dans le tableau 2.1. Les trois mécanismes principaux mis en jeu sont la prolifération, la motilité et le changement de phénotype. Différents modes pour chaque mécanisme sont possibles : par exemple, au départ le changement de phénotype est aléatoire, alors que la règle finale stipule que ce changement peut survenir selon la densité de cellules dans l'environnement local.

Malgré les éléments tendant à valider le modèle par rapport aux données expérimentales utilisées, le système possède également des limites, présentées par les auteurs de l'étude. En particulier, les auteurs n'ont pas déterminé de manière exacte les valeurs des paramètres de leur modèle. Cependant, c'est là un attrait majeur de la simulation numérique : l'idée de l'exploration, pour émettre de nouvelles hypothèses. Par exemple, les auteurs expliquent que l'hypothèse selon laquelle la densité de cellules influence le changement de phénotype n'a pas de preuve expérimentale. Toutefois, compte tenu des résultats de leurs simulations, on peut imaginer que ce type d'exploration permet de lancer de nouvelles pistes à explorer *in vitro* afin, éventuellement, de confirmer ou d'infirmer cette hypothèse. L'exploration numé-

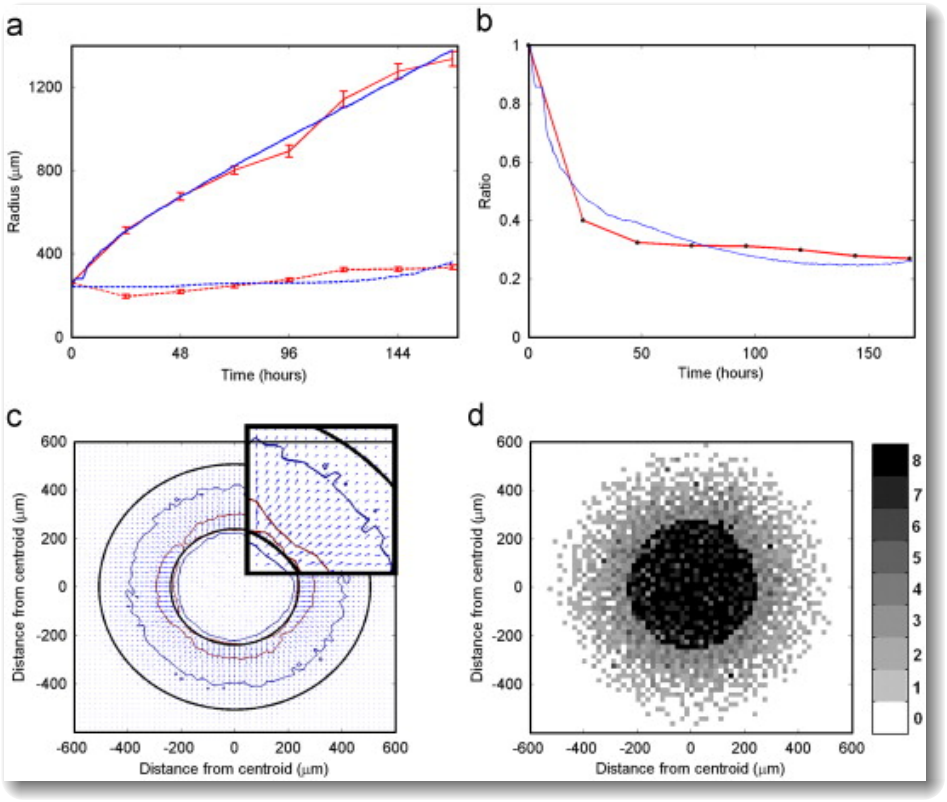


Figure 2.9 – Modélisation de gliome avec un automate de type gaz sur réseau. (a) et (b) comparaison des données obtenues en simulation (courbes bleues) avec les données expérimentales (courbes rouge) : le graphique (a) représente l'évolution au cours du temps des diamètres du cœur (lignes pointillées) et de la zone invasive (lignes pleines) ; le graphique (b) présente les rapports au cours du temps des diamètres du cœur et de la zone invasive ; (c) direction des mouvements des cellules dans le gliome ; (d) visualisation spatiale de la tumeur simulée. Figure extraite de (Tektonidis et coll., 2011).

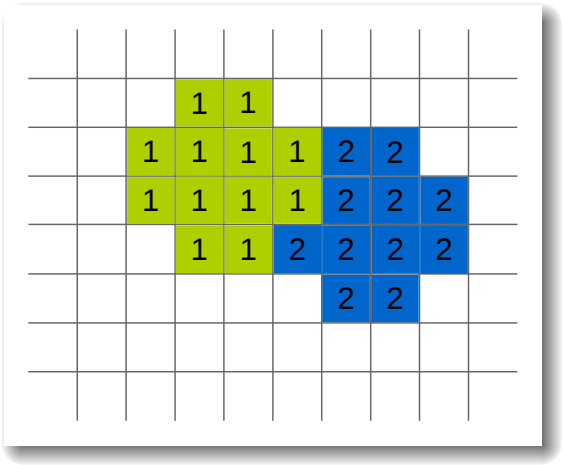


FIGURE 2.10 – Définition de cellules dans le formalisme du modèle de Potts cellulaire : une cellule est composée de sites de la grille marqués par l'identifiant de la cellule.

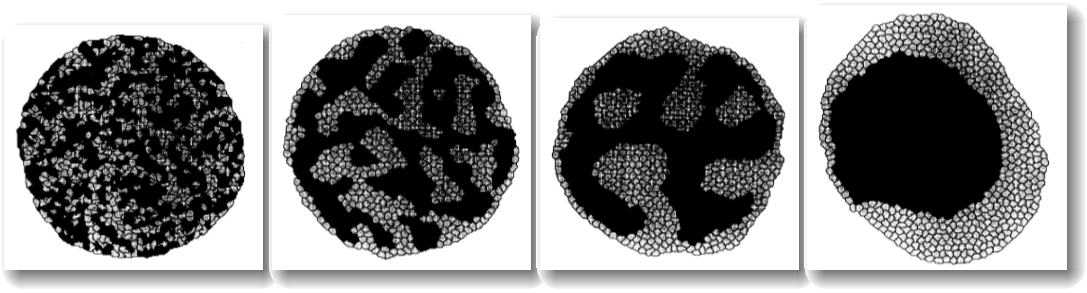
rique permet de dégager de nouvelles pistes ou de nouvelles hypothèses grâce aux modèles utilisés. Cet aspect a ses limites : le mécanisme de répulsion utilisé pour modéliser la motilité est présenté sous forme de « boîte noire », c'est-à-dire dont les spécificités sont opaques : pourrait-il s'agir de répulsion due à des chimio-attracteurs ? des chimio-répulseurs ? d'autres facteurs environnementaux ? D'autres expérimentations sont requises pour aller plus loin dans la compréhension du mécanisme.

b) Modèle de Potts cellulaire

Le modèle de Potts cellulaire (CPM, pour *Cellular Potts Model*) est une extension du modèle de Potts, qui est lui-même une généralisation du modèle d'Ising (Ising, 1925), un modèle de physique statistique utilisé pour étudier les interactions de particules à deux états, comme dans le cas du ferromagnétisme. Le modèle de Potts (Potts, 1952) est une généralisation de ce modèle permettant de pallier la limite du nombre d'états possibles pour les particules.

Le modèle de Potts cellulaire, proposé par Graner et Glazier (1992), est une extension du modèle de Potts pour des applications biologiques. Un CPM est représenté par une grille dont les sites sont associés à des indices, permettant d'identifier les sites occupés par une cellule à un instant donné : des sites différents peuvent donc être identifiés par un même indice. Ces composants discrets composent les cellules et leur re-disposition peut ainsi permettre de modéliser des mouvements cellulaires ou encore des changements de forme. La figure 2.10 montre comment les cellules sont définies dans un CPM : une cellule est constituée de plusieurs sites de la grille qui portent l'identifiant de la cellule. Les sites « vides » de la grille représentent l'environnement des cellules.

La dynamique du système est basée sur la minimisation de l'énergie, autrement dit une changement d'état du système doit conduire à une baisse de l'énergie de ce système. Les énergies prises en compte dans le modèle de Potts original sont les énergies de contact, de surface et de volume. La prise en compte d'autres énergies permet de modéliser des comportement de chimiotaxie ou d'haptotaxie, ou encore d'imposer une forme à la cellule (Tripodi et coll., 2010). L'énergie totale du système sert à calculer la probabilité de transition d'un état à un autre.



(a) (b) (c) (d)

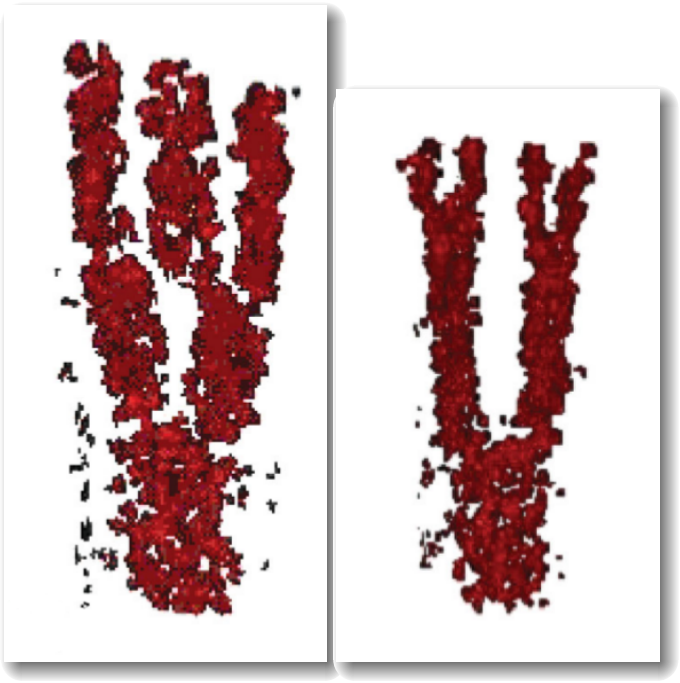
Figure 2.11 – Simulation de tri cellulaire avec le CPM tel que proposé par Graner et Glazier. Les cellules foncées (fortement adhésives) forment un agrégat compact entouré des cellules claires (moins adhésives). Figure extraite de (Graner et Glazier, 1992).

Modélisation de tri cellulaire avec un modèle de Potts cellulaire. Le tri cellulaire est un phénomène se produisant lors de la formation de tissus. Une des hypothèses expliquant ce phénomène est l'adhésion différentielle (DAH, pour *Differential Adhesion Hypothesis*) proposée par Steinberg (1963) : les cellules se ré-arrangent en fonction des adhésions des types cellulaires en présence. C'est un phénomène qui a été beaucoup observé expérimentalement chez l'hydre, par exemple par Rieu et coll. (1998) : une couche externe de cellules ectodermique entoure les agrégats de cellules endodermiques ; en matière d'adhésion cela signifie que les cellules endodermiques sont plus adhésives entre elles que les cellules ectodermiques entres elles et que les cellules ectodermiques avec les cellules endodermiques.

Graner et Glazier (1992) ont utilisé leur extension du modèle de Potts afin de modéliser le phénomène de tri dans un agrégat de cellules. Ils ont montré que la motilité cellulaire et l'adhésion suffisent à obtenir un tissu trié. La figure 2.11 est un exemple de simulation obtenu par Graner et Glazier avec leur modèle. La validation de leurs observations est visuelle ; toutefois dans le cas du tri cellulaire on peut obtenir des éléments de validation quantitatifs en calculant le ratio *nombre de cellules voisines du même type* sur *nombre de cellules voisines total.*

Modélisation de la croissance de membres chez les vertébrés. Chaturvedi et coll. (2005) ont utilisé un CPM pour modéliser et simuler la croissance de membres chez les vertébrés, en particulier chez le poulet, dont le développement est beaucoup étudié. En premier lieu, le bourgeon qui va devenir un membre est constitué de cellules prédifférenciées ayant la capacité de se déplacer, de se diviser et de produire diverses molécules. Différentes zones sont identifiées dans le bourgeon, dans lesquelles les cellules exhibent différents comportements : par exemple, les cellules présentes au bout du bourgeon sont capables de répondre à des morphogènes. Les cellules du bourgeon produisent un facteur de croissance, la TGF-β, ainsi qu'un inhibiteur. Si des cellules perçoivent ce facteur au delà d'un certain seuil, elles se différencient en un type de cellule produisant de la fibronectine, une protéine permettant l'adhésion des cellules à la matrice extra-cellulaire. Le facteur de croissance entraîne une boucle de rétroaction positive chez les cellules le percevant ; ces dernières produisant plus de fibronectine et de facteur de croissance. Les cellules ne pouvant produire de fibronectine répondent tout de même à la présence de cette protéine. Toutes les cellules du bourgeon se divisent, causant la croissance du membre.

Chaturvedi et coll. ont utilisé leur modèle à la fois pour étudier le développement normal de membre, mais également pour étudier le développement anormal, en jouant sur



(a) **(b)**

FIGURE 2.12 – Développement de membre chez le poulet : (a) développement normal ; (b) développement anormal causé par une diffusion anormale de facteurs de croissance. Figures extraites de Chaturvedi et coll. (2005).

les paramètres de leur modèle. En particulier, des diffusions anormales des facteurs de croissance les ont conduit à observer un développement de membre tel qu'observé dans le cas du syndrome d'Apert, une maladie génétique rare conduisant à des malformations des membres et du crâne. La figure 2.12 illustre deux exemples de développement : (a) un développement normal et (b) un développement anormal causé par la diffusion anormale de facteurs de croissance.

Modélisation de la culmination chez le *Dictyostelium Discoideum*. Le *Dictyostelium Discoideum* est un organisme eucaryote unicellulaire qui, en cas d'absence de nutriments, va s'agréger à d'autres cellules afin de former un organisme pluricellulaire. Cet organisme va ensuite migrer, et lorsque des conditions plus favorables sont rencontrées, il va former un corps fructifère, constitué d'une tige surmontée de spores qui seront ensuite disséminées dans l'environnement. Ce cycle de développement est illustré sur la figure 2.13. La formation des différentes formes de l'organisme est réalisée par chiomiotaxie.

L'étape de la culmination se produit après que les cellules aient formé un agrégat. Différents types principaux de cellules sont en jeu : les cellules PstA (*Pre Stalk A*), qui émettent de façon pulsatile de l'adénosine monophosphate cyclique (AMPc) qui va jouer le rôle d'agent chiomiotactique ; les cellules PstO (*Pre Stalk O*) et Psp (*Pre Spore*) qui migrent par chimiotaxie vers les gradients les plus importants d'AMPc, et les cellules de la tige, qui elles ne répondent pas aux gradients d'AMPc. Les cellules PstA se différencient en cellules de type Stalk (des cellules de la tige) au contact de ces dernières, de la même façon que les cellules PstO se différencient en PstA. Différentes adhésions sont en jeu entre les différents types cellulaires.

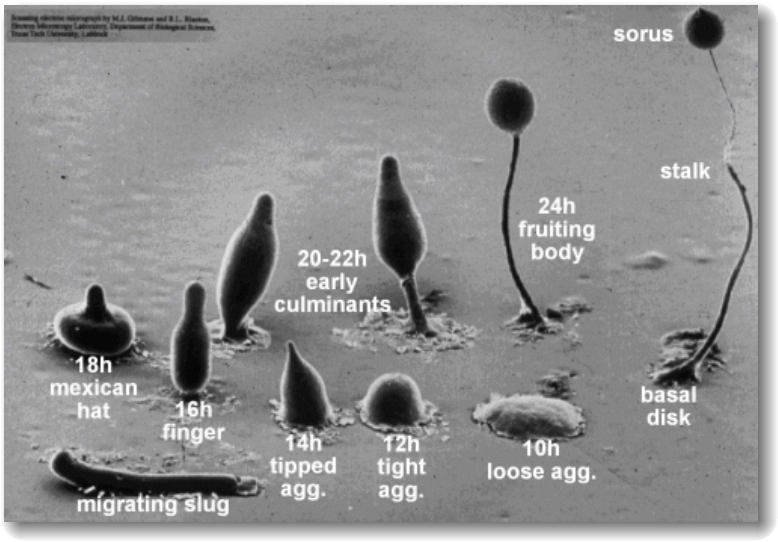


Figure 2.13 – Étapes de développement chez le *Dictyostelium Discoideum*. Figure : M.J. Grimson et R.L. Blanton, Texas Tech University (`http://www.dictybase.org/`).

Marée et Hogeweg (2001) proposent d'utiliser un CPM pour étudier l'étape de culmination. La figure 2.14 illustre une simulation réalisée avec ce modèle. Une hypothèse intéressante émise par les auteurs concerne la formation de la tige du corps fructifère : les études antérieures à celle de Marée et Hogeweg expliquaient la formation et l'allongement de la tige grâce à l'ajout de cellules en haut de celle-ci, par différentiation des cellules PstA en cellules Stalk. Cependant, les simulations réalisées par Marée et Hogeweg ont conduit ces derniers à émettre l'hypothèse que l'élongation de la tige est surtout due à la pression causée par les cellules attirées par les gradients d'AMPc produits par les cellules PstA, ainsi que grâce à l'adhésion différentielle entre les différents types cellulaires — l'adhésion des cellules de la tige entre elles est plus forte que l'adhésion entre les cellules PstO et Stalk — pressant la tige vers le bas, à la manière de mouvements péristaltiques tels que ceux réalisés par le tube digestif. Ces résultats ont été mis en corrélation avec des observations expérimentales montrant que des mouvements chimiotactiques d'intensités réduites ne permettaient pas la formation de la tige (Chen et coll., 1995).

3.3 Systèmes multi-agents et modèles individus-centrés

Les systèmes multi-agents (SMA) sont des modèles individus-centrés composés d'entités (les agents) qui agissent de manière autonome dans un environnement. Contrairement aux méthodes continues, dont l'objet est de décrire dans le modèle le comportement global du système, les SMA reposent sur le principe que seules les interactions locales sont décrites. Ces interactions peuvent avoir lieu entre les agents ou entre les agents et leur environnement. Le comportement global s'observe donc comme un phénomène émergeant des multiples interactions définies dans le modèle. Étant donné que le modèle que nous défendons dans cette thèse est fortement lié à la notion de système multi-agents, une définition plus complète de ces systèmes sera donnée au chapitre 3.

Les SMA ont pour origines principales deux disciplines : l'intelligence artificielle distribuée (IAD) et la vie artificielle (VA). L'IAD est née de l'idée de permettre à des machines ou des programmes reliés par réseau de résoudre des problèmes potentiellement complexes

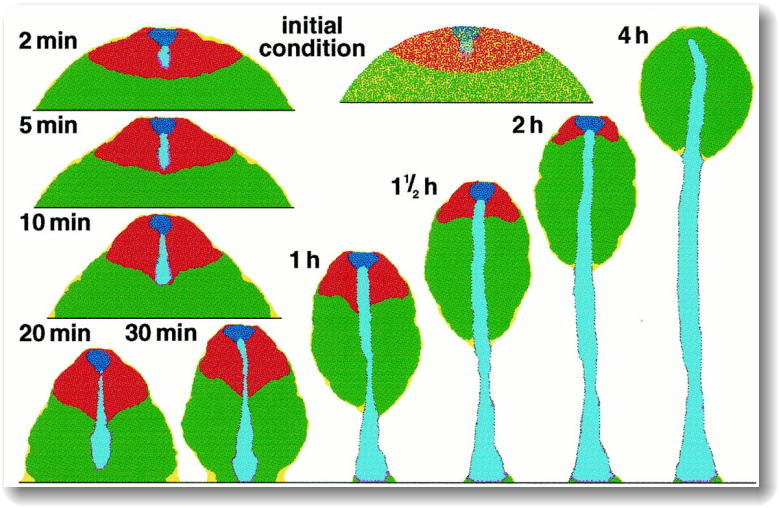


FIGURE 2.14 – Simulation de la culmination de Dictyostelium Discoideum par Marée et Hogeweg. Différents types cellulaires sont : PstA (bleu), PstO (rouge), PsP (vert) et St (cyan). Figure extraite de (Marée et Hogeweg, 2001).

via du raisonnement. Deux travaux majeurs en IAD ont fortement contribué au domaine de la modélisation et de la simulation multi-agents (Decker, 1996) : DVMT (*Distributed Vehicle Monitoring Testbed*), proposé par Lesser et Corkill (1983), est un environnement simulant un réseau de nœuds (des agents) surveillant des véhicules. Les informations perçues peuvent être redondantes ou bruitées et l'architecture a proposé les bases des mécanismes de coopération entre agents afin de pallier les erreurs et les ambiguïtés. MACE (*Multi-Agent Computing Environment*), proposé par Gasser et coll. (1987), est un environnement générique de développement de systèmes d'IAD proposant un ensemble de composants pour la création et l'envoi de message, etc. La VA quant à elle regroupe différents thèmes de recherches, et en particulier l'analyse de la dynamique des phénomènes complexes, l'évolution de populations, la réalisation d'animats et l'étude des phénomènes collectifs (Ferber, 1995).

Les SMA trouvent des applications dans de nombreux domaines comme le trafic routier ; les mouvements de foule ; l'étude des essaims d'oiseaux et des bancs de poissons ; les insectes sociaux comme les fourmis, les termites ou les guêpes ; la biologie moléculaire ; et en ce qui concerne notre travail, la biologie du développement.

Il existe une variété importante de modèles basés sur une approche individus-centrée pour simuler des systèmes biologiques au niveau de la cellule. Tous ces modèles ont en commun d'implémenter, à des degrés divers, la majeure partie des comportements cellulaires et leurs interactions. Finalement, la différence principale que l'on note entre différents modèles se situe dans la *représentation physique de la cellule*, même si la modélisation des différents comportements de la cellule peut faire l'objet de différentes propositions, celles-ci sont trop variées pour proposer une classification claire. Nous avons vu, par exemple avec les automates cellulaires, que la cellule est physiquement réduite à sa plus simple forme : un point. Cette représentation est, on le verra à travers un exemple, également utilisée dans des modèles individus-centrés (il existe une forte proximité conceptuelle entre les automates cellulaires et les approches individus-centrés). Dans le modèle de Potts cellulaire, la forme des cellules devient plus complexe, étant donné que celles-ci peuvent occuper plusieurs sites de la grille. Pour le reste, dans le cas notamment où les cellules sont situées dans un environnement continu, nous verrons que la forme des cellules peut être l'objet de différentes

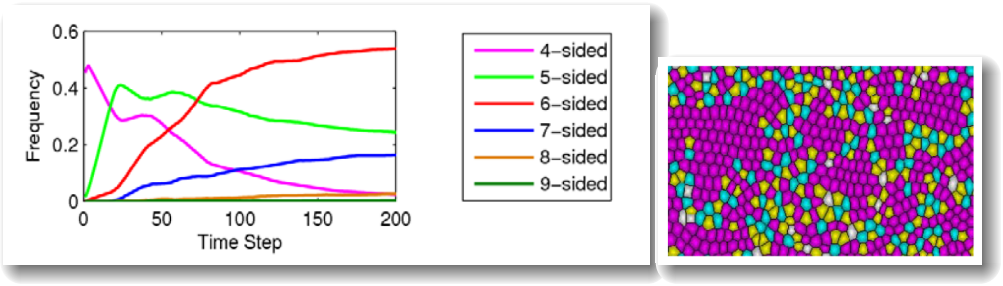
propositions. La forme donnée aux cellules dépend en partie du système que l'on souhaite modéliser. Par exemple des cellules construites avec des polygones sont bien adaptées à l'étude de tissus épithéliaux. Mais de la topologie des cellules peuvent aussi dépendre les éléments que l'on pourra étudier dans le système : contrairement à un modèle dont la cellule est représentée par un point dans l'espace, les modèles biomécaniques permettent de prendre en compte des contraintes physiques issues de l'environnement et dans ce cas la cellule est au minimum représentée par des disques ou des sphères élastiques.

Les modèles à vertices sont des modèles dans lesquels les cellules sont construites à partir de sommets partagés avec ses cellules voisines, la topologie des cellules évoquant les régions des diagrammes de Voronoï. Ces modèles sont souvent utilisés pour étudier la topologie des cellules et des tissus, comme nous allons le voir dans la suite. En dehors de cette catégorie de modèles un peu particulière, un élément important distinguant les modèles les uns des autres est le fait que la cellule aie ou non une forme et soit ou non déformable. On retrouve donc des modèles dont la topologie de la cellule peut aller d'une simple sphère — pouvant ou non être déformable —, voire des particules, à des modèles dont la forme de la cellule est plus complexe. Les paragraphes suivants présentent des modèles selon la forme de la cellule.

a) Modèles à vertices

Étude de la topologie des cellules lors du développement de tissu épithélial. Le tissu épithélial est une catégorie de tissu cellulaire formant les revêtements internes, telle qu'une surface de muqueuse, ou externe, telle que la surface de la peau. Ces tissus sont constitués de cellules juxtaposées très étroitement et ils sont très résistants aux stress mécaniques. Les interactions et la prolifération des cellules influencent fortement la topologie des tissus épithéliaux. La simulation de ce type de tissu doit permettre de mieux comprendre comment le tissu acquiert son intégrité lors de son développement et comment il la conserve. Li et coll. (2012) proposent un modèle de cellule basé sur des polygones afin d'étudier la topologie des cellules dans le tissu épithélial selon les plans de division des cellules et les forces mécaniques se produisant dans le tissu. Les auteurs ont réalisé les observations suivantes : 1) le plan selon lequel les cellules se divisent joue un rôle déterminant dans la topologie des cellules dans le tissu : par exemple, les divisions dont l'axe est pivoté de 90° à chaque génération successive donne lieu à une proportion de 55% de cellules hexagonales dans le tissu — la figure 2.15 est un exemple de simulation dans laquelle on peut visualiser ce résultat — ; 2) les forces mécaniques induites par les cellules jouent également un rôle dans la topologie de celles-ci. En effet, des divisions « organisées », telle que la division orthogonale ou bien selon l'axe coupant le côté le plus long de la cellule, limitent les ré-arrangements des membranes et conduisent à des formes de cellule plus régulières car les stress sur les membranes sont moins importants.

Aegerter-Wilmsen et coll. (2010) utilisent un modèle à vertices pour étudier la topologie des cellules de tissu épithélial également. En particulier, les auteurs ont noté une influence importante des stress mécaniques sur la régulation de la croissance des cellules. Les résultats des expérimentations réalisées avec le modèle sont cohérents avec les données expérimentales utilisées par les auteurs uniquement dans le cas où le modèle inclut une dépendance entre la croissance des cellules et les contraintes mécaniques, sans toutefois que cela signifie que ce soit effectivement ces contraintes qui soient à l'origine de la topologie observée expérimentalement. Cependant, ces simulations apportent des hypothèses et des pistes qui pourront être explorées plus avant dans de nouvelles expérimentations *in vitro* : on retrouve encore une fois l'aspect exploratoire de la simulation numérique.



(a) **(b)**

FIGURE 2.15 – Étude de la topologie de cellules épithéliales avec un modèle à vertices lorsque les divisions successives de cellules s'opèrent orthogonalement à la précédente. (a) fréquence du nombre des cellules de 4 à 9 côtés : les cellules hexagonales sont largement majoritaires (55%) ; (b) simulation de tissu épithélial avec le modèle, les cellules mauves sont les cellules hexagonales. Figures extraites de (Li et coll., 2012).

Dynamique des cellules lors de la croissance de tissus. Cao et coll. (2012) utilisent un modèle à vertices afin d'étudier la dynamique des populations de cellules pendant le développement de tissus. Les mécanismes de prolifération des cellules souches sont liées à des activateurs et inhibiteurs (des facteurs de croissance). Les auteurs veulent pouvoir prendre en compte les informations spatiales liées aux cellules et aux mécanismes de contrôles qui, dans des modèles continus, sont traités comme des moyennes sur la population de cellules, alors que ces facteurs ont en réalité des actions fortement localisées. Leur modèle leur a permis de simuler l'homéostasie dans les tissus. Les cellules différenciées sécrètent un inhibiteur qui limite le renouvellement des cellules souches. Lorsque le tissu comporte un nombre suffisant de cellules différenciées pour assurer ses fonctions, l'homéostasie est atteinte et les cellules souches restantes permettent la régénération du tissu si cela s'avère nécessaire. Le facteur d'inhibition dans le tissu est indispensable au maintien de celui-ci en homéostasie.

Simulation de la morphogenèse de plantes. Les modèles à vertices sont également utilisés dans l'étude du développement de plantes, dont les cellules adhèrent fortement à leurs voisines. Par exemple, Merks et coll. (2011) proposent d'étudier la régulation de la croissance des cellules et de leurs divisions en fonction des signaux chimiques. Comme dans les tissus animaux, la morphogenèse des plantes est en partie contrôlée par des signaux chimiques régulant les divisions cellulaires. Dans cet exemple, une des cellules produit un morphogène tel que l'auxine, qui a un rôle important dans le développement des plantes, et qui diffuse de cellule en cellule. Les comportements des cellules sont dépendants des concentrations du gradient qu'elles perçoivent : si la concentration est au dessus d'un certain seuil, les cellules croissent jusqu'à se diviser quand leur taille a doublé. Si la concentration est en deçà d'un certain seuil, les cellules ne croissent ni ne se divisent. Entre les deux seuils, les cellules croissent seulement. La figure 2.16 est un exemple de simulation de ce modèle : l'idée dans ce cas n'était pas de reproduire un système existant mais de tester les mécanismes mis en place dans le modèle, en particulier la capacité du tissu à se développer grâce à la diffusion de morphogènes.

Rudge et Haseloff (2005) ont proposé un modèle similaire pour modéliser le développement de plantes, en axant leur étude en particulier sur les informations spatiales dans le tissu. Stoma et coll. (2007) proposent quant à eux une approche davantage basée sur les aspects mécaniques intervenant lors de la morphogenèse de plante.

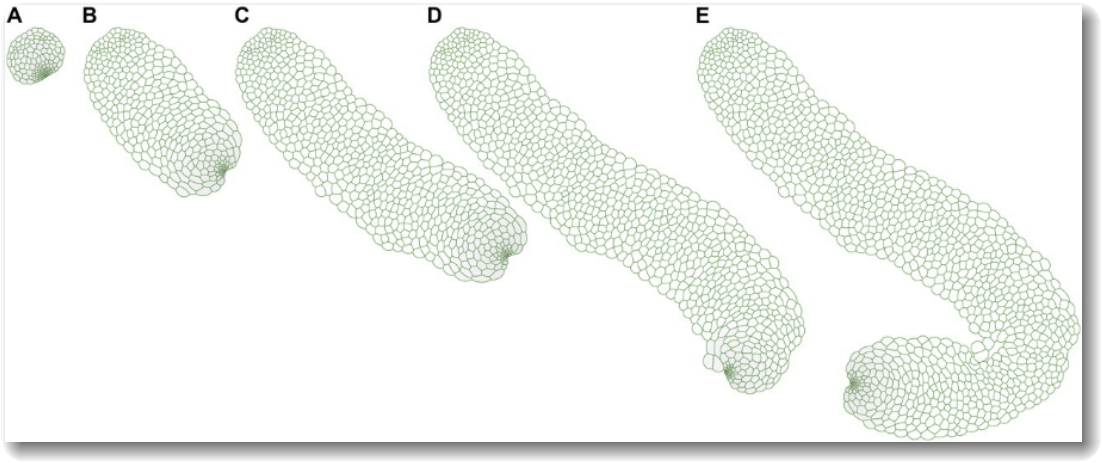


Figure 2.16 – Développement d'un tissu basé sur la diffusion d'auxine. Une cellule au bout du tissu diffuse le morphogène permettant aux cellules de croître et de se diviser là où les concentrations du morphogène sont les plus importantes (c'est-à-dire au bout). Figure extraite de Merks et coll. (2011).

b) Modèles basés sur une cellule de forme simple

Par « forme simple », on entend particules (ou point), disques et sphères. Le plus souvent, les cellules virtuelles construites à partir de ces formes ne sont pas déformables.

Simulation de tri cellulaire avec un modèle de cellule basé sur des particules. Belmonte et coll. (2008) proposent un modèle pour simuler le tri cellulaire en utilisant le principe de l'adhésion différentielle que nous avons présentée plus tôt dans le document (page 37). Les cellules sont représentées par des particules, spatialement repérées par leurs coordonnées et possédant une vitesse. À la différence du tri présenté précédemment — page 37, (Graner et Glazier, 1992) —, ici les particules alignent leurs vitesses sur les vitesses moyennes des voisins (Vicsek et coll., 1995). Les particules sont donc orientées dans des directions semblables. L'alignement des particules peut potentiellement être perturbé par du bruit. La portée de l'alignement des particules est un paramètre du système, au même titre que l'intensité des interactions entre les particules : l'adhésion. L'inclusion de l'alignement des particules est en outre cohérent avec les observations réalisées par Rieu et coll. (1998), lors de l'étude des mouvements des cellules endodermiques et ectodermiques dans des agrégats cellulaire d'hydre (figure 2.17). La figure 2.18 présente des résultats de simulation obtenus avec ce modèle : la figure (a) montre le tri des cellules à mesure que la simulation se déroule ; la figure (b) présente les résultats quantitatifs de la simulation : dans le cas du tri cellulaire, ce type de résultat est obtenu en calculant le ratio *nombre de voisins du même type* / *nombre de voisins total*. Plus le ratio est petit et plus le tissu est trié (il tend vers 0 pour un tissu suffisamment grand). Le comportement d'alignement des cellules permet de faciliter le tri dans l'agrégat.

Tri cellulaire chimiotactique. Eyiyurekli et coll. (2007) proposent également d'étudier le tri cellulaire. L'approche utilisée est toutefois différente que celle que nous avons présenté précédemment : ici le tri n'est pas réalisé en utilisant les propriétés d'adhésion des cellules mais en utilisant des chimioattractants produits par les cellules et diffusant dans l'environnement. Des phénomènes de tri d'agrégats de cellules basés sur la chimiotaxie ont été observés expérimentalement, par exemple par Miura et Shiota (2000). Physiquement, les cellules sont

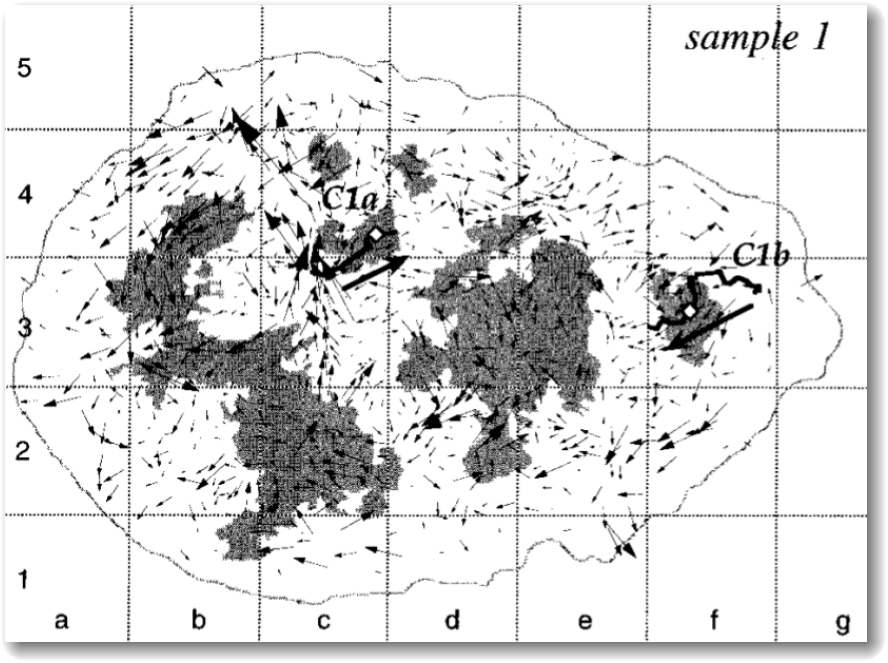
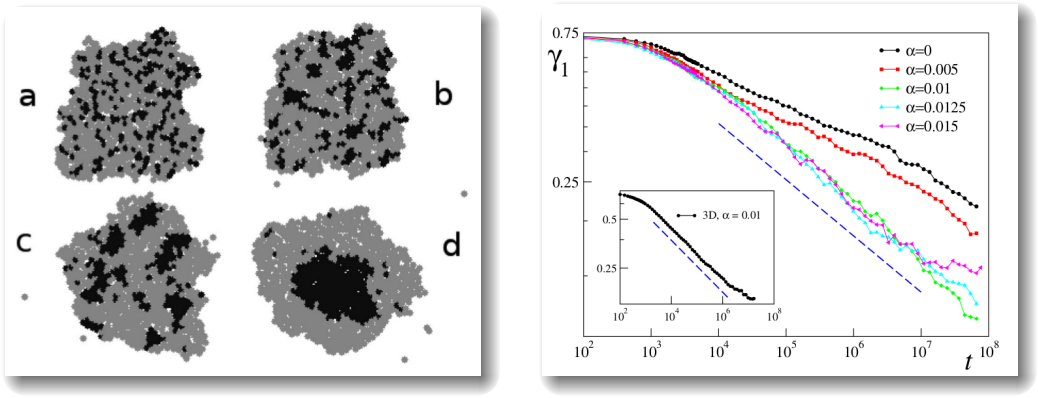


FIGURE 2.17 – Mouvements des cellules observés dans des agrégats cellulaire d'hydre. Les cellules de l'agrégats, des cellules endodermiques et ectodermiques, se déplacent dans des directions semblables à leurs voisines. Figure extraite de (Rieu et coll., 1998).



(a) **(b)**

FIGURE 2.18 – Tri cellulaire avec le modèle basé sur des particules proposé par Belmonte et coll.. La figure (a) montre le tri s'opérant dans le tissu à mesure que la simulation se déroule. La figure (b) montre les résultats quantitatifs de la simulation, obtenus en calculant le ratio *nombre de voisins du même type / nombre de voisins total*. Plus ce ratio est faible et plus le tissu est trié. Figure extraite de (Belmonte et coll., 2008).

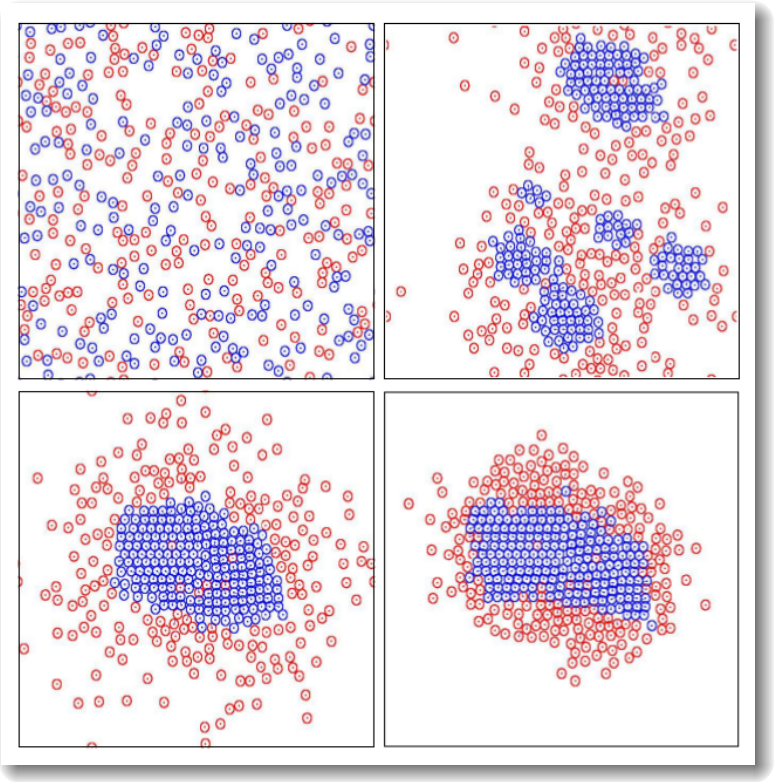
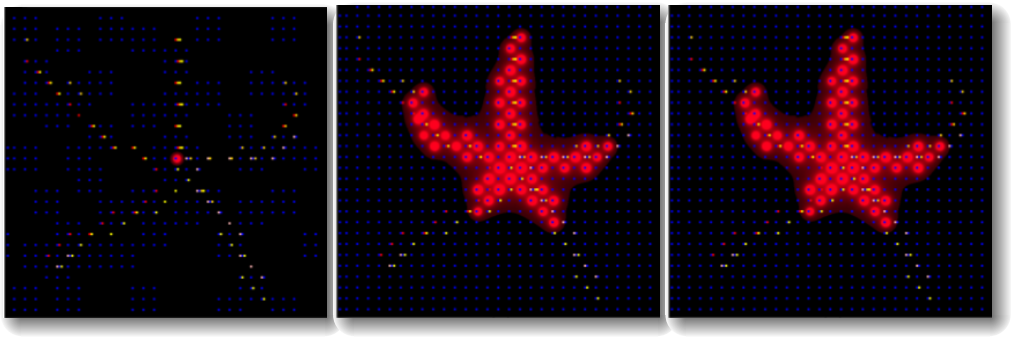


Figure 2.19 – Tri cellulaire basé sur de la chimiotaxie. Les cellules perçoivent et produisent des chimioattractants diffusant dans l'environnement. Les résultats observés sont similaires à ceux obtenus lorsque le tri est basé sur le principe de l'adhésion différentielle. Figure extraite de (Eyiyurekli et coll., 2007).

représentées par des disques non déformables. Le type cellulaire T_1 (resp. T_2) produit un chimioattractant C_1 (resp. C_2). Les réponses des cellules sont différentes selon leur type : les cellules de type T_1 sont très sensibles au gradient C_1 mais ne répondent pas du tout au gradient C_2 ; les cellules de type T_2 répondent aux gradients C_1 et C_2, mais dans des proportions moindres pour ce dernier. Nous retrouvons dans ce système les mêmes relations entre les intensités d'attraction au gradient qu'avec les valeurs d'adhésion différentielle. La figure 2.19 montre un exemple de simulation réalisée avec ce modèle.

Génération de créatures artificielles. Ces travaux sont plus proches du domaine de la vie artificielle que du domaine de la biologie synthétique. Le modèle Cell2Organ proposé par Cussat-Blanc et coll. (2008) est utilisé afin de générer des créatures artificielles à partir d'une unique cellule. L'organisation des cellules entre elles permet de former des groupes de cellules accomplissant une ou des fonctions spécifiques à ces groupes. Autrement dit, ces groupes sont des formes d'organes. Les cellules, des disques, évoluent dans un environnement discret. Elles sont pourvues de capteurs leur permettant de détecter les concentrations de gradients chimiques présents dans leur environnement. Différentes actions peuvent être entreprises par les cellules, comme la transformation de substrat, la duplication ou encore l'apoptose. Un système de sélection d'actions permet aux cellules de choisir une action à entreprendre. Chaque cellule dispose de son réseau de régulation génétique, lui permettant de se spécialiser lorsqu'elle accomplie une division cellulaire.



(a) (b) (c)

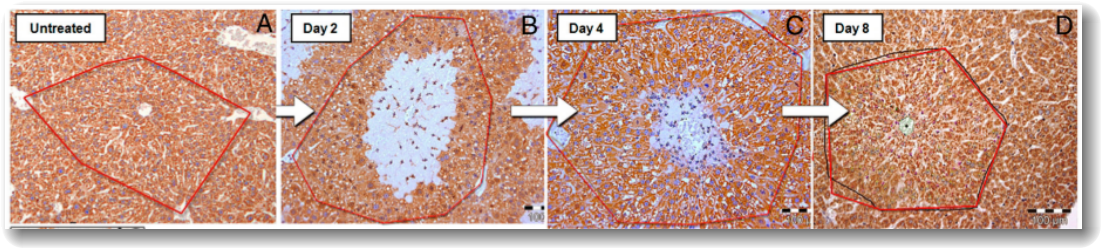
FIGURE 2.20 – Génération de créature artificielle avec le modèle Cell2Organ proposé par Cussat-Blanc et coll.. La forme est pré-définie. Les cellules évoluent grâce à un algorithme génétique qui leur permet de créer la forme désirée. Figures extraites de (Cussat-Blanc et coll., 2008).

Il est par exemple possible, avec ce modèle, de générer des créatures à partir d'une forme pré-définie par l'utilisateur, comme illustré sur la figure 2.20. Des morphogènes diffusent dans l'environnement, les cellules se divisent dans les directions des gradients grâce à leurs capteurs et à leur système de sélection d'actions. Un algorithme génétique permet de générer la créature voulue en faisant évoluer le génome caractérisant l'organisme.

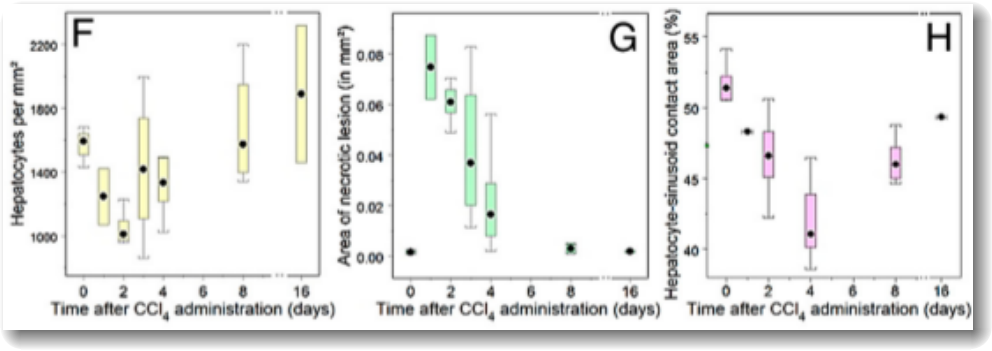
Étude de la régénération du foie. Hoehme et Drasdo (2010) proposent un modèle de cellule basée sur des sphères élastiques pour la modélisation et la simulation de tissus. Ce modèle a par exemple été utilisé pour étudier la régénération du foie (Hoehme et coll., 2010). Le tissu hépatique est constitué de cellules hépatocytes assurant des fonctions métaboliques et de cellules sinusoïdales formant des micro-vaisseaux. Les hépatocytes sont regroupées en lobules. La structure de ces lobules est composée d'hépatocytes en contact direct avec les sinusoïdes, assurant des échanges optimaux de métabolites entre les cellules et le sang. L'objet de l'étude de Hoehme et coll. est d'étudier la régénération du foie chez la souris suite à une intoxication ayant causé la mort des cellules hépatocytes proches de la veine centrale du lobule. La figure 2.21a montre des lobules visualisées en microscopie. En particulier, on observe la mort des cellules hépacocytes centrales après intoxication, ainsi que la régénération du tissu les jours suivants. Lors de la régénération, les cellules hépatocytes issues des divisions cellulaires s'alignent le long des cellules sinusoïdes : en effet, cette micro-architecture permet des échanges optimaux entre les cellules et le sang. Les simulations réalisées sont corrélées aux données expérimentales lorsque le mécanisme d'alignement est utilisé dans le modèle (modèle 3 sur les figures 2.21c et 2.21d) ; les auteurs ont émis l'hypothèse que seul ce mécanisme permet de retrouver la micro-architecture caractéristique des lobules de foie sain. Quant à la cause de cet alignement, les auteurs émettent l'hypothèse que les cellules sinusoïdales émettent des facteurs de croissance servant aussi de chimioattractants permettant l'alignement des cellules ; hypothèse restant à vérifier expérimentalement.

c) Modèles basés sur une cellule de forme complexe

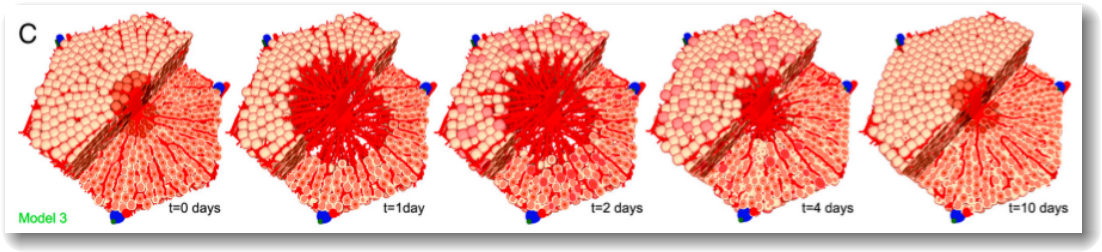
Les formes que nous appelons « complexes » sont les formes de cellules ne se retrouvant pas dans les catégories ci-dessus. Dans le cas présent, les formes de cellules peuvent être construites de manières très variées.



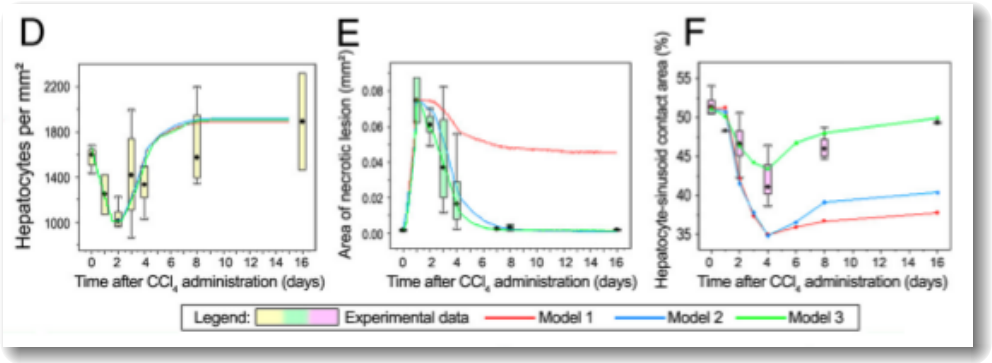
(a)



(b)

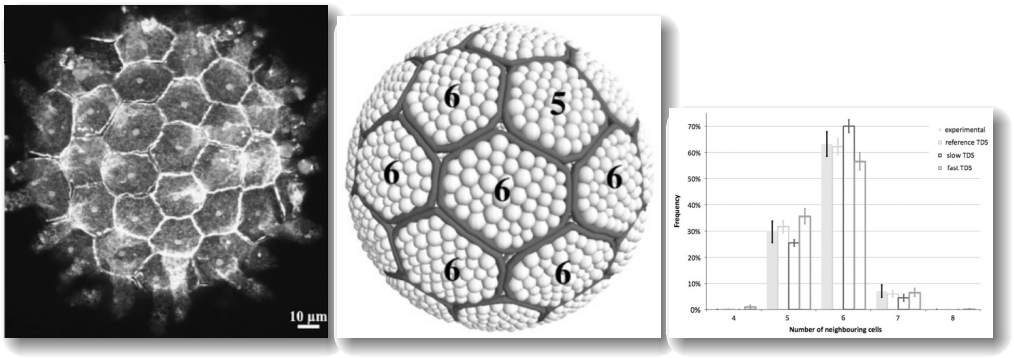


(c)



(d)

Figure 2.21 – Régénération de foie de souris après intoxication. (a) Visualisation au microscope de lobules. Après mort des cellules hépatocytes centrales, le tissu hépatique se régénère. (b) Analyse quantitative de la régénération. (c) Visualisation de la régénération simulée avec le modèle de Hoehme et coll.. (d) Analyse quantitative des simulations comparée avec les données expérimentales. Figures extraites de (Hoehme et coll., 2010).



(a) **(b)** **(c)**

FIGURE 2.22 – Modélisation et simulation d'ovocyte de *Ciona Intestinalis* avec un modèle de cellules à grains. (a) Vue en microscopie de la topologie des cellules folliculaires composant la surface de l'ovocyte. (b) Simulation d'ovocyte. (c) Analyse quantitative du modèle. Figures extraites de (Chélin et coll., 2013).

Simulation de tissu épithélial avec un modèle de cellules à grains. Chélin et coll. (2013) proposent un modèle de cellule dont la structure est composée de grains de deux types : le premier permet de modéliser la membrane de la cellule tandis que le second permet de modéliser la structure interne de la cellule. Les grains sont reliés entre eux par des liens élastiques et la forme de la cellule est flexible. La croissance des cellules est modélisée par l'ajout de grains dans la cellule. Ce modèle a servi à étudier l'organisation de tissu épithélial en fonction de la vitesse de développement du tissu. Le modèle biologique étudié est un ovocyte de *Ciona Intestinalis* qui présente sur sa surface une couche d'une soixantaine de cellules folliculaires (figure 2.22a). La vitesse de développement du tissu correspond à la prise en compte de trois grandeurs : la vitesse d'arrivée des cellules dans le tissu, la vitesse de croissance de l'ovocyte (l'augmentation de son diamètre) et la vitesse de croissance des cellules. La conclusion de l'étude est que la vitesse de développement a une incidence sur la topologie du tissu : les cellules tendent à avoir un plus grand nombre de voisines lorsque la vitesse de développement est rapide. Les figures 2.22b et 2.22c présentent un exemple d'ovocyte simulé avec le modèle ainsi que l'analyse quantitative des simulations.

Modélisation 3D d'épiderme. Christley et coll. (2010) utilisent un modèle de cellule basé sur l'utilisation « d'éléments sous-cellulaires », initialement proposé par Newman (2005). Dans ce modèle, les cellules sont représentées par un ensemble de points en interactions, modélisant la structure interne de la cellule, comme illustré sur la figure 2.23. Le modèle de cellule comprend un réseau de régulation génétique permettant en particulier de modéliser les interactions des cellules entre elles et avec leur environnement. Ce réseau est modélisé avec un système d'équations différentielles. La croissance des cellules est réalisée en ajoutant de nouveaux sous-éléments ; leur division est possible si elles contiennent suffisamment de sous-éléments. Christley et coll. ont utilisé ce modèle pour modéliser le développement d'épiderme en trois dimensions. L'article étant orienté sur la méthodologie (construction de modèle en biologie intégrative et performance des implémentations des modèles), les auteurs ne fournissent pas d'élément de validation de leur modèle.

Migration cellulaire sur un substrat avec un modèle biomécanique de cellule. Ballet et Tracqui (2007) proposent quant à eux un modèle de cellule déformable basé sur un

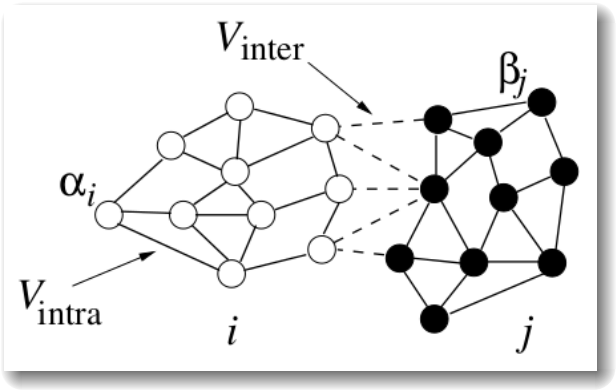


Figure 2.23 – Représentation de cellules avec des éléments sous-cellulaire. La cellule i (resp. j) est constituée des élements représentés en blanc (resp. noir). Les interactions intra-cellulaires sont représentées avec les lignes solides, alors que les interactions cellules/cellules sont représentées avec les lignes pointillées. Figure extraite de (Newman, 2005).

système masses / ressorts. Topologiquement, les cellules sont représentées par des polygones dont les sommets représentent des nœuds (les masses) dans la cellule, qui sont reliés par des liens élastiques permettant de représenter à la fois la membrane des cellules, ainsi que des simplifications de cortex cellulaire et de cytosquelette. La forme de ces cellules est très similaire à celle que l'on retrouve dans les modèles à vertices. En dehors du fait que le polygone est construit avec un système masses / ressorts, la différence principale est que les sommets des polygones ne sont pas partagés entre les cellules : celles-ci sont physiquement totalement indépendantes les unes des autres. Ce modèle a été utilisé pour modéliser la migration cellulaire sur un substrat, tel qu'illustré sur la figure 2.24. Les résultats présentés sur la figure 2.24b sont mis en corrélation avec les données expérimentales issues d'une étude antérieure (Palecek et coll., 1997), ayant montré que les vitesses de migration des cellules varient en fonction de la densité du substrat sur lequel les cellules sont situées (figure 2.24c).

3.4 Synthèse sur la modélisation discrète

À travers ces exemples de modèles discrets, nous avons aperçu l'étendue de l'expressivité de ce type de modélisation. Les systèmes simulés sont variés, les formalismes permettent une liberté importante quant à la manière de modéliser un système. En particulier, les approches individus-centrés de type systèmes multi-agents n'imposent pas de contraintes sur la forme des cellules par exemple, contrairement à un automate cellulaire où une cellule est une case de l'environnement (sauf dans le cas du modèle de Potts cellulaire puisque dans ce cas, les cellules sont étendues sur plusieurs sites de la grille). À la lecture de ces éléments, il apparaît que les formalismes discrets présentés sont proches conceptuellement. Dans tous les cas, les modèles permettent une définition fine des entités du système, de leurs comportements ainsi que des interactions locales de ces entités. La différence majeure se situe notamment dans l'environnement (discret dans le cas d'un automate cellulaire, discret ou continu dans le cas d'un système multi-agents), ainsi que dans la forme des cellules.

Cependant, l'utilisation dans un modèle du seul formalisme discret réduirait l'expressivité du modèle. Le couplage de ce formalisme avec un formalisme continu permet de prendre en compte une palette de comportements plus large ; cette approche « hybride » est l'objet de la section suivante.

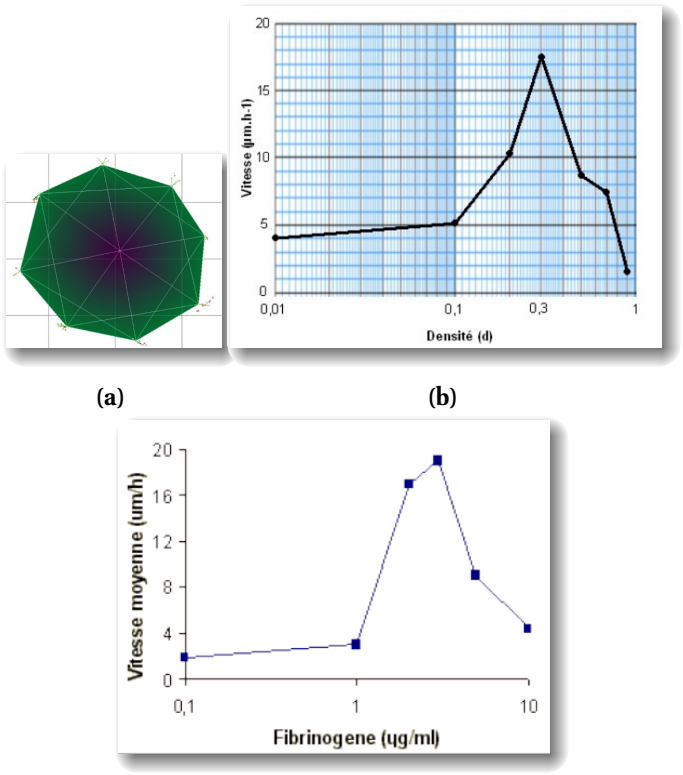


FIGURE 2.24 – Modèle biomécanique de cellule pour l'étude de la migration cellulaire sur un substrat. (a) La structure de la cellule est construite avec un polygone. Les sommets (des masses) du polygone sont reliés par des liens élastiques. Ces liens permettent de modéliser la membrane de la cellule d'une part et une simplification de cortex et de cytosquelette d'autre part. (b) Vitesse de déplacement en simulation sur un substrat de densité variable. (c) Données expérimentales : vitesse de déplacement d'une cellule sur un substrat de densité variable. Figures extraites de (Ballet et Tracqui, 2007).

4 Approche hybride

La plupart des phénomènes biologiques que l'on pourrait vouloir étudier requièrent l'utilisation de modèles hybrides. Les modèles discrets présentés dans la partie précédente sont pour la plupart couplés à des éléments continus afin de prendre en compte des éléments tels que la diffusion de molécules. Même si une approche discrète pourrait permettre de modéliser la diffusion d'une molécule, le nombre d'entités en présence — par rapport au nombre de cellules par exemple — serait bien trop important, et rendrait la simulation du modèle difficile, voire impossible. L'utilisation d'équations de réaction-diffusion autorise la considération des concentrations de molécules plutôt que les molécules elles-même. Cela permet de ramener au même niveau les différents éléments du modèle, à savoir les cellules et les molécules : on passe ainsi d'un modèle de molécules au niveau microscopique à un modèle de molécules au niveau macroscopique.

La plupart des modèles présentés dans la section précédente sont en fait des modèles hybrides, bien que nous n'ayons présentés que les niveaux cellulaires de ces modèles : l'utilisation d'un seul formalisme de modélisation est trop contraignante et compte tenu des avantages et inconvénients des différents formalismes ainsi que leur expressivité, dans le cas de la modélisation de morphogenèse de tissus, l'approche hybride permet de tirer parti des avantages de chaque formalisme. Le seul modèle présenté qui n'est pas un modèle hybride est le CPM proposé par Graner et Glazier, utilisé pour simuler le tri cellulaire basé sur l'hypothèse de l'adhésion différentielle : les adhésions cellulaires, bien qu'ayant lieu grâce à des protéines exprimées à la surface des membranes des cellules, sont modélisées au niveau cellulaire et non pas au niveau moléculaire. On pourrait toutefois vouloir modéliser l'adhésion à un niveau plus bas, par exemple dans l'optique d'étudier comment les défauts d'expressions des protéines impliquées dans l'adhésion affectent celle-ci : encore une fois, le niveau auquel on place la modélisation dépend du phénomène que l'on souhaite étudier. De façon générale, il semble difficile de prendre en compte des phénomènes multi-échelles spatiales ou temporelles avec un formalisme unique. Comme expliqué ci-dessus, les échelles sous-cellulaires peuvent être judicieusement prises en compte avec le couplage du modèle principal à des sous-modèles continus. Quant aux niveaux sur-cellulaires, c'est-à-dire les tissus et les organes, ils ne sont pas modélisés explicitement mais émergent des interactions se produisant au niveau cellulaire.

Pour illustrer ceci, reprenons un exemple que nous avons déjà évoqué, mais dont nous n'avions pas détaillé entièrement le modèle : CompuCell3D (Chaturvedi et coll., 2005) — paragraphe « Modélisation de la croissance de membres chez les vertébrés » page 37 — est un exemple de modèle hybride. Nous avons déjà présenté le formalisme utilisé au cœur de cette architecture : il s'agit du modèle de Potts cellulaire. Ce formalisme est utilisé pour représenter le niveau cellulaire ainsi que les niveaux tissus et organes, mais le modèle global contient également d'autres éléments pour prendre en compte différents niveaux inhérents à la morphogenèse de tissus. Des équations de réaction-diffusion, pour le niveau moléculaire, permettent de modéliser la diffusion de facteurs de croissance. Un automate à états, pour le niveau sous-cellulaire, permet de prendre en compte la régulation génétique, en particulier les capacités de différentiation des cellules. La figure 2.25 synthétise les différents niveaux biologiques que l'on retrouve dans la morphogenèse de tissus, ainsi que les formalismes adoptés par Chaturvedi et coll. pour modéliser ces différents niveaux.

Nous n'allons pas décrire ici en détail chaque modèle hybride que nous avons trouvé dans la littérature. Il est cependant important de savoir que quasiment tous les travaux que nous avons présentés dans ce chapitre couplent le formalisme principal de leur modèle à

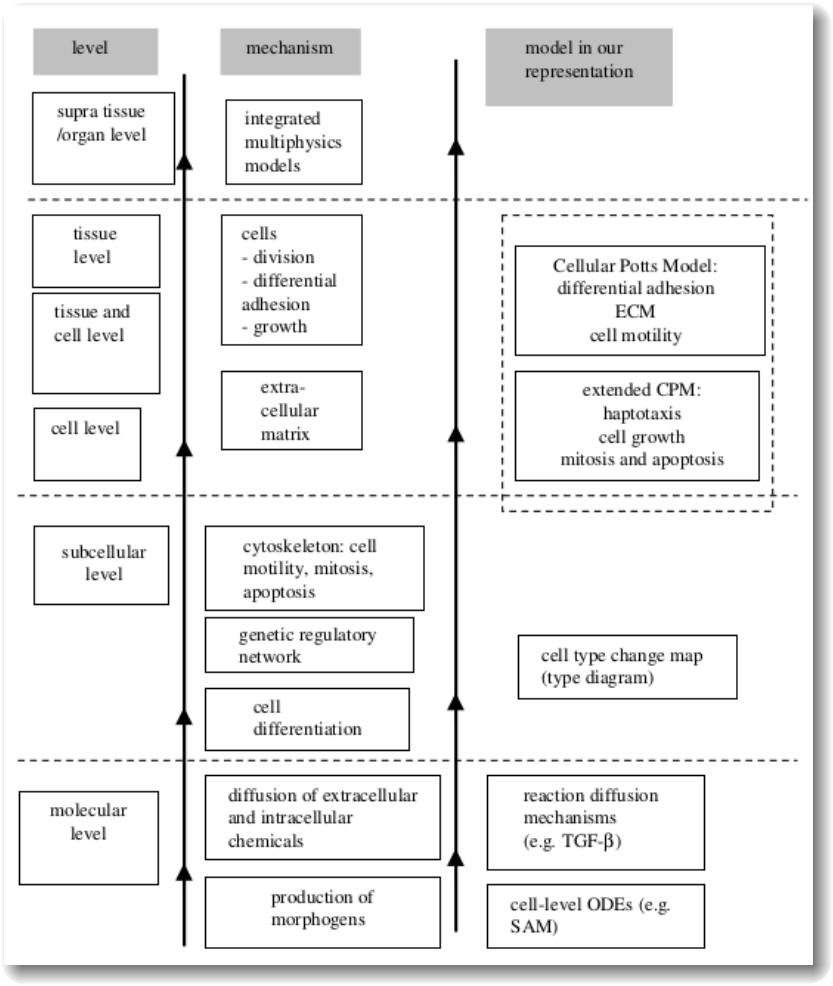


FIGURE 2.25 – Niveaux biologiques dans la morphogenèse de tissus. Ces différents niveaux sont pris en compte par différents modèles couplés : les différents sous-modèles sont ceux utilisés dans CompuCell3D (Chaturvedi et coll., 2005), mais la plupart des modèles créés pour simuler la morphogenèse de tissus utilisent de manière analogue des sous-modèles. Figure extraite de (Chaturvedi et coll., 2005).

un modèle continu, en particulier lorsqu'il s'agit de modéliser la réaction et la diffusion de molécules dans l'environnement. Même si le couplage de différents formalismes s'applique avant tout afin de pouvoir prendre en compte, au moins dans une certaine mesure, des échelles temporelles et spatiales différentes, des équations peuvent aussi être utilisées au même niveau que le formalisme principal, par exemple pour modéliser les mouvements des cellules. Il y a donc une flexibilité importante dans le couplage de formalismes de modélisation.

La section suivante permet de récapituler les différents éléments étudiés dans ce chapitre.

5 Synthèse du chapitre

5.1 Éléments de comparaison des approches continues et discrètes

L'objet de ce chapitre est de confronter les grandes catégories de modèles étudiés : les modèles continus et les modèles discrets. Les avantages et inconvénients de chaque approche (et plus précisément, de chaque type de modèle) doivent nous permettre de choisir le formalisme principal qui sera le cœur du modèle que nous proposons. Les approches continues et discrètes peuvent être comparées sur la base de six différents aspects.

Prise en compte des individus. Nous avons vu au début du chapitre que les approches continues, qu'elles soient basées sur des équations différentielles ordinaires ou des équations aux dérivées partielles, permettent de considérer les populations des entités du système de manière globale. Ce sont les évolutions des *populations* que l'on modélise dans un tel système. En outre, les populations sont traitées de manière continues, ce qui peut poser des problèmes de pertinence (car on peut théoriquement considérer des fractions d'entités) tout comme cela peut conduire à des aberrations lors de la résolution des équations. Pour illustrer ce problème, reprenons le simple problème proies / prédateurs dont l'évolution est caractérisée par le système d'équation 2.1. Modifions le paramètre α, qui modélise le taux de reproduction des proies : $\alpha = 0.1$ au lieu de $\alpha = 1.5$ précédemment. On a donc un taux de reproduction considérablement plus faible que précédemment. L'évolution du système est visible sur la figure 2.26. Le nouveau jeu de paramètres fait que l'on se trouve avec une population de prédateurs qui devrait s'éteindre suite à la forte baisse de la population de proies, dont le taux de reproduction est devenu très faible. Cependant, la population de prédateurs va baisser jusqu'à environ 9.9×10^{-20}, autant dire zéro. Mais le système étant continu, la population de prédateurs augmente à nouveau une fois que la population de proies redevient suffisante. Ce n'est pas un phénomène qui aurait été observé avec une approche où les entités proies et prédateurs seraient modélisées de manière discrète car dans ce cas, il est impossible d'avoir des « fractions » d'entités.

Prise en compte des interactions locales. Le second point concerne la possibilité de modéliser des interactions locales. Étant donné que les équations différentielles ordinaires permettent uniquement de modéliser l'évolution d'un système dans le temps, la notion d'interactions locales n'est pas présente dans ce cas. Les équations aux dérivées partielles par contre permettent de prendre en compte des interactions locales dans une certaine mesure. En effet, nous avons vu à travers le modèle proposé par Sherratt et Chaplain (2001) par exemple, qu'un des termes de l'équation que nous avons décrite permettaient de considérer l'impact des interactions des cellules sur la motilité de celles-ci. Il paraît toutefois évident que seul le fin niveau de granularité offert par les approches discrètes permet de modéliser

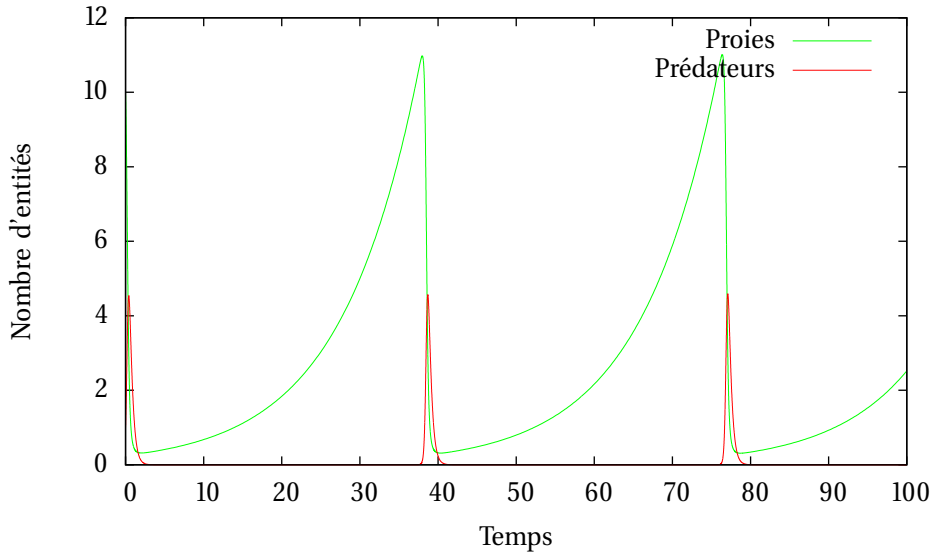


FIGURE 2.26 – Système proies / prédateurs : résolution numérique du système d'équation 2.1. Le taux de reproduction des proies est cette fois très faible : $\alpha = 0.1$ au lieu de $\alpha = 1.5$ précédemment.

aussi finement que voulu des interactions locales, comme on a pu le voir avec les systèmes multi-agents et les automates cellulaires. En effet, les voisinages des entités permettent de déterminer l'état des entités aux pas de temps suivants. Il en va de même pour les automates de gaz sur réseau et le modèle de Potts cellulaire.

Modularité des approches. La capacité d'ajouter des éléments au système n'est pas du tout la même selon que celui-ci soit modélisé avec des équations ou avec des composants discrets. En effet, si dans le cas d'un système discret (et en particulier d'un système multi-agents), l'ajout d'entités, de types d'entité, de comportements ou d'interactions, est très aisé, il n'en va pas de même avec les systèmes d'équations. Dans ce cas là, il est nécessaire d'ajouter de nouvelles équations au système ou de modifier, parfois de manière importante, les équations existantes afin de prendre en compte ces nouveaux éléments. Dans le cas d'un modèle discret, il suffit d'ajouter la description de l'élément au système, cela ne nécessitant pas (ou que très peu) de modification du reste du système.

Aspects quantitatifs et qualitatifs ; validation des modèles. Les aspects quantitatifs et qualitatifs peuvent être observés tant en entrée du modèle qu'en sortie. En entrée des modèles, on qualifie de « quantitatifs » des paramètres numériques utilisés dans le modèle. Les éléments dit « qualitatifs » peuvent sembler plus subjectifs, et dans notre cas, peuvent par exemple correspondre à des comportements attendus des entités ; ces comportements pouvant être basés sur du raisonnement (on reviendra sur cette notion de raisonnement dans le chapitre 3, page 63). Pour reprendre l'exemple du système proies / prédateurs, les paramètres présents dans le système d'équations sont des paramètres quantitatifs. Une modélisation discrète du phénomène permettrait de modéliser différentes techniques de chasse de la part des prédateurs : ce sont des éléments qualitatifs. Les approches discrètes permettent d'incorporer avec aisance tant des paramètres quantitatifs que qualitatifs. Ces derniers sont plus délicats à prendre en compte dans le cas d'un modèle continu. Le caractère qualitatif d'un système peut également être observé en sortie du modèle : par exemple dans le cas du tri cellulaire,

l'observation des agrégats de cellules permet de dire si le tri est ou non effectué. Les comportements d'entités en essaims ou encore de foules sont également qualitatifs. Ces sorties peuvent, dans certains cas, être quantifiées également, comme nous l'avons vu au long de la section 3. Ceci nous amène à la validation des modèles. Si les mathématiques offrent des outils pour analyser les modèles construits sur des équations différentielles (stabilité des solutions du système, comportements stationnaires, etc.), les modèles discrets n'offrent pas moins de pertinence. Pour commencer, ceux-ci intègrent en effet des paramètres issus de données expérimentales — les données peuvent être erronées, mais dans ce cas on sort de la portée de la présente thèse. Si aucune donnée n'est disponible, alors, et c'est vrai tant dans le cas discret que continu, les modélisateurs estiment les paramètres de leur modèle. Celui-ci est ensuite simulé et les paramètres peuvent être raffinés de simulations en simulations afin d'obtenir les résultats voulus. Nous avons vu, au travers des différents exemples de modélisation discrète présentés, que le plus souvent, les auteurs proposent une analyse quantitative fine de leur modèle afin de le valider. Ces analyses varient : il peut s'agir de mesurer la taille d'une tumeur virtuelle ou d'un tissu lors de son développement (Jiao et Torquato, 2011; Tektonidis et coll., 2011; Hoehme et coll., 2010), de calculer des voisinages selon des types cellulaires (Belmonte et coll., 2008; Hoehme et coll., 2010) ou encore de calculer des vitesses de migration de cellule sur un substrat (Ballet et Tracqui, 2007). Comparées à des données expérimentales, ces analyses montrent toute la pertinence des modèles proposés.

Contrôlabilité des modèles. La notion de « contrôlabilité » fait entrer en jeu différents éléments. Le premier de ces éléments concerne la lisibilité et la compréhension du modèle : si de manière conceptuelle un système multi-agents paraît relativement simple, il n'en va pas de même de son formalisme sous forme de programme. Ce dernier peut compter des milliers de lignes de code. Sa lisibilité sera difficile, tout comme sa compréhension, même si l'on a une expérience de la programmation. Ce problème est moins marqué avec les systèmes d'équations, dont le formalisme mathématique permet de faciliter la lecture du système, quand bien même celui-ci compterait des dizaines d'équations. En second lieu, il convient de réfléchir à l'impact des paramètres dans ces modèles respectifs : un système multi-agents comporte par nature un nombre important de degrés de liberté. En outre, son implémentation peut rendre compliquée la détermination de l'impact de paramètres, à cause de la complexité du code (par exemple avec les branchements conditionnels). Les multiples formalismes des SMA complexifient la contrôlabilité du modèle : algorithmes, langage utilisé pour implémenter le SMA, précision de la machine exécutant le SMA, etc. sont autant d'éléments qui complexifient ce type de modèle. En ce qui concerne les systèmes d'équations, on dispose d'outils mathématiques permettant potentiellement d'analyser le système, notamment si l'on peut déterminer des solutions analytiques. Cependant, dans le cas de systèmes complexes, il devient rare de pouvoir déterminer de telles solutions. On doit alors discrétiser le problème pour le simuler numériquement et l'avantage de la contrôlabilité est en partie perdu. Finalement, les frontières sont assez floues entre ces deux grand types de modèles. Il est toutefois important de garder ces éléments à l'esprit lors de la conception d'un modèle et de son implémentation.

Caractère exploratoire des simulations. Un intérêt majeur de la simulation numérique est son caractère exploratoire. Les modèles, une fois construits, peuvent être simulés à volonté — contrairement à la réalisation d'expériences *in vitro* dont le coût et parfois le temps ne permettent pas de réaliser autant de manipulations que voulu. Du point de vue de l'informatique, nous pouvons exprimer cette notion d'exploration à travers l'expression *laboratoire*

TABLE 2.2 – Élements de comparaison des approches continues et discrètes.

	Modèles continus	Modèles discrets
Prise en compte des individus	−	+ + +
Interactions locales	+	+ + +
Aspects qualitatifs / quantitatifs	++	++
Validation des modèles	++	++
Contrôlabilité des modèles	+ + +	−
Caractère exploratoire des simulations	++	+ + +

virtuel. De la même façon que dans la réalité, le modélisateur et l'expérimentateur peuvent, sur la base de leurs observations, déduire ou raffiner des modèles, jouer sur les paramètres d'un système, ou même, comme nous l'avons également vu au long de ce chapitre, émettre des hypothèses sur la nature d'un phénomène observé (stratégies de soin lors du traitement de cancers (de Pillis et coll., 2006) ; vitesse de développement de tumeurs en fonction de la présence ou non de tissus environnants (Sherratt et Chaplain, 2001) ; importance de la motilité cellulaire et de la densité de la matrice extra-cellulaire lors du développement de tumeurs (Jiao et Torquato, 2011) ; influence de la densité de cellules sur leur changement de phénotype (Tektonidis et coll., 2011) ; suffisance de la motilité et de l'adhésion cellulaire pour obtenir un tissu trié (Graner et Glazier, 1992) ; formation de la tige lors du développement du *Dictyostelium Discoideum* (Marée et Hogeweg, 2001) ; influence de l'orientation des divisions cellulaires dans la topologie d'un tissu (Li et coll., 2012) ; influence de l'alignement des cellules pour l'obtention d'un tissu trié (Belmonte et coll., 2008) ; alignement de cellules de foie le long de vaisseaux sanguins pour maximiser les échanges dans le foie (Hoehme et coll., 2010)). Il est évident que ces hypothèses ne sont que des hypothèses. Cependant, la relative facilité avec laquelle il est possible de jouer avec le modèle — en particulier dans le cas d'un modèle discret — permet d'émettre de telles hypothèses qui auront vocation à être mises à l'épreuve lors de tests *in vitro* ou *in vivo*. La flexibilité d'un système d'équations est toutefois moindre, comme on l'a vu dans le paragraphe « Modularité des approches » (page 54), ce qui réduit donc la capacité exploratoire de ce type de modèle.
Un récapitulatif des éléments présentés ci-dessus est donné dans le tableau 2.2.

5.2 Conclusions

Nous considérons important de prendre en compte les éléments du système et en particulier leurs interactions, de manière individuelle, afin de capturer des comportements globaux émergeant. L'aspect exploratoire de ce type d'approche permet en particulier de construire des modèles et des hypothèses de manière itérative, grâce aux raffinements successifs de ces modèles et hypothèses à travers des expérimentations numériques permettant de valider ou d'invalider des éléments et de proposer de nouvelles idées sur la base des observations réalisées. Hormis l'aspect local, ces éléments se retrouvent, à différents niveaux, tant dans les approches continues que discrètes. Les éléments de comparaison présentés ci-dessus nous ont orienté vers l'utilisation d'un formalisme principal de nature discrète.

Toutefois, nous avons aussi montré la nécessité de se tourner vers des approches hybrides, c'est-à-dire mêlant des éléments continus et discrets, afin de pouvoir exprimer certains comportements qu'il serait difficile d'exprimer avec un formalisme unique. Un exemple illustrant cet aspect concerne la diffusion de molécules dans un système biologique : une approche individus-centrée permettrait de modéliser ce phénomène, cependant, lorsque l'on parle de

millions de molécules interagissant dans un environnement contenant ne serait-ce qu'une centaine de cellules, on imagine rapidement les difficultés qui vont se poser au moment de simuler le modèle. Une approche continue permettant d'exprimer des concentrations de molécules ainsi que leurs interactions (réaction, diffusion) permet de modéliser la réaction et la diffusion des molécules en se plaçant au même niveau de modélisation que celui des cellules. Des éléments discrets peuvent également se mêler à des éléments continus à d'autres niveaux, comme par exemple pour les mouvements des entités qui, si elles sont modélisées individuellement, peuvent utiliser des comportements modélisés par une approche continue. Nous verrons au cours du chapitre 3 comment ces éléments peuvent être couplés au sein d'un même modèle.

Considérant, encore une fois, que nous nous intéressons à la modélisation et à la simulation de morphogenèse de tissus et partant des éléments que nous avons présentés au long de ce chapitre, nous avons décidé de nous placer au niveau de la cellule (approche *mésoscopique*) en nous basant sur une représentation discrète de celle-ci et en modélisant explicitement les comportements cellulaires et les interactions des cellules. Le couplage de ce modèle discret à un modèle continu, permettant de modéliser l'aspect chimique du système, fait gagner de l'expressivité à notre modèle. Le formalisme discret que nous avons choisi d'adopter est celui des systèmes multi-agents. En effet, ce formalisme nous a semblé être le plus souple parmi les formalismes discrets étudiés : l'environnement du système peut être tant discret que continu et la forme des cellules est totalement libre.

L'objet du prochain chapitre, après une présentation plus détaillée de l'approche *mésoscopique* et du formalisme multi-agents, sera d'introduire le modèle de cellule virtuelle que nous proposons dans cette thèse. Nous avons choisi comme point de départ de modélisation la structure de cellule proposée par Ballet et Tracqui (2007). Notre cellule virtuelle est basée sur une forme complexe de cellule, composée de nœuds reliés par des ressorts. En plus d'être déformable, la structure de notre cellule comprend aussi des simplifications de la structure interne d'une cellule biologique. Le modèle inclut également divers comportements cellulaires ainsi qu'une chimie artificielle.

CHAPITRE 3

Proposition : modèle de cellule virtuelle

Ce chapitre a pour objet de présenter le modèle biomécanique de cellule virtuelle que nous proposons. Notre modélisation est centrée sur la cellule et notre modèle comprend les comportements principaux des cellules : croissance, division cellulaire, interactions entre cellules et entre cellules et environnement. Nous avons également modélisé des éléments de mécanotransduction, permettant à nos cellules virtuelles d'évaluer les contraintes physiques auxquelles elles sont soumises afin qu'elles puissent adapter leur comportement en conséquence. Une chimie artificielle simplifiée est également présente dans notre modèle. Nous avons couplé notre modèle à un système multi-agents afin de bénéficier de la flexibilité inhérente aux systèmes multi-agents.

Sommaire

Des éléments de ce chapitre on fait l'objet de deux publications (Jeannin-Girardon et coll., 2013b, c).

1 Choix du formalisme de modélisation

1.1 Approche centrée sur la cellule

La biologie humaine est constituée de nombreuses échelles spatiales et temporelles. Ces échelles peuvent être abordées via différents formalismes, comme nous l'avons vu au chapitre 2. La proposition d'un modèle permettant d'étudier un système ne requiert pas de partir d'une extrémité ou d'une autre de l'échelle : on peut choisir un point de départ intermédiaire. Dans le cas de la modélisation et de la simulation de morphogenèse de tissus — phénomène émergeant d'interactions locales entre les cellules composant le tissu —, la cellule biologique semble être une échelle intermédiaire adaptée : en effet, une cellule sait interpréter et utiliser à la fois des informations génétiques et environnementales ; elle peut également créer des signaux (donc communiquer, interagir) avec des portées plus ou moins importantes. Les tissus résultent de ces interactions qui peuvent donc être directes (cellules / cellules) ou indirectes (cellules / matrice extra-cellulaire). En outre, une dérégulation se produisant au niveau cellulaire peut avoir des répercussions au niveau macroscopique, comme dans le cas de cancers par exemple.

Une approche centrée sur la cellule — on parlera dans ce document d'approche *mésoscopique*, échelle située entre les échelles microscopique et macroscopique — peut être

utilisée indifféremment avec un formalisme de modélisation continu ou discret, selon les besoins du modèle. Cependant, compte tenu de l'importance à la fois des contraintes spatiales et des interactions locales (cellules / cellules ou cellules / matrice extra-cellulaire) lors du développement de tissus, l'utilisation d'un formalisme discret semble plus adapté car plus pertinent pour capturer les interactions locales.

Le niveau de modélisation ainsi que le niveau de détail du modèle dépend de ce que l'on souhaite modéliser. Des modèles très précis tel que Virtual Cell (Moraru et coll., 2008), à cause de leurs complexités intrinsèques, ne peuvent que difficilement être utilisés pour modéliser un tissu qui compte un nombre important de cellules. Dans le cas où l'on souhaite étudier le développement de tissus cellulaires, est-il nécessaire de modéliser les mécanismes intra-cellulaires menant par exemple à une division cellulaire ou pouvons-nous modéliser ce comportement à un niveau d'abstraction plus élevé ? Merks et Glazier (2005) semblent penser que c'est le cas : l'idée est alors de considérer les comportements cellulaires comme des boîtes noires et de les modéliser à plus haut niveau.

Le cas totalement opposé à la modélisation fine de chaque comportement serait de modéliser toute la population composant le tissu de manière globale, en considérant, à la manière du système proies / prédateurs modélisé par des équations différentielles ordinaires tel que présenté au chapitre précédent, les comportements cellulaires comme des taux (taux de division par exemple). Compte tenue de l'importance des interactions locales dans la formation de tissus cellulaires, il serait dommage de ne pas considérer ces interactions individuellement afin de permettre une modélisation fine du système.

Se placer au niveau de la cellule biologique, à l'aide d'une approche individu-centrée, semble ainsi adéquat pour construire un modèle permettant d'étudier le développement de tissus : ce niveau permet de capturer, sans les détailler, les interactions et comportements individuels des cellules. La prise en compte d'échelles sous-cellulaires reste possible grâce au couplage du modèle principal à un ou des sous-modèles permettant par exemple de prendre en compte la diffusion de molécules dans l'environnement.

La section suivante présente plus en détail le formalisme individus-centré et en particulier les systèmes multi-agents. On verra en particulier que ce formalisme est cohérent avec la notion d'approche mésoscopique et que les aspects à la fois qualitatifs et quantitatifs de ces systèmes permettent de proposer des modèles cohérents avec les systèmes réels que l'on souhaite modéliser.

1.2 Présentation du formalisme multi-agents

a) Intérêt des systèmes multi-agents

Le choix d'un paradigme de modélisation dépend de ce que l'on souhaite étudier et des résultats que l'on souhaite obtenir. Pour reprendre l'exemple des systèmes proies / prédateurs présenté au chapitre 2, dans lequel on a montré comment modéliser le système par un système d'équations différentielles, on observe en simulant ce modèle un résultat auquel on peut intuitivement s'attendre : des oscillations dans les populations. En effet, les prédateurs vont pouvoir fortement proliférer tant qu'il y a des proies, leur prédation faisant baisser la population. Lorsqu'il n'y a plus suffisamment de proies, la population des prédateurs décline et la population des proies, étant moins soumise à la prédation, augmente. Ces événements se répètent en formant un cycle.

Ce comportement global n'est pas nécessairement le seul que l'on puisse capturer. En outre, on peut vouloir ajouter d'autres éléments dans le système, pour capturer d'autres phénomènes observés : variation du taux de prédation au cours du temps ou des temps de

gestation pour chaque espèce. Dans le cas d'un système d'équations, une façon de prendre en compte ce type de phénomènes est d'inclure de la stochasticité, pouvant permettre de considérer des aspects aléatoires du système réel. Cependant, en plus de rajouter des paramètres au système, on peut se demander quel est le sens de ces paramètres par rapport au système réel. Même si, par définition, un modèle est une approximation ou une simplification d'un système réel, ses paramètres sont d'autant plus utiles qu'ils gardent du sens par rapport au système réel.

Lorsque l'on adopte comme formalisme de modélisation un système à base d'équations, d'autres contraintes se posent : tout d'abord, on ne peut modéliser et observer que le comportement global du système. Les paramètres des équations ne prennent pas ou peu en compte les interactions locales des composants du système. Les comportements individuels ne sont pas pris en compte et, comme nous l'avons vu dans la synthèse du chapitre 2, les aspects qualitatifs sont difficilement pris en compte par les approches continues.

Les systèmes multi-agents (SMA) sont apparus à cause, entre autres, du besoin de simuler à un niveau de granularité plus fin que les modèles à base d'équations (MBE). Un SMA n'est pas seulement la modélisation des individus et de leurs comportements, mais aussi des interactions entre les entités du système. Contrairement aux MBE, on ne définit pas le comportement du système de manière globale. Ce comportement global, au contraire, émerge des interactions locales définies entre les entités : c'est donc le résultat du modèle.

En conséquence, on se place dans une démarche radicalement différente de celle propre à la construction de MBE : la démarche est davantage exploratoire, où l'on intervient à des grains fins et où l'ajout de comportements, d'interactions, d'entités ou de types d'entité se fait de manière relativement aisée. Cette démarche peut entrer en jeu dans un processus de recherche ou de raffinement de modèle.

Les SMA sont bâtis autour de quatre concepts principaux (Michel et coll., 2009) :

- l'autonomie des entités : les entités du système agissent de leur propre initiative, en utilisant les ressources dont elles disposent ou avec l'aide d'autres entités ;
- les entités sont sociales, c'est-à-dire qu'elles font partie intégrante de la société dans laquelle elles évoluent, celle-ci étant elle-même issue des interactions des entités qui la composent et modifiant en retour les comportement de ces entités ;
- la construction de cette société dépend donc des interactions de ses entités ;
- l'environnement dans lequel les entités sont situées permet de définir comment celles-ci interagissent et se comportent : l'environnement est un médium d'interaction.

La société d'entités forme donc une boucle de rétroaction — positive ou négative selon les cas — : les entités y agissent de manière autonome, interagissant localement et modifiant l'environnement dans lequel elles sont situées. L'environnement va en retour influencer les comportements des entités.

Un élément important des SMA est leur aspect qualitatif, que ne peut capturer un MBE. Cependant, un SMA peut également présenter un aspect quantitatif car les paramètres du système peuvent être précisément évalués. En outre, un autre élément de flexibilité des SMA est leur couplage aisé à d'autres modes de modélisation : il est en effet possible, et d'une simplicité relative, de coupler un SMA à un système d'équations. Intrinsèquement, les SMA permettent donc de modéliser plusieurs échelles, grâce à la définition des comportements au niveau microscopique qui permettent l'observation de comportements au niveau macroscopique, ainsi que par le couplage à des équations, permettant de représenter d'autres échelles (dans le cas d'un système biologique, cela peut être la représentation de concentrations de molécules).

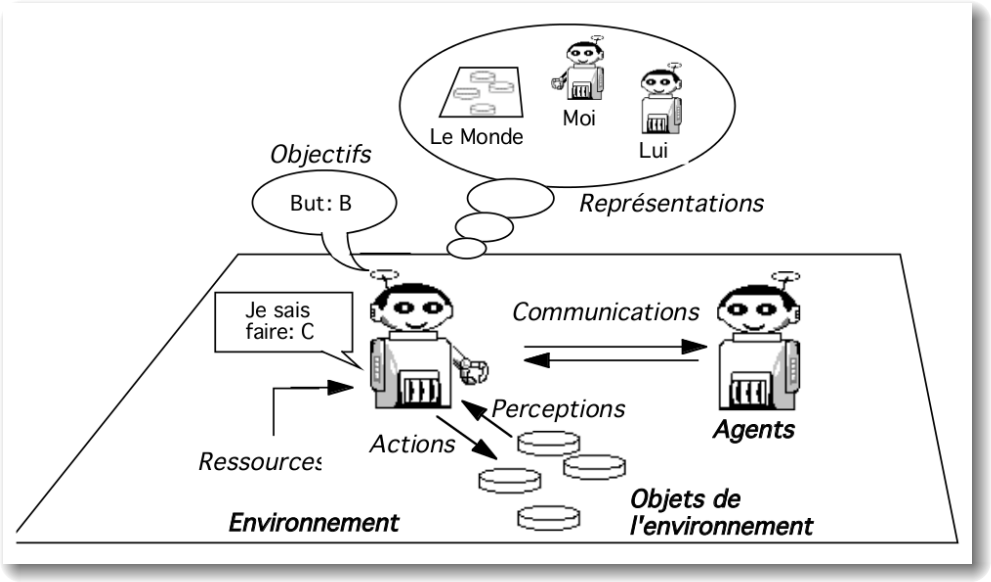


FIGURE 3.1 – Représentation simplifiée d'un agent en interaction avec son environnement. Figure extraite de (Ferber, 1995).

b) Principes des systèmes multi-agents

Demazeau (1995) propose de définir les SMA autour des concepts suivants : **A**gents, **E**nvironnement, **I**nteractions, **O**rganisation. C'est ce que l'on appelle l'approche voyelle.

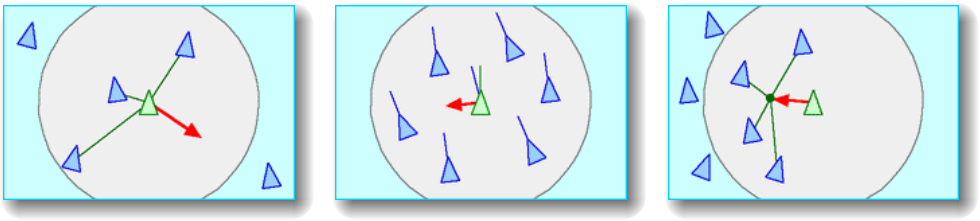
Agents. Ferber (1995) propose de définir les entités, ou agents, comme suit :

> *Un agent est une entité virtuelle ou physique :*
> - *qui est capable d'agir dans son environnement ;*
> - *qui peut communiquer directement avec d'autres agents ;*
> - *qui est mue par un ensemble de tendances (objectifs individuels, fonction de satisfaction / de survie à optimiser) ;*
> - *qui possède des ressources propres ;*
> - *qui est capable de percevoir (de manière limitée) son environnement ;*
> - *qui ne dispose que d'une représentation partielle de cet environnement (éventuellement aucune) ;*
> - *qui possède des compétences et offre des services ;*
> - *qui peut éventuellement se reproduire ;*
> - *dont le comportement tend à satisfaire ses objectifs, selon les ressources et les compétences dont elle dispose et selon ses perceptions, ses représentations et les communications qu'elle reçoit.*

La figure 3.1 illustre une représentation d'un agent en interaction avec son environnement, en particulier avec d'autres agents.

Il existe différents types d'agents. On peut distinguer deux catégories principales, même si en pratique la frontière est ténue.

Les **agents réactifs** sont des agents dont le comportement est de type stimulus / réaction. Ce type d'agent n'a pas de représentation de son environnement (y compris des autres agents) et n'a pas de capacité d'anticipation ou de planification de ses actions, que ce soit sur la base de ses actions passées ou de buts à réaliser. Les agents réactifs sont par exemple utilisés



(a) **(b)** **(c)**

Figure 3.2 – Comportements des agents dans un essaim selon Reynolds : (a) séparation ; (b) alignement ; (c) cohésion. Source : http ://www.red3d.com/cwr/boids/

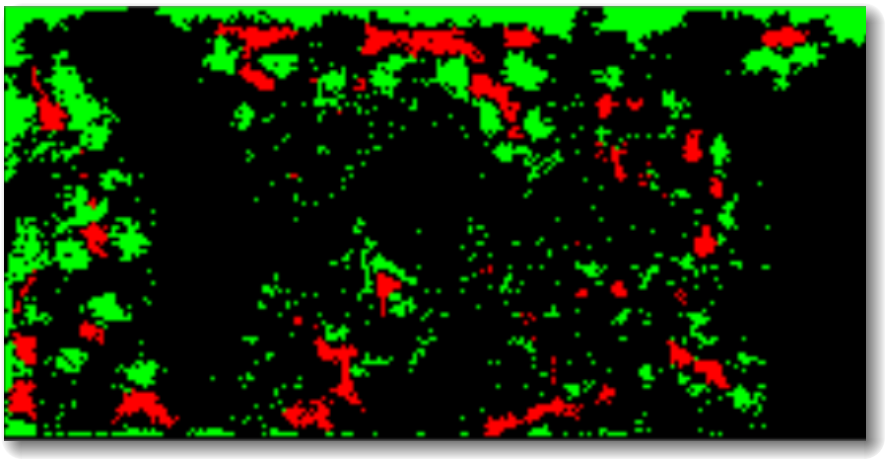


Figure 3.3 – Simulation de deux populations distinctes se comportant en essaim. Des comportements de reproduction et de prédation sont également implémentés dans le système (les agents verts sont les proies et les agents rouges sont les prédateurs). Figure extraite de (Guevel, 2013).

dans des systèmes d'essaims tels que ceux proposés par Reynolds (1987) : ce type particulier de SMA est constitué d'agents effectuant des déplacements en groupe, à la manière de certaines espèces que l'on peut observer dans la nature comme les étourneaux ou les bancs de poissons. Les comportements de ces agents tels qu'ils sont définis par Reynolds sont basés sur trois principes : la séparation (les agents évitent de trop se rapprocher de leurs voisins) ; l'alignement (les agents se dirigent dans la direction moyenne des directions de leurs voisins) et la cohésion (les agents se déplacent vers la position moyenne de leurs voisins). Ces trois comportements sont illustrés sur la figure 3.2. Un exemple de mise en pratique de ces comportements est illustré sur la figure 3.3 (Guevel, 2013) : ce système ne fait cependant pas qu'implémenter des comportements d'agents en essaim car le système est couplé à un système proies / prédateurs. Il y a donc, en plus des comportements de regroupement, de reproduction et de prédation.

les **agents cognitifs** sont des agents plus « complexes », dont le comportement est non seulement affecté par leur environnement mais aussi par le raisonnement, à partir d'une base de connaissances par exemple. Ces agents ont la capacité de planifier des actions futures — dynamiquement si un tel agent est interruptible. L'architecture BDI (Rao et Georgeff, 1992) est une des architectures les plus connues pour construire des agents cognitifs. Ces agents sont mus par leurs *croyances* (« Belief »), leur *buts*, ou désires, (« Desire ») et par leurs *intentions* (« Intention »). Un des objectifs de l'utilisation de ce type d'agent est de se rapprocher du

comportement humain. Les travaux de Bordini et coll. (2007) utilisent l'architecture JASON[1] pour modéliser une équipe d'agents devant extraire de l'or dans un environnement virtuel. Le SMA a à la base été conçu dans le cadre d'un concours[2], dont le scénario était de mettre en concurrence deux équipes de mineurs explorant une même zone (l'environnement), devant éviter les obstacles présents (arbres et bosquets) pour collecter des minerais d'or disséminés dans l'environnement. Chaque équipe devait faire preuve de coordination pour collecter le minerai et le rapporter à un dépôt. Bordini et coll. — qui ont par ailleurs remporté le concours — ont mis au point une équipe de mineurs conduite pas un leader coordonnant leurs actions dans l'environnement, entre autres pour leur affecter des zones distinctes à explorer selon leurs positions de départ. Les mineurs portant déjà un minerai ne peuvent en collecter d'autres avant d'avoir déposé le premier minerai. Toutefois, s'ils trouvent un minerai en chemin, ils communiquent sa position aux autres agents. Ceux-ci, selon leur distance au minerai et leur disponibilité, peuvent se proposer pour aller chercher le minerai. Toutes les propositions sont envoyées au leader, qui va déterminer le meilleur agent pour aller chercher ce minerai. La figure 3.4 montre l'environnement dans lequel évoluent les deux équipes d'agents-mineurs.

Qu'ils soient réactifs ou cognitifs, le comportement des agents est fondé sur le cycle « perception-décision-action ». La perception est le fait de percevoir l'état de l'environnement, comme le voisinage d'un agent ou encore la présence de molécules. L'étape de décision consiste à choisir l'action à entreprendre, en accord avec les informations perçues depuis l'environnement et l'état interne de l'agent. Enfin, l'action consiste en l'exécution effective de la décision prise pendant la phase précédente. Cette action peut avoir pour effet de modifier l'environnement. Ce cycle est illustré sur la figure 3.5.

Environnement. L'environnement d'un SMA est l'ensemble des agents et des objets composant le système. L'environnement est le plus souvent un environnement physique (dans le sens où il comporte une métrique). Les environnements peuvent être continus, dans ce cas la portée des perceptions des agents est continue ; ils peuvent aussi être discrets et construit selon une topologie donnée, telle qu'une grille. Dans ce cas, les perceptions des agents sont les cases de la grille. Comme nous le verrons dans la section 5, il est possible de considérer l'environnement tantôt comme discret, tantôt comme continu.

Interactions. Les interactions dans un SMA peuvent être de nature variée, comme la communication par passage de messages permettant à un agent A de communiquer des informations ou des requêtes à un agent B. La communication peut également se faire par signaux : ce concept est originaire à la fois de la biologie, avec par exemple l'émission de facteurs de croissance par des cellules, et de l'éthologie, avec dans ce cas l'émission de phéromones dans une colonie de fourmis. Ces signaux peuvent également se diffuser dans l'environnement et dans ce cas, il s'agit de communications indirectes.

Organisation. L'organisation d'un SMA, émergente dans le système, est la structuration du système (de ses individus en particulier) en une hiérarchie ou un groupe. Cette structuration découle des interactions des agents du système. L'organisation peut s'observer à différents niveaux dans un SMA (Ferber, 1995) : au niveau « micro-social », où l'on se concentre sur les interactions impliquant un petit nombre d'agents ; au niveau « groupes », où l'intérêt se

1. Architecture libre pour la création d'agents cognitifs, en particulier basés sur le modèle BDI – http ://jason.sourceforge.net/wp

2. https ://www.multiagentcontest.org/

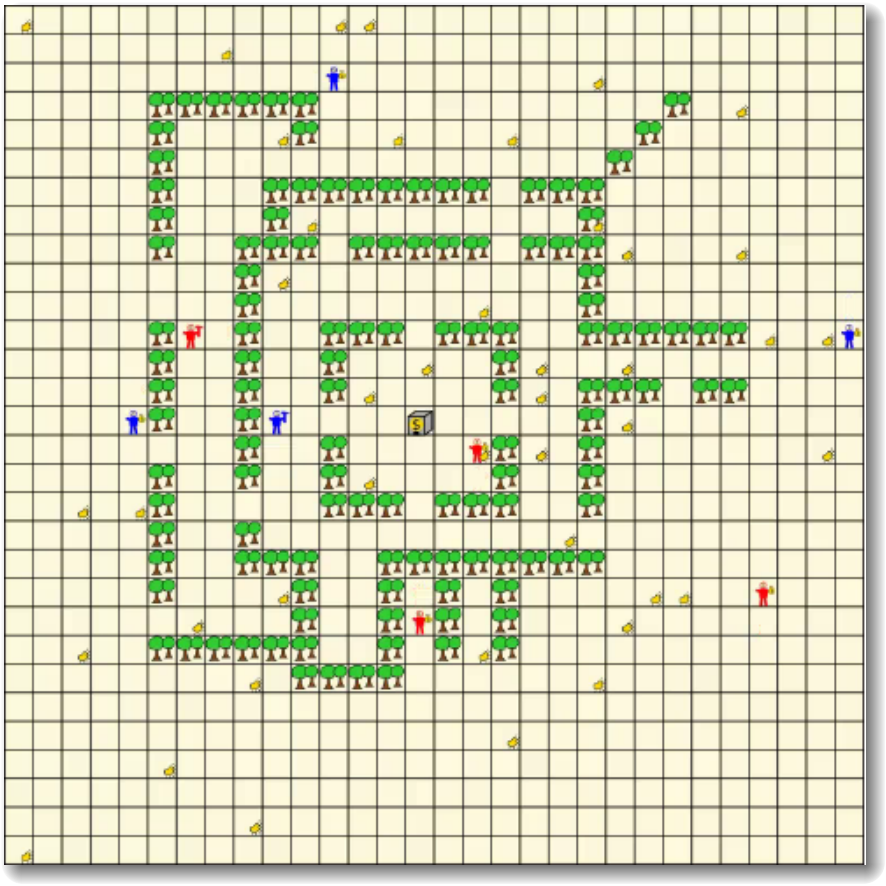


Figure 3.4 – Utilisation d'un SMA basé sur le modèle BDI pour la constitution d'une équipe coordonnées de mineurs d'or en compétition avec une équipe adverse. Source : https ://www.multiagentcontest.org/2006

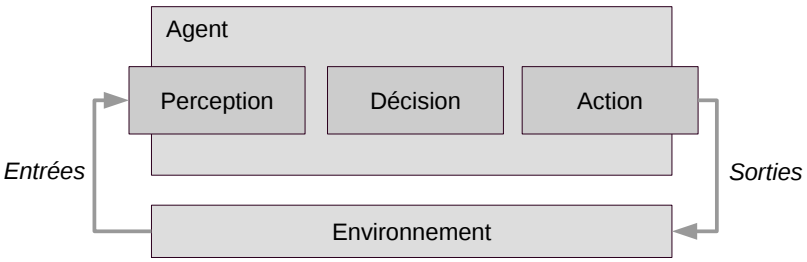


Figure 3.5 – Cycle de perception-décision-action d'un agent. La prise d'information depuis l'environnement permet à l'agent de décider de l'action à entreprendre, en accord avec son état interne. L'exécution de l'action peut en retour modifier l'état de l'environnement.

déplace vers des structures intermédiaires du système et enfin, au niveau « société globale », où l'on se concentre cette fois sur un ensemble considérable d'agents.

La suite de ce chapitre est consacrée à la présentation du modèle de cellule virtuelle que nous proposons dans cette thèse, qui est un modèle biomécanique. Nous avons pris comme point de départ pour notre modélisation la structure de cellule proposée par Ballet et Tracqui (2007). Nous avons modélisé, en étant guidé par des contraintes d'efficacité, les comportements cellulaires principaux. En particulier, nous avons inclus des éléments de mécanotransduction — la prise en compte de contraintes mécaniques issues de l'environnement, ces contraintes ayant une influence sur les comportements des cellules — qui, s'ils sont présents dans les systèmes réels, sont encore peu pris en compte dans les modèles de cellule existants. Notre modèle de cellule virtuelle est construit sur la base d'une forme complexe, en deux dimensions, constituée de nœuds reliés par des liens (un système masses / ressorts). Le couplage du modèle à notre système multi-agents permet en outre d'obtenir un modèle dont l'implémentation est efficace, en plus d'apporter la flexibilité inhérente au formalisme multi-agents : c'est ce que nous verrons dans la suite de ce chapitre.

2 Structure de la cellule virtuelle

Notre cellule virtuelle est un modèle en deux dimensions constitué de nœuds (des masses) reliés entre eux par des liens élastiques, dont la configuration permet de modéliser différents éléments de la structure de la cellule : cela nous permet de représenter de manière simplifiée la membrane, le cortex cellulaire et le cytosquelette. Nous estimons important de proposer une modélisation, même très simplifiée, de la structure interne de la cellule. De cette façon, la structure de notre cellule virtuelle est stable et les tensions et compressions s'appliquant à la cellule s'équilibrent (celle-ci tend à retrouver sa forme de base lorsque des tensions physiques s'appliquent à elle) : on parle alors de *tenségrité*. Cette notion, issue du domaine de l'architecture, joue un rôle important dans différents phénomènes biologiques (Ingber, 2003b) : sur le développement de tissus de manière générale et de manière plus spécifique, sur l'expression des gènes, la différentiation cellulaire ou encore l'apoptose. Certains travaux se penchent sur la mise au point de modèles de cellules en se focalisant sur la modélisation de la structure interne de celles-ci (Ingber, 2003a). Cependant ces modèles sont créés pour l'étude d'une unique cellule et sont donc complexes, alors que nous cherchons à modéliser des milliers de cellules en interactions. La structure interne que nous proposons semble alors être un bon compromis pour disposer à la fois d'une cellule exhibant de la tenségrité et qu'il nous est possible de simuler en nombre important afin d'étudier le développement de tissus.

La cellule est composée de n nœuds membranaires N_i ($0 \leq i < n$) et d'un nœud central C. Ces nœuds sont reliés par des ressorts formant les structures suivantes, illustrées sur la figure 3.6 :

- membrane : ensemble des nœuds $\{N_i \leftrightarrow N_{(i+1)\%n}\}$ (liens verts pointillés) ;
- cytosquelette cortical (ou cortex) : ensemble des nœuds $\{N_i \leftrightarrow N_{(i+2)\%n}\}$ (liens orange) ;
- cytosquelette : ensemble des nœuds $\{C \leftrightarrow N_i\}$ (liens noirs).

La construction de la cellule dépend du nombre de nœuds n sur la membrane (notons que le nombre de nœuds sur la membrane des cellules peut être modifié d'une simulation à l'autre mais est le même pour toutes les cellules lors d'une simulation donnée).

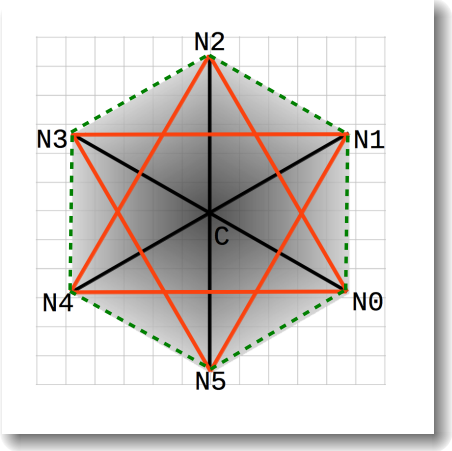


Figure 3.6 – Structure physique de notre cellule virtuelle. Exemple avec $n = 6$ nœuds. Les liens verts pointillés correspondent à la membrane ; les liens orange correspondent au cytosquelette cortical (le cortex) et les liens noirs correspondent au cytosquelette.

La construction est réalisée par de simples opérations géométriques en fonction de n et du rayon R de la cellule. Ce rayon R correspond à la longueur à vide des liens composant le cytosquelette, soit :

$$l_{0,cyto} = R \tag{3.1}$$

Le nœud central de la cellule ayant pour coordonnées $C(x, y)$, les n nœuds de la cellule ont pour coordonnées :

$$N_i \begin{pmatrix} x + R\cos i\dfrac{2\pi}{n} \\ \\ y + R\sin i\dfrac{2\pi}{n} \end{pmatrix} \tag{3.2}$$

Les longueurs à vide des autres éléments de la cellule virtuelle sont calculées sur la base du rayon R de la cellule. La figure 3.7 permet de visualiser les différents éléments utilisés dans les calculs. La nature régulière du polygone constituant la cellule permet en particulier d'utiliser des relations de trigonométrie pour caractériser les longueurs à vide des différents éléments de la structure.

Longueur à vide d'un ressort de membrane. On s'intéresse ici au calcul de la longueur du segment $[N_iN_{i+1}]$. En suivant l'exemple fourni par la figure 3.7, on considère le segment $b = [N_{11}N_0]$. Soit $h = [CH_2]$ la médiatrice de la base b, et $a = [CN_0]$. La longueur de a est connue, il s'agit du rayon de la cellule R. Le triangle CH_2N_0 étant rectangle en H_2, on a : $\frac{b}{2} = a\sin\frac{\alpha}{2}$, soit :

$$l_{0,membrane} = 2a\sin\frac{\alpha}{2} \tag{3.3}$$

Longueur à vide d'un ressort de cortex. Cette fois on s'intéresse au calcul de la longueur du segment $[N_iN_{i+2}]$. Nous prenons pour exemple le segment $b = [N_0N_2]$. Soit $h = [CH_1]$ la médiatrice de la base b et $a = [CN_2]$. Le segment a est le rayon de la cellule, de longueur R. Le triangle CH_1N_2 est rectangle en H_1 et on a : $\frac{b}{2} = a\sin\frac{2\alpha}{2}$, soit :

$$l_{0,cortex} = 2a\sin\alpha \tag{3.4}$$

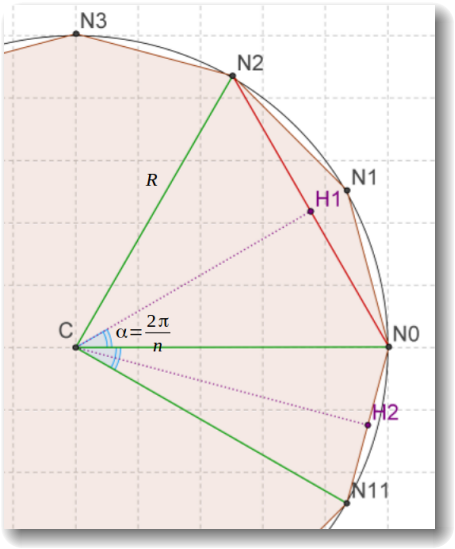


FIGURE 3.7 – Éléments géométriques de la cellule pour la caractérisation des longueurs à vide des ressorts. Exemple pour $n = 12$.

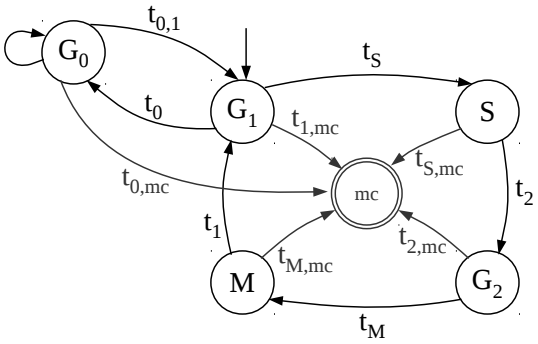


FIGURE 3.8 – Modélisation simplifiée du cycle cellulaire avec un automate à état. L'acronyme « mc » signifie « mort cellulaire ».

3 Comportements de la cellule virtuelle

Dans cette section, nous exposons les différents formalismes de notre modèle permettant de modéliser des comportements cellulaires : en particulier, nous avons modélisé le cycle cellulaire (cela comprend également la croissance des cellules et leurs divisions), les interactions cellulaires (entres cellules, et entre cellule et environnement), ainsi que des éléments de mécanotransduction permettant aux cellules virtuelles de prendre en compte les contraintes mécaniques issues de leur environnement.

3.1 Cycle cellulaire

a) Modélisation du cycle cellulaire

Pour modéliser le cycle cellulaire, nous utilisons un automate à états dont les états sont les états des cellules à différentes phases du cycle cellulaire, tel qu'illustré sur la figure 3.8. Les transitions entre les états peuvent être définies de différentes manières : en fonction du temps moyen qu'une cellule d'un type donné passe dans une phase donnée du cycle

cellulaire, en fonction de la taille de la cellule (pour passer dans l'état mitose), ou encore à l'aide d'un tirage aléatoire. Le cycle cellulaire tel que nous l'avons implémenté est une version très simplifiée du cycle cellulaire réel. Notons que des travaux plus complets sur la modélisation du cycle cellulaire ont été réalisés par Pascalie et coll. (2012).

Biologiquement, des points de contrôle sont réalisés entre les différentes étapes du cycle afin de s'assurer que le matériel génétique de la cellule est intègre tout au long de son cycle. Nous n'avons pas modélisé explicitement ces points de contrôle, mais nous avons inclus dans notre automate des transitions partant des étapes du cycle vers l'état d'apoptose (qui peut de manière plus générale représenter l'état de mort cellulaire).

Le cycle cellulaire, en plus d'incorporer plus de réalisme dans le modèle, permet de *désynchroniser* les cellules lors de la prolifération d'un tissu : le cycle de chaque cellule va dépendre de son automate à état, qui lui même est contrôlé par différents facteurs tant endogènes qu'exogènes. Une synchronicité des divisions cellulaires n'est pas aberrante, c'est un phénomène observé lors des premières divisions cellulaires (Rott et Sheveleva, 1968) mais la modélisation, même simplifiée, du cycle cellulaire permet aux cellules de se désynchroniser. Compte tenu de la nature du cycle cellulaire, il nous a semblé que le moyen le plus simple pour le modéliser est d'utiliser un automate à états.

b) Croissance des cellules

Sachant que les cellules possèdent des propriétés telle que l'âge ou encore une énergie, qui dépend de la quantité de nutriments qu'elles ont reçu, on peut utiliser ces propriétés pour modéliser leur croissance en calculant ce que nous avons appelé un *facteur d'échelle*. Ce facteur permet de faire varier le rayon de la cellule en fonction de la propriété choisie. Ainsi, si l'on considère l'âge comme la propriété choisie, ce facteur d'échelle vaut :

$$facteur\ d'échelle = \frac{1}{2}\left(1 + \frac{age}{age + t}\right) \tag{3.5}$$

Cette relation permet de faire en sorte qu'à l'âge 0, une cellule a un facteur de $\frac{1}{2}$, ce qui donne à la cellule un rayon égal à la moitié de celui de la cellule venant de se diviser. Ce mécanisme permet d'approximer la réduction de la taille de la cellule lorsque celle-ci se divise, ainsi que sa croissance au cours du temps. Le facteur d'échelle tend vers 1 alors que l'âge de la cellule augmente. Pendant son cycle cellulaire, la cellule va doubler sa taille avant de se diviser effectivement (nous considérons uniquement la taille de la cellule, et non sa masse, pour déterminer quand celle-ci peut se diviser : à l'heure actuelle, la masse des cellules reste constante indépendamment de leurs tailles, ceci afin de ne pas avoir à modifier le paramétrage du moteur physique sous-jacent au modèle de cellule). La durée du cycle varie selon les types de cellules. Soit T_d la durée du cycle d'un type cellulaire. Pour que la cellule puisse se diviser, nous voulons que celle-ci ait une surface supérieure ou égale à 90% de sa surface cible ; autrement dit que son facteur d'échelle soit supérieur ou égal à 0.9, soit :

$$\begin{aligned}
&\frac{1}{2}\left(1 + \frac{T_d}{T_d + t}\right) \geq 0.9 \\
\Leftrightarrow \quad &\frac{T_d}{T_d + t} \geq 0.8 \\
\Leftrightarrow \quad &T_d \geq 0.8\,T_d + 0.8\,t \\
\Leftrightarrow \quad &T_d - 0.8\,T_d \geq 0.8\,t
\end{aligned}$$

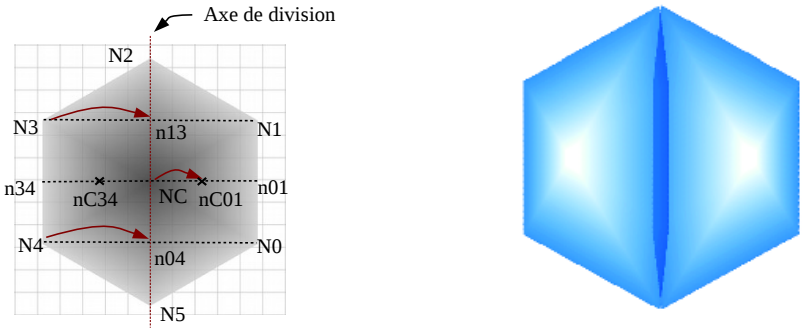


FIGURE 3.9 – Exemple de mitose pour une cellule ayant $n = 6$ nœuds. (a) Division selon l'axe N_5N_2 : redéfinition des nœuds de la cellule mère pour libérer physiquement de la place afin de placer une nouvelle cellule. (b) Division réalisée dans l'environnement virtuel. Une division cellulaire en environnement virtuel peut être visionnée ici : `https://vimeo.com/95404862` (dans le cadre de cette illustration, la division est réalisée sans prise en compte du facteur d'échelle).

$$\Leftrightarrow \quad 0.8\,t \leq 0.2\,T_d$$
$$\Leftrightarrow \quad t \leq \frac{1}{4}T_d$$

À titre d'information, la durée du cycle cellulaire T_d chez les procaryotes est de l'ordre d'une vingtaine de minutes contre 10 à 20 heures chez les eucaryotes (Grunwald et Ronot, 2008).

Il est possible de calculer le facteur d'échelle avec d'autres propriétés de la cellule, comme son énergie ou encore selon les facteurs de croissance présents dans l'environnement : nous avons vu dans le chapitre 2 que des cellules ne recevant pas de facteur de croissance pouvaient ne plus croître et n'allaient pas jusqu'à l'étape de mitose de leur cycle cellulaire. Dans ce cas il convient d'établir une relation entre t et la propriété prise en compte, comme nous l'avons montré pour l'âge.

c) Division cellulaire

La division des cellules est modélisée de la façon suivante : la cellule voulant se diviser est « coupée » en deux moitiés égales, selon un axe $N_iN_{(i+\frac{n}{2})\%n}$. Les points de la cellule mère sont ensuite « déplacés » de façon à ce que celle-ci occupe une surface qui est égale à la moitié de la surface avant division. Cela permet de créer de l'espace pour y placer une nouvelle cellule, issue de la division. La figure 3.9 illustre le déplacement des points de la cellule mère et montre l'exemple d'une mitose réalisée en environnement virtuel.

Formalisation pour une cellule à n nœuds. Formellement, le déplacement des nœuds de la cellule mère est réalisé comme suit : n est le nombre de nœuds sur la membrane. Pour la division, nous considérons un axe $N_iN_{(i+\frac{n}{2})\%n}$ avec $0 \leq i < n$; i représente l'indice du nœud de départ de l'axe de division. Soit q le nombre de nœuds compris entre N_i et $N_{(i+\frac{n}{2})\%n}$ exclus : $q = \frac{n-2}{2}$. Soit p le nombre de nœuds déjà traités dans la cellule : $p = p+2$ à chaque nœud N_k traité ($i < k < (i+\frac{n}{2})\%n$). On cherche à déterminer quel est le nœud symétrique N_{ksym} du nœud N_k afin de placer le nœud intermédiaire sur l'axe de division, sachant que ce nœud intermédiaire se trouve à la moitié du segment $[N_kN_{ksym}]$. Pour illustrer la notion du symétrique d'un nœud N_k, nous pouvons nous reporter à la figure

3.9a : si l'on considère le nœud N_3, alors son symétrique est le nœud N_1, autrement dit $ksym = 1$. L'indice $ksym$ du nœud symétrique de N_k est :

$$ksym = (k + 2q - p)\%n \tag{3.6}$$

Les q nœuds sont ainsi traités de manière successive (algorithme 3.1). La nouvelle position du nœud se trouve donc à la moitié du segment $[N_k N_{ksym}]$, sur l'axe de division de la cellule. De la même façon, les positions des nœuds de la cellule filles sont calculées.

```
1 p ← 0
2 pour chaque nœud d'indice k : i < k < (i + n/2)%n faire
3 |   ksym ← (k + 2q − p)%n
4 |   p ← p + 2
5 |   k ← k + 1
6 fin
```

Algorithme 3.1 - Calcul des indices $ksym$ pour réaliser la division cellulaire.

Le nœud central de la cellule nécessite un traitement particulier selon que q soit pair ou impair. La figure 3.10 illustre les deux cas de figure. On cherche à placer le nouveau centre de la cellule, identifié en rouge sur la figure. Pour cela, il faut trouver un nœud N_r qui est le plus proche possible du nouveau centre et qui permet de déterminer un point P, identifié en vert sur la figure, tel que les droites (CP) et $(N_i N_{(i+\frac{n}{2})\%n})$ — l'axe de division — soient perpendiculaires.

Si q est pair, la configuration est celle de la figure 3.10a donc l'indice r du nœud que l'on recherche est :

$$r = (i + \frac{q}{2})\%n \tag{3.7}$$

Le point P est le milieu du segment $[N_r N_{r+1}]$. Le nouveau centre de la cellule sera donc le milieu du segment $[CP]$.

Dans le cas où q est impair, la configuration est celle présentée sur la figure 3.10b : l'indice du nœud que l'on recherche est dans ce cas :

$$r = (i + \left\lceil \frac{q}{2} \right\rceil)\%n \tag{3.8}$$

Dans ce cas, $P = N_r$. Le nouveau centre de la cellule sera placé au milieu du segment $[CP]$.

Exemple. Reprenons la figure 3.9a, qui illustre les déplacements des nœuds d'une cellule à $n = 6$ nœuds. L'axe de la division est $N_5 N_2$ et la nouvelle cellule va être placée à gauche de cet axe. On a $n = 6$, $q = 2$. Il faut trouver les indices des nœuds symétriques aux nœuds dont les indices sont compris entre 2 et 5 exclus.

- Pour $k = 3$: $ksym = (3 + 2 * 2 - 0)\%6 = 1$ ($p \leftarrow p + 2$) : la nouvelle position du nœud N_3 est donc à la moitié du segment $[N_3 N_1]$. Sur la figure 3.9a, cette position est identifiée par le point n_{13}.
- Pour $k = 4$: $ksym = (4 + 2 * 2 - 2)\%6 = 0$: dans ce cas la nouvelle position de N_4 est à la moitié du segment $[N_4 N_0]$, soit la position identifiée par le point n_{04}.
- Reste à replacer le centre de la cellule : ici, $q = 2$, donc nous allons utiliser la relation 3.7. Pour placer le nouveau centre de la cellule mère, il faut inverser les nœuds de l'axe

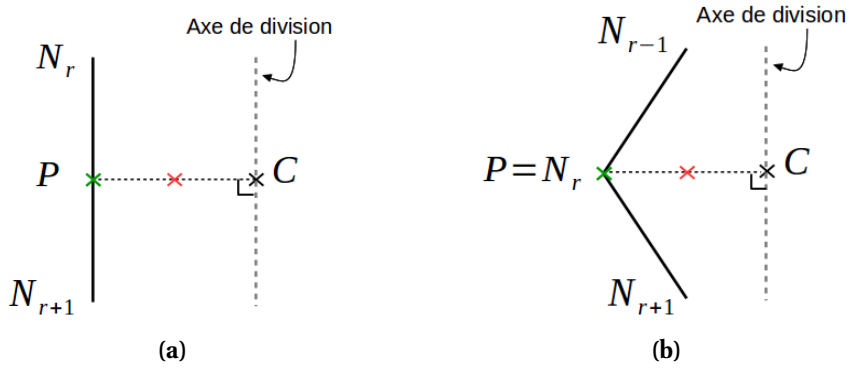


FIGURE 3.10 – Configuration de la cellule pour replacer le nœud central de la cellule mère lors de la division : (a) q est pair ; (b) q est impair.

de division afin de se placer du bon côté de l'axe. Cela signifie que dans la relation 3.7, pour la cellule mère, i vaut $(i + \frac{n}{2})\%n$. On a donc : $r = (5 + \frac{2}{2})\%6 = 0$. Le point P sera donc la moitié du segment $[N_0N_1]$, et la nouvelle position du centre de la cellule mère est à la moitié du segment $[CP]$, identifié par le point $nC01$ sur la figure 3.9a.

Si la nouvelle cellule est à placer à droite de l'axe, la démarche est la même, mais l'utilisation des modulos pouvant rendre l'exemple difficile à suivre, c'est le cas le plus trivial qui a été développé.

Une fois les cellules placées, une force de répulsion et des forces intracellulaires vont leur permettre de reprendre leur forme normale. Ces forces sont détaillées dans la section 3.2 qui suit.

3.2 Interactions cellulaires

a) Forces intracellulaires

Comme nous l'avons vu, la structure de la cellule virtuelle est construite sur des ensembles de liens permettant de proposer une approximation de la structure d'une cellule biologique. Les liens formant la membrane, le cytosquelette et le cytosquelette cortical (le cortex) sont modélisés par des forces de rappel :

$$\vec{F_r} = k(l - l_0)\vec{u} = k\Delta l\vec{u} \tag{3.9}$$

où k est la raideur du ressort, l la distance entre les extrémités du ressort (ses extrémités sont des nœuds de la cellule), l_0 la longueur au repos de ces ressorts et $\vec{u}$ dénote la direction de la force.

Cette force est appliquée sur chaque lien présent dans la cellule : sur la membrane, avec les liens $N_i \leftrightarrow N_{(i+1)\%n}$; sur le cytosquelette, avec les liens $C \leftrightarrow N_i$ et sur le cortex, avec les liens $N_i \leftrightarrow N_{(i+2)\%n}$.

À chaque fois que, pour un nœud i, une force de rappel est calculée, la même force est ajoutée — mais dans la direction opposée — au nœud N_j en interaction avec N_i, selon le principe des actions réciproques stipulé par la troisième loi de Newton :

$$\vec{F}_{N_iN_j} = -\vec{F}_{N_jN_i} \tag{3.10}$$

b) Interactions cellules / cellules

Les interactions cellules / cellules sont modélisées avec l'utilisation de deux forces. La première est une force adhésive :

$$\vec{F_a} = k(l - d_{max})\vec{u} \tag{3.11}$$

permettant aux cellules d'adhérer en fonction d'un coefficient d'adhésion. $\vec{u}$ dénote la direction de la force, k est la raideur du ressort, l est la distance entre les centres des cellules et d_{max} est la distance maximale jusqu'à laquelle la force agira avant que le lien élastique ne se rompe. L'intensité de l'adhésion peut s'exprimer de deux façons : il est possible de faire varier la valeur d_{max} qui permettra à des cellules d'aller « chercher » la cellule avec laquelle elle est susceptible d'adhérer ; on peut également faire varier la raideur k du lien : plus celle-ci sera importante et plus le lien sera difficile à briser. Cette force permet de modéliser *l'adhésion cellulaire*.

Cette force est appliquée sur certains nœuds des cellules en adhésion. Les nœuds en question sont sélectionnés avec une simple opération géométrique : un produit scalaire. La figure 3.11 illustre la démarche que nous présentons : nous voulons créer des ressorts entre des nœuds de la membrane d'une première cellule $Cell_1$ et le centre d'une seconde cellule $Cell_2$. Soit C_1 (resp. C_2) le centre de la cellule $Cell_1$ (resp. $Cell_2$). Nous dénotons l'ensemble des nœuds de la cellule $Cell_1$ par N_i^1 avec $0 \leq i < n$ et nous définissons $\vec{b_i} = C_1N_i^1$ et $\vec{a} = C_1C_2$. Les nœuds de $Cell_1$ sélectionnés pour l'adhésion sont tels que $\vec{a} \cdot \vec{b_i} > 0$, autrement dit l'angle $\widehat{C_2C_1N_i^1}$ doit être aigu. Cette méthode permet de créer des ressorts uniquement avec les nœuds de $Cell_1$ qui sont le plus près de $Cell_2$. En outre, cette méthode, une fois implémentée, ne requiert qu'un temps de calcul et qu'une quantité de mémoire faibles, en opposition avec une méthode de sélection basée sur une recherche explicite des nœuds de la cellule et leur mémorisation.

La seconde force entrant en jeu dans les interactions cellules / cellules est une force de répulsion :

$$\vec{F}_{rep} = k(l - d_{min})\vec{u} \tag{3.12}$$

empêchant en particulier les cellules de se superposer. $\vec{u}$ dénote la direction de la force, l est la distance entre les centres des deux cellules. Si cette distance est inférieure à la distance minimale définie par d_{min}, alors un ressort est créé entre les centres des deux cellules, leur permettant de se repousser. Cette force garantit que les centres des cellules ne se trouveront jamais situés dans un même point de l'espace (c'est important en particulier pour des raisons de synchronisation que l'on discutera plus en détails dans la partie traitant de l'implémentation parallèle du système, au chapitre 5).

Les nœuds des membranes se repoussent également avec cette force — la force issue de cette répulsion est simplement ajoutée à $\vec{F}_{rep}$. De la même façon que pour l'adhésion, les nœuds des membranes sur lesquels vont s'appliquer cette répulsion sont sélectionnés avec un produit scalaire.

Le principe des actions réciproques s'applique également dans ce cas. Toutefois, les réciproques des forces ne sont pas explicitement ajoutées aux cellules en interaction avec une cellule donnée, cela pour des raisons conceptuelles liées au couplage avec un système multi-agents et à son implémentation. Ces raisons sont expliquées dans le paragraphe « Interactions des agents », page 89.

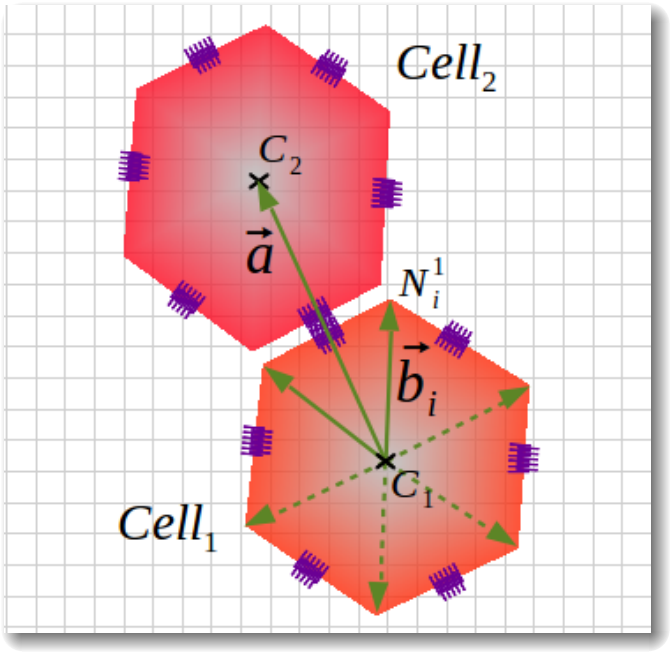


FIGURE 3.11 – Méthode de sélection de nœuds pour la création de ressorts entre deux cellules pour l'adhésion cellulaire : si $\vec{a} \cdot \vec{b_i} > 0$ (l'angle $\widehat{C_2C_1N_i^1}$ est aigu) alors le nœud N_i^1 appartenant à la cellule $Cell_1$ est sélectionné, et un ressort est créé entre C_2 et N_i^1.

c) Interactions cellules / environnement

L'environnement agit sur les cellules à travers différentes forces. Tout d'abord, une force de Langevin est appliquée. Elle permet de modéliser les collisions aléatoires se produisant entre les nœuds des cellules et les molécules de l'environnement. Cette force s'exprime de la façon suivante :

$$\vec{F_L} = I\vec{u} \tag{3.13}$$

où I est l'intensité de la force appliquée sur tous les nœuds de la cellule, et $\vec{u}$ dénote la direction de la force.

Les bords de l'environnement exercent également une force de répulsion sur les cellules, de façon à ce que celles-ci ne puissent physiquement être en dehors des frontières définies pour l'environnement. Cette force, modélisée avec un ressort, s'exprime comme suit :

$$\vec{F}_{edges} = k(l - d_{cell,edge})\vec{u} \tag{3.14}$$

où $d_{cell,edge}$ est la distance minimale à laquelle la force est appliquée.

Nous avons également, lors d'une étude sur le cisaillement (voir section 3.3), ajouté des forces de flux, $\vec{F}_{flux}$, de quelques micro newton de directions opposées pour induire des forces de cisaillement sur les cellules. En outre, une force $\vec{F}_{chem}$, de quelques micro newton également, permet de modéliser les phénomènes de chimiotaxie et d'haptotaxie lorsque ceux-ci ont lieu. Finalement, on a :

$$\vec{F}_{env} = \vec{F_L} + \vec{F}_{edges} + \vec{F}_{flux} + \vec{F}_{chem} \tag{3.15}$$

Les autres interactions cellules / environnement concernent la consommation et la production de molécules : ces aspects sont traités un peu plus loin dans le document, dans la section 4 de ce chapitre.

Les cellules peuvent également être plus ou moins adhérentes au milieu dans lequel elles évoluent (la matrice extra-cellulaire). Pour modéliser cela, nous utilisons ce que nous avons appelé un *coefficient de fixation*, $\phi \in [0, 1]$: plus le coefficient ϕ est élevé et plus l'adhésion à la matrice extra-cellulaire est importante. Biologiquement, cela peut par exemple être interprété comme une concentration importante de protéines telle que la fibronectine. Cela représente l'adhésion de la cellule avec la matrice extra-cellulaire sans avoir besoin de modéliser les éléments la composant. La matrice extra-cellulaire est alors vue de manière implicite. La baisse du coefficient ϕ peut également être interprétée comme une facilitation de la migration cellulaire suite à la dégradation de la matrice extra-cellulaire par les cellules. Cette interaction des cellules avec l'environnement n'est pas modélisée avec une force, mais en utilisant ce coefficient lorsque la nouvelle position du nœud central de la cellule est calculée : $\vec{x_1} = (1 - \phi)\vec{x_1}$ (l'expression pour calculer $\vec{x_1}$ est donnée dans le paragraphe d) qui suit).

d) Bilan des forces et intégration numérique

Pour résumer, on a donc quatre forces différentes dans le modèle : la force de rappel pour la structure de la cellule $\vec{F_r}$; la force d'adhésion $\vec{F_a}$; la force de répulsion $\vec{F_{rep}}$ et les forces issues de l'environnement $\vec{F_{env}}$. Pour faire évoluer le système, nous mettons en pratique la seconde loi de Newton :

> Les changements qui arrivent dans le mouvement sont proportionnels à la force motrice, et se font dans la ligne dans laquelle cette force a été imprimée.

Mathématiquement, cela se traduit par l'expression bien connue $\Sigma\vec{F} = m\vec{a}$. Dans notre cas, nous avons donc :

$$\Sigma\vec{F} = \vec{F_r} + \vec{F_a} + \vec{F_{rep}} + \vec{F_{env}} = m\vec{a} \tag{3.16}$$

avec l'accélération $\vec{a} = \frac{d\vec{v}}{dt}$ et la vitesse $\vec{v} = \frac{d\vec{x}}{dt}$. Pour intégrer cette expression, nous utilisons la méthode d'Euler explicite. Cette méthode peut exhiber des instabilités, mais l'utilisation d'un pas de temps suffisamment petit et le fait que nous considérions le milieu dans lequel évoluent nos cellules virtuelles comme amorti limite ces instabilités car l'inertie du système est elle-même limitée.

Pourvu que l'intervalle de temps soit petit, nous faisons l'approximation suivante : $\vec{v} = \frac{\Delta\vec{x}}{\Delta t}$. $\Delta\vec{x}$ est la quantité de mouvement réalisée durant l'intervalle de temps Δt. Soit $\vec{x}_1 - \vec{x}_0$ où $\vec{x}_0$ est la position du nœud au temps t et $\vec{x}_1$ est la position du nœud au temps $t + \Delta t$. En reprenant la relation 3.16 :

$$\begin{aligned}
& \vec{F} = m\frac{d\vec{v}}{dt} \\
\Rightarrow \quad & \Delta\vec{v} = \frac{\vec{F}}{m}\Delta t \\
\Leftrightarrow \quad & \vec{v}_1 - \vec{v}_0 = \frac{\vec{F}}{m}\Delta t \\
\Leftrightarrow \quad & \vec{v}_1 = \frac{\vec{F}}{m}\Delta t + \vec{v}_0 \\
\Leftrightarrow \quad & \frac{\Delta\vec{x}}{\Delta t} = \frac{\vec{F}}{m}\Delta t + \vec{v}_0 \\
\Leftrightarrow \quad & \Delta\vec{x} = \frac{\vec{F}}{m}\Delta t^2 + \vec{v}_0\Delta t
\end{aligned}$$

$$\Leftrightarrow \quad \vec{x}_1 - \vec{x}_0 = \frac{\vec{F}}{m}\Delta t^2 + \vec{v}_0 \Delta t$$
$$\Leftrightarrow \quad \vec{x}_1 = \frac{\vec{F}}{m}\Delta t^2 + \vec{v}_0 \Delta t + \vec{x}_0$$

La nouvelle position du nœud est donc :

$$\vec{x}_1 = \frac{\vec{F}}{m}\Delta t^2 + \vec{v}_0 \Delta t + \vec{x}_0 \tag{3.17}$$

Cette expression est utilisée sur tous les nœuds de la cellule. On peut également utiliser un coefficient γ permettant de modéliser de la friction. Dans l'expression 3.17, ce terme s'utilise au niveau de la vitesse initiale : $\gamma \vec{v}_0 \Delta t$.

L'utilisation de la relation 3.17 requiert la définition des conditions initiales du système : pour tous les nœuds de toutes les cellules, on a $\vec{v}_0 = 0$ à $t = 0$. En fin de chaque cycle de simulation la vitesse initiale est mise à jour avec la vitesse nouvellement calculée $v_0 \leftarrow v_1$: il y a donc la vitesse initiale injectée en début de simulation et la vitesse initiale de chaque cycle. Les positions des cellules sont par contre variables, puisque nous pouvons positionner celles-ci n'importe où dans l'environnement : seule la position du nœud central est à définir ; les autres nœuds de la cellules seront positionnés selon la relation 3.2 page 67.

Détermination du pas de temps Δt Nous avons défini une relation nous permettant de déterminer le pas d'intégration maximal Δt qu'il est possible d'utiliser dans le système. Pour rappel, un pas d'intégration trop important peut conduire à des instabilités dans le système. Dans le pire des cas, tous les ressorts de la cellule sont orientés dans la même direction et le mouvement de la masse centrale s'amplifie au lieu d'être régulée (notons que ce pire cas ne peut à priori pas se produire dans notre système car notre modèle garantit l'intégrité de la forme de la cellule virtuelle). Nous voulons éviter des oscillations du nœud central : celui-ci ne doit pouvoir se déplacer que dans l'intervalle $[l_0, x_1]$ (visible sur la figure 3.12) entre les temps t and $t+1$ (nous ne considérons que les forces de rappel étant donné que les autres forces du système sont constantes). Deux cas de figure doivent être considérés : le lien a, au temps t, une longueur supérieure (resp. inférieure) à la longueur au repos l_0. Les résultats s'appliquant aux deux cas de figure, nous ne considérons que le premier cas. Les inégalités suivantes sont à résoudre : $l_0 \leq x_1 \leq x_0$. L'inégalité de droite est la plus simple à résoudre : la force $\vec{F_r}$ étant négative, la position x_1 ne peut qu'être inférieure à la position x_0. L'inégalité de gauche est résolue en substituant x_1 par sa valeur en x_0 déterminée dans l'équation 3.17 :

$$\begin{array}{ll} & l_0 \leq x_0 + \frac{F_r}{m}\Delta t^2 \\ \Leftrightarrow & \Delta t^2 \leq (x_0 - l_0)\frac{m}{F_r} \\ avec: & F_r = nk(x_0 - l0) \\ \Leftrightarrow & \Delta t \leq \sqrt{\frac{m}{nk}} \end{array} \tag{3.18}$$

3.3 Évaluation de contraintes physiques

Les cellules virtuelles sont également capables d'évaluer les contraintes mécaniques qui leur sont appliquées. Il est largement admis que ces contraintes mécaniques ont un rôle important lors de la morphogenèse de tissus. Les forces sont transmises aux tissus par la matrice extra-cellulaire et les adhésions entre cellules. Ces forces permettent de moduler

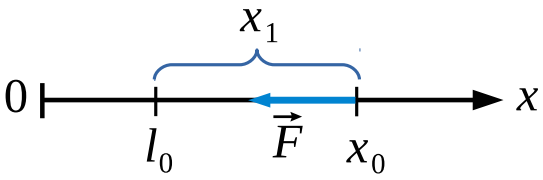


Figure 3.12 – Calcul du pas d'intégration maximal du sytème : définition de l'intervalle $[l_0, x_1]$ à l'intérieur duquel le nœud à la position x_0 peut se déplacer.

Figure 3.13 – Illustration des contraintes physiques prises en compte dans notre modèle : (a) compression ; (b) étirement ; (c) cisaillement.

et de réguler les comportements cellulaires tels que la différentiation (Holle et coll., 2013), l'apoptose (Rosenblatt et coll., 2001), la prolifération (Liu et coll., 2006b) ou encore la migration cellulaire (Lauffenburger et Horwitz, 1996). Ces influences ont lieu au niveau cellulaire, mais peuvent avoir des implications au niveaux des tissus lors du développement de ceux-ci : étudier comment ces contraintes mécaniques affectent le développement des tissus pourrait permettre de mieux comprendre le développement de certaines maladies, les cancers notamment (Jaalouk et Lammerding, 2009).

En particulier, les contraintes mécaniques pourraient jouer un rôle primordial lors du développement de tissus dans les premiers stades de l'embryogenèse. Des mesures *in vitro* de flux et de déformation de tissus chez le poulet vont dans ce sens (Fleury, 2011) : les constantes de diffusion des signaux biochimiques qu'expriment les cellules semblent trop lentes pour expliquer les rapides changements observés et mesurés par Fleury. Yennek et coll. (2014) ont observé expérimentalement que les tensions différentes s'exerçant sur des cellules souches en division ont un rôle important dans la nature de la division qui peut être symétrique (une cellule souche donne deux cellules souches ou deux cellules différenciées) ou asymétrique (une cellule souche et une cellule différenciée sont issues de la division). En corrélation avec les résultats mesurés par Fleury, Yennek et coll. apportent des éléments consolidant le fait que les contraintes physiques s'exerçant sur les cellules jouent un rôle prépondérant dans leur différentiation. Dans un registre quelque peu différent, les travaux de (Rolfe et coll., 2014) ont mis en évidence l'influence de la stimulation mécanique sur l'expression de gènes qualifiés de « mécanosensitifs » lors du développement du squelette chez la souris : une stimulation mécanique défaillante conduit à des anomalies de développement lors de l'ostéogenèse.

Dans notre système, les contraintes mécaniques que les cellules peuvent évaluer sont leur compression, leur étirement et leur cisaillement. Ces trois contraintes sont illustrées sur la figure 3.13.

a) Évaluation des contraintes de compression et d'étirement

Les contraintes de compression et d'étirement sont évaluées en calculant la surface de la cellule. La surface de la cellule est calculée en faisant la somme des aires T_i des triangles $CN_{i\%n}N_{(i+1)\%n}$ qui constituent la cellule (cette décomposition est visible sur la figure 3.6 page 67). Par construction, les triangles composants la cellule sont isocèles mais

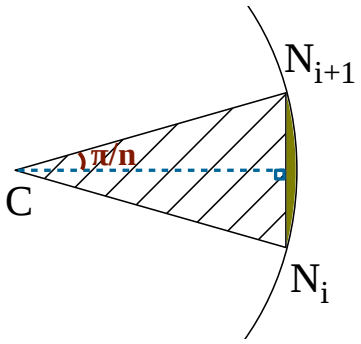


FIGURE 3.14 – Éléments géométriques pour la définition du ratio permettant de prendre en compte l'approximation dans le calcul de la surface de la cellule.

les déformations que subit la cellule rendent ces triangles quelconques, nous calculons donc les aires de ces triangles avec une formule adaptée pour de tels triangles : connaissant les coordonnées des nœuds de la cellule, nous pouvons calculer les distances $CN_{i\%n}$, $CN_{(i+1)\%n}$ et $N_{i\%n}N_{(i+1)\%n}$. Ces distances, une fois triées, nous permettent d'utiliser la formule de Kahan pour calculer l'aire du triangle T_i. La formule de Kahan est une version plus stable numériquement que la formule de Héron dont elle dérive. La forme générale de cette formule est la suivante :

$$T = \frac{1}{4}\sqrt{(a+(b+c))(c-(a-b))(c+(a-b))(a+(b-c))} \tag{3.19}$$

où $a > b > c$. Le parenthèsage peut sembler excessif, mais il est utile lors de l'utilisation de cette formule de manière numérique, car cela permet d'éviter les potentiels problèmes d'associativité de la formule.

Les cellules, selon leur type, ont une surface cible à atteindre. Il est alors possible de comparer la surface « courante » avec cette surface cible. Il faut noter qu'il convient d'adapter cette surface cible avec le $facteur\ d'échelle$ présenté à la relation 3.5 page 69, afin de prendre en compte le fait que la cellule n'a pas une surface identique tout au long de sa vie mais sans pour autant qu'elle subisse des contraintes de compression ou d'étirement. Nous utilisons ensuite le ratio calculé entre la surface courante et la surface cible afin de déterminer si la cellule subit ou non des contraintes de compression ou d'étirement. Il faut noter que, pour des raisons de simplification des calculs, la surface cible correspond à l'air du cercle circonscrit au polygone représentant la cellule. Compte tenu de cette simplification, la surface calculée est légèrement inférieure à la surface telle qu'elle serait si elle avait été calculée en appliquant la relation ΣT_i. Afin de prendre en compte cette légère approximation, nous avons adapté le ratio des cellules ayant pour surface la surface cible. Par exemple, pour une cellule à six nœuds, le ratio est de 1.2. Pour calculer ce ratio, nous utilisons la formule suivante :

$$Ratio = \frac{\pi}{n\ sin(\pi/n)\ cos(\pi/n)} \tag{3.20}$$

où n est le nombre de nœuds de la cellule. Notons que le ratio ne dépend pas du rayon R de la cellule. Cette relation est déduite de la figure 3.14 : soient A_1 l'aire du triangle CN_iN_{i+1}, hachuré sur la figure : $A_1 = Base * Hauteur/2 = R^2 * sin(\pi/n) * cos(\pi/n)$; et A_2 l'aire du triangle CN_iN_{i+1} plus l'aire de la partie en verte sur la figure : $A_2 = {}^{\pi R^2}/_n$. Le ratio d'une cellule à n nœuds est donc ${}^{A_2}/_{A_1}$.
Par exemple pour $n = 6$, on a : $Ratio = \pi/(6 * sin(\pi/6)\ cos(\pi/6)) \simeq 3.14/2.58 \simeq 1.2$.

Afin d'illustrer la capacité des cellules à évaluer les contraintes qu'elles subissent, nous modifions la couleur de celles-ci en fonction du ratio que nous avons calculé. Une telle illustration est visible sur la figure 3.15. Alors que la figure 3.15a montre des cellules avec un rendu consistant simplement à afficher une couleur par type cellulaire, la figure 3.15b montre quant à elle des couleurs adaptées en fonction du ratio $surface\ courante/surface\ cible$ calculé par chacune des cellules virtuelles. On observe que les cellules rouges/orange sont les cellules subissant les contraintes les plus importantes, au centre du tissu virtuel. Les cellules jaunes subissent des contraintes modérées et les cellules vertes subissent peu ou pas de contraintes. La figure 3.15c est un graphique dans lequel les cellules sont représentées par leur coordonnées dans l'environnement à deux dimensions. Nous avons, au moment de capturer l'état du système tel que présenté, exporté les ratios calculés par chaque cellule et utilisé ces données quantitatives dans le graphique.

b) Évaluation des contraintes de cisaillement

Pour évaluer les contraintes de cisaillement qu'elles subissent, les cellules calculent le couple qui s'appliquent à elles. Soit $\overrightarrow{C_k}$ le couple d'une cellule k. Ce couple s'exprime de la manière suivante :

$$\overrightarrow{C_k} = \Sigma(\overrightarrow{CN_i} \wedge \overrightarrow{F_{i,\perp}}) \tag{3.21}$$

où C est le centre de la cellule, N_i est un nœud de la cellule, $\overrightarrow{F_i}$ est la somme des forces appliquées sur le nœud i et $\overrightarrow{F_{i,\perp}}$ est la projection de $\overrightarrow{F_i}$ sur $\vec{v}$. Ces différents vecteurs sont illustrés sur la figure 3.16. En pratique, nous utilisons la valeur absolue de la composante z du vecteur $\overrightarrow{C_k}$ afin d'évaluer le couple d'une cellule : ce qui nous intéresse est simplement la valeur de la force de cisaillement, pas son sens car, dans l'état actuel des connaissances en biologie, le sens du cisaillement n'a pas d'influence sur les mécanismes de mécanotransduction.

Afin d'illustrer l'évaluation des contraintes de cisaillement, nous procédons, de la même manière que pour les contraintes de compression, à une modification de la couleur des cellules qui varie selon le cisaillement que celles-ci subissent, comme on peut le voir sur la figure 3.17. Nous « découpons » un tissu virtuel en couches puis nous faisons se déplacer ces couches dans des sens opposés les unes aux autres afin de générer du cisaillement aux frontières de chaque couche. Nous observons que les cellules orange et roses aux frontières de ces couches subissent effectivement un cisaillement important.

4 Chimie artificielle

Notre modèle inclut également une chimie artificielle, c'est-à-dire un système imitant la chimie réelle : présence d'espèces chimiques différentes, réactions entre ces espèces et diffusion des espèces dans l'environnement. Les chimies artificielles sont, comme la plupart des modèles, des simplifications des systèmes réels. Nous avons choisi comme point de départ la description formelle de la chimie artificielle proposée par Dittrich et coll. (2001) et avons adapté ce formalisme à notre modèle. La formalisation de base selon Dittrich et coll. d'une chimie artificielle est la suivante : une chimie artificielle est un triplet (S, R, A) où $S = \{s_1, s_2, \ldots, s_n\}$, $n \in \mathbb{N}$ est l'ensemble des molécules de la chimie (les espèces chimiques) ; R est l'ensemble des règles de réaction se produisant entre les différentes espèces chimiques et A est l'algorithme permettant de décrire comment les règles de réaction sont appliquées. Ces trois éléments sont détaillés ci-après.

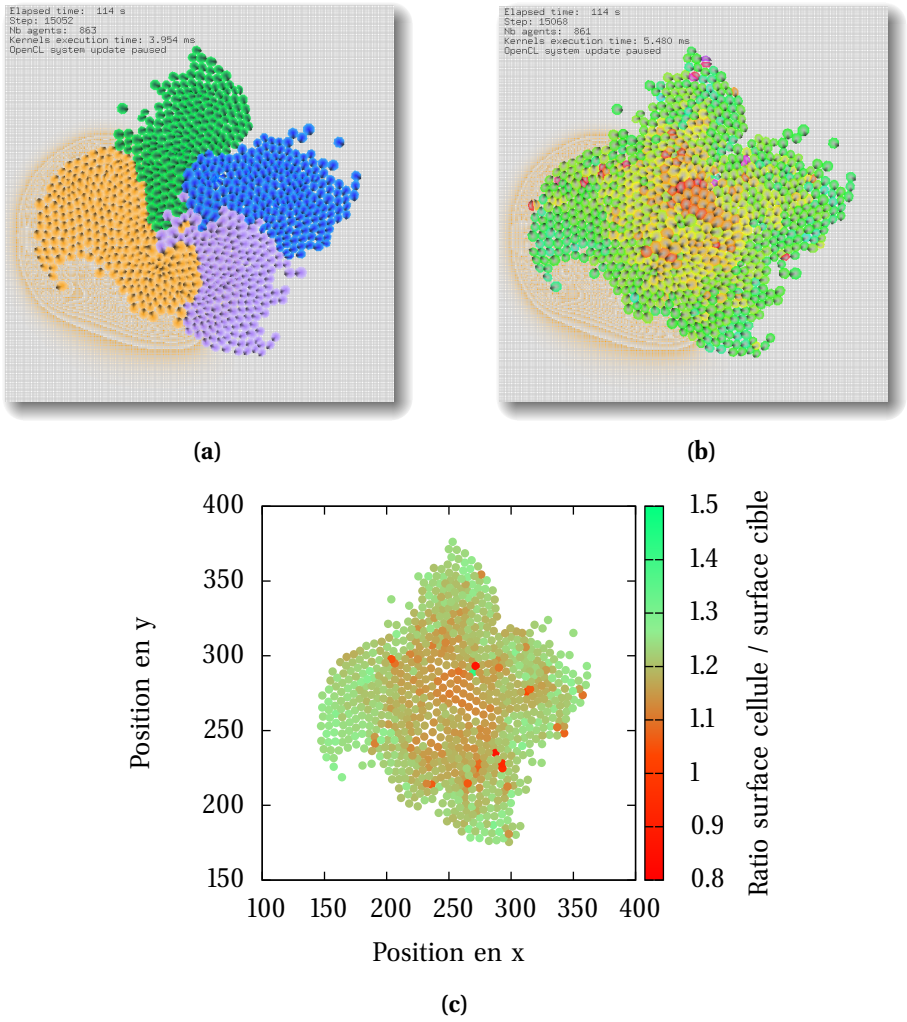


FIGURE 3.15 – Évaluation des contraintes de compression et d'étirement par nos cellules virtuelles : (a) rendu normal, les différentes couleurs dénotent des types cellulaires différents ; (b) mise en avant des contraintes de compression et d'étirement : les cellules orange/rouges sont fortement contraintes, les cellules jaunes sont modérément contraintes et les cellules vertes sont peu voire pas contraintes ; (c) graphique représentant les positions des cellules dans l'espace en deux dimensions : les points sont colorés en fonction de la compression des cellules à partir des données quantitatives exportées lors de la simulation.

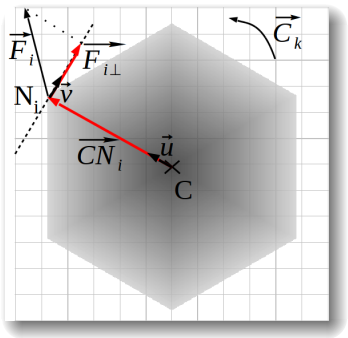


FIGURE 3.16 – Définition des vecteurs utilisés pour calculer le couple appliqué à une cellule afin d'évaluer les contraintes de cisaillement qu'elle subit.

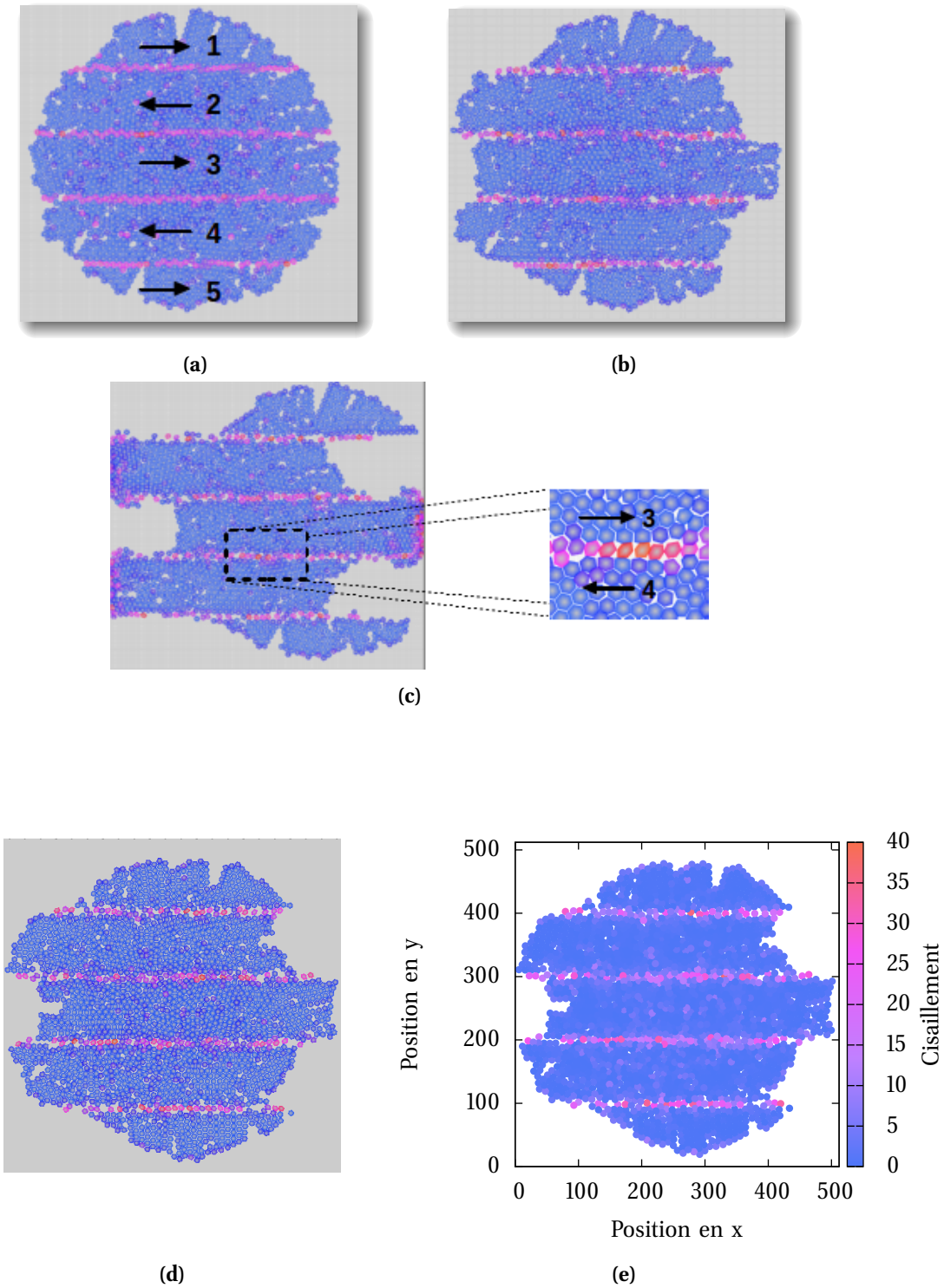


Figure 3.17 – Évaluation des contraintes de cisaillement par les cellules : (a) le tissu artificiel est « coupé » en couches qui se déplacent dans des sens opposés les unes aux autres. (b) et (c) À la frontière de ces couches, on observe que les cellules orange et roses sont des cellules subissant d'importantes contraintes de cisaillement. (d) et (e) : Utilisation de données quantitatives extraites de simulation la simulation présentée en (d) dans le graphique (e) représentant les positions des cellules dans l'espace en 2 dimensions et colorées selon les données extraites.

4.1 Ensemble de réactions

Les règles de réaction permettent de déterminer la façon dont les espèces chimiques interagissent les unes avec les autres. Une règle $r \in R$ s'écrit de la façon suivante :

$$s_1 + s_2 + \cdots + s_n \rightarrow s'_1 + s'_2 + \cdots + s'_m \tag{3.22}$$

Les n réactifs dans la partie gauche de la relation réagissent de façon à produire les m produits de la partie droite de la relation. La relation 3.22 est une réaction d'ordre n. La condition basique pour que cette règle puisse être utilisée est que tous les réactifs de la règle soient présents. D'autres conditions, telles que de la spatialité, des probabilités de réaction ou encore la consommation d'énergie peuvent également exister.

4.2 Application des réactions

L'application des règles de R est réalisée sur un ensemble de molécules P, contenant des molécules de S, possiblement en plusieurs exemplaires pour certaines d'entre elles tandis que d'autres ne seront pas présentes. Différentes méthodes existent afin d'appliquer les règles de réaction.

Collisions moléculaires stochastiques. Dans ce cas, toutes les molécules sont représentées de manière explicite. Un algorithme va sélectionner un échantillon de molécules de P de façon aléatoire et regarder si une règle de réaction est applicable. Si une telle règle existe, sont remises dans P les molécules produites par la réaction. Cette méthode est flexible et permet de prendre en compte avec une relative facilité les conditions supplémentaires évoquées ci-dessus (spatialité, consommation d'énergie, etc.).

Équations différentielles. Une seconde méthode est d'utiliser un système d'équations différentielles ordinaires afin de décrire l'évolution des concentrations des espèces chimiques du système. Dans ce cas, les règles de réaction peuvent s'écrire de la façon suivante :

$$a_1 s_1 + a_2 s_2 + \cdots + a_N s_N \rightarrow b_1 s_1 + b_2 s_2 + \cdots + b_N s_N \tag{3.23}$$

Dans ce cas, les coefficients a_i et b_i sont les coefficients stœchiométriques de la réaction. Ici on considère l'ensemble de toutes les molécules de S. Si une molécule ne se trouve ni parmi les réactifs ni parmi les produits de la réaction, alors son coefficient stœchiométrique a_i (b_i dans le cas des produits) vaut simplement zéro. L'évolution des concentrations des molécules $s_i \in S$ en fonction des règles de réaction $r \in R$ est donc caractérisée par l'équation suivante :

$$\frac{ds_i}{dt} = \sum_{r \in R} \left[(b_i^r - a_i^r) \prod_{j=1}^{N} s_i^{a_j^r} \right] \ , \ i = 1, \ldots, N \tag{3.24}$$

où a^r et b^r sont les coefficients stœchiométriques propres à la règle de réaction r.

Autres approches. Il existe d'autres méthodes pour appliquer les règles de réaction de la chimie artificielle. En particulier, l'approche « métadynamique » stipule que le nombre d'espèces chimiques et de règles de réaction peuvent changer au cours du temps. Un seuil au niveau de la concentration permet de prendre ou non en compte certaines molécules.

En conséquence, les règles de réaction sont également prises en compte de manière variable. Il existe également des approches dites « mixtes » : dans ce cas, certaines molécules sont considérées de manière explicite tandis que l'on considère les concentrations d'autres molécules.

4.3 Adaptation du formalisme pour notre chimie artificielle

Nous avons basé notre modèle de chimie artificielle sur le formalisme proposé par Dittrich et coll.. Dans ce qui suit, nous utilisons un ensemble de quatre molécules $S = \{A, B, C, D\}$. Cet ensemble n'est toutefois pas figé et il est possible en pratique d'utiliser un ensemble plus important d'espèces chimiques. Les règles de réaction sont construites sur le principe de la règle présentée à la relation 3.23, avec l'utilisation de coefficients stœchiométriques. Les règles de réaction ne sont pas figées dans le système, elles peuvent être définies à la volée. Pour le moment, nous faisons cela en interprétant du code informatique « fait à la main » mais il est tout à fait envisageable de générer automatiquement ce code à partir d'une interface de haut niveau permettant de décrire une réaction de manière formelle. Par exemple, on pourrait définir des relations telles que $2A + B \rightarrow C$. Nous verrons dans le chapitre 5 que nous avons déjà inclus une fonctionnalité de génération de code grâce à des fichiers de configuration. Elle ne s'étend cependant pas encore à la définition de réactions pour la chimie artificielle.

Pour appliquer ces règles, nous utilisons un système d'équations aux dérivées partielles, nous permettant de prendre en compte des aspects spatiaux (diffusions et réactions locales). Nous avons déjà présenté, au chapitre 2, une équation basique de réaction / diffusion pour une espèce chimique i. La voici de nouveau pour rappel :

$$\frac{\partial i(x,t)}{\partial t} = D_i \Delta i(x,t) - R_{i(x,t)} \tag{3.25}$$

où D_i est le coefficient de diffusion de l'espèce chimique i ; Δ est l'opérateur laplacien et $R_{i(x,t)}$ sont les réactions locales liées à l'espèce chimique i. En pratique, nous discrétisons cette équation avec la méthode d'Euler explicite. Dans un environnement à deux dimensions, nous définissons $I(x, y)$ comme étant la concentration de l'espèce chimique i à la position (x, y). Nous avons donc, pour la diffusion isotropique de l'espèce chimique i (c'est-à-dire une diffusion uniforme dans toutes les directions), la relation discrète suivante :

$$\begin{aligned} I(x,y) = \quad & (1-\tau)[\delta_i I(x-1,y) \\ & +\delta_i I(x+1,y) \\ & +\delta_i I(x,y-1) \\ & +\delta_i I(x,y+1) \\ & +(1-4*\delta_i) I(x,y)] \end{aligned} \tag{3.26}$$

Où $\tau \in [0, 1]$ est le taux de dégradation de l'espèce chimique i à chaque pas de temps et δ_i est la vitesse de diffusion par pas de temps de l'espèce chimique i. La figure 3.18 est un exemple de diffusion dans un espace à deux dimensions, basée sur la relation 3.26.

Pour illustrer la notion de réaction dans la chimie artificielle, nous utilisons l'exemple de la réaction bimoléculaire suivante : $aA + bB \rightarrow cC + dD$. Les termes a, b, c et d représentent respectivement les coefficients stœchiométriques des espèces chimiques A, B, C et D, et $[A]$, $[B]$, $[C]$ et $[D]$ les concentrations des espèces chimiques de la réaction. Le taux de réaction R de cette réaction correspond au nombre de collisions efficaces entre les réactants et s'exprime de la façon suivante :

$$R = k(T)[A][B] \tag{3.27}$$

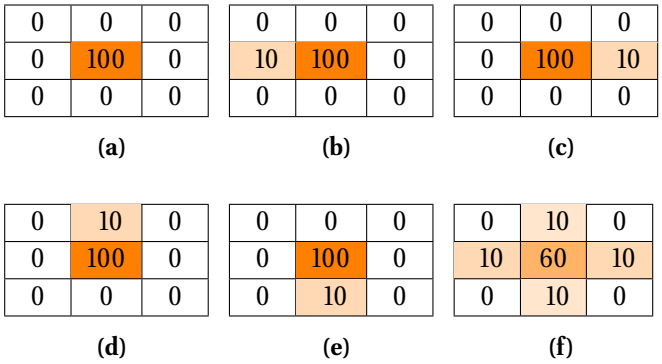


FIGURE 3.18 – Exemple de diffusion en 2 dimensions. (a) L'environnement contient une concentration initiale d'une espèce chimique à la position (x, y) avec $\delta_i = 0.1$. (b) à (e) De manière simultanée, toutes les concentrations de l'environnement sont réévaluées sur la base de l'état de l'environnement initial en (a) et en fonction du coefficient de diffusion de l'espèce chimique ainsi que de son taux de dégradation (dans cet exemple, $\tau = 0$). Dans l'exemple présenté, on considère les concentrations aux positions (x, y), $(x-1, y)$, $(x+1, y)$, $(x, y-1)$ et $(x, y+1)$ car seules ces concentrations vont varier. (f) État final de l'environnement après diffusion.

où $k(T)$ est la constante de vitesse de réaction, dépendant de la température. Le taux de réaction peut s'exprimer en fonction des variations de concentrations au cours du temps. Ce taux de réaction est d'une part négatif pour les réactants (puisque les concentrations de ceux-ci vont diminuer à mesure que les réactions se produisent) et d'autre part positif pour les produits de la réaction (les concentrations des produits augmentent à mesure que les réactions se produisent) (Raff, 2001, chap. 19). Le taux de réaction s'exprime alors comme suit :

$$R = -\frac{1}{a}\frac{d[A]}{dt} = -\frac{1}{b}\frac{d[B]}{dt} = \frac{1}{c}\frac{d[C]}{dt} = \frac{1}{d}\frac{d[D]}{dt} \tag{3.28}$$

Les données nous intéressant sont les variations des concentrations des espèces chimiques de la réaction. En se basant sur les relations 3.27 et 3.28, nous en déduisons les relations suivantes :

$$\begin{aligned} \tfrac{d[A]}{dt} &= -a\,R &&= -a[A][B]\,k(T) \\ \tfrac{d[B]}{dt} &= -b\,R &&= -b[A][B]\,k(T) \\ \tfrac{d[C]}{dt} &= c\,R &&= c[A][B]\,k(T) \\ \tfrac{d[D]}{dt} &= d\,R &&= d[A][B]\,k(T) \end{aligned} \tag{3.29}$$

Il est à noter que les principes utilisés pour modéliser les réactions de notre chimie artificielle ne nous permettent, pour l'instant, de considérer uniquement des réactions d'ordre deux.

Environnement de la chimie artificielle. Les concentrations de molécules sont stockées dans une grille en deux dimensions, ces dimensions étant les mêmes que celles de l'environnement dans lequel évoluent les cellules (pour plus de détails à ce sujet, se reporter à la section 5.4). En pratique, nous utilisons une grille par espèce chimique et au final l'environnement s'apparente à un ensemble de couches superposées tel qu'illustré sur la figure 3.19.

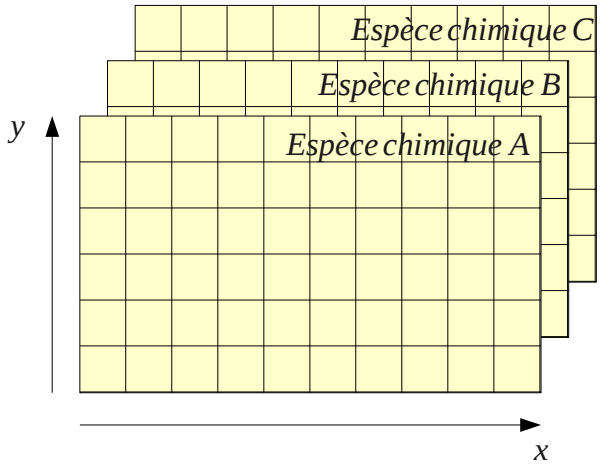


Figure 3.19 – Environnement de la chimie artificielle. Une grille, de mêmes dimensions que celles de l'environnement dans lequel évoluent les cellules, contient les concentrations des différentes espèces chimiques dans l'espace. Chaque espèce chimique dispose de sa propre grille, donnant lieu au final à une superposition de différentes couches.

Consommation et production de molécules par les cellules. La consommation et la production de molécules est un des comportements des cellules virtuelles. Celles-ci peuvent détecter les gradients de molécules présents dans l'environnement grâce à des récepteurs présents sur leur membrane. En pratique, nous n'avons pas modélisé les récepteurs explicitement, mais les nœuds de la membrane jouent ce rôle. En ce qui concerne la production de molécules, les cellules peuvent ajouter des quantités de molécules dans les différentes couches que nous avons présentées au paragraphe précédent. Les quantités de molécules produites sont déposées à la position du nœud central de la cellule, dans la grille correspondant à l'espèce chimique produite.

La section suivante présente le couplage du modèle de cellule avec un système multi-agents qui est à la base de l'implémentation que nous proposons.

5 Couplage à un système multi-agents

Afin de réaliser le couplage de notre modèle à un système multi-agents, il est nécessaire d'examiner différents aspects : la granularité du système, la mémoire des agents, l'ordonnancement des agents lors des cycles de simulation, les interactions entre les agents et, pour finir, l'environnement des agents.

5.1 Granularité du système multi-agents

Le modèle nous ayant servi de base, proposé par Ballet et Tracqui, est couplé à un système multi-agents. En particulier, deux types d'agents sont présents dans le modèle : des agents « physiques », qui représentent des nœuds ou des ressorts de la cellule virtuelle et des agents « composites » permettant soit d'attribuer des comportements à des groupes d'agents physiques ou bien d'ajouter ou de retirer des agents physiques du système. Un agent physique peut appartenir à un ou plusieurs composites. Par exemple, un agent « nœud » appartient à la fois à un composite « membrane » et à un composite « cellule » ; chacun de ces composites

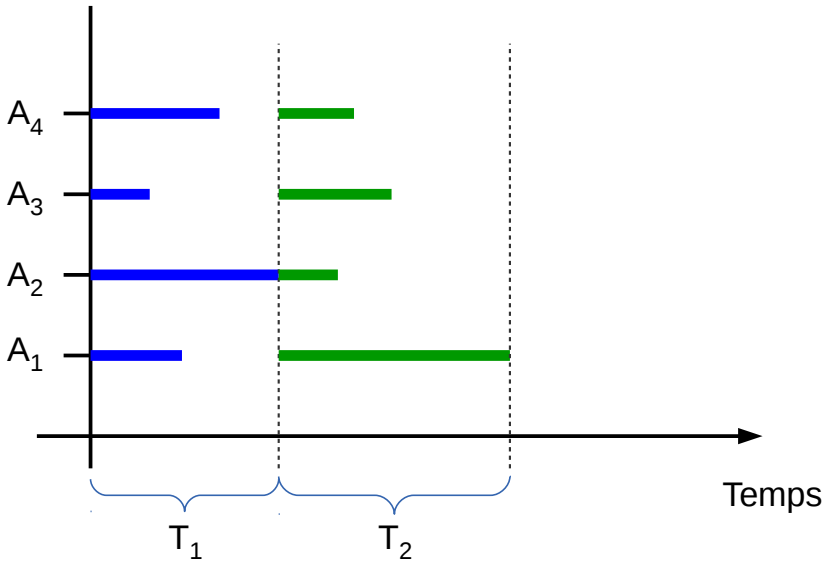


FIGURE 3.20 – Exemple d'ordonnancement des agents de notre système. Sur cet exemple, les quatre agents du système ont deux tâches à réaliser. Les agents sont synchronisés à la fin de chaque tâche, cela signifie que les agents ayant terminé doivent attendre les autres agents avant de commencer la tâche suivante. Une fois que toutes les tâches du cycle ont été réalisées, un nouveau cycle de simulation peut commencer.

ayant des comportements spécifiques. Il existe également des agents physiques « substrat » permettant de mettre en œuvre l'adhésion des cellules virtuelles au substrat dans lequel elles évoluent. Dans ce cas la granularité du système est donc très fine, avec la définition d'agents dans différents niveaux. Une unique cellule est donc constituée d'un certain nombre d'agents qu'il convient d'ordonnancer (c'est-à-dire dont il faut faire exécuter le comportement selon certaines règles).

Quant à nous, nous avons décidé de coupler notre modèle de cellule virtuelle à un système multi-agents dont la granularité sera plus élevée : nous considérons pour notre part que *1 cellule virtuelle = 1 agent du système*. Cette granularité a donc nécessité en particulier d'adapter la mémoire de l'agent, qui est constituée d'un ensemble d'éléments. Cette approche a cependant l'avantage de nécessiter un nombre total d'agents bien moins important que dans le cas ou chaque sous-élément de la cellule est représenté par un agent. Cela apporte notamment plus de simplicité lors de l'ordonnancement des agents du système. En outre, un nombre d'agents plus faible permet aussi de limiter les ressources de calcul utilisées, permettant en fin de compte de pouvoir exécuter plus de cellules.

5.2 Ordonnancement des agents et évolution du système

Nous avons pensé l'ordonnancement de notre système de façon à ce que celui-ci soit cohérent par rapport à l'implémentation sur architecture multi-cœurs que nous avions prévu de réaliser.

La simulation du système est constituée d'une succession de *cycles*. Lors d'un de ces cycles, tous les agents du système vont réaliser leur comportement. L'ordonnancement des actions des agents dans le système peut être réalisée selon deux modes principaux : synchrone et asynchrone.

Dans le cas d'un ordonnancement synchrone, tous les agents du système prévoient leurs actions et calculent leur nouvel état mais attendent que tous les autres agents aient fait de même avant de mettre à jour leur état. En conséquence, cela peut donner lieu à des conflits, les agents respectifs ne sachant pas qu'une ressource n'est pas ou plus disponible, ou bien qu'elle a été modifiée, car la mise à jour de l'environnement n'a pas encore eu lieu.

Dans le cas d'un ordonnancement asynchrone, les agents vont immédiatement mettre leur état à jour après avoir prévu leur action, sans attendre les autres agents. On distingue alors deux façons de considérer les actions des agents : dans le premier elles sont séquentielles, ou plus précisément un agent exécute sont comportement, *puis* un second, *puis* un troisième, etc., jusqu'à ce que tous les agents présents aient effectué leur action. Si cela permet de ne plus avoir de notion de conflits, il peut se produire un biais qui peut être fortement amoindrit en rendant l'ordonnanceur chaotique, permettant ainsi d'ordonner les agents aléatoirement : on peut par exemple imaginer un système dans lequel deux agents a_1 et a_2 veulent accéder à une ressource, la rendant inaccessible durant tout le cycle courant. L'agent a_1 est toujours exécuté avant a_2, causant un biais dans l'exécution du système car a_2 n'aura jamais accès à la ressource. Dans le second cas, les agents exécutent leur comportement de manière simultanée — ce qui, d'ailleurs, est plus proche de systèmes réels : les cellules d'un même tissu agissent de manière autonome sans « attendre » les autres cellules en présence —, ce qui permet de ne plus avoir de biais, mais qui peut causer des conflits si des agents tentent d'accéder à une même ressource en même temps.

Dans notre système, le comportement — et par extension les cycles de simulation — des agents est divisé en différentes tâches. Dans une même tâche, les agents agissent de manière asynchrone et accomplissent la tâche en partant d'un même état de l'environnement. Les tâches par contre sont synchronisées : les agents ne commencent pas la tâche suivante tant que tous les agents n'ont pas terminé la tâche courante. Au final, on peut considérer l'ordonnancement des agents comme *synchronisé*. La figure 3.20 est un exemple d'ordonnancement tel qu'il existe dans notre système : quatre agents, A_1, A_2, A_3 et A_4, ont deux tâches T_1 et T_2 à exécuter pendant leur cycle de simulation. Tous démarrent au même moment, avec les mêmes informations sur l'environnement. Les agents exécutent la tâche avec une vitesse variable, mais les agents ayant terminé doivent attendre que tous les autres agents terminent avant de pouvoir démarrer la tâche suivante. Une fois que toutes les tâches du cycle ont été réalisées par les agents, un nouveau cycle de simulation peut commencer. Pour illustrer cela, prenons comme exemple les deux tâches de base que les cellules virtuelles de notre système doivent réaliser : calculer les forces qui s'appliquent à elles (tâche n^o 1) et application de ces forces sur les nœuds de la cellule (tâche n^o 2). La synchronisation des agents entre ces deux tâches permet de ne pas altérer l'état de l'environnement pendant que chaque agent calcule les forces s'appliquant à lui : l'état de l'environnement est le même pour tous les agents peuplant cet environnement. Cet ordonnancement est particulièrement adapté à une implémentation parallèle qui est par ailleurs l'objet du chapitre 5. En ce qui concerne les conflits pouvant survenir, un accès à une même ressource ne se produit pas sur deux tâches successives mais toujours au sein d'une même tâche. Au sein d'une même tâche, les agents sont asynchrones, donc si l'état de l'environnement doit être modifié, cette modification est répercutée au moment où l'agent effectue son action. Encore une fois, la notion de parallélisme que nous avons évoqué va nécessiter quelques précautions sur les ressources partagées de l'environnement, mais nous ne détaillons pas ces éléments dans le présent chapitre ; ils sont présentés au cours du chapitre 4, section 1.6.

L'évolution temporelle du système est réalisée de manière discrète (inhérente à de tels systèmes) : le temps est discrétisé et le système avance par pas de temps Δt. $t_2 = t1 + \Delta t$

TABLE 3.1 – Mémoire d'un agent

Informations nécessaires à toute simulation
Position des n nœuds de la cellule
Forces appliquées sur les n nœuds de la cellule
État de la cellule[3]
Graine pour le générateur de nombres aléatoires
Vitesse initiale v_0, utilisée pour l'intégration numérique
Autres informations
Âge
Énergie
$facteur\ d'échelle$
Type de la cellule
...

représente le passage du temps t_1 au temps t_2. Le pas de temps que nous considérons est celui que nous avons introduit lors de l'intégration numérique de l'équation différentielle 3.16 dans la relation 3.17. Pour rappel, cette relation permet de calculer les nouvelles positions des nœuds des cellules en fonction des forces appliquées à celles-ci. L'utilisation d'une équation différentielle pour faire évoluer une partie du système nous permet donc, même si dans le cas de simulation numérique celle-ci est discrétisée, d'avoir une « vraie » notion du temps dans notre système : cela permet de faire le lien entre un cycle de simulation et l'évolution temporelle du système.

5.3 Mémoire des agents

Nous avons vu au cours de ce chapitre que notre cellule virtuelle est physiquement composée de différents éléments, dont le nombre est de surcroît variable. Chaque agent doit donc connaître au minimum les positions des nœuds qui compose sa cellule et les forces appliquées à chaque pas de temps sur ses nœuds. Mais chaque cellule est également caractérisée par diverses informations, qui peuvent varier selon le système modélisé. Le tableau 3.1 liste les données minimales nécessaires à toute simulation et donne quelques exemples de données que l'on peut rajouter dans le système (la liste n'est pas exhaustive et est simplement donnée à titre d'exemple).

Les agents accèdent également à des données communes, données qui contiennent les valeurs de l'adhésion cellulaire selon le type de l'agent et les longueurs à vide des éléments de la cellule (segments de membrane, de cytosquelette et de cytosquelette cortical). Il faut savoir que, pour des raisons de cohérence des données, seuls les agents eux-mêmes peuvent modifier leur données « personnelles ». Les données communes sont accessibles seulement pour consultation et ne sont donc pas modifiables. Compte tenu de la nature synchrone de l'ordonnancement de notre système, il peut toutefois y avoir des cas de figure où des ressources accessibles à tous peuvent également faire l'objet de modifications de la part des agents, comme l'environnement : ces cas de figure et leurs résolutions sont détaillés dans le chapitre 5.

3. Nous verrons au cours du chapitre 5 la nécessité de cette donnée, qui ne sert pas uniquement à modéliser le cycle cellulaire de l'agent mais qui sert aussi pour la mise en œuvre d'un algorithme particulier.

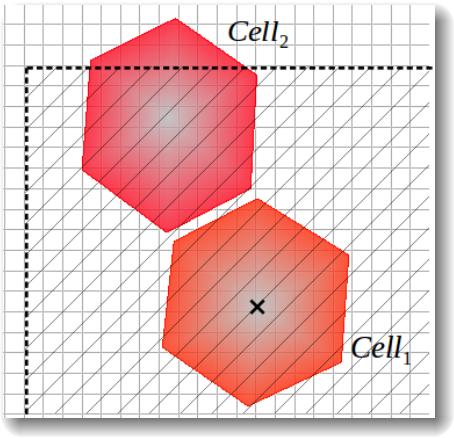


Figure 3.21 – Utilisation d'une grille discrète pour que les cellules détectent leur voisinage immédiat. Les identifiant des cellules sont stockés dans cette grille, à l'emplacement de leur nœud central. Chaque cellule explore une zone, définie autour d'elle : sur la figure, la cellule 1 explore la zone hachurée, qui contient la cellule 2.

5.4 Environnement des agents

Les agents du système évoluent dans un environnement à deux dimensions et borné. Nous avons vu, dans le chapitre 2, qu'un des avantages des systèmes multi-agents concernant l'environnement est que celui-ci peut être continu ou discret. Nous avons choisi d'utiliser un environnement que l'on considère tantôt comme continu, tantôt comme discret : nous pouvons donc qualifier cet environnement d'hybride. L'aspect continu est dû au fait que les déplacements des cellules dans l'environnement sont réalisés grâce à la résolution numérique de l'équation différentielle 3.16.

Nous utilisons également une grille discrète superposée à notre environnement qui nous sert à établir les voisinages des agents. Cette grille est illustrée sur la figure 3.21 : les cellules examinent leur environnement local afin d'y détecter d'éventuelles cellules voisines. Dans le cas de cette illustration, la zone hachurée est examinée par la cellule 1 : la cellule 2 fait partie de son voisinage (et réciproquement). La seule information que nous enregistrons dans cette grille est l'identifiant de la cellule qui est placé aux coordonnées du centre de la cellule : $\lfloor \vec{x} \rfloor$. Cela permet de limiter les informations stockées dans la grille : l'identifiant seul de la cellule permet d'accéder à tout type d'information concernant la cellule. La taille de la zone à explorer doit avoir un rayon d'au moins deux fois le rayon des cellules.

Cette méthode permet de conserver la notion d'environnement local propre aux systèmes multi-agents. De plus, cela permet de ne pas avoir à parcourir la liste de toutes les cellules présentes afin de dresser et de mettre à jour une liste des voisinages des cellules.

L'utilisation d'une telle grille a un coût en matière de mémoire. Cependant, aujourd'hui la quantité de mémoire disponible sur les ordinateurs augmente bien plus rapidement que leurs ressources de calcul. Nous avons donc choisi d'utiliser davantage de ressources mémoire afin de gagner en efficacité en terme de calcul.

5.5 Interactions des agents

Les interactions des agents se résument aux interactions cellulaires que nous avons présentées dans la section 3.2 : interactions cellules / cellules et cellules / environnement. Profitons-en pour revenir sur le principe des actions réciproques, stipulant que $\vec{F}_{A/B} =$

$-\vec{F}_{B/A}$, que nous avions évoqué dans la section 3.2. Dans le cas des interactions intracellulaires, nous avions précisé que, lors du calcul d'une force d'un nœud A vers un nœud B, nous ajoutions explicitement une force de même intensité, mais de direction opposée, au bilan des forces du nœud B. Ce principe est également à l'œuvre dans le cas des interactions cellules / cellules : dans ce cas il est implicite, car chaque agent effectue son bilan des forces : nous avons vu qu'un agent C_1 ayant pour voisin un agent C_2 est lui-même le voisin de C_2 du point de vue de ce dernier. L'ordonnancement du système étant synchrone, ces deux agents en interaction disposent exactement des mêmes informations globales au temps t : les forces issues des interactions cellules / cellules entre ces deux agents seront donc identiques au signe près et le principe des actions réciproques est bien appliqué.

6 Synthèse du chapitre

Ce chapitre était consacré à la présentation de notre modèle pour l'étude de la morphogenèse de tissus. Partant du niveau mésoscopique, nous proposons un modèle mécanique de cellule virtuelle bâti autour du formalisme des systèmes multi-agents. Notre modèle présente quatre aspects distincts : la structure physique de la cellule virtuelle, la formalisation des différents comportements de la cellule virtuelle, la chimie artificielle et le couplage du modèle à un système multi-agents. Ces aspects sont présentés synthétiquement dans le tableau 3.2.

6.1 Paramètres du système

Le système dans sa globalité comporte un certain nombre de paramètres. Chaque cellule du système peut théoriquement avoir un jeu de paramètres distinct des autres. Cependant, pour des raisons de simplification, nous avons défini un jeu de paramètres commun pour tous les agents du système. Cela n'empêche pas de personnaliser ces paramètres pour certains agents si cela est nécessaire, par exemple pour définir des types cellulaires distincts (taille, rigidité de la structure cellulaire, etc.). Les paramètres pour l'adhésion cellulaire ne sont pas fixés mais sont à définir en fonction des simulations réalisées.

Le tableau 3.3 présente les paramètres principaux de notre système fixés sur la base de données biologiques extraites de la littérature. Ces valeurs sont à priori valides et figées dans le système, réduisant le nombre de paramètres à déterminer effectivement. Les valeurs que l'on donne sont le plus souvent des plages, puisque certain paramètres dépendent du type cellulaire. En ce qui concerne l'adhésion, une valeur d'adhésion, telle que définie à la section 3.2 b), est associée entre chaque type cellulaire. De plus, ces valeurs ne sont pas nécessairement réflexives ; autrement dit on peut avoir $adhesion_{type_1,type_2} \neq adhesion_{type_2,type_1}$.

6.2 Conclusions

Ce chapitre nous a permis de formaliser les différents éléments de notre modèle. En effet, celui-ci est au final relativement complexe dans le sens où différents éléments sont couplés pour donner lieu au modèle final. La cellule virtuelle est une cellule biomécanique basée sur une forme complexe déformable constituée de nœuds reliés par des liens élastiques. Les comportements cellulaires principaux sont inclus dans le modèle : un cycle cellulaire simplifié inclut la croissance et la division des cellules. Nous avons modélisés, à l'aide de forces physiques, différentes interactions : entre cellules, et entre cellule et environnement. Nous avons également inclus des éléments de mécanotransduction, c'est-à-dire la capacité

Table 3.2 – Synthèse de notre modèle de cellule virtuelle

Éléments du modèles		**Remarques**
Structure	Système masses / ressorts	Modélisation de différents éléments de la cellule biologique : membrane, cytosquelette, cortex ; éléments géométriques pour construire une cellule à n nœuds
Comportements cellulaires	Cycle cellulaire simplifié, croissance des cellules et division cellulaire	Automate à états pour le cycle cellulaire ; modèle de division généralisé pour des cellules dont le nombre de nœuds peut être modifié d'une simulation à une autre
	Interactions cellulaires : cellule / cellule ; cellule / environnement	Utilisation de forces mécaniques ; adhésion cellule / cellule modélisée de manière originale à l'aide d'un produit scalaire ; coefficient de fixation pour l'adhérence à la matrice extra-cellulaire ; mouvements des cellules calculés avec la 2^{e} loi de Newton
	Évaluation des contraintes mécaniques : compression, étirement, cisaillement	Modélisé grâce à des opérations géométriques simples (formule de Kahan, produit vectoriel)
Chimie artificielle	Chimie simple, basée sur le formalisme de Dittrich et coll.	Réaction et diffusion modélisées avec un système d'équations aux dérivées partielles discrétisées dans une grille à 2 dimensions
Couplage à un SMA	Granularité	1 cellule = 1 agent : ressources de calcul utilisées moindres que lorsque la granularité est plus fine (par exemple 1 agent = 1 nœud de la cellule)
	Mémoire	Données « de base », comme la position des nœuds de la cellule, et données de nature variable selon les simulations
	Ordonnancement	Cycles de simulation découpés en tâches ; ordonnancement synchrone des agents entre les tâches ; ordonnancement asynchrone des agents au sein d'une même tâche
	Environnement	Environnement continu, mais utilisation d'une grille discrète contenant les identifiants des cellules, permettant d'accéder facilement aux voisinages des celles-ci
	Interactions	Application des interactions cellulaires décrites dans la section 3.2

TABLE 3.3 – Paramètres de notre système

Paramètre	**Valeur**
Raideur des ressorts $k_{membrane}$, $k_{cytosquelette}$, k_{cortex}	1 à 3 $\mu N.\mu m^{-1}$ (Mogilner et Oster, 2003) L'ordre de grandeur est plus élevé que dans la référence car nous considérons, dans le cadre de notre modèle, des groupes de 10^3 à 10^4 liaisons.
Rayon cible des cellules	De 4 à 10 μm (Rosenbluth et coll., 2006; Krombach et coll., 1997)
Masse d'un nœud	1 pg (Park et coll., 2008)
Pas d'intégration Δt	0.2 min
Distance minimale d_{min} pour que deux cellules se répulsent	10 μm
Distance maximale d_{max} à laquelle l'adhésion entre deux cellules a encore lieu	Quelques μm ; variable selon les types cellulaires en présence
Vitesse de diffusion des molécules δ_i, par unité de temps	De l'ordre de 1 $mm.min^{-1}$ (Yu et coll., 2009) Nous considérons une diffusion uniforme dans 4 directions (haut, bas, droite, gauche) ; l'environnement dans lequel les molécules diffusent est discret et chaque case a une surface de $1\mu m^2$. Pour une diffusion de 50 $\mu m^2.s^{-1}$, on a, dans une direction, une diffusion de 12.5 $\mu m.s^{-1}$, soit 750 $\mu m.min^{-1} = 0.75\ mm.min^{-1}$
Dégradation des molécules τ, par unité de temps	De l'ordre de 0.001 molécules par minute Nous nous sommes basés sur une durée de vie d'approximativement 10h pour une molécule d'eau (Dellago, 2005)

des cellules à évaluer les contraintes physiques qu'elles subissent afin d'adapter leur comportement : nos cellules virtuelles peuvent évaluer les contraintes de compression, d'étirement et de cisaillement issues de leur environnement. Les interactions cellulaires sont modélisées avec des forces. Nous avons utilisé de simples opérations géométriques nous permettant de modéliser de manière efficace certains processus complexes, comme l'adhésion cellulaire. Nous avons détaillé le couplage du modèle de cellule à un système multi-agents à travers différents aspects, synthétisés dans le tableau récapitulatif 3.2. Les paramètres de notre système sont des données biologiques tirées de la littérature et permettent de limiter le nombre de degrés de liberté du système, malgré le nombre potentiellement important d'agents qu'il peut contenir.

Un point que nous n'avons pas abordé jusqu'à maintenant est le temps de calcul nécessaire pour simuler le système multi-agents, à cause du nombre d'entités dont il faut exécuter le comportement. En effet, un tissu biologique comporte, au mieux, des milliers de cellules en interactions. Pour faire le passage à l'échelle d'une cellule à un groupe potentiellement important de cellules, on pourrait imaginer que le modèle le plus adapté est un système d'équations différentielles car cela permet de considérer la population de cellules dans son ensemble. Nous avons cependant insisté sur la notion d'interactions locales et sur la capacité d'un individu à influencer le groupe ou une partie du groupe. De ce point de vue, les systèmes multi-agents sont mieux adaptés. Nous avons alors pensé le modèle dans son ensemble afin de pouvoir en réaliser une implémentation efficace. En particulier, nous nous sommes fixés comme but de l'implémenter sur architectures parallèles afin de pouvoir disposer d'une puissance de calcul plus importante permettant de réduire les temps de calculs requis pour simuler un système multi-agents comportant beaucoup d'entités.

On peut considérer chaque agent comme un programme informatique indépendant des autres (dans une certaine mesure), et cela nous conduit alors naturellement vers l'utilisation d'architectures matérielles parallèles pour exécuter les simulations du modèle. Ces architectures offrent aujourd'hui une puissance de calcul très importante pour un coût en euros très bas : c'est en particulier vrai pour les processeurs graphiques, ou GPU (Graphics Processing Units), qui comportent jusqu'à plusieurs milliers d'unités de calculs fonctionnant en parallèle. Ce type de matériel est présent dans tous les ordinateurs grand public, ce qui le rend intéressant à utiliser puisque qu'il est disponible facilement. Cependant, il faut utiliser un langage de programmation particulier pour programmer des processeurs graphiques. Nous nous sommes en particulier intéressés au langage OpenCL car il permet de programmer une large gamme de matériel qui ne s'arrête pas aux cartes graphiques mais inclut notamment les micros processeurs et, depuis récemment, les plateformes reconfigurables que sont les FPGA.

Le prochain chapitre est consacré aux intérêts de la programmation parallèle. Nous y présentons également différents langages et plateformes de développement pour programmer en parallèle différents matériels, et nous revenons plus en détail sur la plateforme de développement OpenCL.

Deuxième partie

Simulateurs parallèles, applications et résultats

Introduction à la partie II

La seconde partie de ce document est consacrée à la simulation de modèles sur processeurs multi-cœurs. Le chapitre 4 dresse une revue des différentes plateformes que l'on peut utiliser pour programmer des architectures parallèles. Cette revue nous a amené à choisir OpenCL comme base sur laquelle faire reposer notre implémentation parallèle. Le chapitre 5 est dédié à la présentation du simulateur parallèle que nous proposons, non sans avoir au préalable étudié l'état de l'art concernant les simulateurs que l'on trouve dans la littérature. Ce chapitre se termine par la présentation de cas d'étude que nous avons réalisés avec notre modèle de cellule virtuelle dans le simulateur parallèle que nous proposons. Cela nous permet de mettre en avant l'expressivité de notre modèle à travers des cas d'étude variés, tout en apportant des éléments de validation aux différents éléments inclus dans notre modèle. Enfin, nous présentons également une étude de performances de notre simulateur parallèle.

CHAPITRE 4

Intérêts de la programmation parallèle

Ce chapitre traite de la programmation parallèle et de ses intérêts, ceci dans un cadre général et dans le cadre de cette thèse en particulier. Dans un premier temps, nous proposons une introduction à la notion de programme parallèle. Dans un second temps, nous présentons quelques bibliothèques pour la programmation parallèle, en mettant l'accent sur les outils pour la programmation de processeurs graphiques qui sont les architectures nous ayant le plus intéressées de prime abord, car elles sont hautement parallèles, même si nous avons fini par privilégier les systèmes hétérogènes de manière générale. Le reste du chapitre est quant à lui consacré, d'une part, à la présentation des cartes graphiques en se plaçant à un assez haut niveau et, d'autre part, à la présentation d'OpenCL, qui nous permet de mettre en évidence des éléments qui nous ont finalement amené à proposer l'architecture logicielle que nous verrons au chapitre 5.

Sommaire

1 Qu'est-ce que la programmation parallèle ?

1.1 Introduction générale au parallélisme

La notion de parallélisme, au sens général du terme, consiste en l'exécution de tâches de manière simultanée. En général, il s'agit de sous tâches d'un problème. En guise d'illustration, prenons l'exemple d'une recette de cuisine. Ce type d'exemple est déjà un parallèle souvent utilisé avec l'algorithmique de façon générale, car il s'agit d'une succession d'étapes (d'instructions) à réaliser. Paralléliser un problème consiste à extraire un ensemble d'étapes (possiblement toutes) qu'il est possible de réaliser en même temps. Par exemple pour faire une mousse au chocolat, il faut d'un côté faire fondre du chocolat avec du beurre (tâche T_1) et de l'autre monter des blancs d'œufs en neige (tâche T_2) : ces deux étapes (ou instructions ou tâches) sont *indépendantes*. La troisième étape consiste à incorporer le chocolat fondu dans les blancs d'œufs (tâche T_3). Cette étape ne peut être réalisée qu'après que les étapes 1 et 2 soient elles mêmes terminées : dans ce cas on parle de *dépendance*. On peut donc réaliser cette recette de deux manières différentes : on peut réaliser les tâches T_1, T_2 et T_3 successivement les unes aux autres (figure 4.1a) : dans ce cas on parle de *séquentialité*. On peut réaliser les tâches indépendantes simultanément (figure 4.1b) : dans ce cas il s'agit de *parallélisme*.

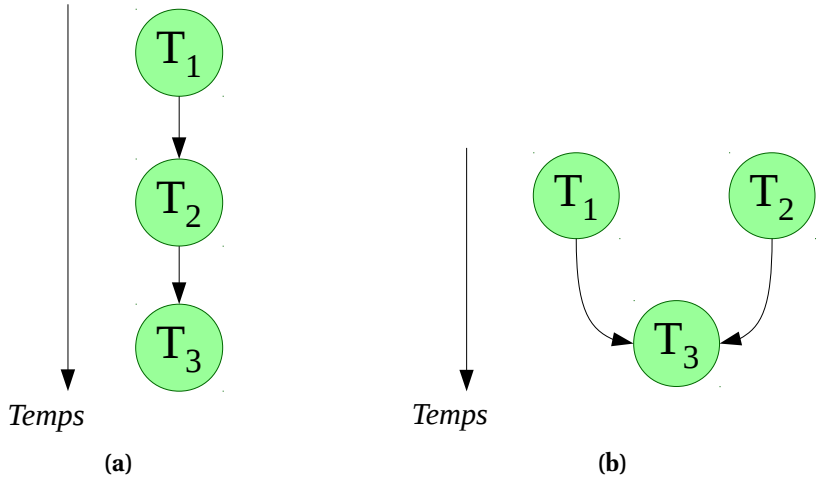


FIGURE 4.1 – Notion de *séquentialité* et de *parallélisme*. (a) Les trois tâches T_1, T_2 et T_3 sont réalisées séquentiellement. (b) Les tâches T_1 et T_2 sont réalisées en parallèle.

Cet exemple est cependant incomplet : le problème a été découpé en sous-tâches, certaines sont indépendantes et de fait peuvent être réalisées en parallèle. Mais avec quelles ressources physiques ? Lors d'une recette, on peut vouloir faire chauffer plusieurs éléments potentiellement en même temps, mais cela nécessite d'avoir plusieurs ustensiles. Le problème est similaire avec le découpage d'un programme informatique : la réalisation parallèle de tâches nécessite l'emploi d'un matériel adapté.

1.2 Parallélisation d'un programme

Il existe deux manières principales pour paralléliser un programme. En premier lieu, le parallélisme de tâches est celui que l'on vient d'évoquer pour introduire la notion de parallélisme : il s'agit d'identifier des tâches d'un problème pouvant être réalisées en même temps. Deuxièmement, le parallélisme de données consiste quant à lui à exécuter un *même traitement sur des données différentes*, par exemple chauffer (*le traitement*) des ingrédients (*des données*) différents. La figure 4.2 est un exemple de parallélisme de données : on applique sur une image un traitement pixel par pixel (en l'occurence une désaturation) : chaque pixel peut être traité indépendamment des autres. Le parallélisme de flux découle du parallélisme de données. Dans ce cas, une succession de traitements est appliqué à des données ; on utilise également le terme de « pipeline » dans ce cas.

Dans la pratique, le plus souvent un programme ne sera pas entièrement parallèle : il existera des dépendances entre certaines tâches. Le *graphe de dépendances* illustré sur la figure 4.1 illustre cette idée de dépendances partielles : sur la figure (b), illustrant l'exécution parallèle du problème, les tâches T_1 et T_2 sont indépendantes, et peuvent donc être réalisées en même temps. La tâche T_3 dépend par contre des deux tâches T_1 et T_2.

La parallélisation d'un problème dépend à la fois du problème en lui même (le problème est-il décomposable en tâches indépendantes ? peut-on exploiter du parallélisme de données ? de flux ?) ainsi que des ressources de calcul à disposition (autrement dit, le matériel).

Figure 4.2 – Désaturation d'une image. Chaque pixel peut être traité de manière indépendante par un processus.

1.3 Pourquoi calculer en parallèle ?

La première application du calcul parallèle concerne la simulation. Ces simulations peuvent concerner des phénomènes dont l'expérimentation réelle peut s'avérer : dangereuse (armement, pharmacologie), lente (mouvements d'étoiles), complexe (multi-échelles) et onéreuse.

L'avènement de la programmation parallèle est due à la stagnation de la puissance du matériel : la fréquence du matériel, permettant de définir le nombre d'opérations que l'on peut réaliser par seconde, stagne. La loi de Moore, stipulant que le nombre de transistors gravés sur les puces double tous les deux ans, est arrivée à ses limites : depuis quelques années, la fréquence des micro-processeurs stagne autour de 3 ou 4 giga hertz. Augmenter cette fréquence signifie augmenter la consommation d'énergie et la quantité de chaleur dissipée par la puce. Cependant, les applications informatiques demandent toujours plus de capacités de calcul. La solution utilisée pour pallier ces limites consiste à multiplier les unités de calcul plutôt que d'accroître la puissance d'une unité individuelle. Finalement, en 2014 le parallélisme est partout : ordinateurs de bureau, ordinateurs portables, smartphones et tablettes sont tous équipés, à minima, de processeurs multi-cœurs. Malgré l'omniprésence du matériel parallèle, la programmation parallèle reste plus complexe que la programmation séquentielle « classique » : parallélisation du problème, granularité du parallélisme, placement mémoire, ressources partagées et mise en place de mécanismes de synchronisation ainsi que prise en compte des différentes architectures matérielles sont les principales difficultés liées à la programmation parallèle.

1.4 Évaluation d'un programme parallèle

L'évaluation d'un programme parallèle se réalise en calculant son accélération par rapport à sa version séquentielle :

$$S_p = \frac{T_1}{T_p} \tag{4.1}$$

où T_1 est le temps requis pour exécuter le programme en séquentiel et T_p est le temps requis pour exécuter le programme sur p unités de calcul. Idéalement, $S_p = p$. En pratique,

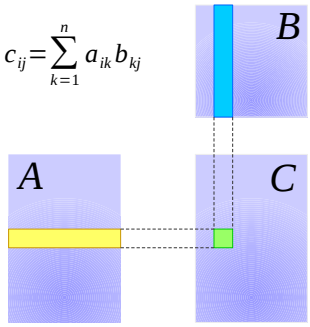


FIGURE 4.3 – Illustration d'un produit de deux matrices : $c_{i,j} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Sur la figure, c_{ij} est identifié en vert ; la ligne a_{ik} est identifiée en jaune et la colonne b_{kj} est identifiée en bleu.

c'est rarement, voire jamais, le cas. En effet, un programme conserve souvent des portions de code séquentiel et la mémoire est un goulet d'étranglement important. En outre, le travail n'est peut-être pas réparti équitablement (*équilibrage de charge*) sur toutes les unités de calcul (*partage de charge*).

Un exemple idéal pour illustrer la notion d'accélération est le produit de deux matrices. Soient deux matrices $A = (a_{ij})$ et $B = (b_{ij})$ de dimensions respectives $n * m$ et $m * p$. Le produit de ces deux matrices est : $C = c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$, de dimensions $n * p$. Le produit de matrices est illustré sur la figure 4.3.

La figure 4.4 (Guevel et coll., 2013) montre l'accélération obtenue avec des versions parallèles de la multiplication de matrices par rapport à une version séquentielle. Les versions parallèles sont exécutées sur un processeurs multi-cœurs et sur une carte graphique : ce dernier type de matériel peut contenir jusqu'à plusieurs milliers de cœurs ; nous reviendrons sur ces aspects dans la suite de ce chapitre. La version parallèle « basique » d'une multiplication de matrices consiste à affecter $n * p$ processus qui vont chacun calculer une case de la matrice résultante $c_{i,j}$. Les courbes de la figure 4.4 contiennent beaucoup d'informations, sur lesquelles nous revenons un peu plus bas. Pour l'heure, intéressons nous aux courbes `seq` (en noir), `cpu_global` [1] (en bleu) et `gpu_global` (en cyan). Pour des matrices de 2000 cases, l'exécution du programme sur le CPU multi-cœurs offre une accélération de 10 tandis qu'exécuté sur le GPU de carte graphique, l'accélération obtenue est presque de 1000.

Les autres courbes ont été réalisées avec des versions du programme utilisant différemment les ressources de calcul et la mémoire du matériel, induisant des coûts de communication pouvant être soit avantageux soit pénalisant. L'idée de ces mesures de temps d'exécution était d'évaluer le gain lors de l'utilisation du cache sur le matériel (pour plus de détails concernant l'utilisation de la mémoire des CPU et GPU, se reporter aux sections 3.3 et 4.3 de ce chapitre). Pour résumer, nous avons observé que pour le traitement d'un nombre important de données, il faut maximiser la taille du cache utilisé afin de ne pas être pénalisé par le coût important du transfert des données depuis la mémoire principale vers le cache et inversement. Si la taille du cache est trop petite, alors il devient plus avantageux de ne pas recopier les données et de les utiliser directement depuis la mémoire principale (voir les courbes `cpu_global` et `cpu_local2` ainsi que les courbes `gpu_global` et `gpu_local2`).

1. La dénomination « global » signifie que c'est la mémoire principale du matériel qui a été utilisée, tandis que la dénomination « localN » signifie qu'une portion de cache de taille N*N a été utilisé.

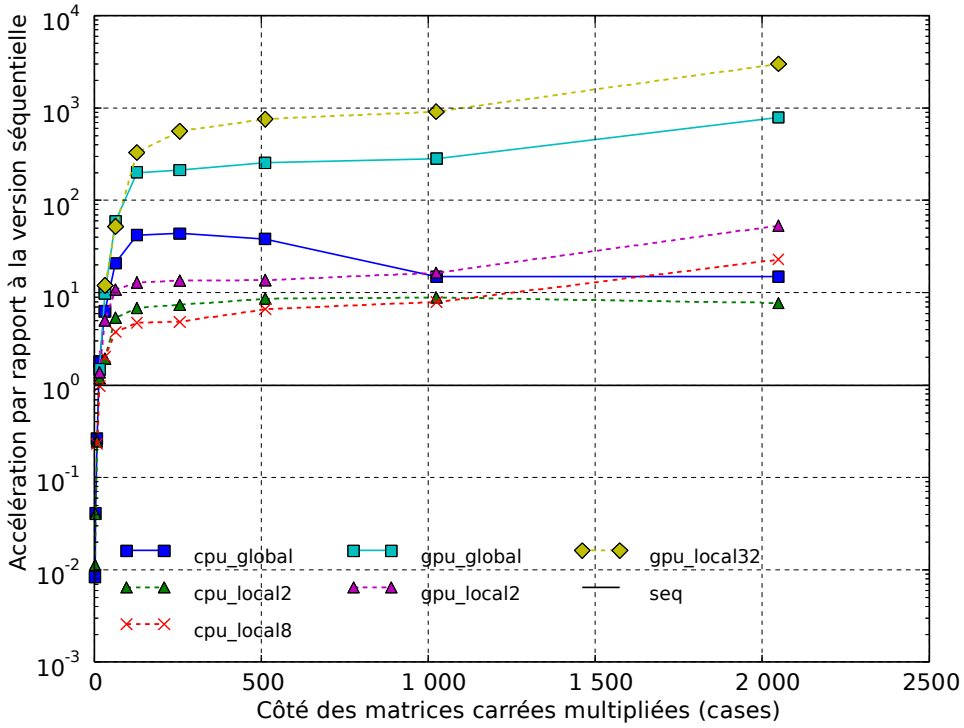


Figure 4.4 – Accélération d'un programme parallèle réalisant une multiplication de matrices. Différents matériels sont utilisés (CPU, GPU) et les différentes utilisations des ressources de calcul et de placement mémoire induisent des performances différentes. Figure extraite de (Guevel et coll., 2013).

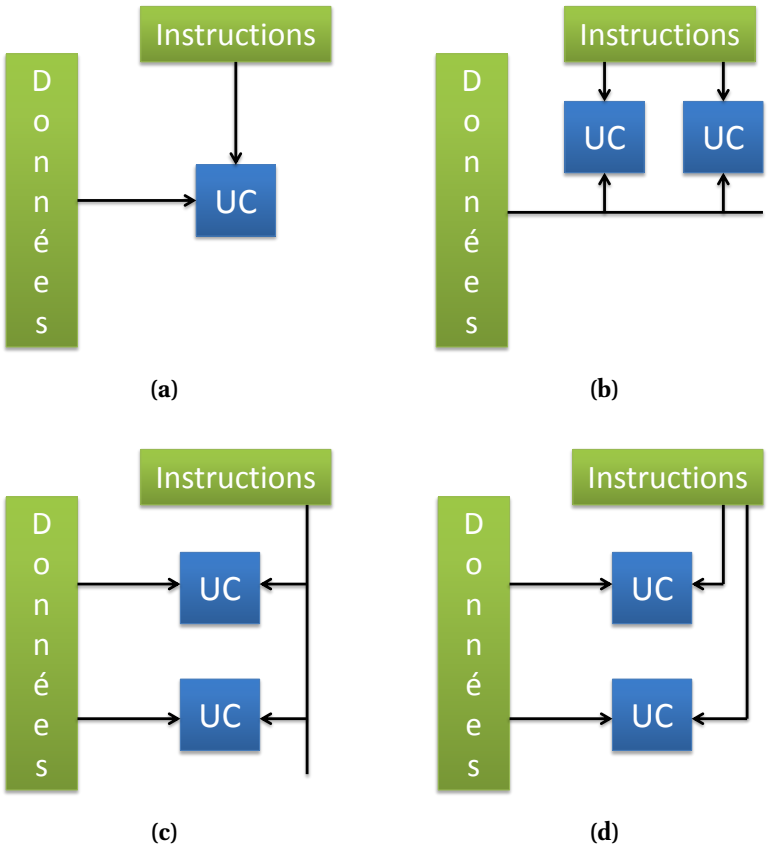
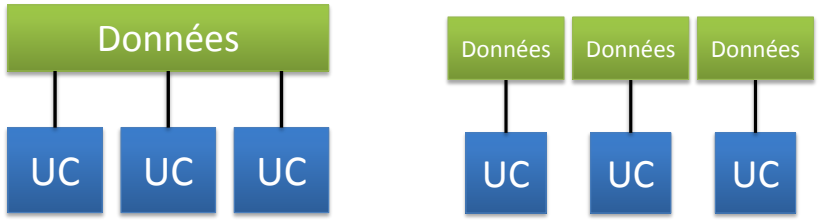


FIGURE 4.5 – Taxinomie de Flynn. L'acronyme UC signifie « Unité de Calul ». (a) Single Instruction Single Data (SISD) ; (b) Multiple Instructions Single Data (MISD) ; (c) Single Instruction Multiple Data (SIMD) ; (d) Multiple Instructions Multiple Data (MIMD).

1.5 Classification de Flynn

La classification proposée par Flynn (1972) permet de décomposer les architectures matérielles en quatre différentes catégories, en fonction des flux d'instructions et de données de celles-ci. La catégorie SISD (*Single Instruction Single Data*) correspond à une architecture mono-processeur purement séquentielle. Une seconde catégorie, MISD (*Multiple Instructions Single Data*) permet d'exécuter un flux d'instructions sur une donnée, comme dans le cas d'un pipeline. Dans le cas de la catégorie SIMD (*Single Instruction Multiple Data*), une même instruction est exécutée sur un ensemble de données. Cette catégorie correspond par exemple aux architectures des cartes graphiques. Enfin, la catégorie MIMD (*Multiple Instructions Multiple Data*) correspond au niveau le plus global de parallélisme puisqu'ici un flux d'instructions est appliqué à des données différentes. Ces différentes catégories sont illustrées sur la figure 4.5.

Depuis la taxinomie proposée par Flynn, d'autres catégories ont été proposées. La catégorie SPMD (*Single Program Multiple Data*) est en particulier intéressante : il s'agit d'une certaine manière d'une extension de la catégorie MIMD, l'idée étant de fournir aux unités de calcul un programme (une suite d'instructions) à exécuter sur les données ; ces instructions n'étant pas nécessairement synchronisées. Cependant il existe une variété importante de catégories proposées depuis la taxinomie de Flynn, telles que les catégories proposées par Shami et Hemani (2012) qui couvrent les architectures parallèles et reconfigurables.



(a) (b)

Figure 4.6 – Mémoire des architectures parallèles. (a) Mémoire partagée par toutes les unités de calcul. (b) Mémoire distribuée entre toutes les unités de calcul.

1.6 Accès mémoire

Selon les architectures, la mémoire peut être soit partagée soit distribuée. Dans le premier cas la mémoire est partagée (figure 4.6a), les unités de calcul (UC) sont indépendantes et communiquent — avec des opérations de lecture et d'écriture — via une mémoire unique. L'usage de ce type de mémoire peut poser des problèmes de cohérence, par exemple si deux UC écrivent au même emplacement mémoire en même temps. La présence d'un cache sur chaque UC permet d'optimiser certains accès mais peut également causer des problèmes de cohérence de la mémoire, si la mémoire partagée est modifiée et que le cache d'UC n'est pas mis à jour. Le verrouillage de la ressource avec une opération atomique garantit que seul un processus à la fois accède à cette ressource : ces opérations sont « indivisibles » ; le processus ayant débuté la procédure de verrouillage ne pouvant être interrompu.

Ces problèmes de cohérence nécessitent l'emploi de synchronisation. Différentes formes de synchronisation existent ; les deux plus employées sont les barrières et les sémaphores binaires. Une barrière est un point de rendez-vous d'un groupe de processus : les processus atteignant la barrière doivent attendre que tous les autres processus soient rendus à ce point avant de continuer. Nous avons déjà évoqué cette méthode de synchronisation dans le chapitre 3, dans la section 5.2 page 86 : même s'il s'agissait de l'ordonnancement des agents de notre système, la finalité était bien de permettre aux agents d'accéder à des données dont on garantit la cohérence à mesure de l'exécution des agents.

Les sémaphores binaires, où mutex, permettent de créer une section critique dans le code : un seul processus à la fois peut accéder à la section critique. Une opération atomique permet de verrouiller le sémaphore afin d'empêcher tout autre processus d'entrer dans la section critique. Une fois exécuté le code de la section critique, le processus déverrouille la ressource qui redevient alors accessible.

Il faut garder à l'esprit que les opérations de synchronisation peuvent être à l'origine d'*interblocages*, c'est-à-dire de situations où au moins deux processus vont se bloquer dans l'attente d'une ressource. Un exemple classique pour illustrer la notion d'interblocage est le suivant : deux processus P_1 et P_2 souhaitent accéder à deux ressources A et B, dans l'ordre A et B pour P_1 et dans l'ordre B et A pour P_2. Il peut se produire la situation suivante : P_1 verrouille l'accès à A, P_2 verrouille l'accès à B. Pour continuer, P_1 a besoin de l'accès à B, mais cette ressource est verrouillée par P_2 qui, pour continuer, doit lui accéder à la ressource A : les deux processus se bloquent mutuellement.

Dans le cas d'architectures à mémoire distribuée, la mémoire est distribuée à toutes les unités de calcul (figure 4.6b). L'échange d'informations entre les unités de calcul se fait alors par passage de messages. La communication peut être réalisée de manière point à point (d'une unité de calcul à une autre, toutes les autres unités ignorant cette communication)

ou par diffusion (*broadcast*) dans les deux cas suivants : une unité de calcul envoie un message à toutes les autres ou bien toutes les unités de calcul envoient un message vers toutes les autres. Les communications peuvent être synchrones ou asynchrones. La mémoire propre à chaque unité de calcul est le plus souvent de la mémoire cache, voire des registres. Cela permet d'optimiser la vitesse des accès mémoire car lire ou écrire des données est moins coûteux avec ces types de mémoire. Si les unités de calculs ne risquent plus d'écrire simultanément à un même emplacement mémoire, la distribution de la mémoire nécessite cependant des communications entre les unités de calculs afin de propager les modifications réalisées localement. Ces opérations de communications peuvent être prises en charge par le système.

Cette introduction a permis d'aborder les éléments essentiels ayant trait à la programmation parallèle. Dans la section suivante, nous présentons quelques bibliothèques et *frameworks* permettant de programmer en parallèle. Par « framework », nous entendons des environnements de programmation n'offrant pas seulement un langage mais également des éléments tels que des modèles d'exécution, des modèles de données et une API.

2 Bibliothèques et frameworks pour la programmation parallèle

Il existe différentes bibliothèques et frameworks permettant de faire de la programmation parallèle. Certains sont dédiés à un type d'architecture matérielle, comme les cartes graphiques, certains sont liés à un modèle mémoire (distribuée ou partagée) alors que d'autres sont un peu plus généralistes. Cette section permet de donner un aperçu de ces différentes méthodes de programmation ainsi que de quelque unes des bibliothèques les plus utilisées. Durant cette thèse, nous nous sommes particulièrement intéressés à la programmation de matériels hétérogènes, comprenant les cartes graphiques. La présentation des architectures de cartes graphiques et de leur programmation sera plus fournie que la présentation des autres environnements de développement évoqués dans les premiers paragraphes de cette section. En effet, les processeurs graphiques sont les matériels qui nous ont le plus intéressés de prime abord. En outre, étudier les différents moyen de les programmer nous a conduit à utiliser OpenCL qui présente l'avantage de pouvoir être utiliser pour programmer de nombreux types de matériels différents.

2.1 Threads

Les threads sont des tâches lancées par des processus. La mise en œuvre des threads est le plus souvent réalisée par le système d'exploitation et ils sont utilisables grâce à des bibliothèques. Il existe de nombreuses bibliothèques permettant l'utilisation de threads. Par exemple les threads POSIX, ou pthreads, sont fournis par une bibliothèque C disponible sous les systèmes de type UNIX. Ils permettent la gestion des threads et l'utilisation de mécanismes de synchronisation tels que les barrières et les sémaphores. Une utilisation typique de threads concerne les interfaces graphiques : ils permettent de garder la main sur le programme en séparant l'exécution d'opérations de celle de l'interface en elle-même.

2.2 OpenMP

OpenMP (*Open Multi-Processing*) (Chapman et coll., 2007) est une API permettant de réaliser des applications parallèles en C et C++ avec des machines à mémoire partagée. L'API

permet en particulier la création de threads, la distribution des tâches aux différents threads, la synchronisation des threads. OpenMP est utilisé à travers des directives pré-processeur. Le code qui suit est un exemple de parallélisation de boucle avec OpenMP :

```
1  int main(int argc, char *argv[])
2  {
3      const int t = 1000;
4      int i, a[t];
5
6      #pragma omp parallel for
7      for (i = 0; i < t; i++)
8          a[i] = i;
9
10     return 0;
11 }
```

Dans cet exemple, chaque affectation $a[i]$ sera réalisée par un thread. Ceux-ci sont alloués automatiquement par le système.

2.3 MPI

MPI (*Message Passing Interface*) (Gropp et coll., 1999) est un protocole de communication utilisé dans la programmation de machines parallèles à mémoire distribuée. Les implémentations d'API dans différents langages (C, C++, Fortran, etc.) font de MPI un modèle standard pour la communication. De manière simplifiée, la base de programmes utilisant le protocole MPI consiste à faire calculer une sous-partie de l'algorithme à des sous-processus. Chaque processus envoie ensuite son résultat à un processus racine : ce résultat peut-être par exemple envoyé grâce à des implémentations de fonction de diffusion (*broadcast*). Pour information, le nombre de processus à instancier est précisé au lancement du programme, via un argument de la ligne de commande.

Il est possible d'utiliser MPI conjointement à OpenMP, permettant d'utiliser différents niveaux de parallélisme : à titre d'exemple, ce parallélisme hybride est utilisé avec des grappes de SMP (Jost et coll., 2003). Les SMP (*Symmetric Multi-Processors*) sont des machines équipées de multiples processeurs accédant à une même mémoire (mémoire partagée), au niveau duquel est utilisé OpenMP. Les nœuds de la grappe communiquent grâce à MPI.

2.4 PVM

PVM (*Parallel Virtual Machine*) (Geist et coll., 1994) est, à l'instar de MPI, utilisé pour programmer des machines parallèles hétérogènes à mémoire distribuée. Les machines sont connectées en réseau et sont vues comme une unique machine à mémoire distribuée : la machine virtuelle. Les communications se font via des sockets TCP/IP. Les programmes PVM sont par certains aspects assez semblables aux programmes utilisant le standard MPI. Une différence notable est la façon dont les processus sont instanciés : dans le cas de PVM, un processus racine va lancer les autres processus. Les sources sont souvent divisées en deux parties : les sources du processus racine, dont on peut dire qu'il coordonne l'exécution des autres processus en les instanciant et en attendant les résultats calculés par ceux-ci.

2.5 Frameworks et outils pour la programmation de cartes graphiques

Les cartes graphiques, GPU (*Graphics Processing Units*) sont utilisées pour des opérations de rendu graphique. Le développement des GPU, notamment grâce aux jeux vidéos, a permis de développer du matériel puissant, hautement parallèle et peu onéreux. Les cartes graphiques sont présentes sur tout type d'ordinateurs ce qui en fait un matériel très accessible. Depuis quelques années, on peut utiliser les GPU pour faire du calcul généraliste, c'est-à-dire exécuter des programmes « classiques » n'ayant pas nécessairement trait au traitement graphique : on parle de GPGPU (*General Purpose computing on Graphics Processing Units*). Cette section présente différents frameworks et outils pour le GPGPU.

a) Shaders

Dans l'ensemble d'étapes de calculs réalisés par un GPU, certaines d'entres elles sont programmables au moyen de *shaders*. Ce sont des programmes exécutés sur les GPU, permettant de réaliser des opérations sur des pixels ou des textures : ils sont donc avant tout dédiés à la réalisation d'opérations graphiques. Dans les années 2000, des langages de shader ont été proposés par différents groupes : Cg (*C for graphics*) de NVidia, HLSL (*Higl-Level Shading Language*) de Microsoft et GLSL (*OpenGL Shading Language*) d'OpenGL. Ces langages haut niveau ont ainsi permis de commencer à démocratiser le GPGPU, même si l'utilisation de langages à priori dédiés à des opérations graphiques peut s'avérer compliquée pour la mise au point de programmes généralistes.

b) Langages de haut niveau

Sont ensuite apparus des environnements de développement (*frameworks*) haut niveau pour le GPGPU. En particulier, ces frameworks ont apporté un certain niveau d'abstraction, permettant alors de programmer des GPU sans qu'il soit nécessaire d'avoir une connaissance approfondie de leur architecture. CUDA (*Compute Unified Device Architecture*), de NVidia, a été lancé début 2007. Compte tenu de son ancienneté, c'est un des frameworks les plus aboutis en matière de fonctionnalités. CTM (*Close To Metal*), proposé par ATI/AMD, a été la première mouture de ATI Stream qui est lui sorti fin 2007. Fin 2008, OpenCL (*Open Computing Langage*) du Khronos Group voit le jour. Quant à Direct Compute de Microsoft, c'est un framework basé sur HLSL. DirectCompute a été rendu disponible avec la version 11 de DirectX, parue fin 2009.

c) Choix d'un framework de programmation

Compte tenu, d'une part, de la disponibilité et du faible coût des cartes graphiques et d'autres part, que leur programmation est devenue plus aisée grâce aux divers frameworks existants, c'est vers ce type de matériel que nous nous sommes en partie tournés durant cette thèse. La programmation parallèle présente toutefois des difficultés par rapport à la programmation classique : il faut identifier les portions de code parallélisables, gérer la mémoire explicitement, gérer l'ordonnancement des tâches et la communication entres ces tâches. Les frameworks disponibles pour la programmation GPGPU présentent chacun différents avantages et inconvénients. Nous avons choisi d'utiliser OpenCL car c'est un standard ouvert et maintenu par un consortium réunissant un grand nombre d'acteurs des nouvelles technologies : Apple, Intel, AMD, NVidia, Altera et ARM entre autres. En conséquence, le principal avantage de ce standard est qu'il est utilisable sur différents types de matériel : micro-processeurs (Intel, AMD, ARM) ; GPU (NVidia, ATI), FPGA (Altera, Xilinx). Cependant, si

l'amélioration de programmes obtenue grâce à leur parallélisation dépend des programmes eux-mêmes, elle dépend aussi grandement de la prise en compte de l'architecture du matériel utilisé. Aussi, l'hétérogénéité de la large gamme de systèmes que l'on peut programmer avec OpenCL rend difficile l'optimisation de programme de manière portable. En outre, différentes implémentations d'OpenCL sont disponibles selon les systèmes utilisés, certaines caractéristiques peuvent donc être disponibles sur certains systèmes mais pas sur d'autres. Des optimisations « génériques » peuvent toutefois être apportées, en particulier en ce qui concerne l'optimisation des ressources de calcul et des ressources mémoire du matériel utilisé (Guevel et coll., 2013). Le débuggage d'applications parallèles peut s'avérer délicat. Ça n'est cependant pas le propre d'OpenCL et aujourd'hui certains constructeurs mettent à disposition des outils pour débuguer les programmes OpenCL, tel que OpenCL Emulator-Debugger d'AMD [2] ou encore avec le débugger Gnu GDB (utilisable avec les processeurs AMD et Intel).

d) Bilan

Finalement, l'hétérogénéité des plateformes utilisables avec OpenCL rend celui-ci très avantageux par rapport aux autres frameworks de développement GPGPU, en particulier dans une optique de diffusion des programmes. Même si OpenCL permet l'utilisation d'architectures matérielles variées, nous nous sommes notamment focalisés sur les GPU, compte tenu du degré de parallélisme offert par ces architectures. En effet, le formalisme multi-agents que nous avons adopté lors de la mise au point de notre modèle a une nature distribuée que nous exploitons à travers de la programmation du GPGPU.

La section suivante a pour objet de présenter, en se plaçant à un haut niveau, l'architecture d'un GPU : cela permettra de mieux comprendre les différents modèles composant le framework OpenCL.

3 Généralités sur le fonctionnement des GPU

Cette section a pour objet de présenter les GPU et leur fonctionnement, tout en restant à un haut niveau. Le pipeline graphique est présenté en premier, puis nous présentons les différences architecturales majeures entre GPU et CPU. Un dernier point concerne la mémoire des GPU.

3.1 Pipeline graphique

Le processus de rendu dans une carte graphique est bâti autour d'un pipeline, une suite d'opérations menant au rendu final de la scène à calculer. Ces calculs sont effectués sur quatre éléments de base : des vertices, des primitives, des fragments et des pixels. Ce pipeline est visible sur la figure 4.7. Les étages rouges du pipeline sont programmables. De manière simplifiée, le pipeline se déroule comme ceci : les vertices à utiliser dans le pipeline sont extraits de la mémoire. Chaque vertex est ensuite traité de manière indépendante dans l'étage programmable *Vertex Processing* : il peut par exemple s'agir de transformer les vertices ou encore de les projeter par rapport à l'emplacement de la caméra dans la scène. Les vertices sont assemblés en primitives (points, lignes, triangles, quads). Ces primitives sont traités de façon indépendante dans l'étage programmable *Primitive Processing* : un exemple de traitement est le « clipping », consistant à enlever des objets (ou des parties d'objets)

2. `http://developer.amd.com/tools-and-sdks/open-source/open-source-tools/opencl-emulator-debugger/`, en ligne, consulté en mai 2014

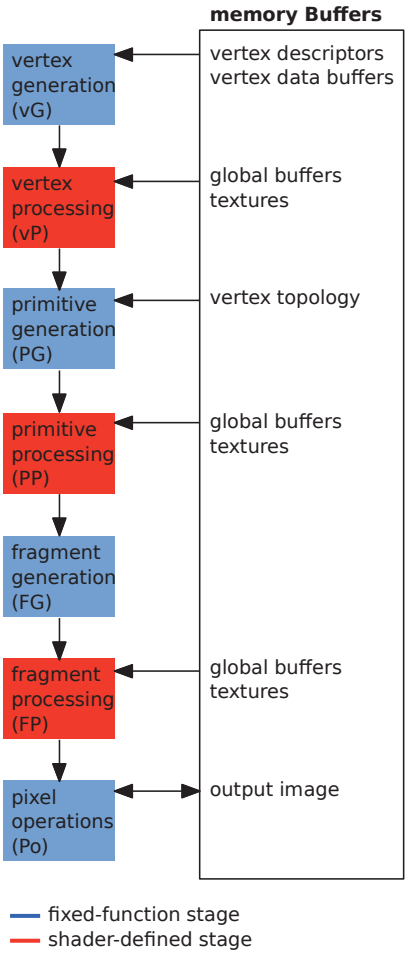


FIGURE 4.7 – Simplification du pipeline graphique dans les GPU. Les étages en rouge sont programmables. Figure extraite de (Fatahalian et Houston, 2008).

qui ne seront pas dans la vue. Les primitives sont ensuite transformées en fragments, des pixels auxquels sont associés des propriétés (couleur, profondeur, etc.). C'est ce qu'on appelle la rastérisation. Ces fragments sont traités dans l'étage programmable *Fragment Processing*, permettant l'application de texture ou encore le paramétrage de l'éclairage. L'avantage des plateformes de développement telles que CUDA ou OpenCL est qu'elles proposent une abstraction de ce pipeline, qui rend plus aisée la programmation des GPU.
En partant de ce pipeline, on peut déduire des éléments qui composent une application GPGPU de manière générale : il s'agit d'une application comportant un nombre importants de données (des vertices originellement, mais avec le GPGPU ces données peuvent être de nature variée). Ces données seront traitées en flux par des fonctions (les shaders) et il faut de préférence limiter les dépendances entre les données.

3.2 Architecture des GPU

En matière d'architecture, il est difficile de présenter un schéma d'architecture GPU précis qui permet de décrire différents GPU : chaque fabriquant construit ses propres architectures, et différentes architectures peuvent exister chez un même fabriquant. Afin de saisir les

Figure 4.8 – Comparaison simplifiée des architectures de type CPU face aux architectures de type GPU : les différences principales résident dans le nombre d'unités de calcul de chaque type de matériel ainsi que dans leurs fréquences d'horloge respectives. D'un côté les CPU possèdent quelques unités de calculs (des « cœurs ») bien adaptés à du parallélisme de tâches et gérant bien du code « complexe » contenant des branchements conditionnels. En outre ces unités de calcul ont une fréquence d'horloge élevée (plusieurs giga hertz). De l'autre, les GPU sont constitués de milliers d'unités de calcul très simples et adaptées à du parallélisme de données. Les branchements dans le code sont à éviter sur ce type d'architecture. Les fréquences des processeurs graphiques montent rarement au delà de un giga hertz. Concernant la mémoire, les deux architectures possèdent une mémoire hiérarchisée, constituée de registres, de caches et d'une mémoire principale. Les caches sont toutefois plus importants sur les CPU.

différences architecturales principales entre CPU et GPU, on peut toutefois simplifier les architectures des plateformes respectives, comme illustré sur la figure 4.8. Les CPU disposent d'un nombre réduit d'unités de calcul, mais celles-ci sont très bien adaptées pour du parallélisme de tâches. Les GPU contiennent quant à eux jusqu'à plusieurs milliers d'unités de calcul adaptées à du parallélisme de données : pour faire la comparaison avec le pipeline d'exécution présenté ci-dessus, le parallélisme se situe au niveau des vertices, des primitives et des fragments qui, pour un même type, sont traités en parallèle. Les différences architecturales entre CPU et GPU rendent évidemment la programmation de ces différentes plateformes différentes : les branchements conditionnels sont mal adaptés au GPU car leurs performances sont optimales si toutes les unités artithmétiques et logiques d'une unité de calcul SIMD travaillent.

3.3 Mémoire des GPU

La mémoire des GPU est, de manière classique, organisée en une hiérarchie. Les constructeurs ne dévoilent que les grandes lignes de leurs architectures mais dans les GPU modernes nous trouvons les niveaux de mémoire suivants (NVidia, 2012; AMD, 2012) : de la RAM, la mémoire disponible en quantité importante et accessible par toutes les unités de calcul du GPU. Cependant les accès à cette mémoire sont très coûteux, de l'ordre de plusieurs centaines de cycles. On trouve également différents niveaux de cache, permettant aussi bien de disposer de mémoire partagée entre quelques unités de calcul, que de caches en lecture seule, accessibles à toutes les unités de calcul pour accélérer les accès mémoire. Ces caches sont accessibles en quelques cycles (au pire quelques dizaines). Si dans les CPU la latence doit être minimisée, elle est *masquée* dans les GPU : une part des caches est réservée au stockage de contextes de threads, permettant ainsi de permuter les contextes si un thread se trouve en attente alors qu'un autre est prêt : l'idée qui ressort de cela est qu'il faut prévoir

bien plus de threads que ne peut effectivement en exécuter le GPU. Il est important de noter que la mémoire requise pour une application doit être allouée avant que l'application ne démarre et libérée en fin d'exécution. En outre le placement mémoire est explicite avec les frameworks de développement présentés dans la section précédente. Cela demande donc d'organiser ses données, de prévoir des structures adaptées pour l'exécution sur GPU, ainsi que des mécanismes pour protéger les données accessibles à toutes les unités de calcul, sous peine de générer des inconsistances dans les données de la simulation.

L'objet de la section suivante est de présenter le framework OpenCL. Nous y abordons tout d'abord des notions générales concernant la structure du framework, l'optimisation de code et le débuggage d'application, ensuite nous détaillerons les différents éléments composant le framework OpenCL.

4 Présentation du framework OpenCL

OpenCL (*Open Computing Language*) est un environnement de développement ouvert proposé par le groupe Khronos en 2009. Ce groupe est en fait un consortium d'entreprises et d'académiques travaillant au développement et à l'amélioration du framework. On retrouve des membres tels que Apple, AMD, Intel, Nvidia, Xilinx ou ARM mais aussi des entreprises du web telles que Google ou Mozilla et du jeux vidéo avec EA, Valve, etc. Une liste complète des membres contributeurs à OpenCL peut être consultée sur la page internet `https://www.khronos.org/members/`. C'est donc un framework largement porté par l'industrie des nouvelles technologies. OpenCL permet la programmation *unifiée* de plateformes parallèles, offrant un avantage indéniable en matière de portabilité du code. Les architectures sous-jacentes sont masquées par les différents éléments du framework que nous présentons dans cette section :

- modèle de plateforme (modèle matériel abstrait pour exécuter des fonctions OpenCL) ;
- modèle d'exécution (définissant comment les fonctions OpenCL sont exécutées) ;
- modèle mémoire (hiérarchie mémoire abstraite utilisée par les fonctions OpenCL).

Ces différents modèles permettent ainsi de réaliser des programmes sans nécessairement avoir à connaître les architectures sur lesquelles le code va être exécuté. Cette portabilité a cependant un coût : comme on l'a vu, la variété des architectures parallèles rend presque indispensable l'optimisation du code pour une architecture donnée. L'attrait de la portabilité fournie par OpenCL s'estompe rapidement si il faut ré-écrire le code d'une application lorsque l'on utilise une architecture différente. L'optimisation des implémentations s'avère donc problématique avec OpenCL, et le mieux reste encore d'optimiser le modèle à implémenter, par exemple comme nous l'avons fait avec notre modèle de cellule virtuelle dont certains éléments ont été conçus ou inclus de façon à ce que l'implémentation en soit améliorée. Des travaux visent toutefois à apporter des éléments d'optimisation aux programmes OpenCL. Certains sont dédiés à des champs d'applications restreints, comme c'est le cas pour le framework Parallel SURF (Mistry et coll., 2011), dédié à la vision par ordinateur. D'autres travaux visent l'optimisation des programmes OpenCL sur une gamme d'architectures : Du et coll. (2012) proposent des éléments pour optimiser les programmes OpenCL exécutés sur GPU. Sylvain et coll. (2012) proposent un support exécutif afin de gérer automatiquement le placement de tâches avec OpenCL. Enfin, Guevel et coll. (2013) ont, dans une rapide étude, montré que la granularité du calcul et l'optimisation de la localité des données permettent d'améliorer les performances de programmes OpenCL à la fois sur CPU et sur GPU. Ces optimisations entraînent toutefois un coût, ne serait-ce qu'en matière de conception : le placement des tâches et de la mémoire dépend grandement de l'application. Une application

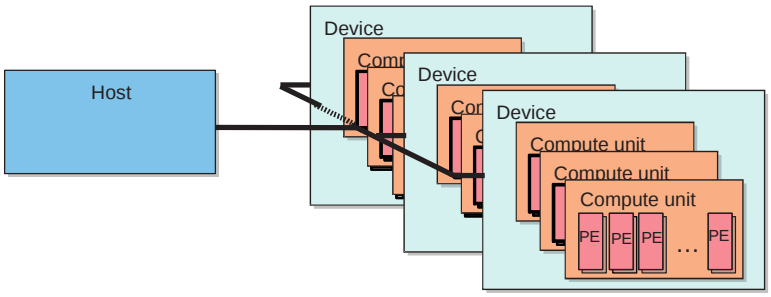


Figure 4.9 – Modèles de plateforme OpenCL. Une plateforme regroupe différents périphériques pouvant communiquer. Figure extraite de (Gaster et coll., 2011).

comme un produit de matrices est idéale pour faire des tests d'optimisation car le problème peut très facilement être découpé en sous-problèmes. Ça n'est plus vrai pour des problèmes plus complexes, pouvant demander un temps considérable pour proposer un découpage adéquat, si un tel découpage peut être trouvé. Nous verrons toutefois au chapitre 5 que notre implémentation, même sans optimisation particulière en matière de placement mémoire et de tâches[3], offre de bonnes performances.

OpenCL permet de coder des applications présentant du parallélisme de données ou de tâches (voire les deux en même temps). Nous nous sommes intéressés en particulier au parallélisme de données, car celui-ci est plus proche du concept de systèmes multi-agents que nous avons présenté au chapitre 3 : de manière simplifiée, tous les agents du système exécutent un même programme.

Dans la suite de cette section, nous présentons les différents modèles constituant le framework OpenCL : modèle de plateforme, modèle d'exécution et modèle mémoire. Nous présentons également l'environnement d'exécution d'OpenCL à travers lequel sont concrètement mis en œuvre les notions de plateforme, de mémoire et d'exécution.

À titre de complément, le lecteur intéressé trouvera dans les ouvrages de Scarpino (2011), Gaster et coll. (2011) et Munshi et coll. (2012) des informations précises et détaillées concernant le framework OpenCL dans sa globalité.

4.1 Modèle de plateforme

Le modèle de plateforme, illustré sur la figure 4.9, permet de fournir une abstraction du système hétérogène sous-jacent. En particulier, une plateforme est le plus souvent spécifique à une marque de matériel et correspond aux implémentations OpenCL présentes sur la machine. Par exemple on peut imaginer un ordinateur équipé d'un micro-processeur Intel et de deux cartes graphiques : une carte ATI et une carte NVidia. Le modèle de plateforme va regrouper de manière abstraite les périphériques pouvant communiquer entre eux. Sur cette machine imaginaire, on peut supposer que l'on aura donc au moins deux plateformes : une plateforme ATI et une plateforme NVidia. Les éléments de la plateformes (« OpenCL devices ») sont connectées à un hôte, qui permet de coordonner l'exécution des programmes OpenCL.

3. D'autres optimisations ont toutefois été mises en œuvre, comme on le verra.

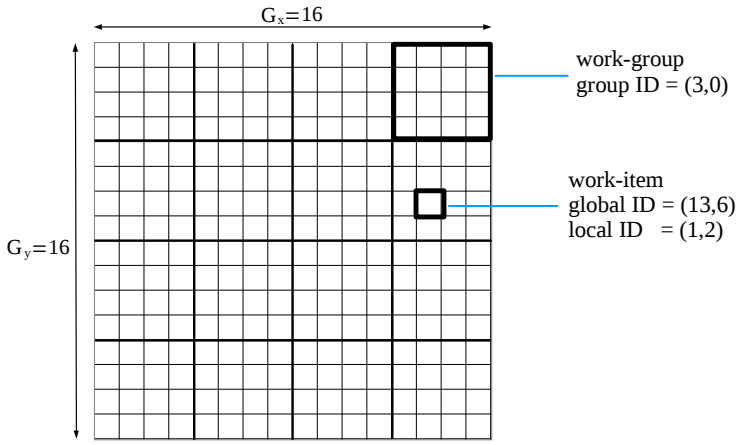


FIGURE 4.10 – Modèle d'exécution d'OpenCL organisé en une grille de dimensions N ($1 \leq N < 3$). Ici, $N = 2$; la dimension de l'espace d'index est $16 * 16$, les groupes sont de taille $4 * 4$. La grille, ou espace d'index, est divisée en groupes (work-groups) eux-mêmes constitués de work-items. Les work-items sont des instances de kernels. Groupes et work-items sont identifiés dans la grille. Dans le cas des work-items, il existe une identification globale dans la grille, ainsi qu'une identification locale, dans le work-group auquel appartient le work-item.

4.2 Modèle d'exécution

Le modèle d'exécution d'OpenCL est illustré sur la figure 4.10. Il est construit sur la base d'un espace d'index permettant d'indexer les instances de *kernels* (des fonctions contenant le code parallèle). La dimension de cet espace d'index est N ($1 \leq N < 3$). Cet espace d'index est en outre hiérarchisé : il est découpé en groupes (les work-groups) qui sont eux-mêmes composés d'unités plus petites : les work-items. Les work-items sont donc les instances de *kernel*. Nous verrons dans la section 4.4 comment ces kernels sont gérés par l'hôte. Les work-groups sont identifiables localement, par des coordonnées en 1, 2 ou 3 dimensions dans l'espace d'index. Les work-items sont eux identifiables à deux niveaux : globalement, avec leurs coordonnées dans l'espace d'index, en au plus trois dimensions également, et localement au groupe auquel ils appartiennent, là aussi en au plus trois dimensions. Des fonctions sont fournies pour permettre au programmeur de récupérer facilement les identifiants, dans les N dimensions, des work-groups et des work-items. Un exemple d'usage d'une telle fonction est montré dans la section 4.4.

Les dimensions et la taille de l'espace d'index sont spécifiés dans le code hôte. La taille de l'espace d'index ne doit pas être limitée au nombre d'unités de calcul du matériel utilisé : on a par exemple vu qu'une exécution sur GPU peut tirer avantage de la permutation de contextes lorsque des threads sont en attente. La spécification de la taille des groupes de travail est optionnelle : si elle n'est pas spécifiée, elle sera attribuée automatiquement par OpenCL.

La synchronisation de work-items n'est possible au sein d'un même kernel qu'entre work-items appartenant à un même groupe : il n'est pas possible de synchroniser, avec une barrière par exemple, des work-items appartenant à des groupes différents. On parle de synchronisation locale. Une synchronisation globale est possible, en divisant un kernel en sous kernels et en plaçant une barrière entre des exécutions successives des deux kernels, de façon à ce que le second kernel ne débute que lorsque toutes les instances du premier kernel seront terminées.

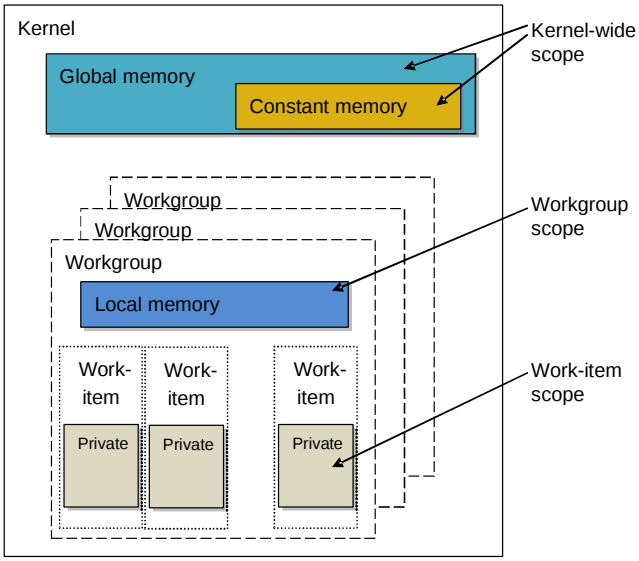


Figure 4.11 – Hiérarchie mémoire avec OpenCL : trois niveaux de mémoire sont présents. La mémoire globale est une mémoire dont l'accès nécessite beaucoup de cycles mais elle est accessible à tous les work-items de l'espace d'index, quel que soit leur work-group d'appartenance. Chaque work-group dispose de mémoire locale, dont l'accès est rapide et accessible seulement aux work-items appartenant au groupe. La mémoire la plus rapide d'accès est la mémoire privée à chaque work-item, mais elle n'est disponible qu'en faible quantité. Figure extraite de (Gaster et coll., 2011).

4.3 Modèle mémoire

Le modèle mémoire d'OpenCL (figure 4.11) est construit autour d'une hiérarchie, constituée de différents types de mémoire : la mémoire globale est accessible à chaque work-item, indépendamment du groupe dans lequel il se trouve. L'accès à cette mémoire nécessite beaucoup de cycles. La mémoire constante est un sous ensemble de la mémoire globale accessible en lecture seule et dont les accès sont plus rapides. Chaque work-group dispose d'un peu de mémoire localement. Seuls les work-items du groupe peuvent accéder à cette mémoire dont l'accès est assez rapide. Enfin, chaque work-item dispose d'une petite quantité de mémoire privée, le plus souvent de type registre. C'est une mémoire rapide, mais disponible en faible quantité. L'utilisation des différents types de mémoire est *explicite*, c'est-à-dire qu'il appartient au programmeur de réaliser lui même le placement de ses données dans le type de mémoire qu'il souhaite. La figure 4.12 présente une analogie entre le modèle mémoire d'OpenCL et la mémoire telle qu'elle est présente dans un GPU AMD Radeon HD6970.

4.4 Environnement d'exécution

L'exécution de kernels OpenCL nécessite au préalable la préparation de différents éléments. En premier lieu, un *contexte*, associée à une plateforme, doit être créé sur l'hôte. C'est à travers ce contexte que l'on gère 1) les interactions entre l'hôte et les périphériques utilisés pour l'exécution du code parallèle, 2) les objets mémoire créés pour stocker les données de la simulation et 3) l'ordonnancement des kernels. Le second objet utilisé est une file de commandes, qui est associée à un matériel donné (plusieurs files permettant donc l'utilisation d'autres périphériques disponibles avec la plateforme associée au contexte). Des files de commandes différentes peuvent être utilisées en parallèle, permettant alors de faire du pa-

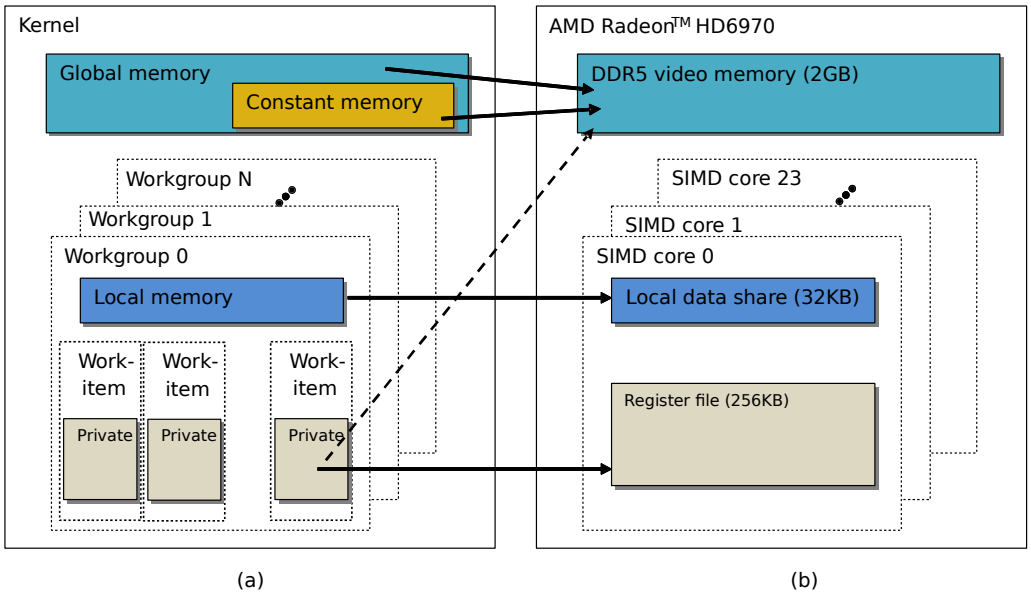


FIGURE 4.12 – Comparaison entre le modèle mémoire d'OpenCL (a) et la hiérarchie mémoire telle qu'elle est présente dans une carte AMD Radeon HD9670 (b). Figure extraite de (Gaster et coll., 2011)

rallélisme de tâches. Il convient ensuite de créer les objets mémoire. Ceux-ci peuvent être de deux sortes : des *buffers*, ou tampon, qui sont l'équivalent de tableaux en langage C ; et des *images*, qui sont des objets dont l'utilisation peut être optimisée sur GPU grâce aux caches des unités de calcul. Il faut ensuite créer un objet *programme* associé à des sources OpenCL, et le compiler. Le *kernel* associé est ensuite créé avec ce binaire. Après avoir passé au kernel ses arguments, celui-ci est exécuté sur le matériel choisi. Il est évidemment possible de créer plusieurs kernels : les sources permettant de créer l'objet programme peuvent contenir plusieurs fonctions kernel, dont le nom est utilisé lors de la création d'objets kernels. La figure 4.13 synthétise ces différents éléments, constituant l'environnement d'exécution d'OpenCL.

4.5 Exemple de programme OpenCL

Afin de faire le lien entre tous les éléments présentés dans cette section, nous montrons ici un exemple simple de code OpenCL (version 1.2). En particulier, nous détaillons l'environnement d'exécution. Cet exemple permet de réaliser une transposée de matrice à deux dimensions, le langage utilisé est du C et les commentaires permettent de suivre le fil du programme. Nous ne détaillons pas l'usage des fonctions de l'API, mais de plus amples informations sur l'API OpenCL peuvent être consultées sur la page de Khronos `https://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml/`.
Nous proposons un exemple complet de programme[4] car cela permet de dégager des éléments importants qui ont conduit aux développements proposés dans le chapitre 5 : la mise en place de l'environnement d'exécution peut s'avérer longue et fastidieuse, comme illustré dans l'exemple, mais il s'avère que la plus grande partie de ce code est redondante d'une application à l'autre. Nous avons choisi d'exploiter cette redondance dans une architecture logicielle que nous présentons au chapitre 5. Voici en premier lieu le code OpenCL qui va

4. Ce code est original et sert de support de cours pour les étudiants de l'université de Brest.

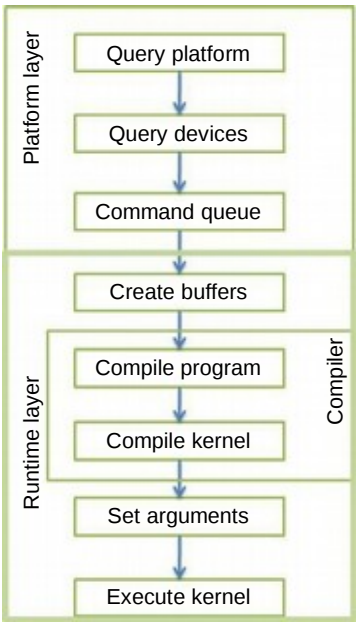


Figure 4.13 – Environnement d'exécution d'OpenCL. Une suite d'objets est à créer afin de pouvoir exécuter le code parallèle : une plateforme et un périphérique associé à cette plateforme sont les premiers éléments. Un *contexte* est associé à la plateforme (permettant la gestion des objets mémoire ou encore de l'ordonnancement du code parallèle). Une *file de commandes* est associée à un matériel, permettant la communication entre l'hôte et le matériel. Un programme est ensuite compilé à partir d'un code source fourni et les kernels sont créés à partir de ce programme. Après avoir passé aux kernels les arguments nécessaires, ceux-ci sont ordonnancés sur le matériel parallèle choisi. Figure extraite de (Gaster et coll., 2011).

s'exécuter sur un des périphériques de la machine ; chaque work-item va réaliser une portion du travail en exécutant une instance de kernel :

```
/*
 * Fichier kernel.cl
 */
__kernel void k_transpose(__constant float *inputMatrix,
                          __global float *outputMatrix,
                          uint width,
                          uint height)
{
  // Transposition des lignes en colonnes
  // get_global_id(0) retourne la composante x du work-item courant
  // get_global_id(1) retourne la composante y du work-item courant
  uint target_id = get_global_id(0)*height + get_global_id(1);
  uint source_id = get_global_id(1)*width  + get_global_id(0);
  outputMatrix[target_id] = inputMatrix[source_id];
}
```

Dans le cas d'une transposée de matrice, le kernel est simple et très court. Il n'en est pas de même pour le code hôte, qui contient différentes phases avant de pouvoir finalement exécuter le code parallèle : il faut avant tout mettre en place tout l'environnement d'exécution présenté à la section 4.4 :

```
/*
 * Fichier hote.c
 */
int main(int argc, char *argv[])
{
  // Dimensions de la matrice à transposer et des work_groups
  int dim1 = /* ... */;
  int dim2 = /* ... */;
  int block_size1 = /* ... */;
  int block_size2 = /* ... */;
  int *inputMatrix  = /* allocation d'une matrice dim1*dim2 */;
  int *outputMatrix = /* allocation d'une matrice dim1*dim2 */;

  /* Initialisation de la matrice "inputMatrix" */
  /* ... */

  // Pour récupérer les codes d'erreur
  cl_int err;

  /*
   * 1. Environnement d'exécution
   */

  // Récupération de la première plateforme disponible
  cl_platform_id platform;
  err = clGetPlatformIDs(1, &platform, NULL);
```

```
27
28  // Note:
29  // la variable err doit être utilisée pour tester
30  // le succès de la fonction.
31  // Si une fonction s'est exécutée avec succès,
32  // err vaut CL_SUCCESS
33
34  // Récupérer le premier périphérique disponible
35  // dans la plateforme (en pratique on peut récupérer tous
36  // les périphériques de la plateforme afin de choisir
37  // un périphérique parmi ceux disponibles ou bien pour
38  // répartir le travail sur plusieurs matériels)
39  cl_device_id device;
40  err = clGetDeviceIDs(platform, CL_DEVICE_TYPE_ALL,
41                       1, &device, NULL);
42
43  // Création du contexte associé à la plateforme
44  cl_context_properties props[3] = {CL_CONTEXT_PLATFORM,
45          (cl_context_properties)platform, 0};
46  cl_context context = clCreateContext(props, 1, &device,
47                                       NULL, NULL, &err);
48
49  // Création d'une file de commandes pour le
50  // périphérique choisi
51  cl_command_queue queue = clCreateCommandQueue(context, device,
52                                                0, &err);
53  /*
54   * 2. Les données
55   * Comme on peut le voir, les buffers de données sont liées
56   * à un contexte d'exécution
57   */
58
59  // La variable inputMatrix est une matrice d'entiers, de taille
60  // dim1*dim2 préalablement initialisée sur l'hôte, et dont le
61  // contenu est recopié dans le buffer
62  cl_mem inputBuffer = clCreateBuffer(context,
63                                      CL_MEM_READ_ONLY |
64                                      CL_MEM_COPY_HOST_PTR,
65                                      dim1*dim2*sizeof(int),
66                                      inputMatrix,
67                                      &err);
68
69  // Nous créons aussi un buffer de sortie, où seront écrits
70  // les résultats; pas besoin d'initialisation cette fois
71  cl_mem outputBuffer = clCreateBuffer(context,
72                                       CL_MEM_WRITE_ONLY,
73                                       dim2*dim1*sizeof(int),
74                                       NULL,
```

```
75                                                  &err);
76
77      /*
78       * 3. Le programme
79       * Le programme est lui aussi lié au contexte d'exécution
80       */
81
82      // La variable "source" contient le code OpenCL:
83      // il s'agit d'une variable de type chaîne de caractères
84      // dans laquelle on a préalablement écrit le contenu du
85      // fichier source "kernel.cl"
86
87      cl_program program = clCreateProgramWithSource(context,
88                                                      1,
89                                                      (const char*)&source,
90                                                      NULL,
91                                                      &err);
92
93      // Le programme est ensuite compilé (le binaire résultant est
94      // compatible avec tous les périphériques associés au contexte)
95      // En cas d'erreur de compilation, il est possible de récupérer
96      // un journal de compilation
97      err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
98
99      // On peut maintenant créer des kernels à proprement parler.
100     // Dans le cas présent, un seul kernel est nécessaire pour
101     // réaliser une transposée de matrice
102     cl_kernel kernel = clCreateKernel(program, "k_transpose", &err);
103
104     /*
105      * 4. Ordonnancement
106      */
107     // Avant d'exécuter le kernel, il faut lui passer ses arguments
108     err  = clSetKernelArg(kernel, 0, sizeof(cl_mem),  &inputMatrix);
109     err |= clSetKernelArg(kernel, 1, sizeof(cl_mem),  &outputMatrix);
110     err |= clSetKernelArg(kernel, 2, sizeof(cl_uint), &dim1);
111     err |= clSetKernelArg(kernel, 3, sizeof(cl_uint), &dim2);
112
113     // Définition de l'espace d'index
114     size_t global[2] = {dim1, dim2};
115     size_t local[2]  = {block_size1, block_size2};
116
117     // Exécution du kernel
118     err = clEnqueueNDRangeKernel(queue, kernel, 2 NULL, global, local,
119                                  0, NULL, NULL);
120
121     // Attente que les commandes dans la file se terminent
122     // (dans le cas présent l'exécution du kernel)
```

```
123     clFinish(queue);
124
125     /*
126      * 5. Transfert des résultats sur l'hôte
127      */
128     // Le buffer contenant la matrice transposée est lu;
129     // son contenu est placé dans la matrice "outputMatrix"
130     // préalablement allouée
131     err = clEnqueueReadBuffer(queue, outputBuffer, CL_TRUE, 0,
132                               dim2*dim1*sizeof(int),
133                               outputMatrix, 0, NULL, NULL);
134
135     // Les ressources allouées sont libérées
136     clReleaseMemObject(inputBuffer);
137     clReleaseMemObject(outputBuffer);
138     clReleaseProgram(program);
139     clReleaseKernel(kernel);
140     clReleaseCommandQueue(queue);
141     clReleaseContext(context);
142     free(inputMatrix);
143     free(outputMatrix);
144 }
```

On observe une disproportion importante entre l'opération à réaliser, une transposée de matrice, et tous les éléments à initialiser côté hôte afin de pouvoir exécuter le code parallèle. Il faut mettre en place tous les éléments de l'environnement d'exécution mais on s'aperçoit vite que l'on va retrouver, d'un programme à l'autre, de nombreux éléments de ce code. Par exemple, si l'on souhaite maintenant réaliser un produit de matrices au lieu d'une transposée, tout ce qui concerne la plateforme, les périphériques, le contexte et la file de commandes sont réutilisables sans changement. L'organisation des données va nécessiter un peu de travail, mais savoir créer un buffer est suffisant. En ce qui concerne l'exécution du kernel, il suffit d'adapter le kernel présenté ci-dessus pour qu'il contiennent les opérations nécessaires à une multiplication de matrices. Il est donc possible de tirer parti de la redondance de code observée. Nous allons présenter avec plus de détails ces éléments dans le chapitre 5, qui est consacré à la présentation du simulateur que nous avons mis en place dans le cadre de ces travaux.

La section suivante est une synthèse des éléments que nous avons présentés dans ce chapitre.

5 Synthèse du chapitre

Ce chapitre, consacré à la programmation parallèle en général, nous a permis de faire un rapide tour d'horizon des différentes solutions, tant au niveau des matériels que des environnements de développement, s'offrant à nous. La parallélisation d'un problème consiste en premier lieu à dégager des sous-tâches de ce problème, qu'il est possible de réaliser simultanément. Dégager ces sous-tâches n'est en général pas trivial, en tout cas dans les problèmes complexes. D'un côté, il existe différentes manières de paralléliser un problème (parallélisme de tâches ou de données) et, de l'autre, il existe différents types de matériels

(machines à mémoire partagée, distribuée, ou encore les deux, microprocesseurs multi-cœurs, GPU, ou même FPGA). De manière générale, si la parallélisation d'un problème n'est pas triviale, l'implémentation de cette parallélisation ne l'est pas non plus : il faut gérer l'ordonnancement des tâches et leurs dépendances, et la gestion mémoire peut s'avérer problématique à cause de problèmes d'accès concurrents par des tâches distinctes.

En outre, il est nécessaire de choisir un matériel adapté à une exécution parallèle. De ce point de vue, nous nous sommes en particulier intéressés aux cartes graphiques, ou GPU, car ce sont des architectures matérielles qui ont très largement bénéficié du développement du jeu vidéo, permettant ainsi de disposer de matériel à la fois puissant, peu coûteux et présent sur tout type d'ordinateur. De plus, la nature distribuée du système multi-agents auquel nous avons couplé notre modèle (voir chapitre 3) fait, selon nous, de l'architecture GPU l'architecture parallèle la plus adaptée à l'exécution de notre modèle. Cependant, l'utilisation d'OpenCL va nous permettre, comme on le verra dans le chapitre suivant, d'utiliser des types de matériels différents tels que des micro-processeurs multi-cœurs. Finalement, les matériels utilisés lors de cette thèse pour simuler le modèle proposé sont hétérogènes.

Différents environnements de programmation permettent de mettre en œuvre des applications parallèles. Alors que certains sont spécialisés pour certains types de matériel (OpenMP pour les machines à mémoire partagée, MPI pour les machines à mémoire distribuée ou encore CUDA pour la programmation de cartes graphiques NVidia, etc.), d'autres offrent davantage de portabilité, comme OpenCL. OpenCL est fortement soutenu par l'industrie des nouvelles technologies et le monde académique s'y intéresse de plus en plus. Nous avons toutefois vu, au cours de la section 4, que cet environnement de programmation présente quelques inconvénients : le premier est que le code produit est difficilement optimisable pour différentes architectures sans perdre en portabilité. Toutefois, nous avons également vu que quelques travaux ont permis de proposer des optimisations à travers des supports exécutifs ou bien des suggestions de granularité de calcul et de placement mémoire. Ensuite, le débuggage d'application OpenCL s'avère souvent fastidieux ; mais c'est généralement vrai avec les applications parallèles. Certains constructeurs proposent toutefois des outils afin d'aider le développeur dans sa phase de débuggage. Finalement, nous avons également vu, à travers un exemple de programme OpenCL complet, la disproportion importante entre la préparation de l'environnement d'exécution (le code hôte) et le programme parallèle en lui même. Évidemment, dans le cas d'un modèle complexe, le code parallèle sera autrement plus long que dans le cas d'une transposée de matrice. Nous avons également montré que le code hôte présente de la redondance d'un programme parallèle à un autre. Nous avons décidé de retourner à notre avantage cette redondance, en concevant une architecture logicielle permettant de masquer la plus grosse partie du code hôte. Cette architecture logicielle est l'objet du prochain chapitre : elle est la base du simulateur que nous proposons, et dans lequel nous avons implémenté notre modèle. En particulier, nous proposons des structures de données adaptées à la fois au masquage du code hôte et à l'utilisation d'architectures GPU pour l'exécution du modèle. Nous proposons également des mécanismes permettant de gérer la réplication des entités de notre système, afin de pallier l'absence d'allocation dynamique de mémoire sur les GPU. Une troisième caractéristique importante de notre simulateur est la présence d'un générateur de code, qui permet de manipuler le modèle de manière plus aisée qu'en codant directement, afin de rendre ce simulateur plus accessible, au moins dans une certaine mesure.

CHAPITRE 5

Proposition d'un simulateur parallèle & exemples d'application

L'objet de ce chapitre est d'introduire le simulateur que nous proposons. Dans un premier temps, nous proposons une revue de simulateurs de systèmes biologiques issus la littérature. Cette revue nous permet de dégager les avantages et inconvénients des différentes approches existantes. Une fois le bilan de cette revue établi, bilan nous permettant de justifier notre approche, nous détaillons la construction de notre simulateur parallèle : celui-ci est bâti sur une architecture logicielle générique. Il contient des structures de données adaptées à l'utilisation d'OpenCL, des mécanismes de gestion mémoire permettant la réplication d'entités lors d'une simulation, ainsi qu'un fichier de paramètres accessible aux non programmeurs afin de manipuler le modèle. Une fois le simulateur présenté, nous montrons concrètement comment nous y avons implémenté notre modèle de cellule virtuelle puis nous présentons divers cas d'application que nous avons réalisés avec nos outils.

Sommaire

Des éléments de ce chapitre on fait l'objet de deux publications (Jeannin-Girardon et coll., 2013a, b).

1 Introduction

Depuis plusieurs dizaines d'années, des quantités très importantes de données ont été extraites en biologie. Cette discipline comporte plusieurs branches et, de manière plus générale, il n'est pas rare que d'autres disciplines abordent des questions de biologie, comme l'informatique, la physique ou les mathématiques. La compréhension des systèmes biologiques peut passer, aujourd'hui, par l'utilisation de simulateurs permettant d'une part, l'intégration des données biologiques à des modèles exécutés dans des simulateurs et, d'autre part, l'étude des systèmes biologiques à travers des simulations numériques. Cela permet d'accroître la connaissance du système étudié, ou encore d'émettre de nouvelles hypothèses liées au système (noter que nous avons abordé les intérêts de la simulation numérique dans le chapitre 2).

À l'instar des modèles que nous avons décrits (voir à nouveau le chapitre 2), les simulateurs proposés pour étudier des systèmes biologiques sont souvent spécialisés pour l'étude d'un système en particulier. Il existe peu d'environnements que l'on pourrait qualifier de

génériques pour l'étude de systèmes biologiques. Il paraît ambitieux d'imaginer qu'une plateforme générique unique (et un modèle générique unique) puisse permettre d'étudier tout type de systèmes biologiques. Pour s'en convaincre, il suffit de constater l'étendue de la biologie, qui concerne l'étude du vivant dans son sens le plus large : biologie moléculaire, biologie cellulaire, évolution, écologie, ou encore botanique.

En ce qui nous concerne, nos intérêts se sont portés sur l'étude du développement de tissus biologiques à travers une approche basée sur la cellule. Nous avons donc limité les aspects que nous souhaitons étudier mais nous avons vu que les modèles proposés pour étudier les aspects de morphogenèse de tissus sont aussi nombreux que variés ; la plupart sont également très spécialisés. Nous avons proposé notre propre modèle de cellule virtuelle, que nous avons pensé pour qu'il permette une expressivité suffisamment large grâce à sa construction hybride (différents sous-modèles permettent d'exprimer des aspects différents du développement de tissus). Les possibilités du modèle sont toutefois limitées, cela est inévitable.

En matière d'environnements de simulation, les problèmes rencontrés sont les mêmes. Il n'est pas raisonnable de penser qu'un unique simulateur pourrait permettre d'accueillir toutes les variétés de modèles possibles. Très souvent, un modèle spécifique est même « livré » avec un simulateur donné. La suite de ce chapitre est consacrée à la revue de quelques simulateurs présents dans la littérature ainsi qu'à la présentation du simulateur que nous proposons.

2 Revue d'environnements de simulation

Cette section est consacrée à la présentation de plateformes et de simulateurs présentés dans la littérature. Nous avons classé ces simulateurs en deux catégories distinctes : dans un premier temps nous présentons des simulateurs séquentiels, c'est-à-dire n'incluant pas d'élément d'algorithmique et de programmation parallèle (ou n'apportant pas de précision à ce sujet). Dans un second temps nous présentons des plateformes construites autour de l'idée de parallélisme ou incluant au minimum quelques éléments d'algorithmique parallèle. Finalement, nous dressons un bilan de cette revue rapide, en dégageant les avantages et inconvénients des approches vues et afin de mettre en évidence les points que nous avons souhaité résoudre en proposant notre propre simulateur.

2.1 Plateformes et simulateurs séquentiels

a) GemCell : Generic Executable Modeling of Cells

Cet environnement de simulation a été proposé par Amir-Kroll et coll. (2008). Il repose sur quatre modules, illustrés sur la figure 5.1. Le module A est le modèle présent dans l'environnement. Il s'agit d'un modèle générique centré sur la cellule qui se concentre sur les comportements haut niveaux de la cellule biologique, sans modélisation de sa dynamique interne. Ce modèle est couplé à une base de données (module B sur la figure 5.1) qui permet d'extraire des données issues de la biologie pour paramétrer le modèle. Les deux modules A et B sont connectés à l'exécution (C) afin d'adapter le modèle générique fourni. Enfin, un dernier module permet de visualiser les sorties et d'interagir dynamiquement avec le modèle. La manipulation de modèle est réalisée avec Statecharts (Harel, 1987), ce qui permet de définir les comportements des objets à travers des diagrammes. Les diagrammes sont ensuite traduits en code exécutable (C, C++ ou Java). L'architecture de l'environnement est formalisée avec un diagramme de classes. Pour synthétiser, GemCell présente les différents aspects suivants :

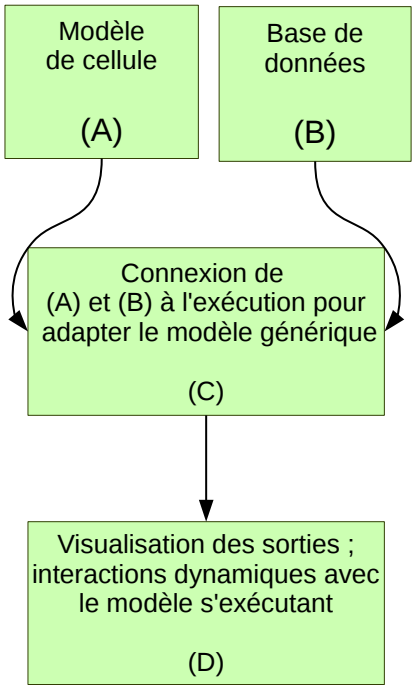


Figure 5.1 – Composition de l'environnement GemCell : un modèle de cellule générique (A) est couplé à une base de données contenant des données biologiques (B) via un module (C). Les sorties du modèle sont visualisables, et il est également possible d'interagir dynamiquement avec le modèle en cours d'exécution (D).

- la plateforme contient un modèle de cellule « générique », mais dont on ne connaît pas les détails (peut-on avoir plusieurs formes de cellules ? la cellule peut-elle percevoir les contraintes physiques issues de l'environnement dans lequel elle évolue ? combien de cellules peut-on simuler en même temps ?) ;
- la plateforme permet de coupler le modèle à des données biologiques pour gagner en pertinence ;
- l'environnement de simulation est rendu accessible à travers la manipulation du modèle via des diagrammes qui sont ensuite traduits en code ;
- il n'y a pas d'information sur la portabilité de l'environnement (possibilité d'exécution sur différents systèmes d'exploitation) ;
- il n'y a aucun cas d'étude présenté pour illustrer l'environnement et présenter des éléments de validation ;
- des aspects *in virtuo* sont présents sur la plateforme, c'est-à-dire qu'il y a possibilité pour un utilisateur d'interagir dynamiquement avec le modèle lors de son exécution.

Il nous semble que le principal intérêt de GemCell est de proposer un couplage modèle / base de données biologiques pour accroître la pertinence des simulations. Néanmoins, l'absence d'exemple ne permet pas de bien appréhender les possibilités de la plateforme.

b) VirtualLeaf

VirtualLeaf, proposé par Merks et coll. (2011), est un environnement de modélisation basé sur la cellule pour l'étude de la morphogenèse de plantes. Le framework comprend un modèle, que nous avons présenté dans le chapitre 2, page 42. L'environnement peut être utilisable à travers une interface graphique (GUI, *Graphical User Interface*). La structure du

logiciel n'est pas détaillée dans la publication, mais le code source de l'environnement est libre et donc accessible à tout un chacun. En matière d'accessibilité à des non-programmeurs, l'utilisation de VirtualLeaf nécessite de pouvoir programmer un minimum en C++. Cependant, des squelettes de code sont disponibles pour démarrer et une API propose quelques fonctions permettant la définition du modèle. Un tutoriel d'aide est disponible afin de prendre ces éléments en main. Enfin, les paramètres du système sont ajustables.
Les éléments que l'on retire de cet environnement sont les suivants :

- un modèle est livré avec l'environnement. Notons qu'il est paramétrable et que ses comportements sont définissables ;
- une connaissance basique de C++ est requise pour manipuler le modèle, mais le tutoriel permet de faciliter l'accès à la définition de comportements dans le modèle ;
- quelques cas d'étude sont présentés : cela permet d'apporter des éléments de validation de l'environnement de simulation ainsi que de montrer les possibilités du système ;
- l'environnement de simulation est exécutable sur différents systèmes d'exploitation ;
- la conception de l'environnement n'est pas formalisée, mais son code est libre et donc consultable facilement ;
- les temps de calcul du modèle sont mentionnés mais aucun détail n'est donné sur les capacités de calcul de manière générale.

Le simulateur est fourni avec une douzaine de modèles de croissance de tissus de plantes et le paramétrage compte une soixantaine de données, toutes modifiables (visualisation, export, mécanique cellulaire, moteur d'intégration).

c) Virtual Cell

Virtual Cell (Loew et Schaff, 2001) est dédié à l'étude de la dynamique intra-cellulaire. Cet outil permet aux biologistes de réaliser des simulations de systèmes biologiques, moyennant des connaissances en physique et mathématiques. Une interface graphique permet de manipuler les structures du modèle (les molécules, les réactions, etc.). L'environnement est également accessible aux bio-mathématiciens car il permet de définir directement des équations en utilisant un langage déclaratif proposé par les auteurs : VCMDL, pour *Virtual Cell Mathematics Description Language*. Le code VCMDL est ensuite traduit en code C++ et envoyé à un solveur. La figure 5.2 illustre le processus de modélisation avec Virtual Cell. Les modèles conçus peuvent être stockés en ligne dans une base de données et être partagés si l'utilisateur le souhaite. L'environnement est bâti sur une architecture de type client / serveur, qui permet notamment d'utiliser des systèmes hautes performances pour exécuter les modèles construits. En matière de bilan, nous retenons donc les éléments suivants :

- le modèle est basé sur des mathématiques et de la physique et requiert des bases dans ces disciplines ;
- une interface graphique est toutefois utilisable pour manipuler le modèle ;
- l'utilisation d'un langage déclaratif permet de définir des équations ;
- des cas d'étude permettent de montrer les possibilités de la plateforme ;
- il est possible d'exécuter les modèles sur des machines hautes performances en ligne ;
- il n'y a pas d'information disponible sur l'implémentation de l'environnement (algorithmes ou structures de données) ;
- Des binaires de l'environnement sont disponibles pour différents systèmes d'exploitation.

Il s'agit d'un des plus anciens simulateurs pour l'étude des mécanismes sous-cellulaires. Il est bien documenté et largement éprouvé.

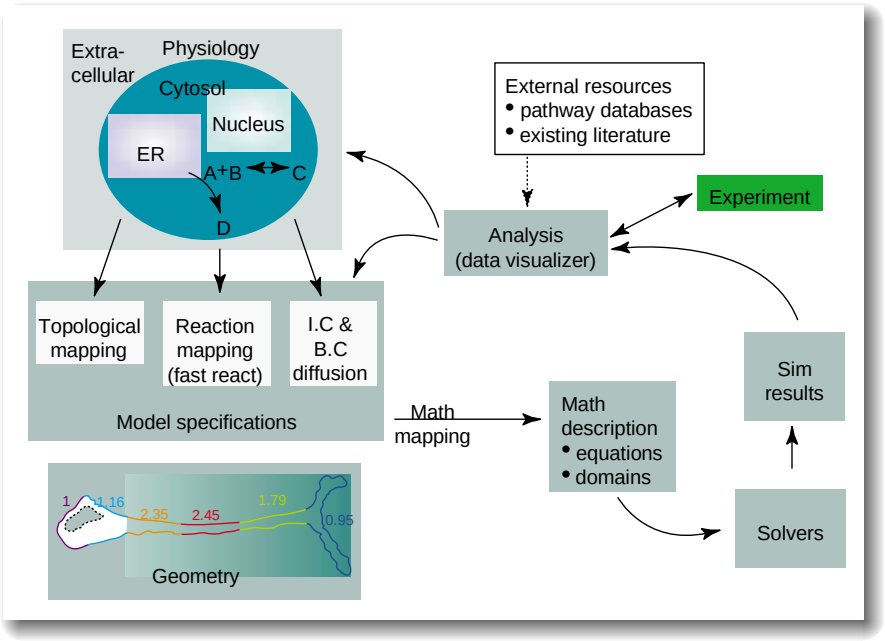


Figure 5.2 – Processus de modélisation avec l'environnement Virtual Cell. Figure extraite de (Loew et Schaff, 2001).

d) E-Cell

Proposé par Tomita et coll. (1999), E-Cell est également un environnement de simulation pour la modélisation et l'étude de la dynamique interne des cellules à travers la définition de fonctions de protéines ou encore de leurs interactions. Ces fonctions et interactions sont traduites en règles de réaction qui sont elles-mêmes traduites en équations différentielles. Le modèle est construit autour de trois éléments : les substances (constituant les cellules et le milieu dans lequel elles évoluent), les règles de réaction pouvant avoir lieu dans la cellule, et les structures spatiales ou fonctionnelles de la cellule et de son environnement. Des versions plus récentes d'E-Cell proposent d'aider au développement de modèles grâce à un IDE (*Integrated Development Environment*, soit environnement de développement intégré), disponible sous le système d'exploitation Windows.
En résumé, l'environnement de simulation E-Cell a les caractéristiques suivantes :

- il contient un modèle flexible ;
- il est très orienté pour les biologistes, grâce à la définition de règles de réaction qui seront transformées en équations différentielles par l'environnement ;
- un effort est réalisé en matière d'accessibilité à travers l'IDE permettant de développer des modèles ;
- l'environnement de simulation est disponible sous différents systèmes d'exploitation ;
- les publications ne donnent pas de détails quant aux algorithmes présents dans l'environnement, mais le code est ouvert et donc consultable par tous ;
- de nombreux cas d'étude ont permis d'apporter des éléments de validation de l'environnement de simulation.

E-Cell permet de coupler des voies métaboliques avec l'activation ou la désactivation d'un gène dans un réseau de gènes déjà pré-câblés, facilitant l'étude d'un modèle dans un contexte plus large.

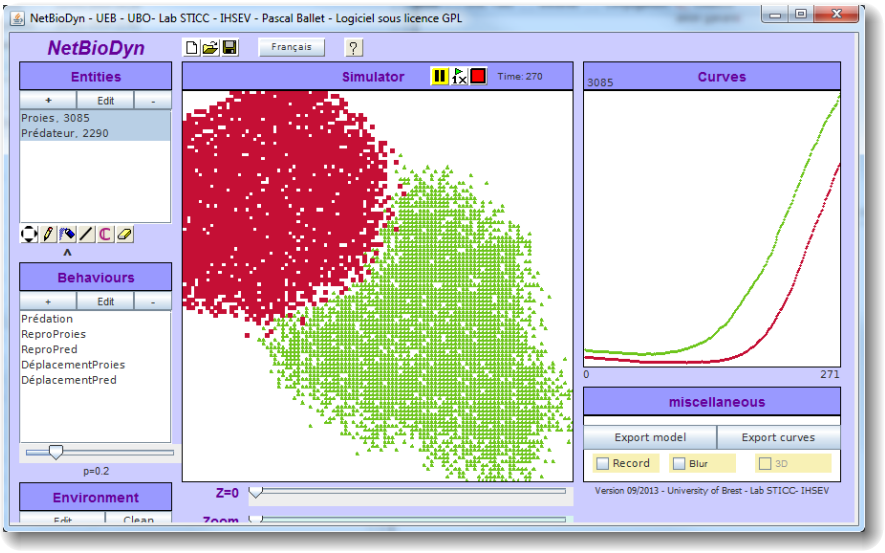


FIGURE 5.3 – Interface du logiciel NetBioDyn (Ballet, 2012).

e) Suite « Biodyn »

Proposée par Ballet (2012), la suite BioDyn comprend différents logiciels à vocation pédagogique, à destination à la fois des étudiants et des enseignants. Ils permettent également de faire du prototypage de modèles. La suite est avant tout orientée simulation multi-agents. Le modèle générique BioDyn est construit autour de quatre classes principales permettant de définir les propriétés spatiales des modèles (classe *Environnement*), les entités du modèle (classe *Entités*), les comportements des entités (classe *Moteur*) et d'exécuter les comportements définis dans le système (classe *Opérateur*). Les logiciels de cette suite contiennent des interfaces graphiques élaborées dans l'optique de faciliter l'utilisation des logiciels, la création de modèles ainsi que leur manipulation. Certains logiciels de la suite BioDyn sont livrés avec un modèle, comme par exemple dans le cas de SimBioDyn : ce logiciel permet de simuler une cellule élastique déformable ; des modules peuvent être codés en python et une interface graphique permet de manipuler le modèle. Un second exemple est NetBioDyn. NetBioDyn permet de réaliser des simulations de modèles basés sur les systèmes multi-agents et qui met à disposition une interface avancée, visible sur la figure 5.3, afin de définir et manipuler des modèles. Les points que nous retirons de la suite BioDyn sont les suivants :

- la suite comporte différents environnements de simulation empaquetés dans différents logiciels et permet d'implémenter et de manipuler différentes sortes de modèles ;
- la suite est construite sur la base d'une volonté d'accessibilité, notamment à travers la mise au point d'interfaces graphiques avancées ;
- les logiciels de la suite sont les seuls, à notre connaissance, à avoir une vocation avant tout pédagogique ;
- les logiciels BioDyn sont disponibles sur différents systèmes d'exploitation ;
- des cas d'études variés ont été implémentés avec les différents logiciels de la suite.

De part son accessibilité, il est difficile (bien que possible) de mettre en œuvre des simulations autres que qualitatives avec cette suite de logiciels.

2.2 Plateformes et simulateurs parallèles

a) CellSys

CellSys, proposé par Hoehme et Drasdo (2010), est un environnement de simulation pour l'étude de la croissance et de l'organisation de systèmes multi-cellulaires. Le modèle livré avec l'environnement est un modèle 3D de cellule élastique et adhésive. L'environnement de simulation est bâti autour de différents modules, permettant de définir le modèle, des variantes des algorithmes implémentant les comportements cellulaires, des algorithmes pour modéliser l'environnement des cellules et des algorithmes de visualisation. Les modules et les paramètres sont contrôlables à travers une interface graphique. Les algorithmes principaux de CellSys sont parallélisés avec OpenMP.
De cet environnement de simulation, nous retiendrons donc :

- le modèle livré avec l'environnement est spécifique mais très paramétrable ;
- l'interface graphique permet de manipuler le modèle et l'environnement de simulation ;
- l'utilisation d'OpenMP restreint le parallélisme à l'utilisation de machines à mémoire partagée ;
- il n'y a pas de détails sur les algorithmes implémentés dans l'environnement ;
- différents cas d'étude permettent d'apporter des éléments de validation du modèle et de l'environnement de simulation.

Notons que CellSys permet d'obtenir un rendu 3D de qualité et qu'un fort paramétrage des simulations est possible.

b) Plateforme parallèle pour la simulation de systèmes multi-cellulaires

La plateforme proposée par Da-Jun et coll. (2005) est plus ancrée dans le milieu de l'informatique que de la biologie : l'idée principale est avant tout de proposer un environnement de simulation permettant de simuler un nombre important d'entités (jusqu'à plusieurs dizaines de milliers). Le modèle fourni est orienté systèmes multi-agents et centré sur la cellule. La dynamique interne de celle-ci repose sur l'utilisation d'un logiciel tiers. Concernant l'environnement de simulation, l'emphase est mise sur les aspects d'équilibrage de charge : les agents naissant et mourant au cours d'une simulation doivent être équitablement répartis sur les nœuds de la grappe de machines utilisée.
Nous retenons les points suivant à propos de cet environnement de simulation :

- un modèle est fourni mais très peu de détails sont donnés à son sujet ou à propos de son paramétrage ;
- aucune information n'est donnée sur l'accessibilité de l'environnement de simulation aux non-programmeurs ;
- ce travail est clairement orienté vers l'informatique, et propose des algorithmes adaptés à la répartition de la charge de calcul sur les nœuds d'une grappe d'ordinateurs ;
- les auteurs ne donnent pas d'information sur la conception logicielle de l'architecture (classes, schéma UML, ...) ;
- il n'y a pas de cas d'étude permettant d'apporter des éléments de validation à l'environnement de simulation proposé.

Cette étude met en avant la capacité du système à simuler un grand nombre d'entités de type bactérie.

c) Plateforme pour la simulation multi-agents large échelle

L'environnement de simulation proposé par Lysenko et D'Souza (2008) est dédié à la modélisation et à l'exécution de simulation de systèmes multi-agents [1] sur GPU. C'est, à notre connaissance, un des premiers environnements de simulation proposés pour des exécutions sur GPU (si ce n'est *le* premier). L'idée première de ce framework était d'aller plus loin que les frameworks existants pour la simulation de systèmes multi-agents, en exploitant le parallélisme inhérent aux systèmes multi-agents grâce à l'utilisation des GPU. C'est un travail qui est maintenant un peu « ancien », utilisant des shaders en guise de kernels et stockant les données en tant que textures (voir la description des shaders et des GPU du chapitre 4 pages 108 et 110). L'intérêt de ces travaux réside dans le fait que leurs auteurs y proposent différentes techniques pour réaliser des implémentations efficaces de systèmes multi-agents, comme par exemple un système de priorités pour l'exécution des comportements des agents ou encore des mécanismes pour la gestion de la réplication des agents (ce dernier mécanisme nous a beaucoup intéressé, nous y revenons dans la section 3.3 de ce présent chapitre). Il s'agit donc d'un environnement que l'on peut qualifier de novateur, relativement à l'année à laquelle il a fait l'objet d'une publication. Nous retenons, de cet environnement, les différents points suivants :

- l'environnement de simulation est dédié aux systèmes multi-agents ;
- l'approche est totalement orientée vers l'informatique : il n'y a pas de module ou d'élément dans l'environnement de simulation pour permettre de faciliter son utilisation par des chercheurs de disciplines autres que l'informatique, telle que la biologie ;
- cet environnement est novateur : c'est un des premiers environnements de simulation à viser les GPU comme plateforme d'exécution ;
- des algorithmes dédiés à l'utilisation de GPU sont donc proposés, en particulier des mécanismes de gestion mémoire ;
- il n'y a pas d'information disponible sur l'architecture logicielle de l'environnement de simulation ;
- l'implémentation de systèmes multi-agents existants a permis d'apporter des éléments de validation de l'environnement de simulation.

L'application de cette plateforme à la biologie cellulaire reste ici anecdotique puisque l'étude adresse les systèmes multi-agents au sens large.

d) Flame : Flexible Large-scale Agent Modeling Environment

L'environnement Flame (Kiran et coll., 2010) est dédié à la simulation de systèmes multi-agents sur machines hautes performances. Les entités du modèle sont définies avec le langage de description XML, permettant ainsi de faciliter la manipulation de modèles et leur compréhension. Il est également nécessaire de connaître quelques bases en C puisque les comportements des agents sont décrits dans ce langage. Une API simple, décrite dans la documentation de Flame, permet de prendre en main la programmation des agents. Les agents sont modélisés par des X-machines, qui sont des variantes de machines à état, et communiquent entre eux avec MPI. Une version dédiée à l'exécution sur GPU avec CUDA a également été proposée (Richmond et coll., 2009a, b) : cette version utilise des structures de données adaptées au GPU et propose une utilisation judicieuse de la mémoire partagée du GPU pour optimiser les communications entre agents. Le bilan de cet environnement de simulation est le suivant :

1. Notons qu'il existe un nombre très important d'environnements de simulation multi-agents, parallèles ou non (NetLogo, Mason, Repast, ...). Une revue de quelques un de ces logiciels est proposée par Allan (2009).

- les modèles implémentables et simulables sont orientés systèmes multi-agents ;
- la définition et la manipulation de modèles est réalisée avec le langage de description XML, mais il est nécessaire d'utiliser le langage C pour décrire les comportements des agents ;
- l'environnement est disponible sur de multiples plateformes, logicielles ou matérielles ;
- la version GPU repose sur des structures de données adaptées et propose des algorithmes dédiés pour la communication des agents ;
- différents cas d'étude permettent d'apporter des éléments de validation de l'environnement de simulation.

Comme dans le cas du système précédent, ici la biologie cellulaire n'est pas ciblée en particulier. Cela rend l'utilisation de cette plateforme peu pratique pour notre sujet.

e) CompuCell 3D

CompuCell 3D (Swat et coll., 2012) est un environnement de simulation pour l'étude de la morphogenèse de tissus. Nous avons déjà évoqué cet environnement, dans le chapitre 2, section 4 page 51, notamment pour présenter le modèle hybride utilisé dans cet environnement de simulation. Il s'agit d'un modèle de Potts cellulaire couplé à des sous-modèles afin de modéliser différents niveaux présents dans les systèmes biologiques. Ce modèle est flexible et manipulable dans CompuCell 3D à travers une interface graphique (visible sur la figure 5.4), le langage de description XML et des scripts Python. Les algorithmes au cœur de CompuCell 3D ont été parallélisés avec OpenMP. À notre connaissance, cet environnement de simulation est l'un des plus complets et des plus réputés pour simuler des tissus biologiques. Les éléments clefs de CompuCell 3D sont les suivants :

- l'environnement de simulation est livré avec un modèle hybride et flexible ;
- ce modèle est manipulable à l'aide d'une interface graphique avancée, du langage déclaratif XML et de scripts python ;
- OpenMP est utilisé pour paralléliser les algorithmes au cœur du simulateur ;
- les publications traitant de CompuCell 3D ne détaillent pas la conception de l'environnement de simulation, mais celui-ci est libre et ses sources sont donc accessibles à tous ;
- CompuCell 3D est disponible pour différents systèmes d'exploitation ;
- de nombreux cas d'étude ont été réalisés avec CompuCell 3D.

Il s'agit ici d'un des simulateurs les plus aboutis et pertinents pour notre domaine d'étude.

f) Matrix Studio

Matrix Studio est un environnement de développement intégré pour OpenCL (où *Integrated Development Environment*, IDE) proposé par Ballet (2012). Contrairement aux logiciels et environnements présentés jusqu'à maintenant, aucun modèle n'est livré avec l'environnement. La vocation première de Matrix Studio est d'être utilisé à des fins pédagogiques par des étudiants en informatique découvrant OpenCL. Cet IDE peut également être utilisé à des fins de prototypage de modèles. Son avantage principal réside dans le fait que le code hôte (voir chapitre 4, section 4.4) est masqué à l'utilisateur, qui n'a qu'à définir ses données et développer ses kernels. L'interface graphique de l'IDE permet de définir les matrices qui contiendront les données, ordonnancer les kernels créés et définir des fonctions tierces pouvant être utilisées dans les kernels. Il est également possible de choisir le périphérique sur lequel exécuter le code OpenCL. Voici les éléments à retenir concernant Matrix Studio :

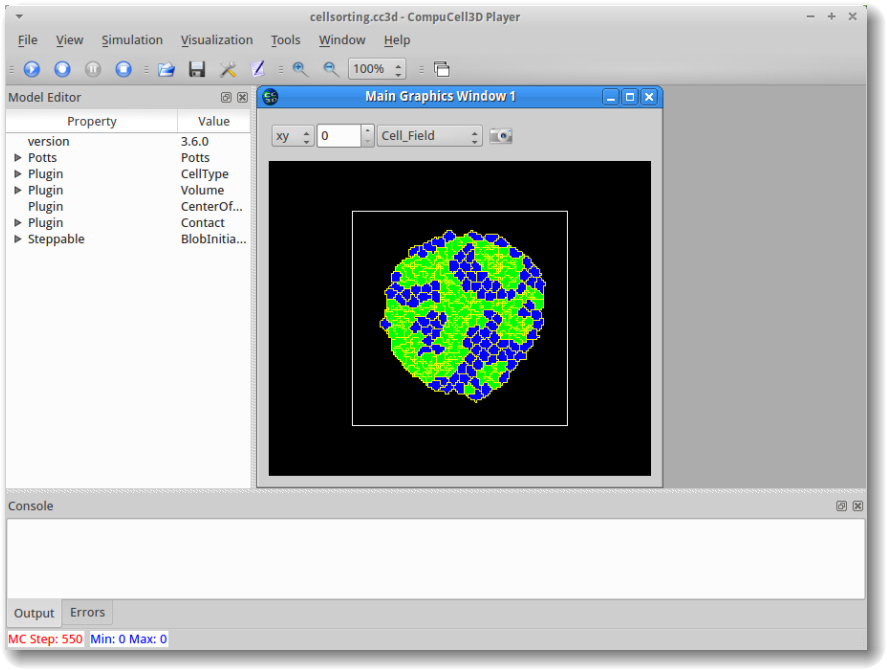


FIGURE 5.4 – Interface de l'environnement de simulation CompuCell 3D.

- tout type de modèle est implémentable (système multi-agents, modèle de Potts cellulaire, modèle continu pour la diffusion de molécules, etc.) ;
- l'IDE s'adresse à des informaticiens : l'intégralité du code OpenCL est à écrire « à la main » pour définir un modèle ;
- l'interface graphique permet de facilement définir ses données et l'ordonnancement des kernels créés ;
- l'IDE est portable et utilisable sur différents systèmes d'exploitation ;
- l'utilisation d'OpenCL permet d'exécuter les modèles créés sur une variété importante de périphériques.

Cet environnement a une vocation généraliste et donc ne propose aucune facilité pour l'étude des systèmes cellulaires.

2.3 Synthèse et définition des besoins

La liste d'environnements de simulation présentée ci-dessus n'est pas exhaustive : il existe beaucoup de plateformes de simulation et il n'est pas possible, ni même pertinent, de les présenter toutes. À la place, nous avons choisi les plateformes qui nous semblaient les plus pertinentes afin tout d'abord de savoir dans quelles mesures il pouvait être nécessaire de proposer un autre environnement de simulation. Cela nous a permis de dégager les avantages et inconvénients des approches étudiées pour proposer un simulateur adapté.

Au vu des éléments présentés au cours de cette section, nous avons élaboré une liste de caractéristiques que l'on retrouve (ou non) dans les environnements de simulation. Cette liste est consultable dans le tableau 5.1, qui donne également des précisions associées à chaque caractéristique. Ces critères sont utilisés dans le tableau 5.2 afin de distinguer les différents travaux étudiés dans cette section.

Table 5.1 – Éléments principaux permettant de caractériser les environnements de simulation.

Modèles implémentables	Les environnements de simulation sont le plus souvent spécifiques à un modèle, au mieux à un type de modèle
Utilisation de l'environnement	Les concepteurs de l'environnement peuvent mettre en place des interfaces graphiques ou encore l'utilisation de fichier de configuration, afin de permettre au plus grand nombre d'utiliser le simulateur (en opposition avec la nécessité de coder « en dur »)
Portabilité en matière de systèmes d'exploitation	Toujours dans une optique d'être utilisable par le plus grand nombre, les concepteurs peuvent proposer des binaires pour différents systèmes d'exploitation, voire distribuer les sources de leur simulateur
Portabilité en matière de matériel	Cette caractéristique est une des plus délicates à mettre en place, notamment lorsque le simulateur propose l'exécution parallèle du modèle créé / implémenté : il est avantageux de proposer un simulateur qui peut s'exécuter sur des plateformes matérielles variées (CPU et GPU notamment)
Formalisation de la conception du logiciel	L'utilisation d'un standard comme UML pour décrire l'architecture logicielle de l'environnement de simulation permet de formaliser celle-ci
Algorithmes dédiés à la résolution de problèmes spécifiques	Des algorithmes très spécifiques peuvent être requis pour pallier certains problèmes inhérents au modèle, au langage utilisé pour implémenter l'environnement ou à la plateforme matérielle utilisée
Éléments de validation à travers des cas d'étude	Proposer quelques cas d'étude permet d'obtenir des éléments de validation de l'environnement de simulation ou du modèle qui y est implémenté, ainsi que de mieux appréhender les possibilités du simulateur et du modèle

TABLE 5.2 – Classification des architectures décrites selon les critères définis dans le tableau 5.1. Certains travaux ne donnent pas d'informations sur des critères mais quelques un sont des outils dont le code source est disponible, l'information est donc potentiellement accessible.

	Modèles implémentables	Utilisation de l'environnement	Portabilité (systèmes d'exploitation)	Portabilité (matériel)	Formalisation de l'architecture	Algorithmes dédiés	Cas d'étude
GemCell (Amir-Kroll et coll., 2008)	✗	✓	✗	✗	✓	✗	✗
VirtualLeaf (Merks et coll., 2011)	✗	✗	✓	✗	✗	✗ (code libre)	✓
VirtualCell (Loew et Schaff, 2001)	✗	✓	✓	✗	✗	✗	✓
E-Cell (Tomita et coll., 1999)	✗	✓	✓	✗	✗	✗ (code libre)	✓
Suite BioDyn (Ballet, 2012)	✓	✓	✓	✗	✓	✗	✓
CellSys (Hoehme et Drasdo, 2010)	✗	✓	✓	✓	✗	✗	✓
(Da-Jun et coll., 2005)	✗	✗	✓	✓	✗	✓	✗
(Lysenko et D'Souza, 2008)	✗	✗	✓	✓	✗	✓	✓
Flame (Kiran et coll., 2010)	✗	✓	✓	✓	✓	✓	✓
CompuCell 3D (Swat et coll., 2012)	✓	✓	✓	✗	✗	✗	✓
Matrix Studio (Ballet, 2012)	✓	✓	✓	✓	✓	✗	✓

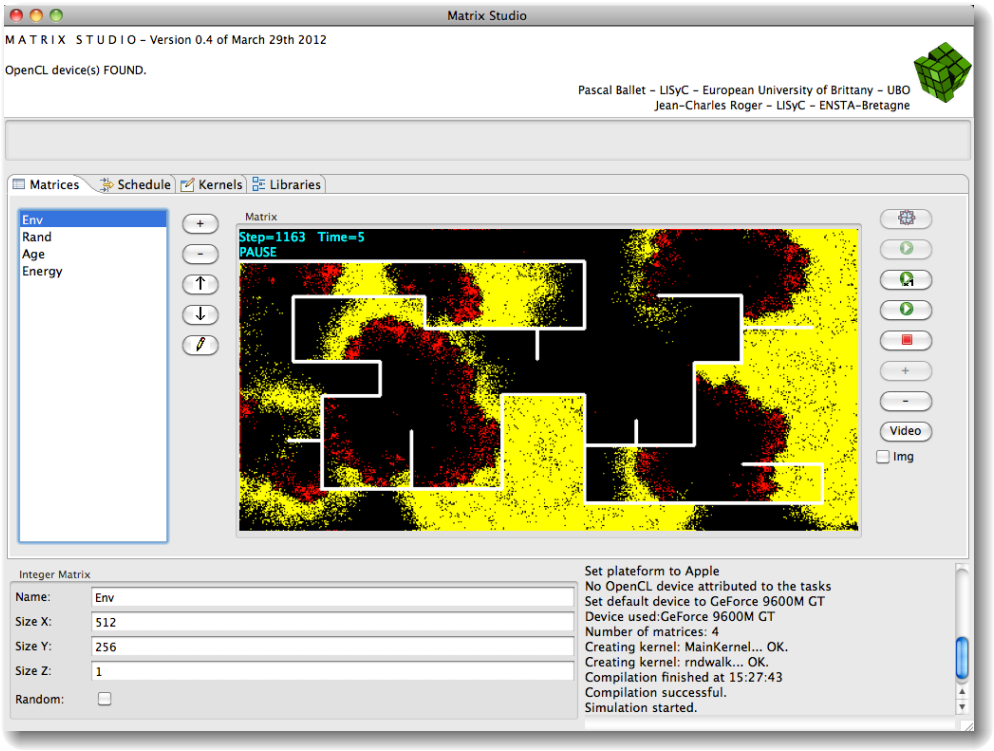


Figure 5.5 – Interface graphique de Matrix Studio.

Orientation de nos travaux. Nous souhaitons modéliser et simuler une vaste gamme de tissus cellulaires, tant dans leur phase de morphogenèse que dans leur phase de méta-stabilité (renouvellement tissulaire). Cette gamme de tissus pourrait être composée de types cellulaires différents comme des leucocytes circulant dans le système veineux, des cellules endothéliales bien maintenues entre elles et montrant des caractéristiques d'adhésion différentielle ou encore des macrophages capables de se déformer, de migrer, d'envoyer et de recevoir des messages chimiques. De plus, nous voulons que le nombre de cellules composant les tissus virtuels soit proche du million. Enfin, il est important que l'implémentation de notre système soit capable de suivre l'évolution technologique des systèmes multi-cœurs, moyen actuel privilégié par les constructeur de processeurs (CPU et GPU) pour augmenter les puissances de calcul. Autrement dit, notre implémentation doit utiliser l'architecture multi-cœurs des processeurs en maximisant automatiquement l'utilisation des cœurs en fonction des disponibilités variables du matériel utilisé, soit présent, soit futur.

Le tableau 5.2 nous permet de dégager les éléments principaux sur lesquels nous avons souhaité mettre l'accent lors de la conception de notre simulateur. Nous avons en particulier retenu deux points : tout d'abord, les différentes plateformes présentées ne permettent pas, pour la plupart, d'implémenter différents types de modèles. Même si, comme précisé au début de ce chapitre, il n'est pas possible de concevoir une plateforme de simulation totalement générique, il nous semble important que celle-ci puisse accueillir différents types de modèles compte tenu du caractère hybride des systèmes biologiques. La capacité de simuler autant des aspects discrets que continus nous paraît essentielle. Le second point concerne la portabilité en terme de matériel. La plupart des plateformes étudiées dans la revue de cette section sont portables en matière de systèmes d'exploitation mais notre opinion est qu'il faut aller plus loin et proposer également de la portabilité en matière de matériel. Cette portabilité

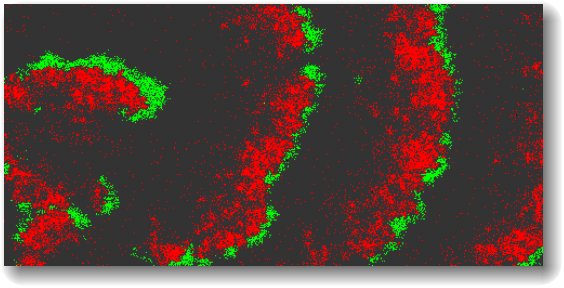
est présente dans certaines plateformes. Cependant, nous pensons également que l'utilisation du framework OpenCL peut permettre d'aller plus loin car les constructeurs proposent de plus en plus des implémentations d'OpenCL dans leurs différents matériels ; ça n'est le cas pour aucun autre framework de programmation à l'heure actuelle.

Compte tenu des éléments dégagés de cette partie et synthétisés dans les tableaux 5.1 et 5.2, ainsi que des éléments présentés dans les chapitres précédents, nous avons décidé de proposer notre propre environnement de simulation. En premier lieu, nous avons proposé, au chapitre 3, notre propre modèle de cellule virtuelle. Compte tenu de la spécialisation de la plupart des environnements de simulation, il nous a paru plus judicieux de réaliser nous-même l'implémentation du modèle dans un environnement personnalisé. En outre, nous avons également, comme précisé au chapitre 4, illustré notre intérêt pour la programmation parallèle avec OpenCL. À notre connaissance, il n'existe pas de simulateur multi-agents dédiés à la biologie se reposant sur OpenCL. Il existe bien quelques environnements de développement, comme Matrix Studio (Ballet, 2012) que nous avons évoqué ci-dessus, cependant les IDE sont dédiés au développement et donc visent à priori un public de programmeurs. Nous avons préféré orienter notre simulateur vers un public plus hétérogène en essayant de proposer une manipulation simplifiée d'une partie du simulateur à travers une architecture logicielle dédiée et formalisée, ainsi qu'une manipulation simplifiée du modèle à travers un fichier de paramètres haut niveau, dont la syntaxe est accessible aux non programmeurs. En outre, notre simulateur contient des algorithmes et structures de données spécifiques adaptés à l'utilisation d'OpenCL, concernant en particulier la gestion de la mémoire (nous avons vu au chapitre 4 qu'OpenCL ne permet pas l'allocation dynamique de mémoire à l'exécution, il a donc fallu pallier cette limite pour assurer autant que possible aux entités du système qu'elles puissent se répliquer, la réplication étant un comportement primordial dans l'étude du développement de tissus). Notre proposition est orientée vers la simulation de systèmes multi-agents, mais l'aspect hybride de notre modèle, avec la chimie artificielle d'un côté et de l'autre les déplacements continus de nos cellules nous ont conduit à inclure des solveurs afin de prendre en compte les équations nous ayant permis de modéliser ces éléments. Nous verrons également que les structures de données proposées permettent d'implémenter des entités de formes variées : nous avons quelques cas d'étude où l'entité est représentée par un point dans l'espace (une particule), alors que dans d'autres cas, nous utilisons notre cellule virtuelle qui a une forme plus élaborée et complexe : la figure 5.6 présente quelques exemples de visualisation de modèles implémentés dans notre simulateur.

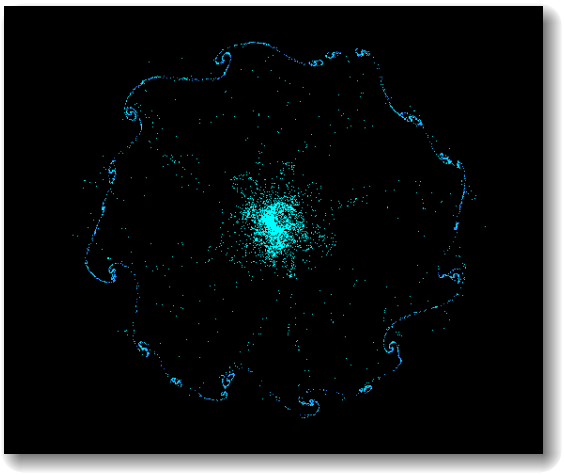
L'objet de la section suivante est donc de présenter les différents éléments que nous avons souhaité inclure dans notre simulateur. Nous faisons également le lien avec notre modèle de cellule virtuelle en montrant comment celui-ci a concrètement été implémenté dans notre simulateur.

3 Proposition de simulateur

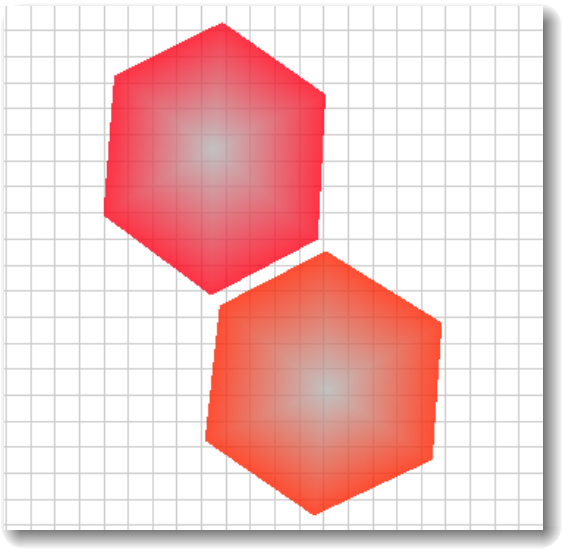
Nous avons basé la conception de notre simulateur sur le flot de conception que nous proposons sur la figure 5.7. Chaque étape lors de la conception d'un simulateur, de son architecture logicielle à la simulation résultante, demande des compétences variées. Durant cette thèse, et bien que nous ne soyons pas des spécialistes de la biologie, nous avons décidé de mettre l'accent sur l'aspect modèle computationnel ainsi que sur l'aspect architecture logicielle de notre simulateur. Nous avons en particulier travaillé à la mise en place de ce que l'on a considéré comme un pont, afin de masquer les aspects « architecture » et « modèle », pour faciliter l'intégration de modèles ainsi que leur simulation en environnement



(a)



(b)



(c)

FIGURE 5.6 – Visualisation d'exemples de modèles implémentés dans notre simulateur. (a) Système proies/prédateurs discret, implémenté avec un système multi-agents. Les agents du systèmes sont physiquement représentés par des particules. (b) Modèle continu : utilisation des équations de mouvements de Newton pour modéliser les interactions dans un problème à n corps. Les corps sont physiquement représentés par des particules. (c) Modèle de cellule virtuelle que nous proposons : dans ce cas le modèle est hybride (des éléments discrets sont couplés à des éléments continus). La représentation physique des cellules est complexe : il s'agit d'un polygone à N sommets (6 dans le cas présent).

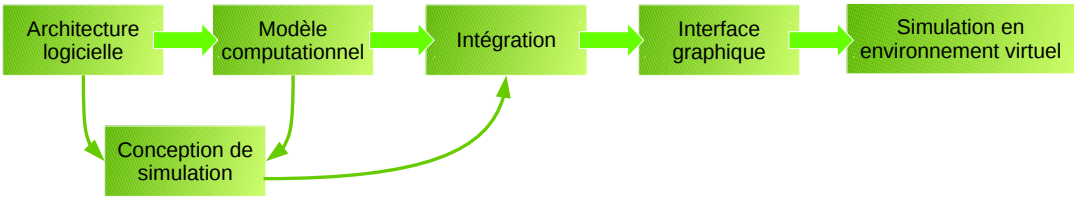


FIGURE 5.7 – Flot de conception de simulations. Nous avons basé notre simulateur sur l'idée d'abstraire l'architecture logicielle (qui est générique) sous-jacente du simulateur, ainsi que le modèle computationnel (qui repose sur des structures de données adaptées à l'utilisation de différents modèles).

virtuel. Bien que nous ayons vu, dans la section précédente, qu'une interface graphique puisse grandement faciliter la manipulation à la fois du simulateur et du modèle, nous n'en proposons pas à l'heure actuelle pour la manipulation du modèle. Nous verrons cependant dans la section 3.4 que nous avons mis en place un fichier de paramètres utilisant une syntaxe plus simple à manipuler que du code. Nous avons toutefois mis en place une interface simple permettant de visualiser l'environnement virtuel et ce qu'il s'y déroule à travers une fenêtre de visualisation.

Dans la suite de cette section, nous présentons 1) l'architecture logicielle générique de notre simulateur, 2) les structures de données que nous utilisons, 3) les mécanismes de gestion mémoire qui permettent de répondre au besoin de réplication du système multi-agents et 4) l'utilisation du fichier de paramètres mis en place pour faciliter la manipulation du modèle.

3.1 Architecture générale

Nous avons établi qu'il n'était pas réaliste de proposer un simulateur capable d'accueillir tout type de modèle (section 1 de ce chapitre). Compte tenu du caractère hybride du modèle que nous proposons, il nous a toutefois paru nécessaire de proposer une architecture logicielle aussi générique que possible, afin d'élargir la palette d'éléments qu'il est possible de simuler avec un tel simulateur : par exemple, on peut vouloir réaliser le déplacement d'une entité de manière discrète dans l'environnement ou encore d'utiliser une équation pour modéliser ce mouvement de manière continue.

Ainsi, pour construire notre simulateur, nous nous sommes reposés sur l'utilisation d'un patron de conception. Les patrons de conception sont utilisés dans l'optique de réutiliser du code ainsi que de disposer de logiciels plus aisés à maintenir. Il existe différentes catégories de patrons de conception : nous nous sommes intéressés aux patrons de type Comportements (pour plus de détails concernant les patrons de conception, les lecteurs intéressés peuvent se reporter à l'ouvrage de Debrauwer (2013)). Le patron de conception « stratégie » permet de définir des comportements pouvant être déclinés en différentes versions selon les besoins. Ce patron de conception est illustré sur la figure 5.8 : la classe abstraite *AbstractSimulation* permet de définir automatiquement certains paramètres communs à toute simulation, comme par exemple le nombre d'entités maximal de la simulation. La classe *ModuleStrategy* représente le cœur du patron de conception et permet de définir les comportements génériques du simulateur : un tel comportement peut être l'affichage des entités dans l'environnement virtuel, dont on imagine bien qu'il sera différent selon que les entités soient des particules, des sphères ou encore des polygones. La spécialisation des comportements est réalisée grâce aux classes *ConcreteStrategy*.

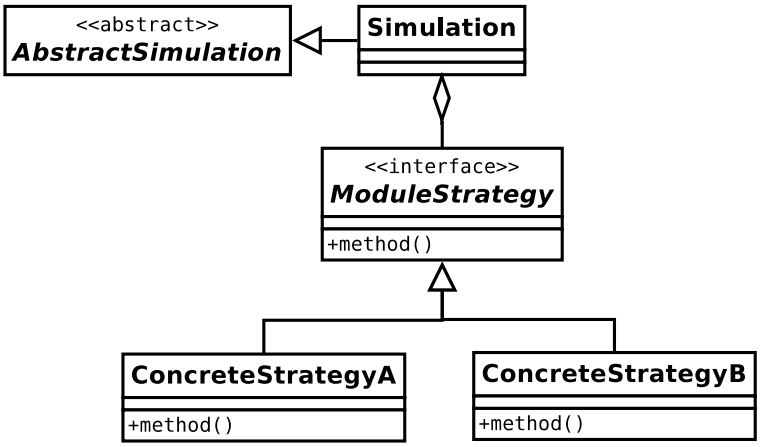


Figure 5.8 – Architecture logicielle générique de notre simulateur : utilisation du patron de conception « stratégie ». La classe abstraite *AbstractSimulation* permet de définir automatiquement des paramètres communs à toute simulation. La classe *ModuleStrategy* représente un comportement, qui doit être décliné et spécialisé dans les classes *ConcreteStragety* en fonction des besoins de la simulation.

Notre simulateur repose donc sur la définition de modules, qu'il convient de spécialiser en fonction des simulations que l'on souhaite réaliser. Pour l'heure, nous avons défini trois modules principaux, illustrés sur la figure 5.9 :

- le module de rendu permet d'afficher les éléments du système dans un environnement virtuel en trois dimensions. Ce rendu est réalisé en utilisant la bibliothèque OpenGL (Wright et coll., 2011). Cette bibliothèque ouverte est utilisable sur différents types de systèmes d'exploitation, différents matériels, et elle permet surtout un couplage avec OpenCL (nous revenons sur ce point dans la section 3.3) ;
- le module *RunModel* contient l'implémentation même du modèle que l'on souhaite simuler. En particulier, il s'agit d'un côté de l'ordonnancement des kernels OpenCL pour ce qui concerne le côté hôte et de l'autre de l'implémentation des comportements dans les kernels OpenCL ;
- le module *ExportData* permet, en cours de simulation, d'exporter et de sauvegarder des données en format texte, pour une analyse à postériori. Par exemple, on peut vouloir exporter, à intervalles réguliers, l'état des cellules afin d'étudier leur évolution dans leur cycle cellulaire. C'est avec ce module que nous avons exporté les données relatives aux contraintes mécaniques présentées dans le chapitre 3 (page 76).

Nous avions également évoqué, au chapitre précédent, que nous souhaitions masquer le plus de code hôte possible, afin de faciliter l'implémentation de modèles. Ceci peut être fait de manière générique pour tout ce qui concerne : la récupération des plateformes OpenCL ainsi que des périphériques associés, la création de contextes, la création de la file de commandes, ainsi que la création du programme OpenCL. Il reste à instancier les éléments qui dépendent directement du modèle à simuler : les kernels, qu'il faut également ordonnancer, ainsi que les buffers de données, qui devront être envoyés sur le périphérique choisi.

Notre architecture logicielle est illustrée sur la figure 5.10. Les modules de rendu, d'exécution de modèle et d'export sont des méthodes abstraites génériques qu'il faut redéfinir et implémenter dans la classe *Simulation*. Les classes abstraites contenant ces méthodes abstraites contiennent également d'autres éléments : la classe *CL_object* contient le code hôte « masqué » ainsi que des attributs génériques hérités par *Simulation* ; la classe *GL_object*

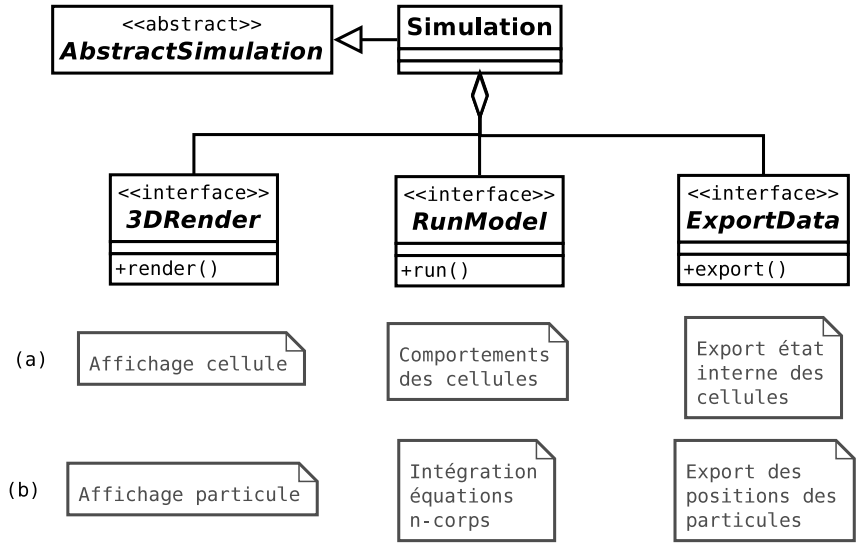


FIGURE 5.9 – Architecture logicielle générale de notre simulateur. Les interfaces correspondent à des méthodes abstraites qu'il faut définir dans chaque simulation différente. Pour illustrer cela, on donne l'exemple de méthodes que l'on peut définir différemment selon le système : la ligne (a) correspond à l'implémentation des modules pour le modèle de cellule virtuelle ; la ligne (b) correspond à l'implémentation des modules pour un modèle de particules se déplaçant selon les équations de mouvement de Newton pour N corps.

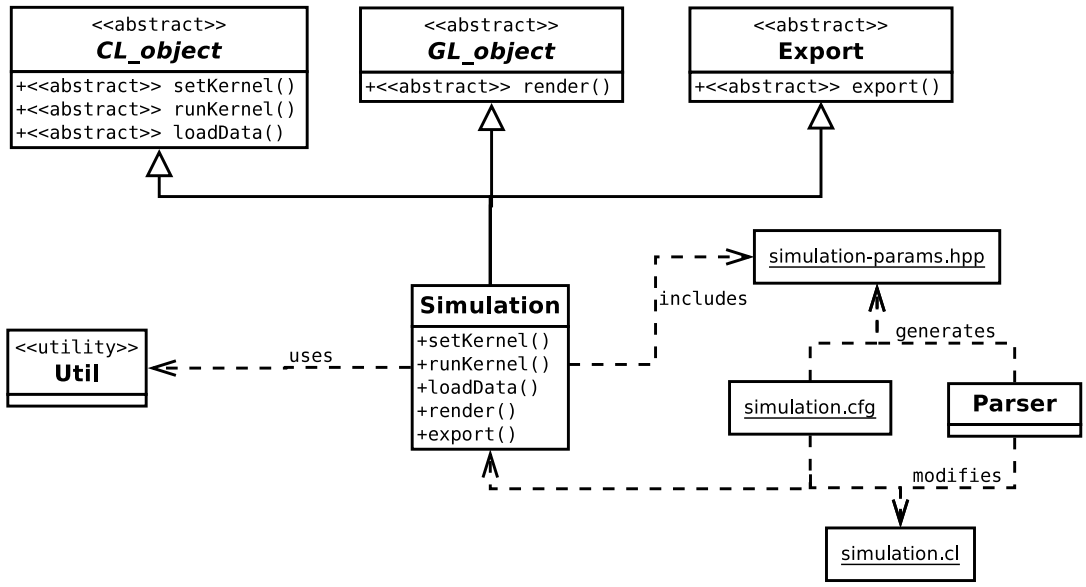


FIGURE 5.10 – Implémentation de notre simulateur. Les modules pour le rendu, l'exécution du modèle et l'export de données sont implémentés comme des méthodes abstraites à redéfinir et implémenter dans la classe *Simulation*. La classe *CL_object* contient également le code hôte commun à tout programme OpenCL. La classe *GL_object* gère également la fenêtre d'affichage de l'environnement virtuel (attributs et fonctions de rappel). Le fichier de paramètres `simulation-params.hpp` est généré grâce à un parseur et au fichier de configuration `simulation.cfg`. Sur la base du fichier de configuration, le code source de la classe *Simulation* et de `simulation.cl` (contenant les kernels OpenCL) vont être modifiés automatiquement, afin de permettre à l'utilisateur de ne pas avoir à reprogrammer certains éléments en cas de modifications des paramètres.

contient des attributs génériques ainsi que le code pour la gestion de la fenêtre d'affichage de l'environnement virtuel (attributs de la fenêtre, de l'environnement et fonctions de rappel notamment). Il est à noter que le module *RunModel* est implémenté à travers trois méthodes distinctes : `loadData()` permet de créer les buffers de données et de les envoyer dans la mémoire du périphérique choisi ; `setKernel()` permet de créer les objets kernels selon les fonctions kernels écrites dans le code source OpenCL, ainsi que d'envoyer les arguments de ces kernels ; enfin, la méthode `runKernel()` concerne l'ordonnancement des kernels créés (c'est dans cette méthode qu'est implémenté notre mécanisme de gestion mémoire, présenté dans la section 3.3).

Les paramètres de la simulation sont définis dans le fichier `simulation.cfg`. Ce fichier est traité par un parseur qui permet de générer le fichier de paramètres inclus dans *Simulation*. Le parseur permet également de modifier les codes sources de la classe *Simulation* et des kernels OpenCL en cas de modification des paramètres (ces aspects sont détaillés dans la section 3.4).

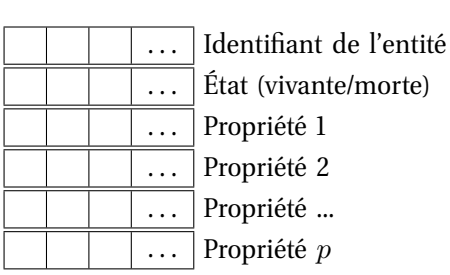
Nous avons donc pu définir une architecture logicielle généraliste pour la conception de simulations de systèmes biologiques. En pratique, l'instanciation de l'architecture pour un modèle donné nécessite l'implémentation de différentes méthodes, mais nous avons pu masquer une partie du code OpenCL (qui concerne des opérations à réaliser pour mettre en place le contexte d'exécution), ainsi que l'affichage de l'environnement virtuel. L'architecture logicielle proposée repose sur des structures de données optimisées pour l'utilisation d'OpenCL et, à fortiori, pour l'utilisation de matériels hétérogènes tels que les CPU et les GPU pour l'exécution du modèle.

3.2 Structures de données

Compte tenu du fait que dans notre simulateur, les modèles sont implémentés avec OpenCL, les structures de données utilisées permettent tout d'abord de faire en sorte (1) que les work-items soient aussi actifs que possible (partage de charge) et (2) que les accès mémoires soient efficaces. Nous détaillons ces deux points dans ce qui suit.

a) Activité des work-items

Étant donné qu'OpenCL ne permet pas l'allocation dynamique de mémoire à l'exécution, nous allouons, de manière statique, suffisamment de mémoire pour pouvoir stocker au plus M entités. Cette approche est justifiée grâce à la notion d'homéostasie, caractérisant la stabilité d'un système. Par exemple, dans le cas d'un tissu biologique sain, le nombre de morts et de naissances de cellules est tel que la taille du tissu est stable. Nous n'avons donc pas besoin de pouvoir créer une infinité d'entités. Il est cependant indispensable de permettre à nos entités de pouvoir se répliquer. Durant une simulation, une entité est associée à un work-item (qui correspond donc à une ressource de calcul ainsi qu'à une ressource mémoire associée). Les entités peuvent être dans deux états de base : active (autrement dit vivante) ou inactive (autrement dit morte). Les work-items des entités inactives ne travaillent pas, ce sont donc des ressources disponibles pour l'allocation d'une nouvelle entité. En attribuant à M une valeur adéquate, en fonction du système à simuler, il est possible de faire en sorte, d'une part, que nous ne gaspillons pas trop de ressources (ressources de calcul et ressources mémoire) et, d'autre part, que les entités auront quand même la possibilité de se répliquer. La détermination de M n'est pas triviale et ne peut à priori être réalisée qu'empiriquement en testant le système à simuler.



(a)

```
struct data{
  ushort state[NB_AGENTS];
  type prop_1[NB_AGENTS];
  type prop_2[NB_AGENTS];
  /* ... */
  type prop_p[NB_AGENTS];
};
```

(b)

FIGURE 5.11 – Structures de données de notre simulateur. (a) Représentation formelle des données : celles-ci sont stockées dans des tableaux (autant que de propriétés). Les propriétés sont variées : il peut s'agir par exemple de l'âge, de l'énergie, du facteur d'échelle que l'on a défini au chapitre 3, ou de toute autre propriété nécessaire à la simulation. (b) En pratique, les tableaux sont encapsulés dans une structure C, facilitant ainsi l'accès à chaque propriété. Il faut noter qu'il n'est pas nécessaire de stocker un tableau avec les identifiants des entités, comme dans la représentation formelle des données : étant donné qu'une entité est associée à un work-item, si i est l'identifiant du work-item alors l'accès à la propriété p est réalisé comme ceci : `data->prop_p[i]` : l'identifiant i d'un work-item peut facilement être obtenu avec la fonction `get_global_id()` fournie par OpenCL.

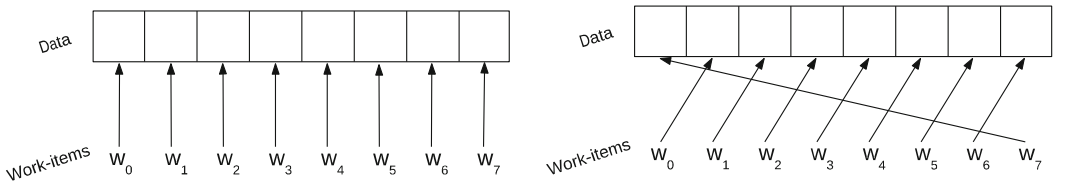
b) Accès mémoire

Les données des entités sont stockées dans la mémoire globale. Nous avons fait ce choix car il n'est pas du tout trivial de découper les données de façon à optimiser les accès grâce à la mémoire locale, si toutefois un tel découpage est possible (cela dépend du problème traité). Au lieu de se focaliser sur le découpage des données, nous avons donc choisi d'optimiser au mieux les accès à la mémoire globale. La mémoire globale est organisée en banques (des positions séquentielles dans la mémoire) qui ont la particularité de pouvoir être accédées en une fois par un groupe de travail : c'est ce qu'on appelle la coalescence. Avant de revenir sur cet aspect, précisons dans quel type de structures sont stockées les données : nous utilisons des tableaux, encapsulés dans des structures C, comme illustré sur la figure 5.11. Cette encapsulation permet de grandement faciliter l'accès aux données. Il est à noter qu'il est couramment admis que l'utilisation de structures de tableaux est une bonne façon de stocker des données globales avec OpenCL (Richmond et coll., 2009a; Chen et coll., 2011).

La notion de coalescence est maintenant plus claire : cette façon de stocker les données permet à des work-items consécutifs d'accéder à des positions mémoires consécutives (les banques). La figure 5.12 illustre cette notion d'accès consécutifs.

La structure de données présentée permet de stocker les propriétés des entités. L'enregistrement de la position des entités dans l'environnement se fait avec une structure de données semblable. Dans le cas de notre modèle de cellule virtuelle, nous utilisons autant de tableaux que de nœuds dans la structure physique de la cellule. Nous avons donc une structure telle que celle-ci :

```
struct nodes{
  float4 centers[NB_AGENTS];
  float4 nodes0[NB_AGENTS];
  /* ... */
  float4 nodesN[NB_AGENTS];
};
```



(a) **(b)**

Figure 5.12 – Exemple d'accès coalescents (a) et non coalescents (b) en mémoire globale : (a) le $k^{\text{ième}}$ work-item accède à la donnée en position k, une seule transaction est réalisée pour les 8 work-items de l'exemple ; (b) les accès ne sont pas alignés : 8 transactions sont réalisées pour les 8 work-items de l'exemple.

(a) **(b)**

Figure 5.13 – Fragmentation mémoire lors d'une simulation. Les cases noires représentent des entités vivantes, les cases blanches des entités mortes. (a) En début de simulation, les entités vivantes sont stockées de manière contiguës en mémoire. (b) Au fur et à mesure que les pas de simulation s'enchaînent, des entités meurent ou se répliquent et les cases vides et non vides se « mélangent » dans la mémoire : on parle alors de fragmentation.

Dans une structure identique, nous stockons les couleurs des cellules, pour le rendu dans l'environnement virtuel. Concernant le stockage de la position des nœuds et des couleurs des cellules, il existe un autre avantage à utiliser la mémoire globale : comme nous l'avons déjà mentionné, nous utilisons OpenGL pour le rendu graphique des simulations. *OpenGL et OpenCL peuvent fonctionner en collaboration, en partageant les données d'un même buffer.* En pratique, le contexte mis en place lors de la définition de l'environnement d'OpenCL est partagé entre OpenCL et OpenGL. Cela signifie qu'il n'est pas nécessaire de réaliser de transfert de données lorsque l'on souhaite réaliser un rendu ou lorsque l'on veut exécuter des kernels OpenCL qui modifient les données. Entre deux appels à des primitives OpenGL, une barrière permet de synchroniser les données.

Pour récapituler, nous avons donc choisi de stocker au plus M entités dans des structures de tableaux. Il a été nécessaire de mettre en place des mécanismes de gestion mémoire afin de pouvoir réutiliser les ressources de calcul et les ressources mémoires rendues disponibles par la mort d'entités ; c'est l'objet de la sous-section suivante.

3.3 Mécanismes de gestion mémoire

L'allocation dynamique de mémoire n'étant pas possible avec OpenCL et sachant qu'il faut permettre aux entités de pouvoir se répliquer si des ressources mémoire sont disponibles, nous proposons une stratégie basée sur deux méthodes différentes afin d'autoriser l'allocation de nouvelles entités. Comme nous l'avons vu, les données des simulations sont stockées dans des tableaux. Au fur et à mesure d'une simulation, alors que des entités disparaissent et que d'autres naissent, les tableaux vont se fragmenter, comme le montre la figure 5.13b. Pour allouer une nouvelle entité, il faut pouvoir trouver un emplacement disponible dans la mémoire, mais la recherche d'une ressource disponible dans un tableau fragmenté a un coût

important. Pour limiter ce coût au maximum, tout en faisant en sorte de garantir autant que possible aux entités qu'elles pourront se répliquer, nous proposons une stratégie basée sur deux méthodes couplées et nous permettant de composer avec la fragmentation mémoire. La première méthode que nous proposons consiste à trier la mémoire, afin de la défragmenter, de façon à ce que celle-ci soit de nouveau dans l'état visible sur la figure 5.13a.

En algorithmique, les opérations de tri peuvent être implémentées de différentes manières pouvant être plus ou moins efficaces et coûteuses. Tripodi et coll. (2012), qui ont proposé une version parallèle de modèle de Potts cellulaire avec OpenCL, ont proposé de trier les données de simulation afin de défragmenter la mémoire du système mais en utilisant une méthode « brute », coûteuse et ne tirant pas avantage du parallélisme offert par OpenCL. Nous proposons donc, pour défragmenter la mémoire de notre système, d'utiliser un algorithme de tri parallèle. L'algorithme qui a retenu notre attention est un tri de type « merge-exchange », initialement proposé par Ken Batcher et formalisé par Knuth (1973). C'est un algorithme qui est parallèle par nature et dont la complexité est en $\mathcal{O}(n \log n)$. Présenté dans l'algorithme 5.1, cet algorithme n'est pas trivial, mais son implémentation avec OpenCL est immédiate : l'opération que l'on peut réaliser en parallèle est l'opération « comparer/échanger » (ligne 13). Pour un tableau de taille N, N work-items sont alloués pour réaliser l'opération. Le tri est réalisé sur la base de l'état des entités : l'état « vivant » vaut 0 et l'état « mort » vaut 1. De cette manière, les tableaux de données peuvent être ordonnés de telle sorte que l'on retrouve la configuration illustrée sur la figure 5.13a.

La figure 5.14 montre un exemple d'exécution de l'algorithme *Batcher's merge-exchange* (Knuth, 1973) pour un tableau de taille 16. Tout au long du déroulement de l'algorithme, on observe que les opérations « comparer/échanger » se font toutes de manières indépendantes les unes des autres.

Une fois que les données sont triées, nous plaçons un pointeur sur la prochaine case disponible dans le tableau. De cette façon, une entité voulant se répliquer peut accéder directement à cette ressource puis faire passer le pointeur sur la case suivante. Cependant, il faut protéger le pointeur, pour garantir que plusieurs entités ne le modifient pas en même temps. Pour cela, l'accès au pointeur est réalisé avec une opération atomique. Cette première méthode garantit qu'une entité voulant se répliquer pourra vraisemblablement le faire si des ressources sont disponibles : à chaque pas de simulation, les entités souhaitant se répliquer peuvent être placées en attente mais il est à noter que le cycle cellulaire tel que nous l'avons modélisé (voir chapitre 3, section 3.1) permet de désynchroniser les entités et donc de limiter le nombre d'entités voulant se répliquer durant un pas de simulation donné. Nous appelons cette première méthode de réplication la *méthode par pointeur*.

La deuxième méthode que nous utilisons pour l'allocation de nouvelles entités a été proposée par Lysenko et D'Souza (2008), dont nous avons décrit le framework pour la simulation d'agents à la section 2.2c) de ce chapitre. Trouver des ressources pour l'allocation d'une nouvelle entité avec cette méthode ne dépend pas de l'état de la mémoire, en particulier le fait qu'elle soit fragmentée n'a pas d'influence. Cette méthode ne nécessite pas non plus d'utiliser des mécanismes de synchronisation. Le but de cette méthode est de mettre en correspondance des entités voulant se répliquer avec des emplacements disponibles dans les tableaux contenant les données. Pour ce faire, une fonction invertible est utilisée : $f : A \to A$, où $A = \{a_0, a_1, \ldots, a_{N-1}\}$ est l'ensemble de toutes les entités :

$$f(a_i) = a_{i+r} \qquad f(a_i)^{-1} = a_{a-r} \tag{5.1}$$

où r est un entier aléatoire : $1 \leq r < N$. Les correspondances entre deux entités sont uniques. Le temps requis pour exécuter cette méthode est constant et ne dépend pas de la

```
/* Tri d'un tableau R de taille N (N > 2)                              */
// Étape 1:
p ← 2^(t−1)
/* où t = ⌈log2 N⌉ est le dernier entier tel que 2^t > N               */
/* les étapes 2 à 5 vont être réalisées pour p = 2^(t−1), 2^(t−2), ..., 1 */
// Étape 2:
q ← 2^(t−1)
r ← 0
d ← p
// Étape 3: (l'opérateur ∧ représente un "ou exclusif")
pour i = 0 jusqu'à N − d et i ∧ p = r faire
    /* À ce point de l'algorithme, d est un multiple impair de p et
       p est une puissance de 2, de sorte que i ∧ p ≠ (i + d) ∧ p: en
       conséquence, l'opération "comparer/échanger" ci-dessous
       peut-être réalisée en simultanée pour tous les i, dans un
       ordre indifférent                                               */
    Comparer/échanger R_(i+1) et R_(i+d+1)
    si q ≠ p alors
        d ← q − p
        q ← q/2
        r ← p
    fin
fin
p ← ⌊p/2⌋
si p > 0 alors
    Retourner à l'étape 2
fin
```

Algorithme 5.1 - Algorithme *Batcher's merge-exchange* (Knuth, 1973)

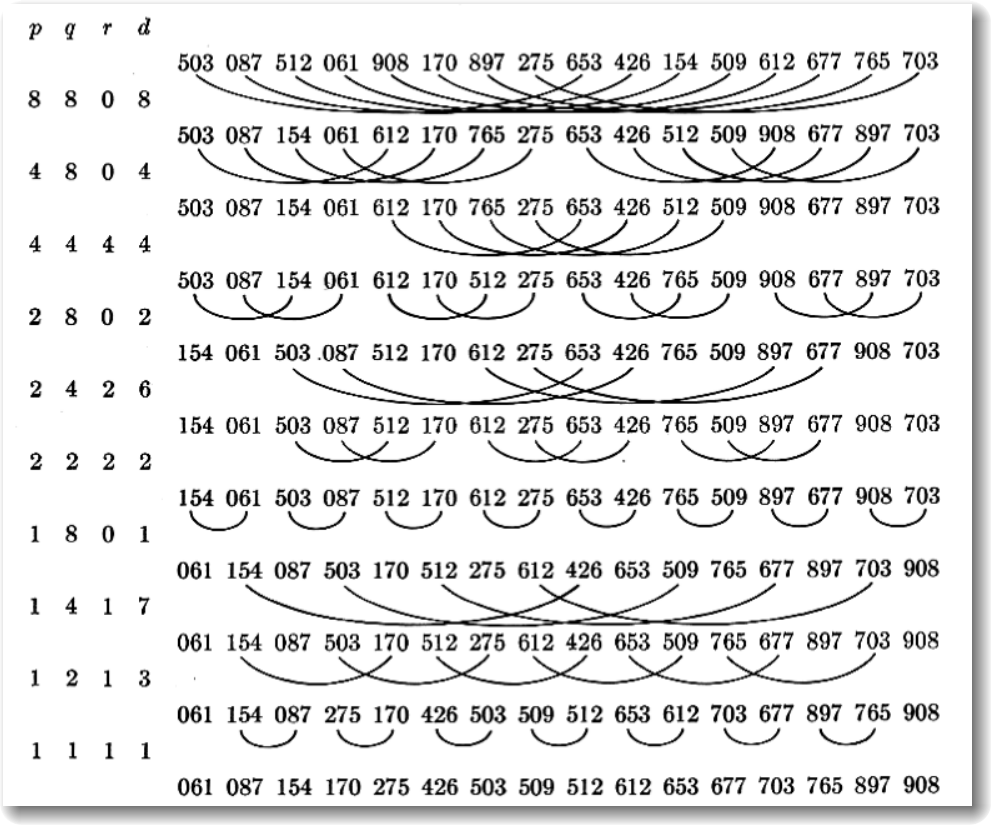


FIGURE 5.14 – Illustration de l'algorithme *Batcher's merge-exchange*. Une séquence de taille $N = 16$ est triée en comparant les nombres deux à deux de manière indépendante. Figure extraite de (Knuth, 1973).

taille de la mémoire. Cette méthode souffre cependant d'un inconvénient important : elle dépend du taux de mémoire utilisé. Plus la mémoire est remplie et moins les entités ont de chances d'être mises en correspondance avec des entités inactives. La probabilité $p(k)$ qu'une entité puisse se répliquer avec succès après k itérations de l'algorithme est :

$$p(k) = 1 - \left(\frac{l}{N}\right)^k \tag{5.2}$$

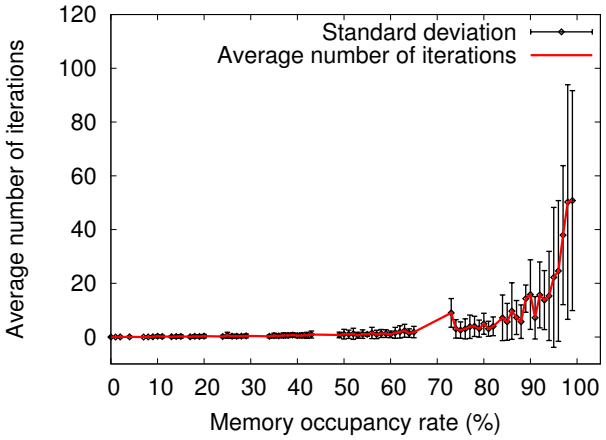
où l est le nombre de cases non vides dans la mémoire. Par exemple, cela signifie que si la moitié de la mémoire est remplie, une entité a 95% de chances de se répliquer après 5 itérations de l'algorithme. Ces chances tombent à 40% après 5 itérations si la mémoire est pleine à 90%. Nous appelons cette seconde méthode de réplication la *méthode stochastique*.

Pour information, OpenCL ne proposant pas de fonction pour générer des nombres pseudo-aléatoires (dont nous avons besoin notamment pour utiliser la méthode de Lysenko et D'Souza), nous avons implémenté un générateur efficace, KISS (*Keep It Simple Stupid*), proposé par Marsaglia (2003). Ce générateur tient sur quelques lignes, et est basé sur l'utilisation de quatres graines. Sa période est de 2^{125} (il permet de réaliser $4, 25.10^{37}$ générations de nombre avant de reboucler).

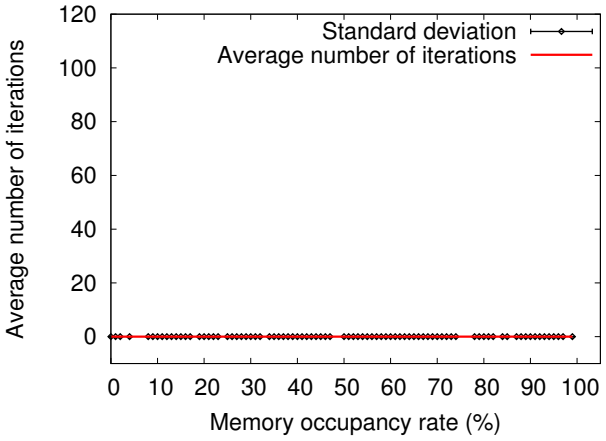
Notre proposition. Nous proposons donc d'utiliser ces deux méthodes de manière couplée, afin de combiner leurs intérêts respectifs en limitant leurs inconvénients. Nous utilisons la méthode stochastique lorsque moins de 70% de la mémoire est remplie : dans ce cas, 10 itérations de l'algorithme sont réalisées, conduisant à 97% de chances de succès lorsque au plus 30% de la mémoire est disponible. Lorsque l'occupation mémoire atteint 90%, nous défragmentons la mémoire et utilisons la méthode par pointeur. Le fait d'utiliser un intervalle pour réaliser l'échange de méthodes plutôt qu'une valeur unique permet de limiter le nombre d'échanges de méthodes réalisés. Il est à noter que ce palier a été déterminé de manière empirique et qu'une étude avancée du système de réplication devrait permettre de déterminer un palier optimal vis à vis du nombre maximal d'essais qu'une entité doit faire pour pouvoir se dupliquer.

Afin d'apporter des éléments de validation sur le bien fondé de cette méthode hybride pour permettre aux entités du système de se répliquer, nous présentons, sur la figure 5.15, des graphiques montrant le nombre d'itérations moyen requis par les entités pour se répliquer, au cours d'une simulation, en fonction de l'occupation de la mémoire (les temps d'exécution des méthodes sont proportionnels au nombre d'itérations et dépendent du matériel utilisé pour l'exécution du modèle). Chacun des trois graphiques correspond à une méthode (stochastique, par pointeur ou hybride). Les écarts types de chaque valeur moyenne sont également montrés sur les courbes. Ces moyennes ont été calculées pour une seule simulation durant laquelle au plus 4096 entités pouvaient être allouées. Comme nous pouvons le voir sur la figure 5.15a, le nombre d'itérations requis augmente fortement lorsque l'occupation mémoire atteint 90%. Les écarts type augmentent en même temps que le nombre d'itérations, à cause de la stochasticité de l'algorithme utilisé. Lorsque la méthode par pointeur est utilisée, seule une itération est nécessaire pour qu'une entité se réplique. Le graphique de la figure 5.15c montre comment on peut prévenir un nombre important d'itérations lorsque l'on change de méthode quand moins de mémoire est disponible. Le tableau 5.3 est une synthèse des mécanismes de réplication que nous proposons.

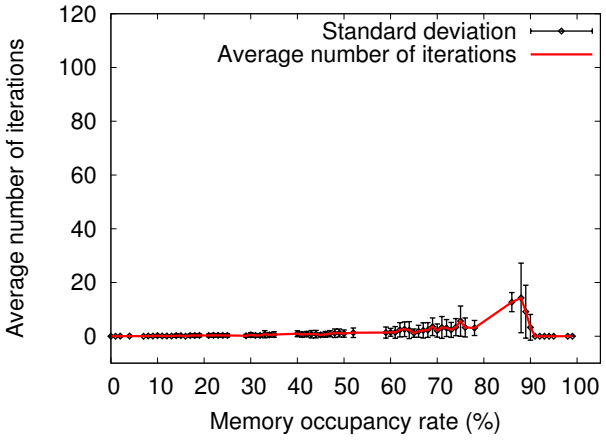
Une étude plus approfondie, avec des données extraites d'un ensemble de simulations, doit être réalisée pour aller plus loin dans la confirmation du bien fondé de ces mécanismes de réplication. Pour l'heure, nous pensons que ces données préliminaires permettent de



(a) Méthode stochastique



(b) Méthode par pointeur



(c) Méthode hybride

FIGURE 5.15 – Nombre moyen d'itérations requis par les entités pour se répliquer en fonction du taux d'occupation mémoire. (a) Le succès de la méthode stochastique repose fortement sur l'espace disponible en mémoire : plus celui-ci est réduit et plus le nombre d'itérations requis augmente pour que les entités se répliquent. (b) Lorsque la méthode par pointeur est utilisée, la réplication est instantanée. (c) La méthode hybride que nous proposons tire avantage des deux méthodes précédentes : la méthode stochastique est utilisée lorsque l'occupation mémoire est inférieure à 90%. Au delà, le nombre d'itérations nécessaires augmente fortement : les données sont défragmentées avec l'algorithme *Batcher's merge-exchange* (Knuth, 1973) et la méthode par pointeur est utilisée.

Table 5.3 – Synthèse des mécanismes de réplication mis en place dans notre simulateur. Les différentes méthodes utilisées peuvent ou non requérir de la synchronisation pour ne pas corrompre les données potentiellement accédées par plusieurs entités en même temps. Chaque méthode a également un lien avec l'occupation mémoire du système (selon que cette occupation soit importante ou non).

Méthode	Synchronisation ?	Occupation mémoire
Méthode stochastique (Lysenko et D'Souza, 2008)	non	doit être peu importante
Défragmentation mémoire (Knuth, 1973)	non	indifférent
Méthode par pointeur	oui	indifférent
Méthode hybride (Jeannin-Girardon et coll., 2013a)	**partielle**	**indifférent**

donner une tendance représentative de ce qu'il se passe dans le système lors de simulations. Nous pensons que la méthode hybride que nous proposons est un bon compromis entre la méthode stochastique et la méthode par pointeur : d'un côté, nous tirons avantage de l'efficacité computationnelle de la méthode stochastique et de l'autre, nous garantissons que les entités voulant se répliquer pourront le faire, grâce à l'échange de méthodes lorsque l'occupation mémoire devient importante.

3.4 Paramètrage du système et fichier de configuration

Afin de faciliter la manipulation du modèle et de ses paramètres, nous avons mis en place un fichier de configuration (voir schéma 5.10 page 140). Ce fichier est traité avec la bibliothèque libre `libconfig`[2]. Pour l'heure, ce module est implémenté pour le modèle de cellule virtuelle uniquement. Le fichier de configuration contient des paramètres du système. En particulier, il peut être utilisé pour définir : le nombre de nœuds sur la membrane de la cellule, la définition des types cellulaires ainsi que différentes propriétés propres à ces types (raideur des structures internes, coefficients d'adhésion, rayon cible), de même que les propriétés des agents qui sont stockées dans la structure de tableau (voir figure 5.11 page 142).

Lorsque les paramètres sont définis, le parseur que nous avons implémenté lit le fichier de configuration et réalise ensuite deux opérations : il va en premier lieu générer sous forme de code C++ le fichier de paramètres de la simulation. Puis, il va modifier le code source existant en se basant sur les nouveaux paramètres : il est en particulier nécessaire de modifier le code des opérations géométriques appliquées aux cellules selon le nombre de nœuds défini dans le fichier de configuration (division, mesure du cisaillement, calcul et application des forces sur les nœuds). Des exemples de définition de paramètres sont visibles dans la figure 5.16. La syntaxe offerte par `libconfig` est facilement accessible aux non-programmeurs et est, à notre avis, plus claire et facile à utiliser que les syntaxes des langages de descriptions utilisant des balises, comme XML. Pour l'instant, l'utilisation de `libconfig` s'arrête à la définition des paramètres du système et, suite à l'extraction de ces paramètres, à la génération de quelques portions de code source (C++ et OpenCL) dans les fichiers de simulations. L'utilisation de ce fichier de configuration pourrait être étendue, notamment par un couplage à une interface graphique qui ne rendrait plus nécessaire de manipuler le fichier lui même, même si celui-ci offre une syntaxe de haut niveau facile à manipuler.

2. `http://www.hyperrealm.com/libconfig/`

Il serait en outre intéressant de proposer à l'utilisateur de construire des simulations avec le modèle de cellule virtuelle en choisissant parmi un ensemble de comportements, implémentés dans des kernels, qu'il ordonnancerait soit via le fichier de configuration soit à travers une interface. Pour l'heure, la modification des comportements exécutés nécessite de modifier directement le code source : l'utilisation du fichier de configuration est encore à améliorer pour rendre modèle et simulateur accessibles à un public plus large.

3.5 Utilisation de notre simulateur

Notre simulateur s'utilise, jusqu'à présent, en ligne de commande. Une invite de commande permet de choisir la plateforme OpenCL ainsi que le périphérique à utiliser pour la simulation : sur l'exemple de la figure 5.17, deux plateformes OpenCL sont disponibles, contenant respectivement un et deux périphériques. L'utilisateur est invité à entrer le numéro du périphérique qu'il ou elle souhaite utiliser pour exécuter son modèle. Quelques informations concernant le périphérique choisi sont affichées, de même que le journal de compilation du programme OpenCL.

La figure 5.18 montre l'environnement de simulation affiché après avoir choisi le périphérique pour l'exécution. Les cellules virtuelles sont affichées avec OpenGL. La fenêtre affiche également quelques informations : le temps réel écoulé, le pas de simulation, le nombre d'agents actuellement présents et le temps d'exécution du dernier pas de simulation.

Nous avons utilisé notre simulateur avec succès sous système d'exploitation Linux, MS Windows et Mac OS. Nous avons également pu utiliser différents périphériques : microprocesseur Intel Core i7 860 et cartes graphiques NVidia 440 GT et GTX 690 et AMD FirePro 7800.

La section suivante a pour objet de donner quelques précisions quant à l'implémentation de notre modèle de cellule virtuelle dans le simulateur que nous proposons.

4 Implémentation de notre modèle de cellule virtuelle

Le but de cette section est de faire le lien entre le modèle de cellule virtuelle que nous proposons (présenté au chapitre 3) et le simulateur introduit dans le présent chapitre.

4.1 Implémentation des cellules virtuelles

Les agents du système, les cellules, sont mis en correspondance avec des work-items OpenCL : `une cellule = un work-item`. De manière schématique, l'exécution du système peut se résumer comme illustré sur la figure 5.19 : m kernels OpenCL constituent le comportement de chacune des cellules. Chaque work-item va exécuter les kernels successivement, sachant que deux kernels successifs sont synchronisés avec une barrière. Les autres mécanismes de synchronisation se situent au niveau de l'algorithme de réplication. Nous n'utilisons pas de mécanisme de synchronisation lors des déplacements des cellules : le moteur physique mis au point pour gérer la physique des cellules doit empêcher celles-ci de se « superposer » dans l'environnement, grâce aux forces de répulsion (voir chapitre 3 page 72) : cela fait partie des points d'optimisation sur lesquels nous avons travaillé lors de la conception de notre modèle.

L'algorithme général des cellules virtuelles est présenté sur la figure 5.20. L'algorithme de réplication donne à cet algorithme général une allure un peu particulière : si le comportement de réplication est inclus dans le modèle, il faut également inclure les mécanismes de choix

```
#####  Cell general settings
cell_general =
{
  nb_nodes = 8;
};

#####  Definition of cell types
cell_types =
(
  {
    type = "RedBC"; // Red blood cell
    radius  = 4;    // cell radius
    k_membrane = 1.; // membrane stiffness
    k_cortex   = 1.; // cortex stiffness
    k_cytosk   = 1.; // cytoskeleton stiffness
  },
  {
    type = "WhiteBC"; // White blood cell
    radius = 5;
    k_membrane = 1.;
    k_cortex   = 1.;
    k_cytosk   = 1.;
  }
);

#####  Definition of agents properties
agents_props =
(
  {type = "ushort"; name = "state";},
  {type = "ushort"; name = "type"; },
  {type = "float4"; name = "random";}
);
```

Figure 5.16 – Exemple de définition de paramètres avec le fichier de configuration. La syntaxe simple de ce fichier de configuration permet de définir les paramètres du modèle à un haut niveau.

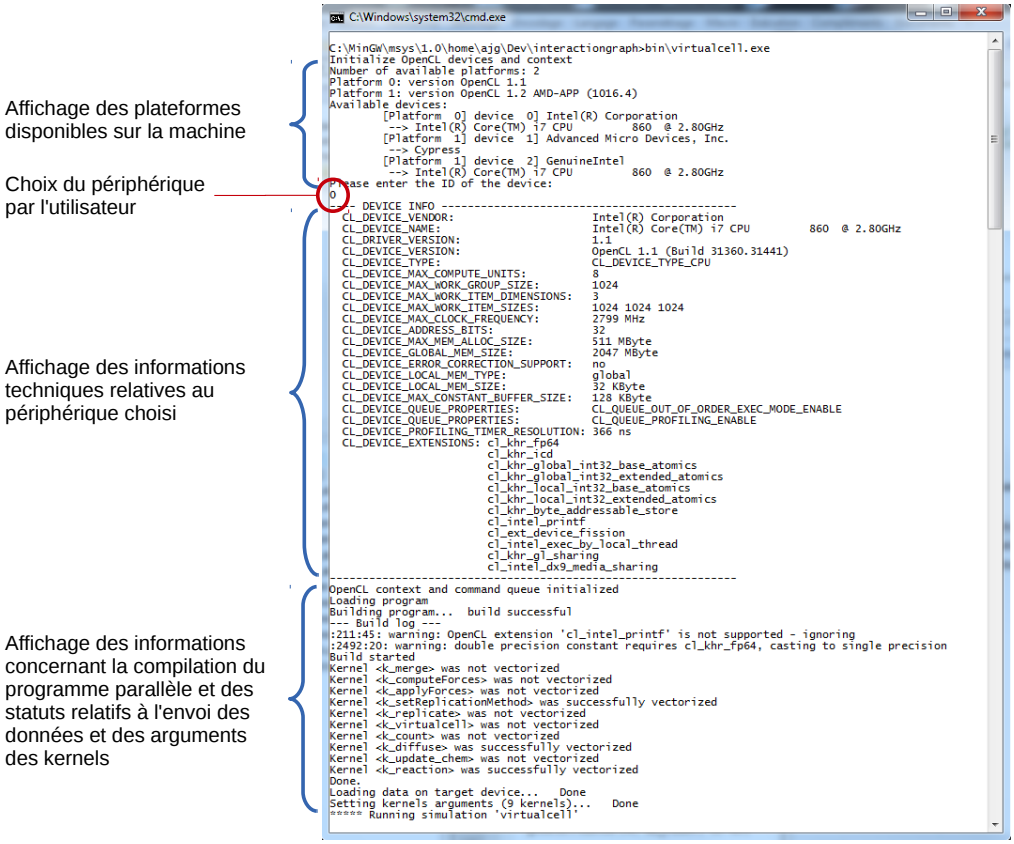


FIGURE 5.17 – Invite de commande pour choisir la plateforme OpenCL et le périphérique sur lequel le modèle va être exécuté.

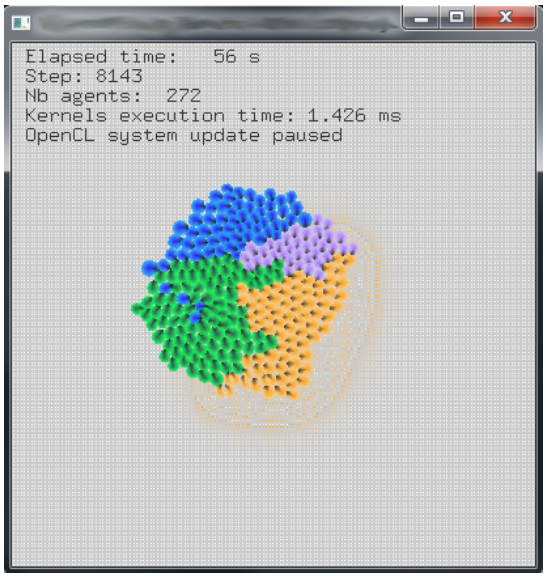


FIGURE 5.18 – Fenêtre affichant l'environnement virtuel. Quelques informations générales sont affichées : le temps réel s'écoulant, le nombre de pas de simulation exécutés, le nombre d'agents actuellement présents et le temps d'exécution du dernier pas de simulation.

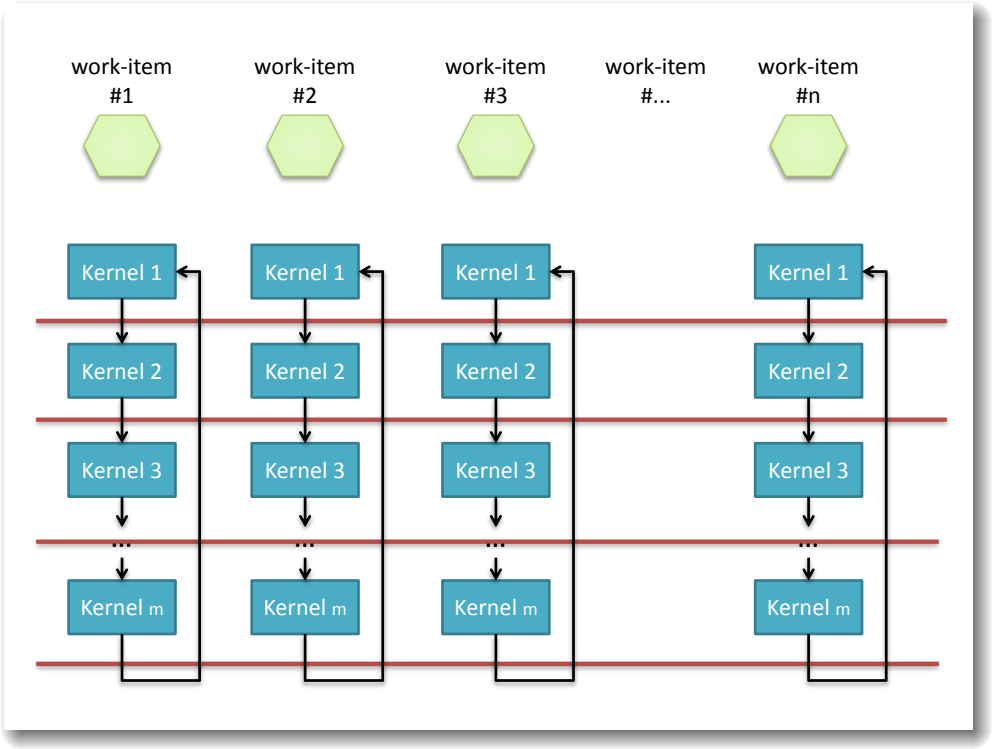


Figure 5.19 – Execution parallèle des agents cellules. Un work-item est associé à chaque cellule, les m kernels constituant le comportement des cellules sont ordonnancés et exécutés successivement (deux kernels successifs sont synchronisés avec une barrière, illustrée en rouge sur la figure). Chaque work-item exécute une instance de kernel.

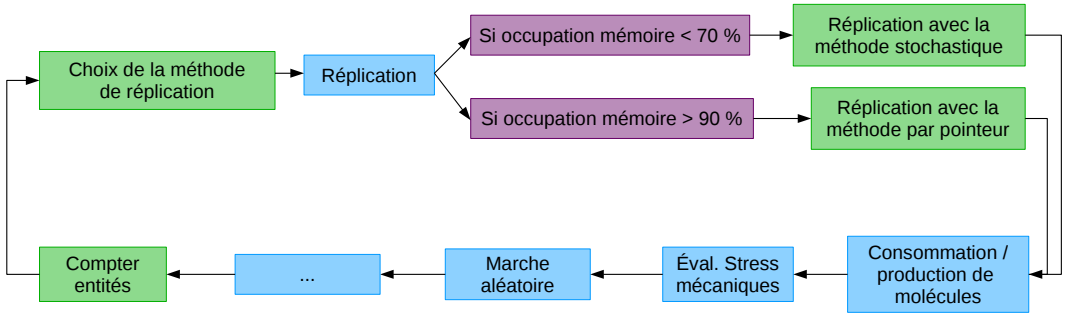


FIGURE 5.20 – Représentation d'un exemple simple d'algorithme d'entités ayant la capacité de se répliquer. Les boîtes bleues représentent des exemples de comportements des entités du système et les boîtes mauves et vertes représentent les fonctions et kernels permettant de choisir l'algorithme pour la réplication, selon l'état de la mémoire. L'espace mémoire disponible est déterminé en comptant les entités actives dans le système : les entités incrémentent elles-mêmes un compteur grâce à une opération atomique, ceci à chaque pas de simulation.

de l'algorithme selon l'occupation de la mémoire (boîtes mauves et vertes sur la figure). L'occupation mémoire est déterminée en comptant les entités actives lors de la simulation. Pour réaliser cette opération, plusieurs options sont possibles :

1. un processus supplémentaire parcours le tableau des entités et dénombre les entités actives. Cette méthode semble peu adaptée : il faut parcourir le tableau en sachant que celui-ci peut être fragmenté ;
2. les entités actives incrémentent un compteur avec une opération atomique, à chaque fin de pas de simulation. C'est la méthode que nous utilisons actuellement ;
3. les entités incrémentent ou décrémentent le compteur évoqué ci-dessus uniquement lorsque leur état change (incrémentation si l'entité devient active ; décrémentation si l'entité devient inactive). Cette méthode semble bien adaptée à notre système.

Mesurer le temps que les entités passent à se compter selon la deuxième ou la troisième méthode permettrait de déterminer quelle solution est la plus optimale.

Toujours sur la figure 5.20, les boîtes bleues représentent les comportements des cellules, qui peuvent être variés : ici nous présentons un exemple simple de cellules consommant et produisant des molécules, évaluant les stress mécaniques issus de l'environnement, réalisant une marche aléatoire et se répliquant.

Le second point traité lors de l'implémentation des cellules virtuelles concerne l'affichage des cellules dans l'environnement virtuel. Comme mentionné auparavant, nous utilisons la bibliothèque OpenGL pour cela, car cette bibliothèque peut coopérer avec OpenCL en partageant les buffers de données à travers un même contexte d'exécution. Cela présente l'avantage indéniable de ne pas avoir à transférer de données entre des calculs OpenCL et des opérations OpenGL.

Nous utilisons un tableau à deux dimensions qui permet d'indexer les positions des sommets constituant la cellule : `index[NB_AGENT][NB_INDEX]` : la première dimension du tableau est le nombre total d'agents qu'il peut y avoir dans le système. La seconde dimension correspond au nombre d'index par cellule (déterminé en fonction du nombre de nœuds sur la membrane des cellules). L'intérêt de ce tableau est de permettre d'accéder aux index par triplets, pour tracer les triangles composant la cellule. L'utilisation d'un tableau d'index permet de dessiner l'ensemble des cellules (dont le nombre total n'importe pas) à travers un unique appel à une primitive OpenGL, ceci indépendamment du nombre total d'agents.

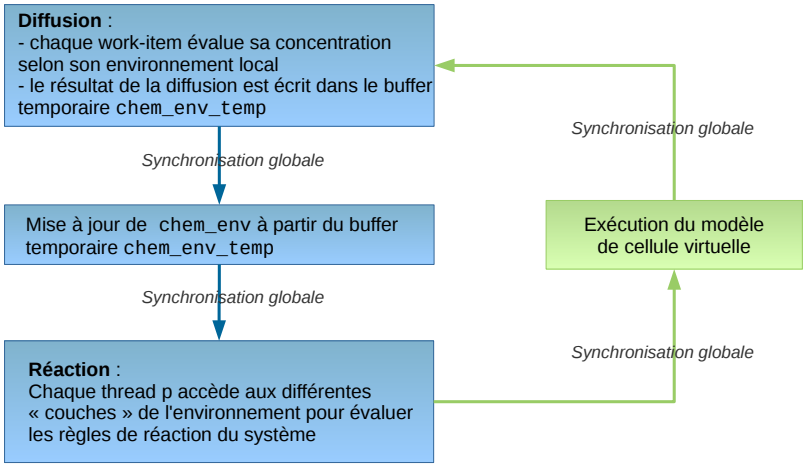


Figure 5.21 – Algorithme pour l'exécution de la chimie artificielle. Chaque work-item calcule en premier lieu sa diffusion et place le résultat dans un buffer temportaire. Celui-ci est mis à jour par un second kernel, une fois que tous les work-items ont évalué leur diffusion. Les work-items évaluent ensuite les règles de réaction du système en accédant aux différentes « couches » (les espèces chimiques) de l'environnement.

4.2 Implémentation de la chimie artificielle

Comme nous l'avons déjà mentionné au chapitre 3, l'environnement de la chimie artificielle est une grille discrète en deux dimensions. Nous utilisons, pour stocker les concentrations des espèces chimiques, le type vectoriel `floatn` fourni par OpenCL. Pour un ensemble de quatre molécules, nous utilisons donc le type `float4`. En pratique, cela signifie que lorsque l'on souhaite accéder à la concentration d'une molécule à la position p de la grille, nous utilisons le code suivant : `chem_env[p].s0` (molécule A), `chem_env[p].s1` (molécule B), `chem_env[p].s2` (molécule C) et `chem_env[p].s3` (molécule D). Pour visualiser cette notation dans l'environnement, rappelons que le type vectoriel permet de stocker les données en « couche » dans le tableau, il s'agit donc d'accéder à une case de la grille dans une couche particulière.

Dans le cas de la chimie artificielle, nous allouons autant de work-items qu'il y a de cases dans la grille contenant les concentrations des molécules. Chaque work-item p calcule la diffusion et la réaction dans l'environnement. Nous utilisons un buffer temporaire afin que tous les work-items utilisent des données similaires à un instant donné, puis ce buffer est recopié dans la grille principale pour mettre à jour l'environnement. L'algorithme global de l'exécution de la chimie est illustré sur la figure 5.21.

Il est à noter que, dans le cas de l'implémentation des agents cellules aussi bien que dans le cas de l'implémentation de la chimie artificielle, nous ne proposons pas de découpage des données pour optimiser les accès mémoire. Ce type de partitionnement n'est en général pas trivial, et nous avons choisi d'optimiser les accès à la mémoire globale, comme nous l'avons montré dans la section 3.2b) de ce chapitre. Il n'en reste pas moins que cela représente une perspective intéressante pour améliorer le simulateur.

Un second point qu'il est important de noter concerne la granularité de la grille utilisée pour la chimie. Pour des raisons pratiques, pour le moment cette grille a la même granularité que la grille discrète utilisée dans l'environnement des agents. La granularité de la grille, en pratique, doit dépendre de la vitesse de diffusion des molécules : plus celles-ci diffusent rapidement et plus la granularité doit être importante, mais cela peut ensuite poser des

problèmes concernant la migration des cellules : une adaptation de la chimie artificielle et de son environnement pour la rendre plus pertinente et réaliste passera alors par une adaptation de l'implémentation des agents du système.

À ce stade, nous avons présenté à la fois notre modèle (voir chapitre 3), notre simulateur et l'intégration du modèle dans celui-ci. Dans la prochaine section, nous proposons quelques exemples d'application permettant d'apporter des éléments de validation des outils que nous avons développé.

5 Exemples d'application

Nous avons, au cours de ce mémoire, présenté les différents éléments des travaux effectués. En particulier, le chapitre 3 était consacré à la présentation de notre modèle de cellule virtuelle tandis que dans le présent chapitre, nous avons présenté le simulateur que nous avons conçu. La section précédente, traitant de l'implémentation de notre modèle, nous permet de faire le pont entre ces deux chapitres.

Dans cette section, nous présentons des simulations réalisées avec nos outils. Ces simulations nous permettent d'apporter des éléments de validation supplémentaires à ceux déjà introduits dans les précédents chapitres, tout en faisant une synthèse cohérente et fonctionnelle.

Les différents cas d'études présentés mettent en application les différents mécanismes intervenant dans la morphogenèse de tissus, sains ou pathologiques : adhésion cellulaire, migration sur un gradient de molécules, prolifération induite par la présence de facteurs de croissance, dégradation de la matrice extra-cellulaire et influence des contraintes physiques issues de l'environnement. Afin d'illustrer ces mécanismes, nous proposons des expérimentations virtuelles variées : tri cellulaire, migration sur un gradient de molécules, formation d'agrégats cellulaires, prolifération et homéostasie dans un tissu et prolifération cellulaire induite par la présence de facteurs de croissance. Une dernière expérimentation traite du bourgeonnement d'un vaisseau sanguin lors du déclenchement du processus d'angiogenèse par une tumeur. De plus, nous présentons également dans cette section une étude de performances de notre système sur différents matériels (micro-processeurs et cartes graphiques). Notons que nous avons pu simuler jusqu'à un million de nos cellules déformables sur carte graphique. Concernant les paramètres, nous avons systématiquement utilisé les ordres de grandeurs présentés dans le tableau 3.3 contenant les paramètres de notre modèle (chapitre 3, page 92).

Il est important de garder à l'esprit que la pertinence biologique des cas d'étude présentés dans cette section est tout à fait discutable ; ces cas d'étude ayant pour première utilité le test des mécanismes modélisés ainsi que leurs implémentations dans le simulateur.

5.1 Tri cellulaire dans un tissu virtuel

Le tri cellulaire est un phénomène biologique que nous avons présenté dans le chapitre 2, page 37. Pour rappel, ce phénomène peut se produire lorsque les tissus se forment. La théorie de l'adhésion différentielle proposée par Steinberg (1963) stipule que les cellules se réarrangent entres elles selon les différentes forces d'adhésions entre les différents types cellulaires en présence. Au long du chapitre 2, nous avons présentés différents modèles proposant des simulations de tri cellulaire : Graner et Glazier (1992) utilisent uniquement le principe de l'adhésion différentielle pour simuler un tri cellulaire avec leur modèle de Potts cellulaire ; Belmonte et coll. (2008) simulent un tri cellulaire à l'aide de particules. Ils ont

ajouté, en plus de l'adhésion différentielle, un principe d'alignement des particules cohérent avec les observations *in vitro* réalisées par Rieu et coll. (1998) ; enfin, Eyiyurekli et coll. (2007) proposent quant à eux une simulation de tri cellulaire basé uniquement sur la chimiotaxie avec des cellules représentées par des disques.

Nous proposons également de simuler le phénomène de tri cellulaire avec notre modèle de cellule virtuelle et le principe d'adhésion différentielle. Pour rappel, nous modélisons l'adhésion de la manière suivante : la structure physique de notre cellule virtuelle est constituée de nœuds reliés par des liens élastiques. Nous avons modélisé les interactions cellulaires avec des forces appliquées sur les nœuds de la cellule. En particulier, en ce qui concerne l'adhésion entre cellules, les nœuds membranaires les plus proches des cellules respectives sont sélectionnés et une force peut ensuite être appliquée. Ces éléments sont détaillés dans le chapitre 3, page 72.

Mise en place de la simulation et résultats. Deux types cellulaires sont en présence. Nous utilisons uniquement le principe d'adhésion différentielle : soit $ad_{type1,type2}$ la valeur de l'adhésion entre les types $type1$ et $type2$. Les types cellulaires correspondent aux couleurs des cellules $bleu$ et $vert$. Les valeurs d'adhésion de ces types cellulaires sont paramétrées de façon à respecter la règle suivante : $ad_{bleu,bleu} < ad_{bleu,vert} = ad_{vert,bleu} < ad_{vert,vert}$. La motilité des cellules est modélisée avec une force de Langevin (présentée formellement au chapitre 3 dans la section 3.2c)). L'intensité de cette force varie de 1 à 10 μN. Initialement, les cellules forment un groupe dans l'environnement et leurs types leur sont attribués de manière aléatoire et de façon à ce que, une fois toutes les cellules initialisées, le tissu soit composé d'environ 60% de cellules bleues et 40% de cellules vertes. Nous attendons donc que lors de la simulation, les cellules vertes se regroupent et forment des agrégats entourés par les cellules bleues. La disposition initiale des cellules montre des similarité avec la disposition initiale des cellules dans le tri cellulaire proposé par Graner et Glazier (1992), dans laquelle les cellules forment initialement un agrégat et dont les types sont distribués aléatoirement.

La figure 5.22 montre un exemple de simulation de tri cellulaire tel qu'obtenu dans notre environnement virtuel. Le nombre de cellules en interaction lors de cette simulation était de 384. La motilité des cellules ainsi que les adhésions différentielles entre les différents types en présence leur permettent de se réarranger de façon à ce que les cellules vertes forment des agrégats entourés de cellules bleues. Cependant, on observe également que les cellules ne forment pas un unique agrégat comme on peut l'observer expérimentalement ou dans les simulations présentées lors du chapitre 2 consacré à un état de l'art de la modélisation de systèmes biologiques. Selon nous, cela provient de la difficulté à équilibrer la motilité des cellules et leurs adhésions (nous avons évoqué la difficulté de paramétrer un système multi-agents dans le chapitre 2 page 55 : « Contrôlabilité des modèles »). Dans le cas présent, augmenter l'adhésion entre les cellules bleues rendra l'agrégat difficile à « casser » afin de réarranger le groupe. De plus, bien qu'étant flexibles, nos cellules sont plus rigides et représentent des types cellulaires moins mélangeables que les cellules du modèle de Potts cellulaire proposé par Graner et Glazier (1992).

La prochaine simulation présentée est une migration cellulaire selon un gradient de molécules.

5.2 Migration cellulaire selon un gradient de molécules

Réaliser une simulation de cellules migrant sur un gradient de molécules permet d'apporter des éléments de validation pour les comportements suivants :

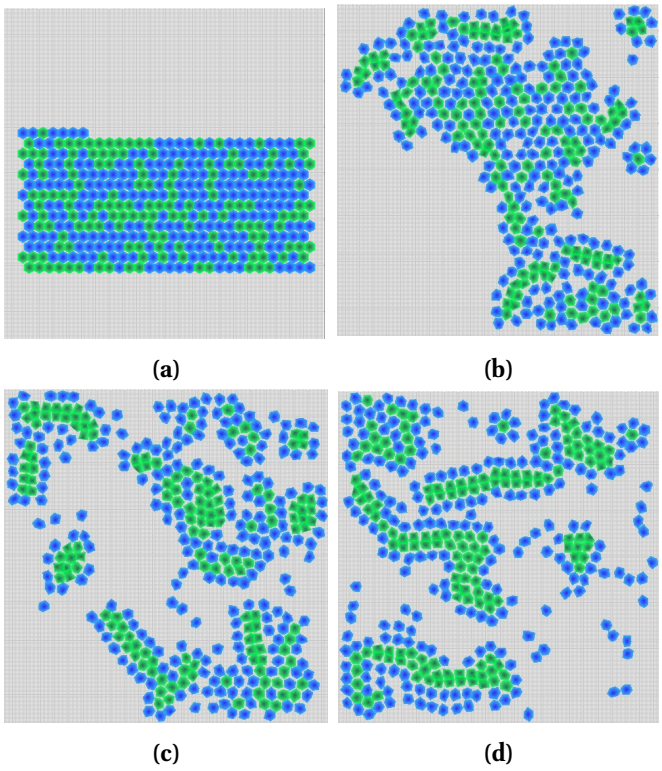


FIGURE 5.22 – Simulation de tri cellulaire avec notre modèle de cellule virtuelle. Ces images sont extraites d'une simulation contenant 384 cellules. De (a) à (d) : des agrégats de cellules vertes se forment et sont entourées de cellules bleues, grâce au principe de l'adhésion différentielle.

- capacité des cellules à se déplacer selon un gradient de molécules (chimiotaxie) ;
- consommation et production de molécules par les cellules ;
- diffusion de molécules dans l'environnement ;
- réaction des molécules selon des règles de réactions définies dans le système.

La figure 5.23 illustre cette simulation. Une source dans l'environnement produit en continu des molécules de type A. Ces molécules diffusent dans l'environnement selon les éléments que nous avons présentés dans le chapitre 3, section 4.3 page 83. Dans le cas présent, une seule cellule est présente dans l'environnement. Cette cellule perçoit et consomme des molécules de type A. Si elle perçoit ces molécules au delà d'un certain seuil, alors elle se déplace dans la direction du gradient le plus important, comme illustré sur les figures 5.23 a à d, sur la ligne du haut). Dans le cas présent, la cellule perçoit les molécules au niveau du nœud 0 (qui est plus foncé que les autres dans le rendu). La consommation suffisante de molécules de type A permet ensuite à la cellule de commencer à produire des molécules de type B (figures 5.23 a à d, ligne du milieu). La présence de molécules de type B dans l'environnement rend possible la réaction $2A + B \rightarrow C$ (figure 5.23 a à d, ligne du bas). Il est à noter que la diffusion des espèces chimique s'arrête au bord de l'environnement (celui-ci n'est pas torique).

Il faut noter que le rendu graphique pour la chimie artificielle peut être trompeur : bien que non visibles sur l'images, les molécules diffusant ont déjà atteint la cellule au niveau de la colonne (a) de la figure 5.23. Les valeurs des concentrations sont simplement très faibles et il conviendrait d'adapter le rendu graphique réalisé avec OpenGL.

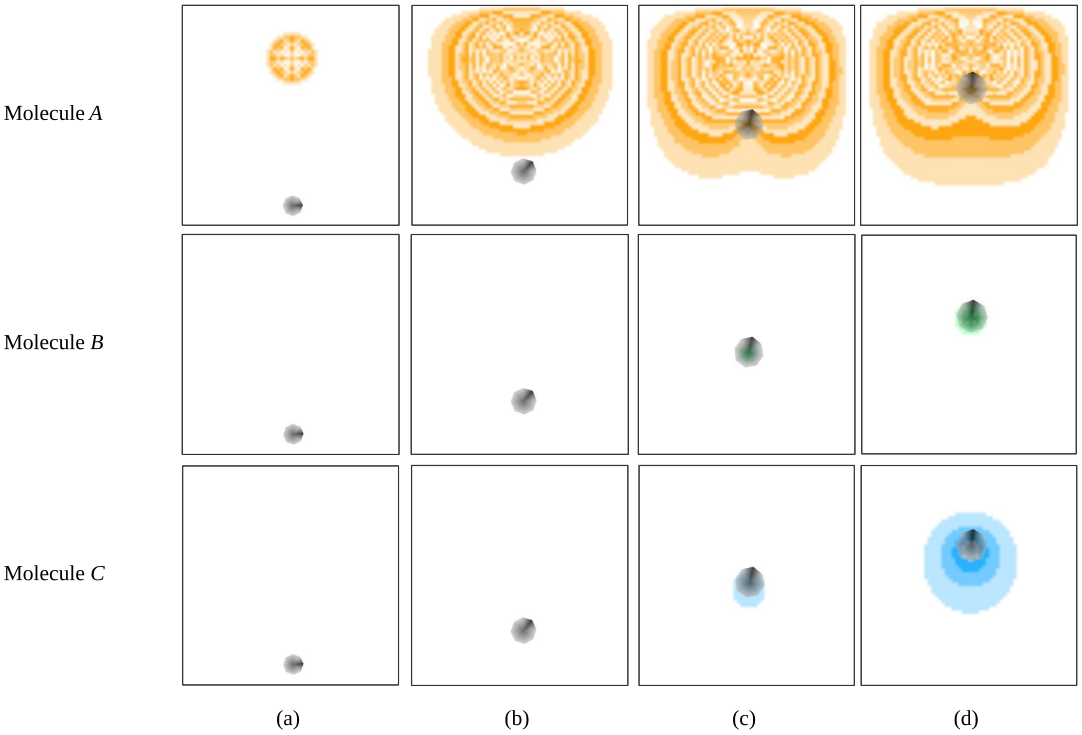


Figure 5.23 – Migration cellulaire selon un gradient de molécules. Dans chaque colonne, la cellule est représentée à la même position mais l'espèce chimique présentée est différente (représentation en « couches » de l'environnement chimique, de haut en bas). Des colonnes a à d : la cellule migre par chimiotaxie sur le gradient de molécules A. Elle consomme également cette molécule, ce qui lui permet de produire des molécules de type B. La réaction $2A + B \rightarrow C$ se produit alors.

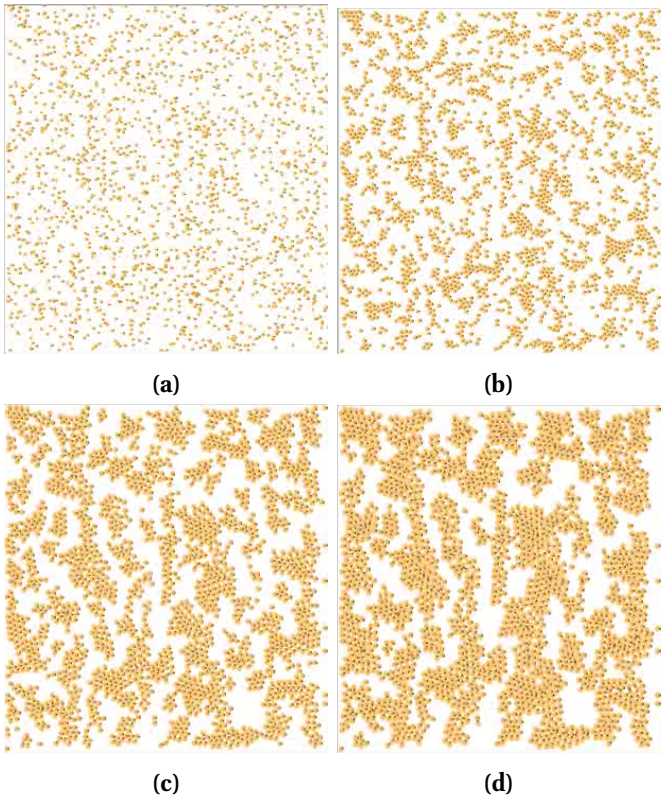


FIGURE 5.24 – Simulation d'agrégation de cellules par chimiotaxie. Initialement, 2048 cellules sont disposées aléatoirement dans l'environnement. Les cellules produisent et consomment une molécule de type A et elles migrent également en remontant ce gradient, formant des agrégats cellulaires à mesure que la simulation se déroule. Une vidéo de cette simulation peut être visualisée à l'adresse suivante : `https://vimeo.com/90620406`.

Cette simulation simple nous a donc permis de tester la chimie artificielle de notre modèle. La simulation suivante exploite ces aspects à plus large échelle pour reproduire une agrégation de cellules grâce à la présence de molécules dans l'environnement.

5.3 Agrégation cellulaire par chimiotaxie

Comme nous l'avons vu au chapitre 2 avec l'exemple de l'organisme *Dictyostelium Discoideum* (page 38), la chimiotaxie est un mécanisme entrant en jeu lors de la morphogenèse de tissus. Pour rappel, dans cet organisme les cellules migrent par chimiotaxie en premier lieu pour former des agrégats jusqu'à l'étape de culmination, suite à laquelle un corps fructifère est formé.

Nous proposons une simulation d'agrégation cellulaire, pendant laquelle des cellules initialement disposées de manière aléatoire dans l'environnement vont migrer en suivant des gradients de molécules. Un seul type cellulaire est en présence. Le nombre de cellules présentes est de 2048 et ces cellules émettent des molécules de type A et migrent par chimiotaxie en direction du gradient le plus important détecté par les cellules. Les cellules finissent donc par former des agrégats compacts. Dans cette simulation, les seuls comportements des cellules sont de migrer et de produire / consommer des molécules : il n'y a pas de cycle cellulaire (donc pas de mort cellulaire ni de prolifération). La figure 5.24 illustre le déroulement de cette simulation.

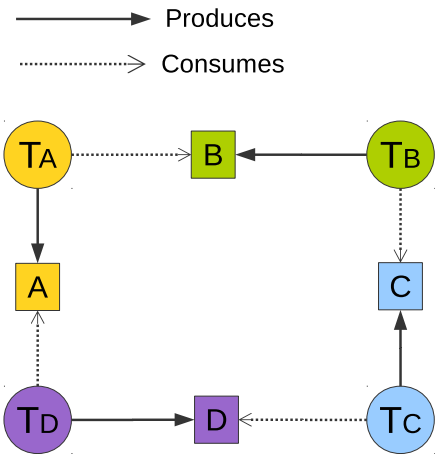


Figure 5.25 – Exemple de graphe d'interaction : les quatre types cellulaire T_A, T_B, T_C et T_D produisent respectivement des molécules de type A, B, C et D et consomment les molécules de type B, C, D et A.

Dans la prochaine simulation, la notion de cycle cellulaire est prise en compte. Le but est de proposer une simulation de la croissance d'un tissu virtuel en utilisant des dépendances entre les types cellulaires.

5.4 Croissance et homéostasie dans un tissu cellulaire

Pour réaliser cette simulation, nous avons défini ce que nous avons appelé un graphe d'interactions. Un tel graphe présente des dépendances entres cellules selon les espèces chimiques qu'elles consomment et produisent. Dans le cas de la simulation que nous présentons, nous avons défini quatre types cellulaires : T_A (en jaune), T_B (en vert), T_C (en bleu) et T_D (en mauve). Ces quatre types cellulaires produisent respectivement les molécules de type A, B, C et D.

Le graphe d'interaction permet de définir quel type cellulaire consomme telle espèce chimique afin de pouvoir se maintenir dans le système : nous interprétons la consommation de molécules comme un signal dont la cellule a besoin afin de survivre et proliférer. Les cellules ne percevant pas suffisamment de signaux chimiques entrent en apoptose. Les divisions cellulaires peuvent avoir lieu sous réserve que les cellules ne soient pas soumises à de trop fortes contraintes de compression — pour rappel, notre modèle inclut des mécanismes pour mesurer les contraintes physiques s'appliquant aux cellules (chapitre 3, page 76).

Pour cette simulation, nous avons défini le graphe d'interaction visible sur la figure 5.25. Dans ce cas :

- Les cellules de type T_A produisent des molécules de type A et consomment des molécules de type B ;
- Les cellules de type T_B produisent des molécules de type B et consomment des molécules de type C ;
- Les cellules de type T_C produisent des molécules de type C et consomment des molécules de type D ;
- Les cellules de type T_D produisent des molécules de type D et consomment des molécules de type A.

Initialement, quatre cellules sont disposées dans l'environnement (une pour chaque type) de manière juxtaposée. La figure 5.26 illustre le déroulement de cette simulation. La première

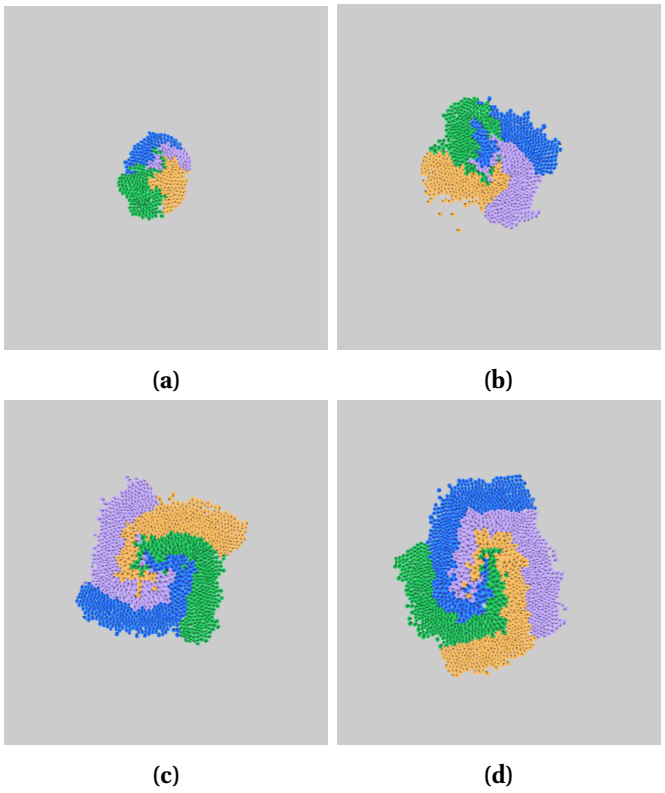


FIGURE 5.26 – Simulation de prolifération cellulaire basée sur un graphe d'interactions. Une vidéo de cette simulation peut être visualisée à l'adresse suivante : `https://vimeo.com/87567320`.

observation que nous pouvons faire est que le tissu prolifère en spirale. Compte tenu du graphe d'interaction, c'est un motif qui peut être attendu lors de la simulation mais le point intéressant est que ce motif n'a pas été défini à priori, il s'agit d'un motif émergeant lors de la croissance du tissu virtuel.

Nous avons, lors de la simulation, mesuré l'évolution de la population de cellule. Cette évolution est visible sur la figure 5.27. Nous observons que la population se stabilise autour de 1800-2000 cellules, grâce à l'équilibre entre les divisions et les morts cellulaires : il s'agit d'homéostasie cellulaire.

Dans la section suivante, nous présentons une simulation de croissance de membres dans un organisme virtuel. Cette simulation prend aussi en compte la notion d'adhésion des cellules à la matrice extra-cellulaire.

5.5 Croissance de membres dans un organisme virtuel

Le développement des membres dans un embryon est un processus complexe faisant intervenir différents mécanismes. Ces différentiations cellulaires permettent de former différents tissus tels que les os, les cartilages ou les muscles mais de surcroît, ces tissus sont formés selon les axes caractérisant la topologie de l'organisme : axes dorso-ventral, antéro-postérieur et droite-gauche. Des protéines différentes entrent en jeu pour le développement de membres selon l'axe concerné (Gilbert, 2000). Les membres se forment à des endroits spécifiques de l'axe ; ces informations sont encodées dans le génome. La formation des membres pendant le développement embryonnaire est le sujet de nombreuses études compte tenu de la complexité du phénomène (voir par exemple des travaux sur l'expression des gènes lors du

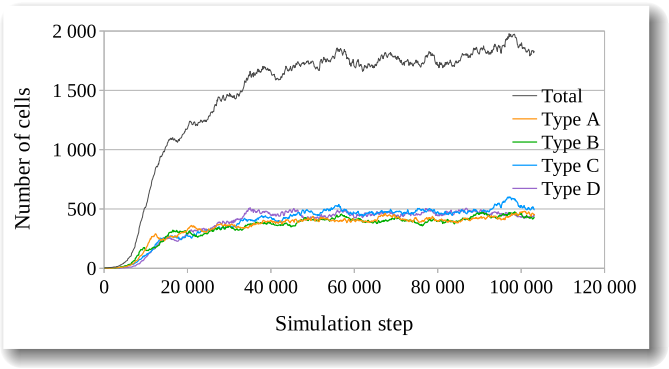


Figure 5.27 – Évolution des populations de cellules lors de la simulation de croissance d'un tissu virtuel. Après une croissance continue jusqu'à environ 30 000 pas de simulation, le tissu se stabilise et la population totale est d'environ 1800-2000 cellules : le système exhibe des taux de morts et de naissances de cellules constants et équilibrés, on peut alors le qualifier de système auto-régulé.

développement de membres (Sheth et coll., 2013), la diffusion des gradients de morphogènes intervenant lors du développement de membres (Sheeba et coll., 2014) ou encore l'étude des lignées de cellules (Altabef et coll., 1997)).

La description du développement de membres proposée par Gilbert dans son ouvrage permet de proposer une simplification du processus de développement afin d'en réaliser une simulation. En premier lieu, la croissance de membres naît des interactions entre deux tissus distincts : l'épithélium qui est un tissu de revêtement et le mésenchyme qui est lui un tissu de soutien. Dans les zones où les membres vont se former, le mésenchyme va secréter des molécules qui vont permettre l'apparition d'une zone appelée *Apical Ectodermal Ridge* (AER) sur l'épithélium. L'AER secrète un facteur de croissance, le Fgf8 (*Fibroblast growth factor*) qui va induire la prolifération des cellules du mésenchyme. En réponse, celles-ci diffusent du Fgf10 qui favorise l'expression de Fgf8 par l'AER : il s'agit donc d'une boucle de rétroaction positive. Ce processus est illustré sur la figure 5.28.

Nous avons mis en place une simulation simplifiée de ce processus afin d'observer la prolifération d'un tissu induite par la présence de facteurs de croissance. Initialement, un tissu est présent dans l'environnement. Ce tissu est composé de 256 cellules. Deux types de cellules sont en présence : des cellules du mésenchyme (qui vont donc proliférer) et des cellules de l'AER : il s'agit de cinq cellules disposées aux extrémités du tissu. Étant donné que l'objectif de la simulation est d'induire la prolifération de cellules grâce à un facteur de croissance, nous avons simplifié la simulation en ne considérant pas les cellules épithéliales normalement présentes lors de la croissance de membres, à l'exception des cellules de l'AER. Le tissu initial est visible sur la figure 5.29. Nous n'avons pas mis en place de boucle de rétroaction positive : le facteur de croissance émis par les cellules de l'AER (le Ffg8) est produit de manière continue par ces cellules. Les cellules du mésenchyme sont fortement adhésives à la matrice extra-cellulaire : nous avions vu, dans le chapitre 2 au paragraphe « Modélisation de la croissance de membres chez les vertébrés » page 37, que la réponse des cellules du mésenchyme est non seulement de proliférer mais aussi de produire de la fibronectine, une protéine favorisant l'adhésion des cellules au substrat dans lequel elles évoluent. Comme nous l'avons présenté au chapitre 3, nous pouvons modéliser l'adhésion à la matrice extra-cellulaire en utilisant le coefficient de fixation ϕ que nous avons défini page 74. Ce coefficient est fixé à 0.96 pour les cellules du mésenchyme et à 0.5 pour les cellules de l'AER.

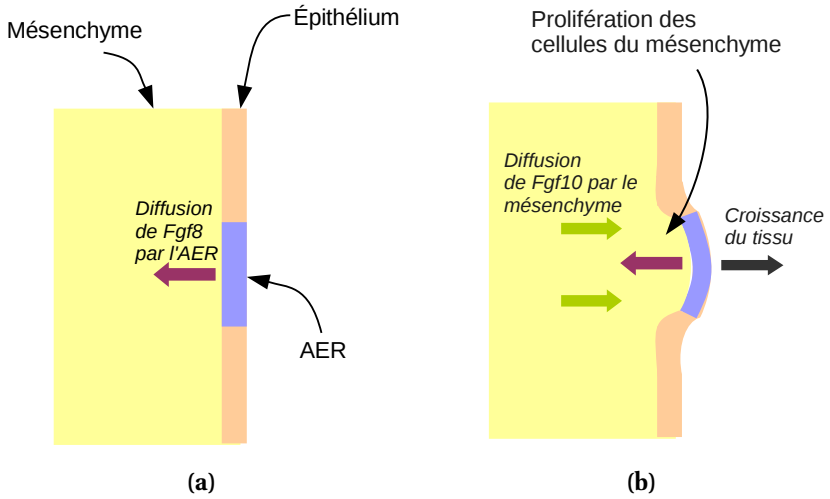


FIGURE 5.28 – Processus simplifié de la croissance de membres. Le Fgf8 diffusé par l'AER a deux conséquences : il induit la prolifération des cellules du mésenchyme permettant de faire croître le membre ; les cellules du mésenchyme répondent également à ce gradient en produisant un autre facteur de croissance, le Fgf10, qui favorise la production de Fgf8 par l'AER.

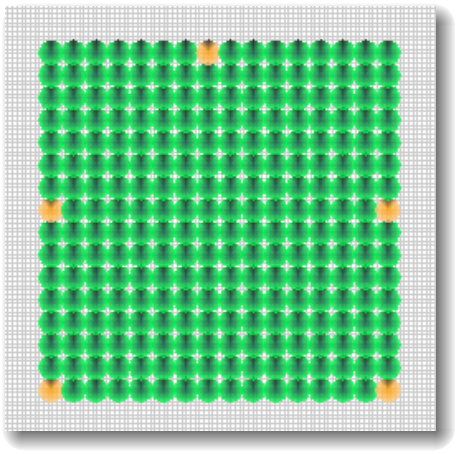


FIGURE 5.29 – Tissu initial pour la simulation de croissane de membre. Les cellules vertes sont les cellules du mésenchyme, répondant au facteur de croissance émis par les cellules de l'AER (en jaune) en proliférant.

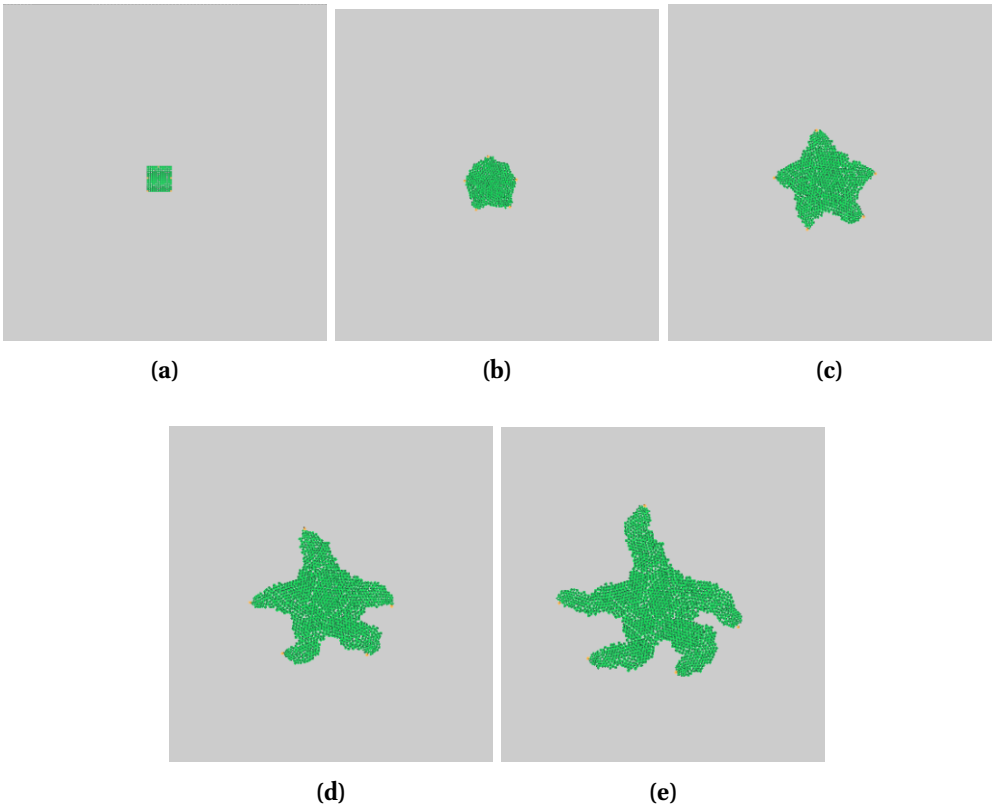


Figure 5.30 – Simulation de croissance de membres. Les cellules de l'AER, placée aux extrémités du tissu initial, produisent un facteur de croissance, le Fgf8, qui induit la prolifération des cellules du mésenchyme le percevant. De (a) à (e), différents stades de la croissance de l'organisme. Une vidéo de cette simulation peut être visualisée à l'adresse suivante : `https://vimeo.com/90620405`.

Les cellules du mésenchyme perçoivent et « consomment » les molécules Fgf8. Ces cellules accumulent le Fgf8 perçu : il s'agit, de manière simplifiée, de la fixation de ces molécules à des récepteurs présents sur la membrane des cellules. Si suffisamment de molécules sont perçues alors la cellule peut se diviser. À chaque pas de simulation, une partie des molécules présentes sur les récepteurs des cellules se dégrade.

La simulation de croissance de membre est illustré sur la figure 5.30. Le placement initial des cellules de l'AER détermine la forme de l'organisme virtuel qui se développe : dans le cas présent, il s'agit d'une forme d'étoile de mer à cinq bras.

Ces études de cas ont permis d'apporter des éléments de validation des comportements cellulaires mis en place dans notre modèle et dans l'implémentation réalisée dans notre simulateur. Ces comportements cellulaires sont ceux que l'on retrouve lors de la morphogenèse impliquant des cellules. Afin de proposer une synthèse des expérimentations virtuelles présentées dans ce chapitre, nous proposons dans la section suivante une simulation simplifiée de bourgeonnement de vaisseau sanguin lors du phénomène d'angiogenèse se produisant lors du développement de cancer.

5.6 Bourgeonnement d'un vaisseau sanguin

La formation de nouveaux vaisseaux sanguin est un processus ayant lieu tant dans des situations normales, lors de l'embryogenèse (dans ce cas on parle de vasculogenèse, processus permettant de donner lieu à de nouveaux vaisseaux sanguins) ou encore de la réparation d'une blessure (dans ce cas on parle d'angiogenèse, processus permettant la création de

nouveaux vaisseaux sanguins, par bourgeonnement, à partir d'un réseau vasculaire existant), que dans des situations anormales, en particulier lors de la morphogenèse de tumeurs. En effet, lorsque la tumeur a atteint une certaine taille, elle doit trouver des ressources supplémentaires afin de poursuivre sa croissance. La mise en place du processus d'angiogenèse permet à la tumeur de devenir vascularisée et d'accéder à de nouvelles ressources : oxygène et nutriments. En outre, la vascularisation est aussi une voie pour permettre à la tumeur de développer des métastases.

Nous avons mis en place une simulation simple pendant laquelle une tumeur déjà formée, hypoxique, va déclencher le processus d'angiogenèse grâce à l'émission d'un facteur de croissance : le VEGF (*Vascular Endothelial Growth Factor*). Deux types de cellules sont en présence : des cellules tumorales et des cellules endothéliales. Par simplification, le vaisseau sanguin n'est constitué que des cellules endothéliales, puisque ce sont ces cellules qui prolifèrent dans le cadre du processus d'angiogenèse. Ces cellules endothéliales sont fortement adhésives et n'ont que de faibles capacités de migration. Toutefois, la présence de VEGF permet à ces cellules de dégrader la matrice extra-cellulaire environnante, permettant ainsi une migration dirigée vers les gradients de VEGF (Tracqui, 2009). Nous pouvons donc utiliser le coefficient de fixation ϕ de notre modèle pour approximer le phénomène de dégradation de la matrice extra-cellulaire permettant alors une meilleure motilité des cellules. Les cellules endothéliales accumulent les molécules de VEGF et elles peuvent entrer en mitose lorsque suffisamment de VEGF est présent sur leur récepteurs. Les seuils de VEGF peuvent vraisemblablement être obtenus auprès de biologistes.

Ce cas d'étude ne tient pas compte de tous les mécanismes biologiques de l'angiogenèse. En particulier, il est à noter que lors de l'angiogenèse, les premières cellules percevant le VEGF émettent un inhibiteur qui empêchera les autres cellules du vaisseau de percevoir le VEGF : cela permet de maintenir la structure du vaisseau sanguin originel. L'expérimentation virtuelle que nous proposons fait donc l'objet de simplifications importantes en regard du système réel. La figure 5.31 montre le déroulement de cette simulation. Il est à noter que la taille de notre tumeur est réduite : la vascularisation d'une tumeur se met en place lorsque celle-ci atteint une taille de quelques millimètres (Ameisen, 2003), hors la tumeur virtuelle n'a un diamètre que d'une centaine de micromètres, ceci afin de simplifier le protocole de simulation sans pour autant nuire aux phénomènes simulés. Le facteur de croissance produit par les cellules tumorales se diffuse dans l'environnement. Nous observons que les cellules endothéliales percevant ce facteur de croissance vont s'orienter et migrer vers ce gradient. Leur prolifération est également induite par ces molécules et on peut ainsi observer le bourgeonnement du vaisseau sanguin virtuel.

5.7 Étude des performances de notre système

a) Étude sur le tri cellulaire

La simulation de tri cellulaire a été pour nous l'occasion de réaliser une étude de performances afin de mieux appréhender les possibilités du système et en particulier de l'architecture logicielle que nous proposons. Nous avons, pour cette étude de performances, utilisé quatre matériels différents : un micro processeur Intel Core i7 860 et les cartes graphiques suivantes : une NVidia GeForce GTX 690 (ce matériel est en fait un bi-GPU, toutefois nous n'avons utilisé qu'un GPU lors de nos simulations), une NVidia GeForce GT440 et une ATI Firepro V7800. Les caractéristiques principales de ces matériels sont présentées dans le tableau 5.4.

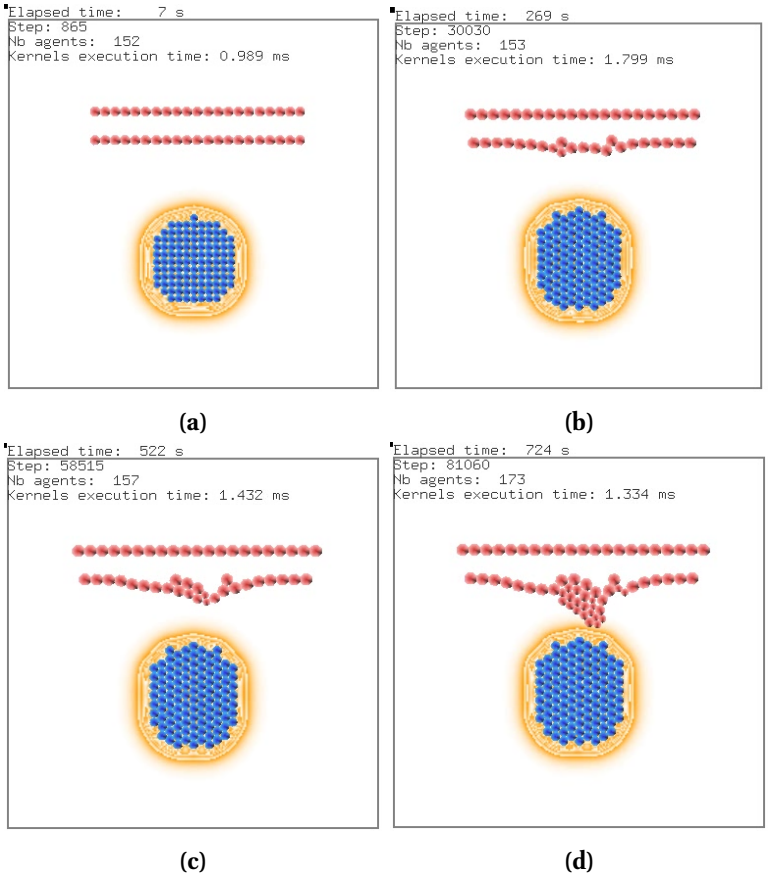


Figure 5.31 – Simulation du bourgeonnement d'un vaisseau sanguin. Les cellules endothéliales (en rouge) perçoivent le facteur de croissance VEGF produit par les cellules tumorales (en bleu). Ce facteur de croissance induit la migration des cellules endothéliales en direction de ce gradient ainsi que leur prolifération, donnant lieu à un bourgeon sur le vaisseau sanguin. Une vidéo de cette simulation peut être visualisée à l'adresse suivante : `https://vimeo.com/103408482`

Table 5.4 – Caractéristiques principales des matériels utilisés pour la simulation de tri cellulaire. Ces données sont extraites des descriptifs fournis par les constructeurs.

	Fréquence	Processeurs de flux	Mémoire partagée
Intel Core i7 860	2.80 GHz	8 (4 unités de calcul)	32 Ko
NVidia GT 400	1600 MHz	96 (2 unités de calcul)	48 Ko
NVidia GTX 690	900 - 1000 MHz	1536 par GPU (8 unités de calcul)	48 Ko
ATI Firepro V7800	700 MHz	1440 (18 unités de calcul)	32 Ko

Nous avons mesuré le temps d'exécution moyen du code parallèle sur 1000 pas de simulation. Pour ce faire, nous avons utilisés des événements OpenCL qui permettent de façon générale de monitorer les files de commandes. De façon plus spécifique, les événements peuvent servir afin d'obtenir des informations concernant les kernels. En particulier, la fonction `clGetEventProfilingInfo()` permet de calculer le temps d'exécution d'un kernel. Pour des raisons de simplification, nous avons décidé de ne calculer que les temps d'exécution des kernels, sans mesurer les temps de calcul lié à la gestion du programme par le code hôte. Il est à noter qu'aucun transfert mémoire n'a lieu après le lancement d'une simulation dans le cas de cette étude.

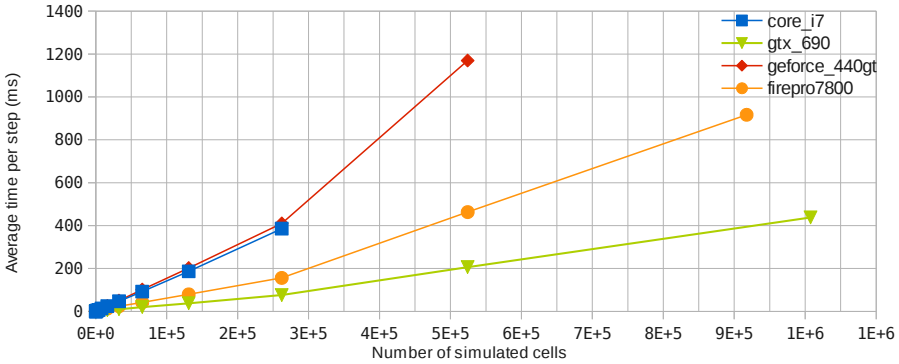
Nous avons lancé différentes simulations en variant le nombre de cellules simulées. Les temps d'exécution du code parallèle sont visibles sur la figure 5.32.

La figure 5.32a montre les performances globales de notre système sur les quatre matériels utilisés. La GTX 690 montre les meilleures performances parmi ces matériels. En terme d'accélération, la simulation de 5.10^5 cellules est 5.66 fois plus rapide avec la GTX 690 que la GT 440 et 2.24 fois plus rapide qu'avec la Firepro. Il est à noter que malgré le fait que la Firepro 7800 ait des spécificités intéressantes, c'est un matériel qui devient vieux, car sorti en 2009. Avec la GTX 690, nous avons pu simuler jusqu'à un million de cellules. Si les autres matériels n'ont pu suivre jusque là, cela est dû au fait que notre modèle a une empreinte mémoire importante et que le matériel, ainsi que son implémentation d'OpenCL peut empêcher l'allocation de la mémoire requise un nombre important d'entités.

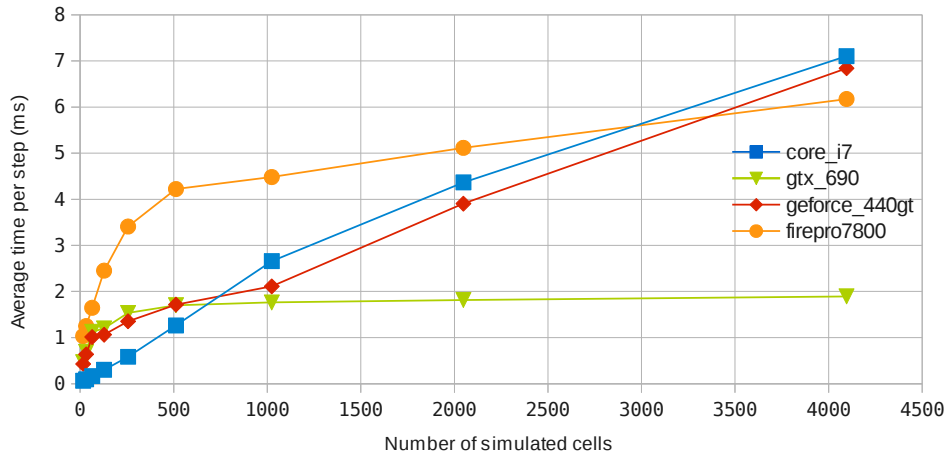
La figure 5.32b présente les mêmes données que la figure 5.32a, il s'agit d'un gros plan sur la partie gauche de la courbe : nous nous sommes intéressés aux temps d'exécution lorsque moins de 5000 cellules sont simulées. En premier lieu, on note les très bonnes performances du Core i7 lorsque le nombre de cellules simulées n'excède pas 700 : ceci est possiblement lié aux branchements conditionnels présents dans le code qui, comme on l'a vu dans le chapitre 4 (section 3.2 page 110), sont moins bien pris en charge par les processeurs spécialisés dans le parallélisme de flux. Une seconde explication peut-être que les unités de calcul du GPU sont sous-utilisées. Des outils de profilage nous permettraient de surveiller l'exécution du code sur ces matériels différents et de mieux comprendre son impact selon le matériel utilisé ou encore la taille du problème traité. En revanche, lorsque le nombre de cellules augmente, le plus important est la capacité du matériel a exécuter du code en parallèle. Le cas de la Firepro 7800 est un peu particulier : ici il faut attendre jusqu'à 3000 cellules simulées pour que sa courbe se croise avec celle du micro processeur. Ce dernier fait 1,68 fois mieux que la Firepro pour simuler 1000 cellules (1,17 fois mieux pour 2000 cellules). Nous n'avons pas d'explication justifiant les performances « passables » de la Firepro pour quelques milliers de cellules, sinon que c'est un matériel vieillissant et que notre application n'est pas optimisée pour un matériel donné.

b) Étude sur la croissance de membres

Nous avons réalisé une seconde étude de performances avec la simulation de croissance de tissus présentée à la section 5.5 : cette étude présente un intérêt supplémentaire car dans ce cas, le système prend en compte la division cellulaire et la chimie artificielle. Nous avons défini un ensemble de simulations en faisant varier le nombre de work-items dans le système et en mesurant le temps total simulé (code hôte et parallèle) nécessaire pour réaliser 200 000 pas de simuation. Nous avons fait varier la taille de l'environnement, c'est-à-dire le nombre de work-items dédiés à l'exécution de la chimie artificielle : 1, 2 et 4 millions. Le nombre maximal d'agents pouvant être actifs dans le système a été fixé à 32 768. Le nombre



(a)



(b)

Figure 5.32 – Étude de performances de la simulation de tri cellulaire avec quatre matériels différents : un CPU Intel Core i7 860 (courbe bleue) et trois cartes graphiques : une NVidia GeForce GT 440 (courbe rouge), une NVidia GeForce GTX 690 (courbe verte) et une ATI Firepro V7800 (courbe orange). (a) Temps d'exécution moyen d'un pas de simulation en fonction du nombre de cellules simulées. (b) Temps d'exécution moyen d'un pas de simulation lorsque moins de 5000 cellules sont simulées.

TABLE 5.5 – Caractéristiques principales des matériels utilisés pour la simulation de croissance de tissus (données extraites des descriptifs fournis par les constructeurs).

	Intel Core i7 4790	ATI Firepro V7800	NVidia GTX 980
Fréquence	3.6 GHz	700 MHz	1278 MHz
Processeurs de flux	8 (4 unités de calcul)	1440 (18 unités de calcul)	2048 (16 unités de calcul)
Mémoire partagée	32KB	32KB	47KB

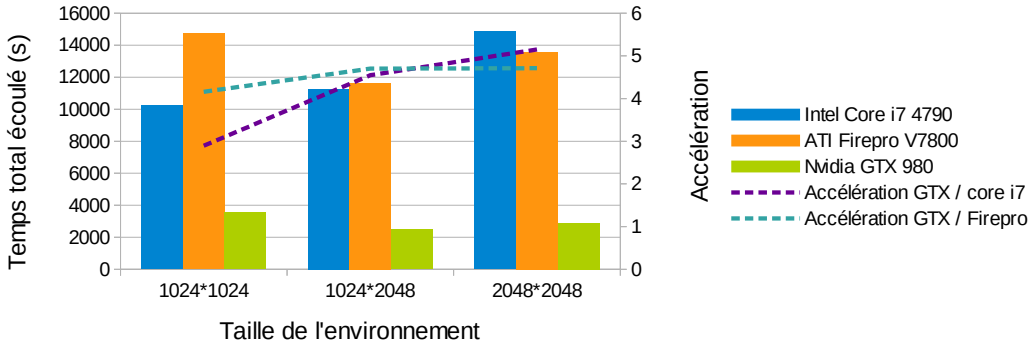


FIGURE 5.33 – Étude de performances de la simulation de croissance de membres dans un tissu. Trois matériels différents ont été utilisés : un CPU Intel Core i7 4790 (bleu) et deux cartes graphiques : une NVidia GeForce GTX 980 (vert) et une ATI Firepro V7800 (orange). Ce graphique présente les temps totaux (code hôte et parallèle) pour exécuter 200 000 pas de simulation selon la taille de l'environnement (c'est-à-dire selon le nombre de work-items instanciés). Le graphique présente également l'accélération du GPU NVidia GTX 980 par rapport aux deux autres matériels utilisés.

d'agents actifs croît lors d'une simulation (jusqu'à environ 16 000), causant une augmentation des temps de calcul parallèle. Nous avons choisi, pour cette étude, de présenter les temps globaux de calcul qui prennent en compte à la fois le nombre de work-items dédiés à la chimie artificielle ainsi que le nombre de work-items (actifs ou non) dédiés à l'exécution des agents.

Trois matériels ont été utilisés pour cette étude : un CPU Intel Core i7 4790, un GPU ATI Firepro V7800 et un GPU NVidia GeForce GTX 980. Les caractéristiques de ces matériels sont synthétisés dans le tableau 5.4. La figure 5.33 présente les performances du système. La NVidia GTX 980 montre, sans surprise, les meilleurs temps de calculs : lorsque le nombre total de work-items dépasse 4 millions, la GTX 980 a une accélération de 5.15 par rapport au Core i7 et une accélération de 4.71 par rapport à la Firepro V7800.

Ces deux études permettent de montrer la capacité de mise à l'échelle de notre système : dans un premier temps, nous avons simulé jusqu'à un million de cellules dans une simulation de tri cellulaire impliquant des comportements simples ; dans un second temps, nous avons simulé plusieurs milliers de cellules en interaction et exhibant des comportements complexes, dont la division cellulaire, dans un environnement chimique. Utiliser du

matériel de type carte graphique permet de profiter de la puissance de calcul d'un matériel très répandu et peu onéreux. En pratique, les temps de calcul peuvent être affectés par d'autres facteurs : le matériel hôte qui peut entraîner une surcharge en temps de calcul causé par l'ordonnancement de code ou bien par des traitements pouvant être réalisés entre deux appels de kernels. Les transferts mémoire peuvent également s'avérer problématiques : comme dans tout système, ce sont les transferts mémoire qui représentent le principal goulet d'étranglement, mais nous n'en avons que très peu, en particulier parce que les données liées à l'affichage destinées à OpenGL sont partagées avec OpenCL.

La prochaine section permet de synthétiser les éléments présentés au cours de ce chapitre sur la simulation de systèmes biologiques.

6 Synthèse du chapitre

Ce chapitre nous a permis, d'une part, de présenter une revue de simulateurs de systèmes biologiques et, d'autre part, de présenter notre propre proposition de simulateur et des applications réalisées avec nos outils. Nous avons vu que les simulateurs sont le plus souvent associés à un modèle particulier, et qu'à l'instar des modèles, il semble peu réaliste de pouvoir utiliser un seul et même simulateur pour simuler tout système biologique. Certains simulateurs permettent toutefois de simuler des éléments tant discrets que continus, comme un modèle de Potts cellulaire couplé à des équations différentielles dans le cas de CompuCell 3D (Swat et coll., 2012). D'autres simulateurs sont dédiés à des modèles discrets, comme dans le cas de Flame (Kiran et coll., 2010), pour la simulation de systèmes multi-agents. D'autres encore permettent la simulation de systèmes continus, comme VirtualCell (Loew et Schaff, 2001) qui est consacré à l'étude de la dynamique interne des cellules au moyen d'équations différentielles. Dans notre cas, le modèle que nous proposons est hybride, notre simulateur permet donc de simuler tant des aspects discrets que continus.

Bilan de notre proposition d'un simulateur. Nous avons choisi de proposer notre propre simulateur. La première raison est qu'il n'existe pas, à notre connaissance, de simulateur reposant sur OpenCL, offrant ainsi la possibilité d'exécuter des simulations sur des matériels hétérogènes. Notre simulateur est construit autour d'une architecture logicielle généraliste, formalisée grâce à l'utilisation d'un patron de conception facilitant sa compréhension et sa maintenabilité. Nous avons inclus des mécanismes dédiés à l'utilisation d'OpenCL en ce qui concerne la gestion mémoire : les données sont stockées dans des structures de tableaux et notre stratégie pour la réplication d'entités repose sur l'utilisation conjointe et couplée de deux méthodes : la première, par pointeur, est basée sur la défragmentation des données que nous réalisons au moyen d'un algorithme de tri performant, le *Batcher's merge-exchange* (Knuth, 1973) ; la seconde méthode est stochastique et ne dépend pas de la fragmentation de la mémoire mais son succès est conditionné au fait que la mémoire ne soit pas trop remplie. L'utilisation couplée de ces deux méthodes permet de garantir que les entités pourront se répliquer, indépendamment de l'état de la mémoire, tout en bénéficiant des bonnes performances de l'algorithme de Lysenko et D'Souza (2008) et du tri *Batcher's merge-exchange* (Knuth, 1973).
Nous avons apporté des éléments de validation corroborant l'utilité de la mise en place de cette stratégie de réplication. En outre, l'utilisation d'un fichier de configuration lu avec la bibliothèque libre `libconfig` permet la manipulation du modèle à un assez haut niveau, grâce à la syntaxe simple de ce fichier de configuration.

Liens entre notre modèle de cellule virtuelle et notre simulateur. Nous avons fait le lien, en matière d'implémentation, entre le modèle que nous avons proposé au chapitre 3 et le simulateur présenté dans ce chapitre. Les exemples d'application introduits montrent l'expressivité de nos outils. Ils nous permettent également d'apporter des éléments de validation de notre modèle. Nous avons testé les différents mécanismes mis en place dans notre système : adhésion différentielle des cellules, fixation des cellules et interactions avec la matrice extra-cellulaire, chimie artificielle, utilisation de gradients de molécules, inter-dépendances et homéostasie dans un tissu ainsi que prolifération cellulaire induite par des facteurs de croissance. Ces mécanismes interviennent tous à différentes étapes de développement de tissus, d'organes, d'organismes ; que ceux-ci soient sains ou pathologiques. Dans l'optique de proposer des simulations de morphogenèse de tissus au sens large (sains comme pathologiques), nous pensons qu'il est indispensable qu'un modèle soit pourvu de ces différents éléments. Dans le cadre des expérimentations sur le tri cellulaire d'une part et, d'autre part, sur la croissance de membres dans un organisme virtuel, deux études de performances nous ont également permis de montrer que notre système supporte l'exécution d'un nombre important de cellules et d'un environnement chimique important, ouvrant la voie à des simulations plus ambitieuses.

Compte tenu des expérimentations présentées dans ce chapitre, les objectifs à poursuivre dans ce cadre sont doubles : en premier lieu il conviendrait de coupler notre modèle à davantage de données expérimentales. Les paramètres de base du modèle sont les valeurs présentées dans le tableau 3.3 (chapitre 3 page 92) mais un ajustement plus fin de ces paramètres (par exemple les seuils de VEGT ou de Fgf8) permettrait à notre modèle de gagner en pertinence. En second lieu, l'idée de descendre à plus bas niveau dans le modèle et d'inclure des notions de régulation génétique, même très simplifiées nous permettrait d'accroitre l'expressivité du modèle : nous avons vu dans le cas du développement de membres que les zones favorisant la prolifération des cellules (les AER) sont « placées à la main » dans le tissu quand les informations de position contenues et exprimées dans le génome permettent aux cellules de se différencier afin de constituer les AER.

D'autres perspectives de travail, à la fois liées au modèle et au simulateur, sont à envisager. Elles sont présentées plus en détail dans le chapitre suivant, permettant de dresser un bilan global des travaux présentés dans ce mémoire ainsi que de présenter les pistes de travail envisagées.

Conclusions

Nous nous sommes intéressés, durant cette thèse, à la modélisation et à la simulation numérique de systèmes biologiques multi-cellulaires. Nous avons privilégié les aspects de morphogenèse de tissus, sains ou pathologiques et avons pour cela développé un modèle biomécanique de cellule auquel de nombreux comportements ont été ajoutés : migration, division, déformation, adhésion différentielle, compression, étirement, cisaillement, production de molécules, consommation de molécules, cycle cellulaire, apoptose, lyse, communication inter-cellules aux travers de signaux biochimiques. Afin de simuler ce modèle, une architecture logicielle parallèle a été mise au point. Elle est dédiée à la simulation de tissus cellulaires dont le nombre de cellules peut atteindre des milliers de cellules (jusqu'à un million de cellules ont été simulées dans le cadre d'une de nos expérimentations *in silico*) et comportant un environnement chimique. Elle est aussi adaptée à l'évolution du matériel informatique utilisant les processeurs multi-cœurs, permettant ainsi de faire croître le nombre de cellules simulées sans revoir la programmation, malgré le fait que l'utilisation d'un même programme sur des matériels hétérogènes rende difficile l'optimisation de ce programme.

Bien que la volonté de reproduire le vivant sur ordinateur date du début de l'informatique (milieu du XX^{e} siècle), ce n'est que depuis récemment que, d'un côté grâce aux nouvelles observations réalisées en biologie et, de l'autre, avec l'avancée des technologies, les simulations de tissus basées sur des principes biomécaniques sont réalisées. Deux points ici sont fondamentaux. Le premier point est la puissance de calcul disponible. Fleischer indique dans sa thèse que la puissance de calcul dont il disposait alors était bien inférieure à ce dont il avait besoin pour représenter quelques centaines de cellules en interaction et ce même si ses modèles n'étaient pas déformables mais des sphères dures adhésives. Outre la puissance de calcul qui devient maintenant plus importante, le deuxième point concerne la connaissance des cellules et des tissus vivants. Les années 1970 à 2000 ont été les années de la biologie moléculaire, faisant suite à la découverte de l'ADN par James Watson et Francis Crick en 1953. La réussite du Human Genome Project, acté par une publication en 2001 dans la revue Nature, a été le point culminant du règne sans partage de la biologie moléculaire. Or depuis les années 2010, et faisant suite au besoin de mieux comprendre la masse importante de données récoltées en biologie, des études jusqu'alors éclipsées reviennent au goût du jour : l'étude des cellules dans leur environnement (aspects épigénétiques), la morphogenèse de tissus (grâce à la microscopie confocale 3D, non léthale pour les cellules), l'ingénierie de tissus (avec les cellules souches induites) apportent de nouvelles données, cette fois-ci à l'échelle cellulaire.

Ces avancées technologiques et scientifiques renforcent le potentiel de modélisation et de simulation de manière importante. L'état de l'art que nous avons écrit au début de ce document nous a permis d'en apprécier la diversité au travers différents modèles de cellule. Ces derniers utilisent soit l'approche continue, soit l'approche discrète ou bien encore une approche hybride qui combine des éléments continus et discrets au sein d'un même modèle. Un autre aspect est celui des modèles qui privilégient soit une approche individuelle sur les cellules (comportement, interactions, forme), soit une vision plus globale où une population

de cellules est l'objet principal d'étude. Notre approche utilise une modélisation hybride combinant une approche individus-centrée (notamment pour notre modèle biomécanique de cellule) et d'une approche par population (pour notre système de chimie artificielle). Notre approche comprend des éléments continus à différents niveaux : calcul de forces physiques, déformation, déplacement des cellules et chimie artificielle sont les éléments continus intégrés dans notre modèle. Les éléments discrets concernent les agents cellules ainsi que le découpage sous la forme d'une grille de l'environnement de chimie artificielle. Cette approche hybride permet de prendre plus facilement en compte certains aspects d'un modèle qui seraient plus difficiles à considérer à travers l'utilisation d'un formalisme unique.

Notre modèle biomécanique de cellule virtuelle est composé de différents éléments : une structure physique (un ensemble de nœuds reliés par des ressorts) bien définie grâce à sa formalisation par des opérations de trigonométrie ; des comportements cellulaires modélisés de manière simplifiée via différents formalismes (cycle cellulaire et automate à états ; croissance des cellules et facteur d'échelle ; division cellulaire et algorithme générique pour une cellule comportant n nœuds sur sa membrane ; interactions cellulaires et forces physiques ; considération des contraintes physiques issues de l'environnement et simples opérations géométriques ; chimie artificielle et équations de diffusion-réaction) et le couplage à un système multi-agents détaillé à travers l'étude de la granularité de ce système ainsi que l'ordonnancement, la mémoire et l'environnement des agents du système. Notre modèle contient également un ensemble de paramètres « de base », dont les valeurs sont extraites de différents travaux issus de la littérature.

L'idée de simuler des milliers d'agents en interaction pour étudier la morphogenèse de tissus sains ou pathologiques a nécessité de proposer un environnement de simulation adapté à ce types de modèle. Nous nous sommes naturellement portés vers l'utilisation de matériels parallèles et nous avons établi que la plateforme de développement OpenCL était la plus adaptée à nos besoins : en effet, OpenCL permet de programmer de manière unifiée des matériels hétérogènes, permettant de simplifier (en partie) la programmation de matériels parallèles. L'établissement d'un état de l'art passant en revue différents simulateurs existants nous a conduit à la conclusion qu'il n'existe pas, à notre connaissance, de simulateur parallèle reposant entièrement sur OpenCL. Partant de ce constat, nous avons décidé de proposer notre propre simulateur parallèle. Comme pour notre modèle de cellule virtuelle, nous nous sommes attachés à proposer une formalisation aussi rigoureuse que possible de ce simulateur à travers différents éléments : nous avons détaillé son architecture logicielle avec le formalisme UML ; présenté les structures de données utilisées en accord avec les activités des work-items (unités de calcul virtuelles fonctionnant en parallèle) et les accès mémoire des agents du système. Nous proposons également des mécanismes de gestion mémoire permettant de pallier le fait de ne pouvoir allouer de la mémoire dynamiquement avec OpenCL ; enfin, nous avons mis en place un fichier de configuration et un générateur de code permettant de faciliter l'utilisation et la simulation du modèle. Quelques cas d'étude permettent de mettre en évidence les comportements que nous avons modélisés dans notre modèle et d'apporter des éléments de validation à la fois de ces comportements et de l'implémentation du modèle dans le simulateur. Deux études de performances nous confortent dans l'idée que notre simulateur parallèle peut potentiellement apporter en souplesse — en matière de tailles de systèmes simulés — dans l'étude de développement de tissus.

Les travaux présentés dans ce mémoire sont loin d'être exempts de défauts et plusieurs perspectives de travail sont à envisager. Tout comme la nature de nos travaux, ces perspectives comportent deux facettes concernant pour l'une, le modèle biomécanique de cellule virtuelle que nous proposons, et pour l'autre le simulateur parallèle que nous avons conçu.

Voici en premier lieu les limites et perspectives concernant notre modèle. Si nous proposons un jeu de paramètres « de base » avec notre modèle, celui-ci n'est en réalité pas complet car des paramètres peuvent être ajoutés ou retirés pour une simulation donnée. Par exemple dans le cas de développement de membres ou de bourgeonnement de vaisseaux sanguins, nous avons fixé de manière empirique les seuils de Fgf8 ou de VEGF. Ces données peuvent très probablement être obtenues auprès de biologistes. Les éléments de validation apportés par les simulations que nous avons réalisées et présentées doivent être approfondis : il convient de comparer ces simulations à des expérimentations réelles. La méthode diffère d'une simulation à une autre ; par exemple dans le cas de la migration de cellule sur un gradient de molécules, il conviendrait de comparer les vitesses de migration d'une cellule virtuelle avec celle d'une cellule réelle. Une seconde perspective consisterait à étendre la modélisation du cycle cellulaire telle que nous la proposons, la dérégulation de ce cycle étant une des causes de développement tumoral par exemple. Une troisième perspective d'amélioration du modèle concerne l'incorporation d'éléments de modélisation à plus bas niveau : nous pensons notamment à l'utilisation d'un génome, même simplifié, qui permettrait de coder différentes informations relatives à la cellule. À titre d'exemple, ces informations pourraient être des informations de positions qui permettraient, dans le cas de la croissance de membres, que les cellules de l'AER se différencient grâce à ces informations. La quatrième perspective découle naturellement de ces travaux : le passage de la 2D à la 3D de notre modèle permettrait de gagner en réalisme. Nous considérerions la forme de la cellule comme une isosphère et les fondements du modèle resteraient identiques : les interactions et comportements seraient modélisés par des forces mais qui seraient cette fois considérés en 3 dimensions. Finalement, une dernière perspective concerne l'amélioration du moteur physique de notre modèle. Pour cela, nous avons deux idées d'amélioration : en premier lieu, il serait intéressant de permettre au moteur d'être plus précis et de faire moins d'approximations, par exemple en ne prenant pas que le nœud central des cellules pour définir certaines interactions mais l'ensemble des nœuds constituant la cellule ; en second lieu, et pour gagner en terme de cohérence dans les tissus simulés, un système de fusion et de séparation des nœuds des cellules pourrait être mis en place : si des nœuds sont très proches, leur fusion permettraient de faire en sorte que la cohérence du tissu résultant soit plus importante.

Un deuxième ensemble de limites et de perspectives concernent le simulateur parallèle. Tout d'abord, il conviendrait d'optimiser l'implémentation du modèle, qui à l'heure actuelle a été implémenté de manière « brute », même si notre modèle se prête bien à une implémentation parallèle grâce à la limitation importante des synchronisations et l'absence de communication entre les entités du système. Nous tirerions beaucoup plus parti du parallélisme offert par OpenCL si nous procédions à un découpage adéquat de nos données de simulation (les agents et l'environnement) afin d'utiliser la mémoire locale qui permet, potentiellement, de grandement accélérer l'exécution d'un programme parallèle (si elle est bien utilisée et cela dépend des modèles et des algorithmes). La seconde perspective concerne les études de performances réalisées dans le système : il serait intéressant de mesurer le surcoût de calcul du code hôte pour, par exemple, optimiser le découpage du modèle en kernels (afin de maximiser l'exécution de portions de code parallèles) et leur ordonnancement sur le matériel (qui doit avoir un coût inférieur à ce que l'on peut gagner en maximisant le parallélisme). Un troisième point concerne l'amélioration de l'allocateur utilisé dans le simulateur pour permettre aux entités de se répliquer dans le système. L'utilisation d'un double tampon pourrait permettre la gestion des identifiants du système sans qu'il soit nécessaire de trier les données : si cela requerrait, d'une part, d'utiliser un peu plus de mémoire pour

l'implémentation du modèle, cela pourrait d'un autre côté permettre de gagner en temps de calcul. Le quatrième et dernier point concerne le paramétrage du système. L'utilisation d'une interface graphique permettrait un usage simplifié du simulateur et le fichier de paramètres servirait d'intermédiaire entre le modèle et la vue. Enfin, la dernière perspective envisagée est également la plus ambitieuse : il s'agit de réfléchir à l'idée de faire « réellement » de la programmation hétérogène, dans le sens ou différents matériels sont utilisés en même temps lors d'une simulation donnée. Pour préciser cette idée, le but serait d'utiliser un matériel pour une portion de modèle donné et un autre matériel pour une autre portion du modèle. Ces portions de modèle présenteraient des caractéristiques différentes dans leur construction et seraient donc prises en charge par un matériel adapté. Pour donner un exemple, prenons le modèle de cellule lui même et le modèle de chimie artificielle. Le code du modèle de cellule est complexe et contient beaucoup de branchements conditionnels qui, à priori, sont mieux pris en charge par des micro-processeurs « classiques ». Pour accroître le parallélisme, il est envisageable de se tourner vers des matériels de type Intel Xeon Phi qui sont des cartes comprenant des grappes d'une douzaine de micro-processeurs. La chimie artificielle, quant à elle, est beaucoup plus simple dans son implémentation et chaque instance de kernel allouée va réaliser exactement le même traitement. De plus, dans l'état actuel des choses, nous allouons plus d'instances de kernels pour la chimie que pour les cellules virtuelles : un matériel hautement parallèle pour le traitement de flux, comme un processeur graphique, semble tout à fait adapté à ce type de traitement. Évidemment cela reste une idée jusqu'à présent et une réflexion plus approfondie devrait nous permettre d'étudier plus avant la faisabilité de distribuer notre modèle sur différents matériels.

De manière générale, les travaux réalisés lors de cette thèse répondent aux objectifs que nous nous étions fixés : d'un côté proposer des briques logicielles (notre modèle de cellule virtuelle) capables de reproduire en simulation certains comportements de cellules réelles lors du développement de tissus sains ou pathologiques ; de l'autre côté proposer un outil de simulation tirant parti des technologies actuelles afin de permettre la simulation de systèmes complexes, comprenant des milliers d'entités en interaction.

Références

Aegerter-Wilmsen T., Smith A.C., Christen A.J., Aegerter C.M., Hafen E. et Basler K., « Exploring the effects of mechanical feedback on epithelial topology », *Development*, vol. 137, n° 3, p. 499–506, 2010.

Allan R., « Survey of agent based modelling and simulation tools », Rapport technique, Computational Science and Engineering Department, STFC Daresbury Laboratory, Daresbury, Warrington WA4 4AD, 2009.

Altabef M., Clarke J. et Tickle C., « Dorso-ventral ectodermal compartments and origin of apical ectodermal ridge in developing chick limb », *Development*, vol. 124, n° 22, p. 4547–4556, 1997.

AMD, *AMD Accelerated Parallel Processing OpenCL Programming Guide*, 2012.

Ameisen J.C., *La sculpture du vivant*, 2003.

Amir-Kroll H., Sadot A., Cohen I.R. et Harel D., « GemCell : A generic platform for modeling multi-cellular biological systems », *Theor. Comput. Sci.*, vol. 391, n° 3, p. 276–290, fév. 2008.

Ballet P., BioDyn, interdisciplinary approach for creating and designing integrated modelling and simulation software in computational biology, 2012, Habilitation à diriger les recherches - Université de Bretagne Occidentale, France.

Ballet P. et Tracqui P., « Migration de cellules virtuelles déformables- modélisation biomécanique multiagent de la migration cellulaire », *RSTI série TSI (N° Spécial ' Modélisation et simulation pour la post-génomique'), Edition Lavoisier - Hermes Sciences - volume 26 - n°1-2/2007*, fév. 2007.

Belmonte J.M., Thomas G.L., Brunnet L.G., de Almeida R.M.C. et Chaté H., « Self-propelled particle model for cell-sorting phenomena », *Phys. Rev. Lett.*, vol. 100, p. 248702, 2008, doi : 10.1103/PhysRevLett.100.248702.

Bertassoni L.E., Cecconi M., Manoharan V., Nikkhah M., Hjortnaes J., Cristino A.L., Barabaschi G., Demarchi D., Dokmeci M.R., Yang Y. et Khademhosseini A., « Hydrogel bioprinted microchannel networks for vascularization of tissue engineering constructs », *Lab Chip*, vol. 14, p. 2202–2211, 2014, doi : 10.1039/C4LC00030G.

Bordini R.H., Hübner J.F. et Tralamazza D.M., Using Jason to implement a team of gold miners, Dans *Computational Logic in Multi-Agent Systems*, Inoue K., Satoh K. et Toni F. (coordinateurs), vol. 4371 de *Lecture Notes in Computer Science*, p. 304–313, Springer Berlin Heidelberg, 2007, doi : 10.1007/978-3-540-69619-3_18.

Breitbach C., Burke J., Jonker D., Stephenson J., Haas A., Chow L., Nieva J., Hwang T., Moon A., Patt R., Pelusio A., Le Boeuf F., Burns J., Evgin L., De Silva N., Cvancic S., Robertson T., Je J., Lee Y., Parato K., Diallo J., Fenster A., Daneshmand M., Bell J. et Kirn D., « Intravenous delivery of a multi-mechanistic cancer-targeted oncolytic poxvirus in humans », *Nature*, vol. 477, p. 99–102, 2011.

Cao Y., Liang C., Naveed H., Li Y., Chen M. et Nie Q., « Modeling spatial population dynamics of stem cell lineage in tissue growth », Dans *34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, p. 5502–5505, 2012, doi : 10.1109/EMBC.2012.6347240.

Chapman B., Jost G. et van der Pas R., *Using OpenMP Portable Shared Memory Parallel Programming*, The MIT Press, 2007.

Chaturvedi R., Huang C., Kazmierczak B., Schneider T., Izaguirre J., Glimm T., Hentschel H., Glazier J., Newman S. et Alber M., « On multiscale approaches to three-dimensional modelling of morphogenesis », *Journal of The Royal Society Interface*, vol. 2, n° 3, p. 237–253, 2005, doi : 10.1098/rsif.2005.0033.

Chélin Y., Azzag K., Canadas P., Averseng J., Baghdiguian S. et Maurin B., « Simulation of cellular packing in non-proliferative epithelia », *Journal of Biomechanics*, vol. 46, n° 6, p. 1075 – 1080, 2013, ISSN 0021-9290, doi : http://dx.doi.org/10.1016/j.jbiomech.2013.01.015.

Chen T., Kowalczyk P., Ho G. et Chisholm R., « Targeted disruption of the dictyostelium myosin essential light chain gene produces cells defective in cytokinesis and morphogenesis », *Journal of Cell Science*, vol. 108, n° 10, p. 3207–3218, 1995.

Chen W., Ward K., Li Q., Kecman V., Najarian K. et Menke N., « Agent based modeling of blood coagulation system : Implementation using a gpu based high speed framework », Dans *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE*, p. 145–148, Aug 2011, doi : 10.1109/IEMBS.2011.6089915.

Christley S., Lee B., Dai X. et Nie Q., « Integrative multicellular biological modeling : a case study of 3D epidermal development using GPU algorithms », *BMC Systems Biology*, vol. 4, n° 1, p. 107, 2010, doi : 10.1186/1752-0509-4-107.

Cussat-Blanc S., Luga H. et Duthen Y., From Single Cell to Simple Creature Morphology and Metabolism, Dans *Artificial Life XI*, p. 134–141, MIT Press, 2008.

Da-Jun T., Tang F., Lee T., Sarda D., Krishnan A. et Goryachev A., Parallel computing platform for the agent-based modeling of multicellular biological systems, Dans *Parallel and Distributed Computing : Applications and Technologies*, vol. 3320 de *Lecture Notes in Computer Science*, p. 5–8, Springer Berlin Heidelberg, 2005.

Debrauwer L., *Design Patterns en Java. Les 23 modèles de conception : descriptions et solutions illustrées en UML 2 et Java*, St-Herblain : Éd. ENI, 2013.

Decker K.S., Distributed artificial intelligence testbeds, Dans *Foundations of Distributed Artificial Intelligence*, O'Hare G.M.P. et Jennings N.R. (coordinateurs), chap. 3, p. 119–138, John Wiley & Sons, 1996.

Dellago C., Transition path sampling, Dans *Handbook of Materials Modeling*, Yip S. (coordinateur), p. 1585–1596, Springer Netherlands, 2005, ISBN 978-1-4020-3287-5, doi : 10.1007/978-1-4020-3286-8_79.

Demazeau Y., « From interactions to collective behaviour in agent-based systems », Dans *In : Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, p. 117–132, 1995.

Dingli D., Cascino M.D., Josić K., Russell S.J. et Željko Bajzer, « Mathematical modeling of cancer radiovirotherapy », *Mathematical Biosciences*, vol. 199, n° 1, p. 55 – 78, 2006, ISSN 0025-5564, doi : http://dx.doi.org/10.1016/j.mbs.2005.11.001.

Dittrich P., Ziegler J. et Banzhaf W., « Artificial chemistries - a review », *Artificial Life*, vol. 7, p. 225–275, 2001.

Du P., Weber R., Luszczek P., Tomov S., Peterson G. et Dongarra J., « From CUDA to OpenCL : Towards a performance-portable solution for multi-platform GPU programming », *Parallel Computing*, vol. 38, n° 8, p. 391 – 407, 2012, ISSN 0167-8191, doi : 10.1016/j.parco.2011.10.002.

Duperray A., Molécules d'adhérence et signalisation cellulaire, Dans *Éléments de biologie à l'usage d'autres disciplines. De la structure aux fonctions*, p. 73–92, Grenobles Sciences, 2008.

Edgar L., Underwood C., Guilkey J., Hoying J. et Weiss J., « Extracellular matrix density regulates the rate of neovessel growth and branching in sprouting angiogenesis », *PLoS ONE*, vol. 9, 2014, doi : 10.1371/journal.pone.0085178.

Engelhart M., Lebiedz D. et Sager S., « Optimal control for selected cancer chemotherapy ODE models : A view on the potential of optimal schedules and choice of objective function », *Mathematical Biosciences*, vol. 229, n° 1, p. 123 – 134, 2011, doi : 10.1016/j.mbs.2010.11.007.

Eyiyurekli M., Lelkes P. et Breen D., « Simulation of chemotaxis-based sorting of heterotypic cell populations », Dans *Life Science Systems and Applications Workshop, 2007. LISA 2007. IEEE/NIH*, p. 249–252, 2007, doi : 10.1109/LSSA.2007.4400931.

Fatahalian K. et Houston M., « A closer look at GPUs », *Commun. ACM*, vol. 51, n° 10, p. 50–57, oct. 2008, ISSN 0001-0782, doi : 10.1145/1400181.1400197.

Ferber J., *Les Systèmes Multi Agents : vers une intelligence collective*, 1995.

Fleischer K.W., *A multiple-mechanism developmental model for defining self-organizing geometric structures*, Thèse de doctorat, California Institute of Technology, 1995.

Fleury V., « A change in boundary conditions induces a discontinuity of tissue flow in chicken embryos and the formation of the cephalic fold », *The European Physical Journal E*, vol. 34, n° 7, p. 1–13, 2011, ISSN 1292-8941, doi : 10.1140/epje/i2011-11073-0.

Flynn M., « Some computer organizations and their effectiveness », *IEEE Transactions on Computers*, vol. C-21, n° 9, p. 948–960, Sept 1972, ISSN 0018-9340, doi : 10.1109/TC.1972.5009071.

Fridlyand L.E., Jacobson D.A. et Philipson L., « Ion channels and regulation of insulin secretion in human β-cells », *Islets*, vol. 5, p. 1–15, 2013, doi : 10.4161/isl.24166.

Gardner M., « Mathematical games : The fantastic combinations of john conway's new solitaire game "life" », *Scientific American*, vol. 223, p. 120–123, 1970.

Gasser L., Braganza C. et Herman N., MACE : A flexible testbed for distributed AI research, Dans *Distributed Artificial Intelligence*, Huhns M.N. (coordinateur), p. 119–152, Pitman Publishing/Morgan Kaufmann Publishers, 1987.

Gaster B., Howes L., Kaeli D., Mistry P. et Schaa D., *Heterogeneous computing with OpenCL*, Morgan Kaufmann, 2011.

Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R. et Sunderam V.S., *PVM A Users' Guide and Tutorial for Network Parallel Computing*, The MIT Press, 1994.

Gilbert S., *Developmental biology, 6th edition*, Sinauer Associates, ISBN 0-87893-243-7, 2000.

Golbert D., Blanco P., Clausse A. et Feijóo R., « Tuning a lattice-boltzmann model for applications in computational hemodynamics », *Medical Engineering & Physics*, vol. 34, n° 3, p. 339 – 349, 2012, ISSN 1350-4533, doi : http://dx.doi.org/10.1016/j.medengphy.2011.07.023.

Graner F. et Glazier J.A., « Simulation of biological cell sorting using a two-dimensional extended potts model », *Physical Review Letters*, vol. 69, n° 13, p. 2013–2016, 1992.

Gropp W., Lusk E.L. et Skjellum A., *Using MPI Portable Parallel Programming with the Message Passing Interface*, The MIT Press, 1999.

Grunwald D. et Ronot X., Croissance et multiplication cellulaire, Dans *Éléments de biologie à l'usage d'autres disciplines. De la structure aux fonctions*, p. 73–92, Grenobles Sciences, 2008.

Guevel E., « Programmation parallèle sur carte graphique avec OpenCL », Rapport technique, Université de Bretagne Occidentale, mars 2013.

Guevel E., Jeannin-Girardon A. et Dezan C., « Du paramétrage de la granularité du calcul et de la localité des données des implémentations sur GPU - Expérimentations OpenCL », Dans *8 e colloque annuel du GDR SOC-SIP*, France, juin 2013.

Hadjiandreou M. et Mitsis G., « Mathematical modeling of tumor growth, drug-resistance, toxicity, and optimal therapy design », *Biomedical Engineering, IEEE Transactions on*, vol. 61, n° 2, p. 415–425, 2014, ISSN 0018-9294, doi : 10.1109/TBME.2013.2280189.

Hanahan D. et Weinberg R.A., « Hallmarks of cancer : The next generation », *Cell*, vol. 144, p. 646 – 674, 2011.

Hardy J., Pomeau Y. et de Pazzis O., « Time evolution of a two-dimensional classical lattice system », *Physical Review Letters*, vol. 31, p. 276–279, 1973.

Harel D., « Statecharts : a visual formalism for complex systems », *Science of Computer Programming*, vol. 8, n° 3, p. 231 – 274, 1987, ISSN 0167-6423, doi : http://dx.doi.org/10.1016/0167-6423(87)90035-9.

Hoehme S. et Drasdo D., « A cell-based simulation software for multi-cellular systems », *Bioinformatics*, vol. 26, n° 20, p. 2641–2642, 2010.

Hoehme S., Brulport M., Bauer A., Bedawy E., Schormann W., Hermes M., Puppe V., Gebhardt R., Zellmer S., Schwarz M., Bockamp E., Timmel T., Hengstler J.G. et Drasdo D., « Prediction and validation of cell alignment along microvessels as order principle to restore tissue architecture in liver regeneration », *Proceedings of the National Academy of Sciences*, 2010, doi : 10.1073/pnas.0909374107.

Holle A.W., Tang X., Vijayraghavan D., Vincent L.G., Fuhrmann A., Choi Y.S., del Álamo J.C. et Engler A.J., « In situ mechanotransduction via vinculin regulates stem cell differentiation », *STEM CELLS*, vol. 31, n° 11, p. 2467–2477, 2013, ISSN 1549-4918, doi : 10.1002/stem.1490.

Ingber D.E., « Tensegrity I. Cell structure and hierarchical systems biology », *Journal of Cell Science*, vol. 116, p. 1157–1173, 2003a.

Ingber D.E., « Tensegrity II. How structural networks influence cellular information processing networks », *Journal of Cell Science*, vol. 116, p. 1397–1408, 2003b.

Ising E., « Contribution to the Theory of Ferromagnetism », *Z.Phys.*, vol. 31, p. 253–258, 1925, doi : 10.1007/BF02980577.

Jaalouk D.E. et Lammerding J., « Mechanotransduction gone awry », *Nature Reviews Molecular Cell Biology*, vol. 10, n° 1, p. 63–73, 2009, ISSN 1471-0072, doi : 10.1038/nrm2597.

Jeannin-Girardon A., Ballet P. et Rodin V., « A software architecture for multi-cellular system simulations on graphics processing units », *Acta Biotheoretica*, vol. 61, n° 3, p. 317–327, 2013a, ISSN 0001-5342, doi : 10.1007/s10441-013-9187-3.

Jeannin-Girardon A., Ballet P. et Rodin V., An efficient biomechanical cell model to simulate large multi-cellular tissue morphogenesis : Application to cell sorting simulation on GPU, Dans *Proceeedings of the Second International Conference on Theory and Practice of Natural Computing (TPNC), Cáceres, Spain, December 3-5, 2013*, vol. 8273 de *Lecture Notes in Computer Science*, p. 96–107, Springer, 2013b.

Jeannin-Girardon A., Ballet P. et Rodin V., « In silico study of mechanical stresses at the cellular level during tissue development », Dans *13th IEEE International Conference on BioInformatics and BioEngineering (BIBE 2013)*, 2013c.

Jiao Y. et Torquato S., « Emergent behaviors from a cellular automaton model for invasive tumor growth in heterogeneous microenvironments », *PLOS Computational Biology*, vol. 7, p. 1–14, 2011.

Jost G., Jin H., an Mey D. et Hatay F., « Comparing the OpenMP, MPI, and hybrid programming paradigm on an SMP cluster », Dans *Fifth European Workshop on OpenMP*, 2003.

Kass L., Erler J.T., Dembo M. et Weaver V.M., « Mammary epithelial cell : Influence of extracellular matrix composition and organization during development and tumorigenesis », *The International Journal of Biochemistry & Cell Biology*, vol. 39, nº 11, p. 1987 – 1994, 2007, ISSN 1357-2725.

Kiran M., Richmond P., Holcombe M., Chin L.S., Worth D. et Greenough C., « Flame : simulating large populations of agents on parallel hardware architectures », Dans *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems : volume 1 - Volume 1*, vol. AAMAS '10, p. 1633–1636, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

Klein G. et Satre M., Compartimentation cellulaire, Dans *Éléments de biologie à l'usage d'autres disciplines. De la structure aux fonctions*, p. 73–92, Grenobles Sciences, 2008.

Knuth D.E., *The art of computer programming, volume 3 : sorting and searching*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, ISBN 0-201-03803-X, 1973.

Kondo S. et Miura T., « Reaction-diffusion model as a framework for understanding biological pattern formation », *Science*, vol. 329, nº 5999, p. 1616–1620, 2010, doi : 10.1126/science.1179047.

Krombach F., Münzing S., Allmeling A., Gerlach J., Behr J. et Dörger M., « Cell size of alveolar macrophages : an interspecies comparison », *Environmental Health Perspectives*, vol. 105, p. 1261–63, 1997.

Lauffenburger D.A. et Horwitz A.F., « Cell migration : A physically integrated molecular process », *Cell*, vol. 84, nº 3, p. 359 – 369, 1996, ISSN 0092-8674, doi : http://dx.doi.org/10.1016/S0092-8674(00)81280-5.

Lee C., Raffaghello L., Brandhorst S., Safdie F.M., Bianchia G., Martin-Montalvo A., Pistoia V., Wei M., Hwang S., Merlino A., Emionite L., de Cabo R. et Longo V.D., « Fasting cycles retard growth of tumors and sensitize a range of cancer cell types to chemotherapy », *Science Translational Medicine*, vol. 4, 2012.

Lesser V. et Corkill D., « The Distributed Vehicle Monitoring Testbed : A Tool for Investigating Distributed Problem Solving Networks », *AI Magazine*, vol. 4, nº 3, p. 15–33, 1983.

Levitt M. et Warshel A., « Computer simulation of protein folding », *Nature*, vol. 253, p. 694 – 698, 1975, doi : 10.1038/253694a0.

Li Y., Naveed H., Kachalo S., Xu L.X. et Liang J., « Mechanisms of regulating cell topology in proliferating epithelia : Impact of division plane, mechanical forces, and cell memory », *PLOS One*, vol. 7, p. 1–10, 2012.

Liu R.T., Liaw S.S. et Maini P.K., « Two-stage turing model for generating pigment patterns on the leopard and the jaguar », *Phys. Rev. E*, vol. 74, 2006a, doi : 10.1103/PhysRevE.74.011914.

Liu W.F., Nelson C.M., Pirone D.M. et Chen C.S., « E-cadherin engagement stimulates proliferation via Rac1 », *The Journal of Cell Biology*, vol. 173, nº 3, p. 431–441, 2006b, doi : 10.1083/jcb.200510087.

Loew L.M. et Schaff J.C., « The virtual cell : a software environment for computational cell biology », *Trends in Biotechnology*, vol. 19, nº 10, p. 401 – 406, 2001, doi : 10.1016/S0167-7799(01)01740-1.

Lolas G., Mathematical modelling of proteolysis and cancer cell invasion of tissue, Dans *Tutorials in Mathematical Biosciences III*, Friedman A. (coordinateur), vol. 1872 de *Lecture Notes in Mathematics*, p. 77–129, Springer Berlin Heidelberg, 2006, ISBN 978-3-540-29162-6, doi : 10.1007/11561606_3.

Lysenko M. et D'Souza R.M., « A framework for megascale agent based model simulations on graphics processing units », *Journal of Artificial Societies and Social Simulation*, vol. 11, nº 4, p. 10, 2008, ISSN 1460-7425.

Marée A.F.M. et Hogeweg P., « How amoeboids self-organize into a fruiting body : Multicellular coordination in dictyostelium discoideum », *Proceedings of the National Academy of Sciences*, vol. 98, n° 7, p. 3879–3883, 2001, doi : 10.1073/pnas.061535198.

Marsaglia G., The KISS random number generator, adresse : `https://groups.google.com/forum/#!msg/sci.math/k3kVM8KwR-s/jxPdZl8XWZkJ`, Février 2003, Message d'un groupe de discussion [en ligne, consulté le 30 juin 2014].

Merks R., Guravage M., Inzé D. et Beemster G., « Virtualleaf : an open-source framework for cell-based modeling of plant tissue growth and development », *Plant Physiology*, vol. 155, n° 2, p. 656–666, 2011.

Merks R.M. et Glazier J.A., « A cell-centered approach to developmental biology », *Physica A : Statistical Mechanics and its Applications*, vol. 352, n° 1, p. 113 – 130, 2005, doi : 10.1016/j.physa.2004.12.028.

Michel F., Ferber J. et Drogoul A., *Multi-Agent Systems and Simulation : A Survey from the Agent Community's Perspective*, chap. 1, p. 3–51, Taylor and Francis Group, 2009.

Mistry P., Gregg C., Rubin N., Kaeli D. et Hazelwood K., « Analyzing program flow within a many-kernel opencl application », Dans *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, vol. GPGPU-4, p. 10 :1–10 :8, New York, NY, USA, ACM, 2011, ISBN 978-1-4503-0569-3, doi : 10.1145/1964179.1964193.

Miura T. et Shiota K., « TGFβ2 acts as an "activator" molecule in reaction-diffusion model and is involved in cell sorting phenomenon in mouse limb micromass culture », *Developmental Dynamics*, vol. 217, n° 3, p. 241–249, 2000, doi : 10.1002/(SICI)1097-0177(200003)217:3<241::AID-DVDY2>3.0.CO;2-K.

Mogilner A. et Oster G., « Force generation by actin polymerization II : The elastic ratchet and tethered filaments », *Biophysical Journal*, vol. 84, n° 3, p. 1591—1605, 2003.

Moraru I.I., Schaff J.C., Slepchenko B.M., Blinov M.L., Morgan F., Lakshminarayana A., Gao F., Li Y. et Loew L.M., « Virtual Cell modelling and simulation software environment », *IET Systems Biology*, vol. 2, n° 5, p. 352–362, sept. 2008, ISSN 17518849, doi : 10.1049/iet-syb:20080102.

Munshi A., Gaster B.R., Mattson T.G., Fung J. et Ginsburg D., *OpenCL Programming Guide*, Addison Wesley, 2012.

Newman T.J., « Modeling multicellular systems using subcellular elements », *Mathematical Biosciences and Engineering*, vol. 2, p. 611–622, 2005.

NVidia, *NVIDIA's Next Generation CUDA Compute Architecture : Kepler GK110*, 2012.

O'Connor C. et Adams J.U., What Is a Cell ? What Are the Essential Characteristics of Cells ?, Dans *Essentials of Cell Biology*, Cambridge, MA : NPG Education, 2010a.

O'Connor C. et Adams J.U., How Do Cells Know When to Divide ?, Dans *Essentials of Cell Biology*, Cambridge, MA : NPG Education, 2010b.

Palecek S., Loftus J., M.H. G., D.A. L. et A.F. H., « Integrin-ligand binding properties govern cell migration speed through cell-substratum adhesiveness », *Nature*, vol. 385, p. 537–40, 1997.

Park K., Jang J., Irimia D., Sturgis J., Lee J., Robinson P., Toner M. et Bashir R., « 'Living cantilever arrays' for characterization of mass of single live cells in fluids », *Lab on a Chip*, vol. 8, p. 1034–1041, 2008.

Pascalie J., Lobjois V., Luga H., Ducommun B. et Duthen Y., « Checkpoint Orientated Cell Cycle simulation - Issues on Synchronized situation (regular paper) », Dans *Artificial Life, Lansing (USA), 19/07/2012-22/07/2012*, Adami C., Bryson D.M., Ofria C. et Pennock R. (coordinateurs), p. 465–472, The MIT Press, juillet 2012.

de Pillis L., Gu W. et Radunskaya A., « Mixed immunotherapy and chemotherapy of tumors : modeling, applications and biological interpretations », *Journal of Theoretical Biology*, vol. 238, n° 4, p. 841 – 862, 2006, ISSN 0022-5193, doi : 10.1016/j.jtbi.2005.06.037.

Polynikis A., Hogan S. et di Bernardo M., « Comparing different ODE modelling approaches for gene regulatory networks », *Journal of Theoretical Biology*, vol. 261, n° 4, p. 511 – 530, 2009, doi : 10.1016/j.jtbi.2009.07.040.

Potts R.B., « Some generalized order-disorder transformations », *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 48, p. 106–109, 1952.

Raff L.M., *Principles of Physical Chemistry*, Prentice Hall, 2001.

Rao A.S. et Georgeff M.P., « An abstract architecture for rational agents », Dans *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, p. 439–449, 1992.

Reynolds C.W., « Flocks, herds, and schools : A distributed behavioral model », *SIGGRAPH Computer Graphics*, vol. 21, n° 4, p. 25–34, 1987, ISSN 0097-8930.

Richmond P., Coakley S. et Romano D., « Cellular level agent based modelling on the graphics processing unit », *High Performance Computational Systems Biology*, vol. 0, p. 43–50, 2009a, doi : http://doi.ieeecomputersociety.org/10.1109/HiBi.2009.12.

Richmond P., Coakley S. et Romano D.M., « A high performance agent based modelling framework on graphics card hardware with CUDA », Dans *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, vol. AAMAS '09, p. 1125–1126, Richland, SC, International Foundation for Autonomous Agents and Multiagent Systems, 2009b, ISBN 978-0-9817381-7-8.

Rieu J.P., Kataoka N. et Sawada Y., « Quantitative analysis of cell motion during sorting in 2D aggregates of hydra cells », *Physical Review E*, vol. 57, p. 924–31, 1998.

Rolfe R., Nowlan N., Kenny E., Cormican P., Morris D., Prendergast P., Kelly D. et Murphy P., « Identification of mechanosensitive genes during skeletal development : alteration of genes associated with cytoskeletal rearrangement and cell signalling pathways », *BMC Genomics*, vol. 15, n° 1, p. 48, 2014, ISSN 1471-2164, doi : 10.1186/1471-2164-15-48.

Rosenblatt J., Raff M.C. et Cramer L.P., « An epithelial cell destined for apoptosis signals its neighbors to extrude it by an actin- and myosin-dependent mechanism », *Current Biology*, vol. 11, n° 23, p. 1847 – 1857, 2001, ISSN 0960-9822, doi : http://dx.doi.org/10.1016/S0960-9822(01)00587-5.

Rosenbluth M.J., Lam W.A. et Fletcher D.A., « Force microscopy of nonadherent cells : A comparison of leukemia cell deformability », *Biophysical Journal*, vol. 90, n° 8, p. 2994 – 3003, 2006, ISSN 0006-3495, doi : http://dx.doi.org/10.1529/biophysj.105.067496.

Rott N.N. et Sheveleva G.A., « Changes in the rate of cell divisions in the course of early development of diploid and haploid loach embryos », *Journal of Embryology and Experimental Morphology*, vol. 20, n° 2, p. 141–150, 1968.

Rudge T. et Haseloff J., A Computational Model of Cellular Morphogenesis in Plants, vol. 3630 de *Lecture Notes in Computer Science*, p. 78–87, Springer Berlin / Heidelberg, 2005.

Scarpino M., *OpenCL in action. How to accelerate graphics and computation*, Manning, 2011.

Schwann T., *Microscopical researches into the accordance in the structure and growth of animals and plants*, vol. The Sydenham Society, Sydenham Society, 1839, traduit de l'allemand par H. Smith (1847).

Shami M. et Hemani A., « Classification of massively parallel computer architectures », Dans *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, p. 344–351, May 2012, doi : 10.1109/IPDPSW.2012.42.

Sheeba C.J., Andrade R.P. et Palmeirim I., « Limb patterning : From signaling gradients to molecular oscillations », *Journal of Molecular Biology*, vol. 426, n° 4, p. 780 – 784, 2014, ISSN 0022-2836.

Sherratt J. et Chaplain M., « A new mathematical model for avascular tumour growth », *Journal of Mathematical Biology*, vol. 43, p. 291–312, 2001.

Sheth R., Grégoire D., Dumouchel A., Scotti M., Pham J.M.T., Nemec S., Bastida M.F., Ros M.A. et Kmita M., « Decoupling the function of Hox and Shh in developing limb reveals multiple inputs of hox genes on limb growth », *Development*, vol. 140, n° 10, p. 2130–2138, 2013.

Steinberg M., « Reconstruction of tissues by dissiciated cells », *Science*, vol. 141, n° 3579, 1963.

Stoma S., Chopard J., Godin C. et Traas J., « Using mechanics in the modelling of meristem morphogenesis », Dans *5th International Workshop on Functional-Structural Plant Models*, p. 52, 1–4, Napier, New Zealand, 2007.

Swat M.H., Thomas G.L., Belmonte J.M., Shirinifard A., Hmeljak D. et Glazier J.A., Multi-scale modeling of tissues using CompuCell3D, Dans *Computational Methods in Cell Biology*, Asthagiri A.R. et Arkin A.P. (coordinateurs), vol. 110 de *Methods in Cell Biology*, chap. 13, p. 325 – 366, Academic Press, 2012, doi : http://dx.doi.org/10.1016/B978-0-12-388403-9.00013-8.

Sylvain H., Denis A. et Barthou D., « Programmation unifiée multi-accélérateur OpenCL », *Techniques et Sciences Informatiques*, vol. 31, p. 1233–1249, 2012, doi : 10.3166/TSI.31.1233-1249.

Tektonidis M., Hatzikirou H., Chauvière A., Simon M., Schaller K. et Deutsch A., « Identification of intrinsic in vitro cellular mechanisms for glioma invasion », *Journal of Theoretical Biology*, vol. 287, n° 0, p. 131 – 147, 2011, doi : 10.1016/j.jtbi.2011.07.012.

Tomita M., Hashimoto K., Takahashi K., Shimizu T.S., Matsuzaki Y., Miyoshi F., Saito K., Tanida S., Yugi K., Venter J.C. et III C.A.H., « E-cell : software environment for whole-cell simulation. », *Bioinformatics*, vol. 15, n° 1, p. 72–84, 1999.

Tracqui P., « Biophysical models of tumour growth », *Reports on Progress in Physics*, vol. 72, n° 5, p. 056701, 2009.

Tripodi S., Ballet P. et Rodin. V., « Self-organization of a virtual multicellular organism by adding a shape model in the cellular potts model », Dans *Artificial Life XII, 12th International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, p. 249–256, 2010.

Tripodi S., Ballet P. et Rodin V., « GPU implementation and performance analysis of reactive agents having division and mobility capacities », *Multiagent and Grid Systems, IOS Press*, vol. 8, n° 4, p. 289–309, 2012.

Turing A.M., « The chemical basis of morphogenesis », *Philosophical Transactions of the Royal Society of London*, vol. 237, n° 641, p. 37–72, 1952.

Vicsek T., Czirók A., Ben-Jacob E., Cohen I. et Shochet O., « Novel type of phase transition in a system of self-driven particles », *Phys. Rev. Lett.*, vol. 75, p. 1226–1229, 1995, doi : 10.1103/PhysRevLett.75.1226.

Volterra V., « Fluctuations in the Abundance of a Species considered Mathematically », *Nature*, vol. 118, p. 558–560, oct. 1926, doi : 10.1038/118558a0.

Walker D.C. et Southgate J., « The virtual cell - a candidate coordinator for middle-out modelling of biological systems », *Briefings in Bioinformatics*, vol. 10, n° 4, p. 450–461, 2008.

Wolf-Gladrow D., 3. lattice-gas cellular automata, Dans *Lattice Gas Cellular Automata and Lattice Boltzmann Models*, vol. 1725 de *Lecture Notes in Mathematics*, p. 39–138, Springer Berlin Heidelberg, 2000, doi : 10.1007/978-3-540-46586-7_3.

Wright R.S., Haemel N., Sellers G. et Lipchak B., *OpenGL SuperBible 5th ed.*, Addison-Wesley, 2011.

Yennek S., Burute M., Théry M. et Tajbakhshe S., « Cell adhesion geometry regulates non-random DNA segregation and asymmetric cell fates in mouse skeletal muscle stem cells », *Cell Reports*, vol. 7, p. 961–970, 2014.

Yu S.R., Burkhardt M., Nowak M., Ries J., Petrášek Z., Scholpp S., Schwille P. et Brand M., « Fgf8 morphogen gradient forms by a source-sink mechanism with freely diffusing molecules », *Nature*, vol. 461, p. 533–536, 2009.

Zhang Y.T., Alber M.S. et Newman S.A., « Mathematical modeling of vertebrate limb development », *Mathematical Biosciences*, vol. 243, n° 1, p. 1 – 17, 2013, doi : 10.1016/j.mbs.2012.11.003.

Résumé

Développement d'un modèle logiciel de cellule sur processeurs multi-cœurs pour la simulation de morphogenèse de tissus

L'objectif principal de cette thèse est de proposer des outils permettant l'étude numérique du développement de tissus cellulaires à travers une approche individus-centrée comprenant des aspects biomécaniques et chimiques. De telles propositions doivent permettre de mieux comprendre les mécanismes régissant le développement de tissus multi-cellulaires grâce à la simulation numérique, en complément d'expérimentations *in vitro* et *in vivo*. La réalisation de ces objectifs est difficile, mais les avancées dans les domaines de la biologie cellulaire et moléculaire permettent l'observation et la collecte d'un grand nombre de données. En outre, le développement de matériels parallèles permet la simulation de systèmes complexes tels que des systèmes multi-cellulaires. Les systèmes multi-cellulaires exhibent essentiellement deux niveaux de complexité lorsque l'on souhaite les simuler : le premier concerne le nombre potentiellement très important de cellules qu'ils contiennent, nécessitant alors une grande puissance de calcul ; le second concerne la multiplicité des comportements des cellules vivantes, tant au niveau individuel que collectif, ce qui requiert des algorithmes bien spécifiques. La conception de modèles, intégrant à la fois des données biologiques pertinentes, des algorithmes adaptés et reposant sur des processeurs puissants permet de résoudre, au moins en partie, ces deux niveaux de complexité, mais doivent reposer sur une architecture logicielle dédiée. Dans l'idée d'étudier, en simulation numérique, le développement de tissus sains et pathologiques, nos travaux apportent deux éléments. Le premier est un modèle biomécanique de cellule virtuelle comportant des processus impliqués dans la morphogenèse de tissus tels que la division, la différentiation, l'adhésion, la migration, la signalisation et l'apoptose. Le second est un simulateur parallèle reposant sur une architecture logicielle généraliste ainsi que sur des structures de données et des algorithmes originaux permettant d'exploiter la puissance de calcul offerte par les matériels multi-cœurs. Nous présentons dans ce mémoire plusieurs cas d'études qui permettent d'apporter des éléments de validation sur la réalisation de notre modèle et de notre simulateur.

Mots clefs : modélisation, simulation, systèmes multi-agents, biologie computationnelle, cellule virtuelle, OpenCL, programmation parallèle, simulation parallèle

Abstract

Development of a virtual cell model on multi-core devices for the simulation of tissue morphogenesis

The main purpose of this thesis is to present tools built in order to numerically study the development of cellular tissues through an individual-based approach that includes biomechanical elements as well as an artificial chemistry. The objective of such proposals is to gain a better understanding of the mechanisms that rule the development of multicellular tissues using numerical simulation as a complement to *in vitro* and *in vivo* experiments. This objective is difficult to achieve. However technological means in the field of cellular and molecular biology allow the observation and the gathering of many data. Moreover, the development of multi-core devices allows the simulation of complex systems, such as multi-cellular systems. Multi-cellular systems exhibit mainly two levels of complexity : the first one concerns the potentially large amount of cells they contain, which requires a considerable computing power when this point is addressed through individual-based approach ; the second level concerns the diversity of biological cells' behaviors. This level requires specific algorithms, for example to deal with complex behavior such as mitosis. The conception of 1) models that integrate biological data and 2) dedicated algorithms adapted to heterogenous and multi-core devices make it possible to solve, at least to some extent, these two levels of complexity. In order to numerically study both healthy and pathological tissue developpement, we propose two elements. The first is a biomechanical cell model that includes the behaviors involved in tissue morphogenesis (mitosis, differentiation, adhesion, migration, cell-cell signaling and apoptosis). The second element is a parallel simulator that relies on a non-specialized software architecture and on dedicated data structures and algorithms used to benefit from the power of multi-core hardware. In this document, we present several case studies that gives some validation elements of both our model and our simulator.

Keywords : modeling, simulation, agent based model, computational biology, virtual cell, OpenCL, parallel programing, parallel simulation