Outils multirésolutions pour la gestion des interactions en simulation temps réel

Thomas Pitiot

N° d'ordre: 3157

UNIVERSITÉ DE STRASBOURG

Laboratoire des sciences de l'ingénieur, de l'informatique et de l'imagerie

École Doctorale Mathématiques, Sciences de l'Information et de l'Ingénieur

THÈSE

présentée pour obtenir le grade de

Docteur de l'Université de Strasbourg
Discipline / Spécialité : Informatique

par

Thomas PITIOT

Outils multirésolutions pour la gestion des interactions en simulation temps réel

Soutenue publiquement le **17 Décembre 2015**

Membres du jury :

M. Philippe MESEURE ..... *Rapporteur*
Professeur à l'Université de Poitiers

M. Julien PETTRE ..... *Rapporteur*
Chargé de recherche HDR à l'INRIA de Rennes

Mme. Céline LOSCOS ..... *Examinatrice*
Professeure à l'Université de Reims Champagne-Ardenne

M. Fabrice JAILLET ..... *Examinateur*
Maître de Conférence HDR à l'Université Lyon 1

M. David CAZIER ..... *Directeur de thèse*
Professeur à l'Université de Strasbourg

ICube UMR CNRS-Unistra 7357

Remerciements

Je remercie tout d'abord David Cazier qui a su me guider depuis mon stage de Master 2 jusqu'à la fin de ma thèse. Il m'a laissé une grande autonomie tout en sachant apporter les conseils et le recul nécessaire aux moments adéquats.

Je remercie également les membres de mon jury d'avoir accepté d'évaluer mes travaux et tout particulièrement mes rapporteurs Philippe Meseure et Julien Pettré.

Merci à toutes les personnes que j'ai pu croiser durant ces années dans l'équipe IGG qui a constitué le cadre principal de cette thèse. Un clin d'oeil particulier à Sylvain Théry et Pierre Kraemer qui ont été mes guides CGo-GNesques, Thomas Jund et Lionel Untereiner qui ont eu la patience de m'expliquer dans les moindres détails de leur travaux et Arash Habibi qui m'a accompagné (parfois tard dans la nuit) pour la réalisation de belles vidéos. Une pensée particulière pour Noura, Sabah, Étienne, Vasyl, Amir, Kenneth et Alexandre avec qui j'ai le plus échangé durant ce temps au laboratoire.

Je tiens à remercier l'équipe MIMESIS qui m'a chaleureusement accueilli durant ma dernière année de thèse. Je pense particulièrement à Stéphane Cotin qui a suivi de près mes travaux et qui représente pour moi un exemple à suivre en tant que chercheur pluri-disciplinaire, dynamique, optimiste et efficace.

Merci également à Jean Ferry et David Cazier (encore) qui m'ont permis de

découvrir les joies de l'enseignement durant deux années à l'IUT d'Haguenau.

Merci à mes amis Strasbourgeois (ou ex-Strasbourgeois) qui m'ont accueilli dans cette belle ville et qui m'ont accompagné durant cette aventure jusqu'à la soutenance.

Merci à Simon, Yacine et l'ensemble de la team Cureuil pour la bonne ambiance, les discussions infinies et parce que, même de loin, le soutien ça compte.

J'aimerais également remercier ma famille. Tout d'abord, mes parents qui m'ont donné très tôt la curiosité scientifique et le goût de l'informatique. Ils m'ont toujours laissé choisir ma voie tout en soutenant mes décisions. Ensuite mes frères et soeur qui constituent pour moi un socle à toute épreuve.

Enfin, un grand merci à Leslie avec qui j'ai la chance de partager mon quotidien. Elle m'a accompagné et soutenu depuis le début de cette thèse jusqu'à la relecture appliquée de ce mémoire. Cet accomplissement n'aurait pas été possible sans les coups de boosts qu'elle m'a apporté dans les moments de doute.

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Contexte et objectifs

Simulation :
La simulation informatique ou numérique désigne l'exécution d'un programme informatique sur un ordinateur ou réseau en vue de simuler un phénomène physique réel et complexe.

Les simulations d'environnements virtuels actuelles sont de plus en plus nombreuses et ambitieuses. Elles recoupent diverses sciences telles que la physique, la mécanique, la biologie et l'informatique pour des applications dans des domaines allant de la médecine aux jeux vidéo en passant par l'urbanisme et le cinéma. Elles nécessitent de représenter des environnements vastes et dynamiques parfois très complexes. On peut distinguer deux catégories de simulations :

La simulation continue, ou simulation a priori, où le système se présente sous la forme d'équations différentielles à résoudre.

La simulation discrète, ou simulation a posteriori dans laquelle le système est soumis à une succession d'événements qui le modifient. Ce type de simulations a vocation à appliquer des principes simples à des systèmes de grande taille. La simulation discrète peut être synchrone ou asynchrone.
Dans une simulation synchrone, on calcule l'arrivée du prochain événement, et on ne simule que d'événement en événement. Dans une simulation asynchrone, on simule à chaque fois le passage d'un pas de temps sur tout le système.

Nous discuterons ici uniquement aux simulations discrètes asynchrones. Par la suite le terme « simulation » sera utilisé pour ce type de simulations. Dans de telles simulations, les objets présents sont soumis à trois modèles : un modèle de rendu qui détient l'aspect des objets, un modèle physique qui détient leurs propriétés physiques et un modèle d'interactions qui permet de détecter les collisions et les contacts. Ainsi, si une collision entre deux objets a lieu, le modèle d'interactions la détecte et permet au modèle physique de calculer les changements à appliquer au modèle de rendu.

Nous nous intéresserons ici plus particulièrement au modèle d'interactions qui s'est complexifié au fil de l'évolution des modèles géométriques utilisés. En effet, les simulations actuelles n'utilisent plus uniquement des primitives géométriques simple pour représenter les objets. Elles utilisent plutôt des maillages détaillés parfois très complexes pour avoir le meilleur rendu visuel possible et simuler la réalité au plus proche. Dans un premier temps, intéressons nous de plus près au fonctionnement d'une simulation pour ensuite nous pencher sur les enjeux de ce modèle d'interactions.

1.1.1 La simulation et le temps réel

Si diverses que soient les simulations, elles respectent un schéma de fonctionnement général relativement standard. Nous allons présenter ici ce schéma d'exécution classique d'un pas de temps de simulation en détail. Par la suite nous discuterons de la complexité de telles simulations.

Le pipeline de simulation

La figure 1.1 présente le schéma classique d'un pas de temps d'une simulation physique interactive. Dans un premier temps, des forces provenant des périphériques d'interaction et du moteur physique et comportemental sont appliquées aux objets. Ces objets sont altérés par ces forces au niveau géométrique (déformation, déplacement), topologique (coupe, fusion , ...) ou visuel (couleur, luminosité).

Une fois les objets altérés, la présence de collisions ou d'interpénétrations est vérifiée. Cette étape précise, sur laquelle nous reviendrons plus en détails, nécessite une connaissance de l'environnement. Ainsi,détecter les collisions requiert d'effectuer pour chaque objet des requêtes de proximité avec les autres entités environnantes participant aux collisions. Une fois ces requêtes établies, on obtient pour chaque objet les entités potentiellement en collision. On peut donc envoyer ces informations au moteur physique et comportemental afin de calculer les forces à appliquer au prochain pas de temps.

Dans les simulations non interactives, les objets sont soit régis par une force globale que l'on souhaite observer (gravité, équation physique à tester, ...) soit autonomes avec un objectif à atteindre. Nous nous intéresserons en particulier à ce cas de simulation d'objets autonomes appelés « agents » dans le cadre de simulations de foules.

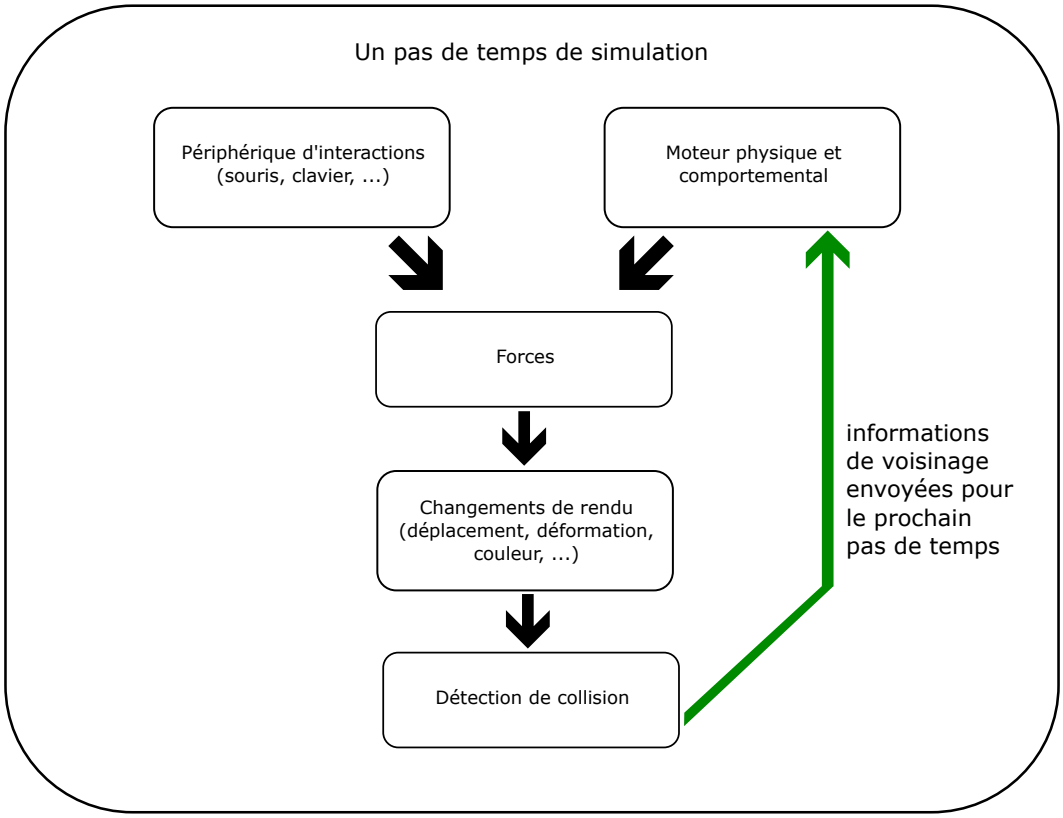

FIGURE 1.1 : Schéma d'exécution classique d'un pas de temps d'une simulation physique interactive

Le temps réel ou interactif

Certaines simulations peuvent être très complexes. Elles peuvent nécessiter un rendu particulièrement précis et de haute définition (cinéma), des calculs physiques lourds (médecine) ou encore être des simulations possédant un grand nombre d'entités (jeux vidéos, simulation de foule).

Dans tous les cas, il s'agit de trouver un compromis entre la précision désirée pour la simulation et le temps d'exécution qu'elle va nécessiter. Certaines simulations pour le cinéma passent outre ce compromis et visent la meilleure qualité possible, même si le rendu doit être calculé pendant plusieurs jours. Si en revanche la simulation est interactive (médecine, jeux vidéos), le compromis à faire prend toute son importance. Pour ce type de simulations, l'ensemble des processus mis en place doit s'exécuter en temps réel, c'est à dire qu'il ne doit pas y avoir de latence trop importante entre deux pas de temps de la simulation.

Le « temps réel » au sens premier n'étant jamais atteint, on utilise le plus souvent ce terme pour parler de temps interactif. Le temps réel ou interactif est un temps d'exécution permettant l'interaction via une contrainte temporelle. Selon les simulations on recherchera un système temps réel strict ou souple.

Pour le temps réel strict, aucun dépassement de la contrainte temporelle du système n'est autorisé. Ce système doit être en place pour les simulations critiques telles que les simulations médicales temps réel. Dans le cas du temps réel souple, la contrainte temporelle doit être respectée pour une moyenne des exécutions, en autorisant donc quelques dépassements appelés « lags ». Cette contrainte temporelle s'exprime sous la forme d'une fréquence de mise à jour de la simulation. Elle varie selon les modes d'interactions utilisés. Si la simulation est uniquement visuelle, il est considéré qu'une fréquence de 30 pas de temps par seconde est suffisante. Si la simulation propose un retour haptique en revanche, cette contrainte sera plutôt fixée autour de 1000 pas de temps par seconde.

1.1.2 Le modèle d'interactions

Détection de collision :
Utilisation d'algorithmes pour tester les collisions (intersection entre solides donnés), pour calculer les trajectoires, les dates d'impact et les points d'impact dans une simulation physique.

Le modèle d'interactions d'une simulation permet de savoir quelles entités entrent en jeu dans la détection de collision et de réaliser cette dernière. Les entités qui entrent en jeux peuvent être très complexes par leur forme, leur topologie et leur évolution au cours de la simulation. Selon le type d'entité traitée les considérations ne sont pas les mêmes. En effet, si l'on traite un objet rigide, on peut pré-calculer un ensemble d'informations le concernant qui ne vont pas changer au cours du temps. En revanche, dans le cas d'objets déformables, il est nécessaire de mettre à jour ces informations régulièrement.

Nous pouvons ainsi établir une échelle de difficulté de simulation selon les caractéristique d'un objet :

— Objet statique : Objet dont la position n'est pas modifiée.
— Objet dynamique : Objet se déplaçant durant la simulation.
— Objet rigide : Objet dont la forme ne change pas.
— Objet déformable : Objet qui peut subir des déformations géométriques grâce à sa plasticité et son élasticité (il peut être tordu ou écrasé).
— Objet à topologie modifiable : Objet pouvant subir des déformations topologiques comme la couture ou la découpe.

Ces entités nécessitent d'être approximées, au moins au niveau de la détection de collision, si l'on veut pouvoir effectuer la simulation en temps réel ou interactif. Voyons donc les techniques utilisées pour approximer ces entités et comment ce choix influe sur la détection de collision en elle-même.

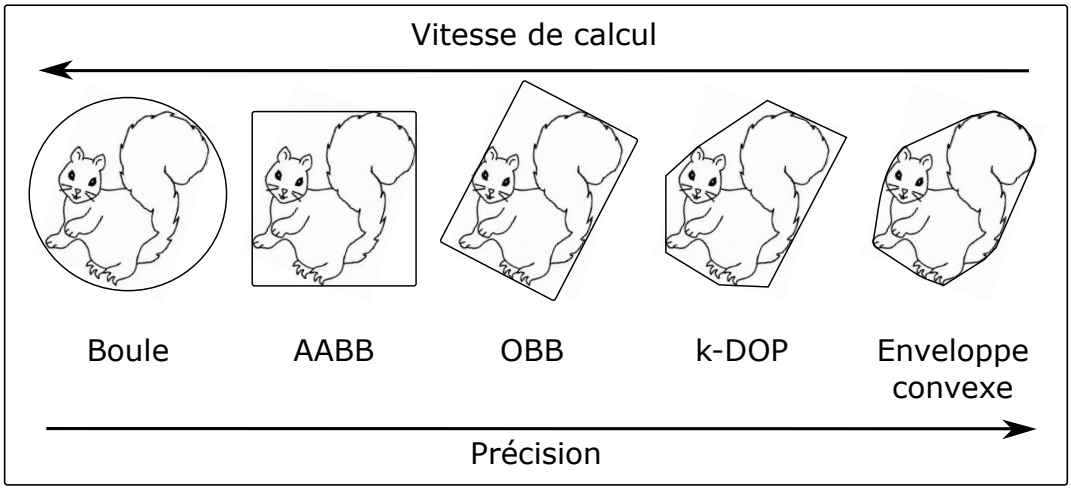

Figure 1.2 : Échelle de précision/vitesse des volumes englobants d'un objet.

La représentation des entités pour l'interaction

Objet convexe :
Un objet est dit convexe si pour toute paire de points de cet objet, le segment qui les joint est entièrement contenu dans l'objet.

Il est courant, et souvent nécessaire, de changer la représentation d'une entité durant les différentes étapes de la simulation. Ainsi, un modèle fin sera utilisé pour le rendu final tandis qu'un modèle grossier est souvent utilisé pour les interactions afin d'accélérer les processus. Ce modèle plus grossier utilisé correspond soit à une version modifiée de l'objet soit à un volume englobant.

Ces volumes englobants prennent diverses formes que l'on peut répartir sur une échelle de précision/coût comme illustré sur la figure 1.2. Plus le modèle est proche de la forme réelle de l'objet plus il est coûteux à calculer et simuler. Inversement, plus le modèle est simple plus la détection de collision sera rapide mais imprécise. On retrouve sur cette échelle non-exhaustive :

— La boule englobante comme méthode la plus rapide.
— La boîte englobante alignée sur l'axe (**A**xis **A**ligned **B**ounding **B**ox).
— La boîte englobante orientée (**O**riented **B**ounding **B**ox).
— Le polytope à orientation discrète de degré k (k-**D**iscrete **O**riented **P**olytope).
— L'enveloppe convexe de l'objet.

Chacune de ces représentations a le mérite de fournir une représentation convexe de l'objet et on choisira une représentation en fonction du nombre d'objets à traiter et de la précision désirée. La stabilité de l'objet simplifié est également à prendre en compte. En effet, pour la simplification d'un objet déformable, il faut être en mesure de percevoir les déformations. Il faut donc avoir une certaine précision sans perdre en efficacité à cause des mises à jour fréquentes du modèle.

Ces représentations plus ou moins simplifiées des objets sont donc utilisées dans les diverses stratégies de détection de collision.

La détection de collision

Voisinage :
On appelle voisinage l'ensemble des entités considérées comme potentiellement en interaction avec une entité donnée.

On peut classer les stratégies de détection de collision dans plusieurs grandes familles : les calculs d'intersections entre objets rigides convexes, les hiérarchies de volumes englobants, les méthodes approximantes et le partitionnement de l'espace.

Les méthodes réalisant des calculs d'intersections entre objets rigides convexes se basent sur la convexité des objets pour déterminer la distance minimale entre deux objets.

Les hiérarchies de volumes englobants permettent de limiter le nombre de tests d'intersection entre paires d'objets. Les volumes englobants des objets ou des parties d'un objet sont ainsi insérés dans une arborescence. Si deux volumes englobants ne s'intersectent pas, alors il n'y a pas de collision entre les deux objets. Les tests d'intersections des boîtes englobantes sont facilités si elles sont composées de plans comme les boîtes OBB ou k-DOP. En effet, ces plans permettent une application directe du théorème de l'hyperplan séparateur de Minkowski.

Théorème de l'hyperplan séparateur de Minkowski :
Soit E un espace affine de dimension finie, A et B deux convexes de E non vides et disjoints. Alors il existe un hyperplan séparant A et B.

Ce théorème s'applique par la présence d'un axe séparateur entre deux polyèdres convexes P et P' disjoints, avec cet axe orthogonal à :

— une face de P
— une face de P'
— une arête de chacun des deux polyèdres

Les méthodes approximantes se basent sur des approximations de représentation pour fournir un résultat de détection de collision en respectant toujours un temps critique d'exécution. Ces méthodes partent du principe qu'avec une approximation suffisante, un humain ne détectera pas les quelques erreurs dues à ces approximations. Parmi ces méthodes, les méthodes stochastiques réalisent un échantillonnage des objets à tester, soit à l'aide de particules évoluant à leur surface, soit en testant de manière aléatoire des paires d'éléments entre objets.

Enfin, **les méthodes basées sur un partitionnement** de l'espace vont réaliser une décomposition de l'environnement en sous-ensembles disjoints. Ce partitionnement permet de tester uniquement les entités présentes dans un ensemble de cellules voisines. Nous nous intéresserons particulièrement à ce genre de techniques et nous les détaillerons dans notre état de l'art. Nous avons basé nos travaux de thèse sur une de ces méthodes : les cartes combinatoires.

Quelque soit la stratégie choisie, on peut décomposer la détection de collision en deux phases. Une première phase de requêtes de proximités où l'on cherche à récupérer des informations de proximité entre les entités pour déterminer des voisinages. Une seconde phase où l'on détermine si des objets voisins sont en collision ou devront être influencés conjointement.

Nous nous intéresserons ici tout particulièrement à cette première phase de requêtes de proximité en explorant les diverses solutions proposées dans la littérature. Nous présenterons par la suite la solution généraliste que nous avons utilisée qui allie les forces des différents modèles de l'état de l'art.

Notre objectif principal est de réaliser de manière optimale des requêtes de proximité entre objets (autonomes ou dirigés interactivement) évoluant dans une simulation en temps réel. Nous présenterons les atouts et définirons le cadre et les limitations de notre méthode en détails.

Autres interactions

Simulation de foule :
La simulation de foule est le processus de simulation du mouvement d'un grand nombre d'entités ou de personnages appelés « agents » dans un environnement virtuel. Ce terme est utilisé communément dans les domaines de l'informatique graphique. Il est notamment destiné aux films et jeux vidéos.

Certaines simulations visent des interactions autres que la détection de collision. Par exemple, dans le domaine de la simulation de foule, plusieurs travaux se sont intéressés à des interactions comportementales évoluées telles que la création de groupes de déplacement ou l'évitement de zones de congestion de la foule.

Ces interactions nécessitent une notion plus étendue de voisinage avec éventuellement différents niveaux. En effet, un agent ne peut pas éviter des zones de congestion s'il prend en compte uniquement les agents les plus proches de lui. Inversement, si un agent a un trop grand voisinage pour détecter de telles zones, ses considérations seront trop grandes pour effectuer de manière efficace la détection de collision.

Cette notion de voisinage, qui est au coeur du sujet de l'interaction, est directement liée à la structure choisie pour la détection de collision. Nous insisterons donc particulièrement sur cette notion et son impact dans la diversité de simulation.

1.2 Contributions

Nos travaux sont dans la continuation des travaux de T.Jund sur le suivi en temps réel de particules dans des environnements partitionnés.

T. Jund a conçu des particules élémentaires capables d'évoluer dans un environnement décomposé en cellules convexes en détectant leurs changements de cellule. Ces travaux permettent de suivre en temps réel ces particules et de détecter les collisions entre les particules et les cellules de l'environnement désignées comme « obstacle ». En revanche, ces particules n'ont pas de notion de voisinage et détectent uniquement leurs collisions avec l'environnement et non avec les autres entités mobiles.

Nous avons réalisé des enregistrements permettant d'effectuer la tâche de requêtes de proximité, fondamentale pour la détection de collision des objets entre eux, et non plus seulement entre les objets et l'environnement. Nos travaux peuvent être décomposés en trois contributions.

Enregistrement de trajectoires dans un environnement complexe : La première contribution majeure est la création d'outils étendant le suivi de particules en enregistrant et en maintenant l'enregistrement d'entités dans des cartes combinatoires. Ces entités peuvent être des objets non-ponctuels tels que des arêtes et des surfaces.

Enregistrement multirésolution et densité de la simulation : La seconde contribution majeure est l'extension de ces outils à des cartes multirésolutions. L'aspect multirésolution de notre carte d'enregistrement permet de conserver des calculs de collisions temps réel, dans des scénarios vastes présentant un grand nombre d'entités. En effet, nous adaptons la taille des cellules d'enregistrement en fonction de la densité d'objets présents dans celles-ci. Nous conservons ainsi un nombre d'entités voisines borné au fur et à mesure de la simulation. De plus, la multirésolution offre la possibilité de traiter des voisinages de différents niveaux pour des interactions plus complexes.

Ces contributions majeures permettent donc de simuler en temps réel un grand nombre d'objets pouvant être représentés par des particules, des arêtes ou des surfaces, évoluant dans une subdivision de l'espace en cellules dont le degré de subdivision est adaptatif. Ces travaux ont été validés dans le cadre d'applications utilisant des maillages surfaciques et volumiques.

Dans le cadre surfacique, notre application de simulation de foule fait évoluer des entités appelées « agents » sur des surfaces adaptatives.

Dans le cadre volumique, nos travaux ont été utilisés pour la simulation d'opération médicales utilisant des maillages décomposés en hexaèdres déformables. Cette contrainte de représentation, qui ne permet pas de garantir la convexité des cellules durant la simulation, nous a mené à étendre notre modèle de suivi de particules a des cellules non-convexes.

Enregistrement dans des cellules non-convexes : La dernière contribution, plus technique, concerne ainsi l'extension du modèle de suivi de particules de T.Jund à des cellules potentiellement non-convexes. Nous avons affaibli la contrainte de convexité à une contrainte géométrique appelée « BC » pour « bien centré », permettant l'approximation de la cellule en un ensemble de tétraèdres. Cette contrainte sera présentée en détails dans la partie 7.2. Cette contribution étend donc nos travaux d'enregistrement aux cellules BC et nous permet de traiter les cas d'hexaèdres déformés.

1.3 Plan du mémoire

Les chapitres 2, 3 et 4 présentent l'état de l'art des travaux concernant la simulation en environnement partitionné.

Le chapitre 2 introduit les différentes structure existantes pour effectuer les requêtes de proximité dans une simulation en environnement partitionné.

Le chapitre 3 détaille la structure que nous avons utilisé pour ces travaux de thèse : les cartes combinatoires.

Le chapitre 4 présente les travaux sur le suivi de particules en temps réel. Ces travaux sont notamment utiles pour optimiser les mises à jour des structures de requêtes de proximité.

Les chapitres 5, 6 et 7 présentent nos contributions en détails.

Le chapitre 5 décrit les techniques utilisées pour effectuer l'enregistrement des différents objets dans notre structure, ainsi que le maintien de cet enregistrement durant la simulation.

Le chapitre 6 décrit les techniques utilisées pour tenir à jour l'enregistrement lors du processus de multirésolution.

Le chapitre 7 présente la définition de notre contrainte BC et les extensions réalisées pour le suivi de particules dans des maillages à cellules élémentaires BC.

Les chapitres 8 et 9 présentent les applications de nos travaux dans divers domaines de la simulation.

Le chapitre 8 montre nos résultats sur une application de simulation de foule en 2D et 2D surfacique. Ces simulations comportent un grand nombre d'objets dans des environnements vastes.

Le chapitre 9 détaille nos extensions en simulations physiques 3D temps réel. Nous y décrivons notre exploration de la simulation de foule 3D ainsi qu'une application médicale en temps réel couplée au moteur d'interactions physiques SOFA. Cette application médicale comporte un environnement fortement déformable, où évoluent des outils chirurgicaux devant être modélisés précisément.

Enfin, ce mémoire se termine par une conclusion portant sur l'ensemble de nos travaux dans laquelle nous proposons quelques perspectives de recherche.

Première partie

État de l'art

Chapitre 2

Les requêtes de proximité en environnement partitionné

Sommaire

Ce chapitre présente un panel des différentes méthodes et structures utilisées dans la littérature pour effectuer des requêtes de proximité en environnement partitionné.

Nous allons distinguer les techniques selon leur approche géométrique ou topologique de la notion de voisinage. Nous verrons ainsi des structures d'arbres ou de graphes, des grilles de voisinages et des grilles d'enregistrement régulières. Nous discuterons des avantages et inconvénients de chaque technique présentée.

2.1 Introduction

Nous rappelons ici que notre objectif est d'effectuer des requêtes de proximité entre diverses entités présentes dans une scène. En d'autres termes, il s'agit de recueillir des informations de distance ou d'interpénétration entre des paires d'entités.

Ici, nous utiliserons le terme « entité » pour parler du modèle de représentation d'une entité comme présenté dans la partie 1.1.2. Ces entités sont donc placées dans une structure permettant d'obtenir des informations de proximité. Nous traiterons uniquement des structures de partitionnement de l'espace.

Un partitionnement de l'espace consiste à subdiviser l'environnement en sous-ensembles disjoints. Il est réalisé soit de manière géométrique, en subdivisant l'environnement en cellules qui vont contenir une ou plusieurs entités, soit de manière topologique, en se basant sur les entités elles-mêmes et leur adjacence.

Cette subdivision permet de tester pour la collision uniquement les entités présentes dans un voisinage défini par un critère. Par exemple, certaines techniques définissent des zones de voisinage par un critère d'adjacence géométrique, minimisant ainsi la distance entre les entités voisines. D'autres utilisent un critère topologique et récupèrent les entités les plus proches dans toutes les directions, sans limite de distance.

Ces différentes approches se mesurent en terme de nombre d'entité testées et de rapidité d'accès à ces voisins détectés. De manière générale, il est plus lent de détecter les voisins les plus importants avec précision mais cette précision diminue le nombre de tests à réaliser sur ces voisins par la suite. Il convient donc de choisir la méthode en fonction des tests que l'on désire réaliser sur les voisins et de notre définition de voisinage. Pour l'évaluation des structures nous noterons N_d le nombre d'entités dynamiques d'une simulation et N_f le nombre d'entités fixes. $N = N_d + N_f$ sera donc le nombre d'entités total à simuler. On notera également N_m le nombre moyen d'entités dans un voisinage. Ce nombre dépend de la densité des entités dans la scène pour certaines structures que nous allons présenter. Avec ces notations, nous évaluerons la complexité temporelle de construction ou de mise à jour des structures et la complexité temporelle de récupération du voisinage d'une entité donnée.

Nous ne présenterons pas ici le modèle de cartes combinatoires qui sera expliqué plus en détails dans le chapitre 3. Explorons donc les différents modèles et leur définition d'un « voisinage ».

2.2 Les arbres et les graphes

Les arbres et les graphes correspondent à une décomposition irrégulière de l'espace. Cette décomposition, centrée sur les entités présentes dans l'espace, est très utile pour représenter des environnements où les entités sont distribuées de manière non-uniforme. Cette méthode de représentation très répandue souffre d'un coût de mise à jour important dans les simulations d'environnements dynamiques. En effet, dans ces scènes, la répartition des entités est constamment en changement. Ainsi, il est souvent moins coûteux de reconstruire l'arbre partiellement ou en entier. Cette construction ou mise à jour est réalisée en $O(N * log(N))$ à chaque pas de temps [4]. Certains travaux proposent des approximations pour réduire le coût et accélérer les simulations mais nous nous focaliserons sur les simulations exactes ici.

Concernant l'accès au voisinage d'une entité donnée, on note une différence entre les arbres et les graphes. En effet, comme montré dans [9] en appliquant N_m fois la recherche du voisin le plus proche, on a un accès pour un arbre est en $O(N_m * log(N))$. Pour les graphes tels que les graphes de Voronoï, l'accès est en $O(N_m)$ car le voisinage d'une entité est directement lié à cette dernière et nécessite d'être parcouru.

Voyons désormais les détails de ces différentes techniques avec des exemples.

2.2.1 Les octrees et quadtrees

Une méthode répandue en partitionnement 3D est celle des octrees qui consiste en une subdivision de l'espace en huit sous-espaces en fonction du nombre d'entités présentes. La figure 2.1 montre un exemple d'une telle subdivision pour un modèle 3D de lapin. La racine de l'arbre est le cube contenant tout l'espace. En partant de cette racine, on crée 8 noeuds en subdivisant l'espace en 8 cubes selon les axes médians. Pour chaque noeud, on répète le processus jusqu'à ce qu'un noeud soit vide ou qu'il ait atteint un critère d'arrêt. Ce critère peut être une taille minimale fixée ou un nombre de primitives contenues fixé. Un tel noeud terminal est appelé feuille. Nous avons donc une arborescence avec des feuilles vides et des feuilles contenant une section élémentaire de la surface du lapin.

Avec cette structure, des algorithmes de parcours de l'arbre permettent d'accéder au voisinage d'une feuille.

L'équivalent en 2D de cette structure est le quadtree. Dans ce modèle, on subdivise des carrés en 4 depuis les milieux des côtés.

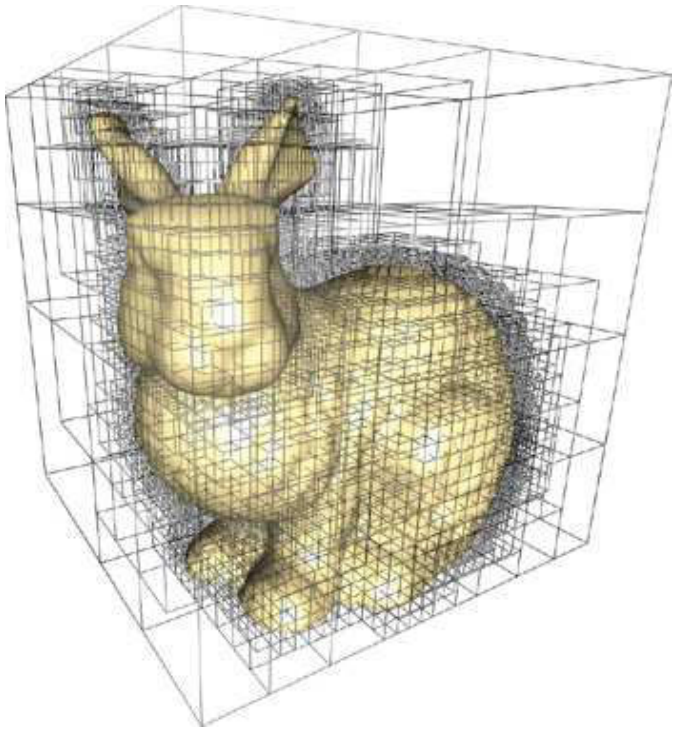

Figure 2.1 : Exemple de partitionnement par un octree.[25]

2.2.2 Binary Space Partitioning

Le Binary Space Partitioning (BSP) est une méthode de partitionnement de l'espace en cellules convexes séparées par des hyperplans. Ces cellules sont stockées dans un arbre binaire qui est rapide à construire et pour lequel les accès sont très efficaces. L'usage de cette technique est répandu, notamment dans le domaine des jeux vidéos. La figure 2.2 présente un exemple d'arbre BSP représentant une scène.

Un cas particulier de BSP souvent utilisé est le kD-tree [12]. Ce partitionnement consiste en un arbre BSP où chaque noeud est un point de dimension k. A chaque étape de la construction, on sélectionne un point ou une entité qui servira de référence pour le découpage de l'espace selon un certain axe. Le processus est réitéré sur les deux sous-espaces créés en changeant d'axe jusqu'à obtenir des feuilles ne contenant qu'un seul point ou des feuilles vides.

Lors de la sélection, on utilise souvent une recherche du point médian dans la direction de l'axe de découpage pour obtenir un arbre équilibré.

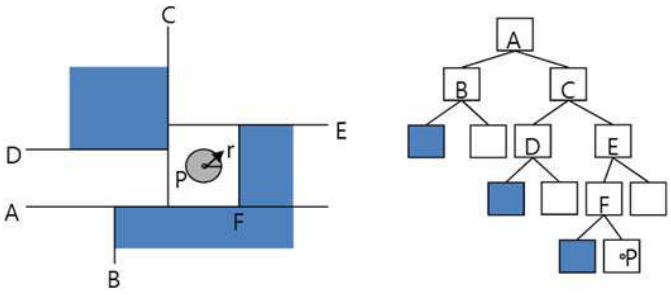

Figure 2.2 : Exemple de partitionnement par un arbre BSP.

2.2.3 Diagramme de Voronoï / Triangulation de Delaunay

Un diagramme de Voronoï est un partitionnement de l'espace à partir de points appelés « germes ». Chaque cellule du partitionnement contient un et un seul germe et représente l'ensemble des points plus proches de ce germe par rapport aux autres germes. Un diagramme de Voronoï est le dual d'une triangulation de Delaunay 2.3. Dans une telle triangulation, aucun sommet d'un triangle n'appartient au cercle circonscrit des autres triangles. De plus, chaque sommet est relié à ses plus proches voisins. Cette structure paraît donc adaptée à notre problème du fait de sa cohérence avec la notion de proximité et [20] fournit un algorithme en $O(N * log(N))$ pour sa construction, quelque soit la distance entre les entités.

Les travaux de Lamarche et Donikian [19] utilisent cette structure pour deux aspects différents de leur simulation de foule 2D.

Leur première utilisation prends les sommets de l'environnement 2D pour réaliser une triangulation. On extrait de cette triangulation une décomposition en zones circulables pour réaliser la recherche de chemin.

Leur deuxième utilisation concerne notre problème de recherche de voisins. Comme présenté dans la figure 2.4, chaque entité ponctuelle représente un sommet de la triangulation de Delaunay. Cette triangulation est ensuite filtrée avec un critère de visibilité (segments en vert). Il en résulte un graphe d'adjacence et de visibilité en noir qui permet de retrouver les voisins d'une entité directement.

Cette structure est donc efficace pour représenter des entités ponctuelles évoluant dans un environnement 2D, grâce à ses propriétés topologiques.

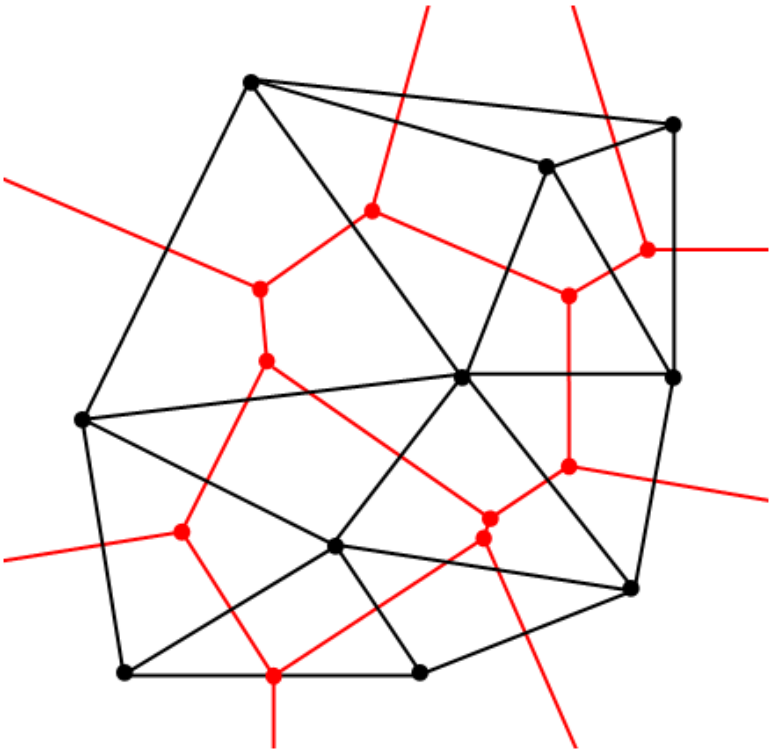

Figure 2.3 : Un diagramme de Voronoï en rouge et sa triangulation de Delaunay duale en noir.

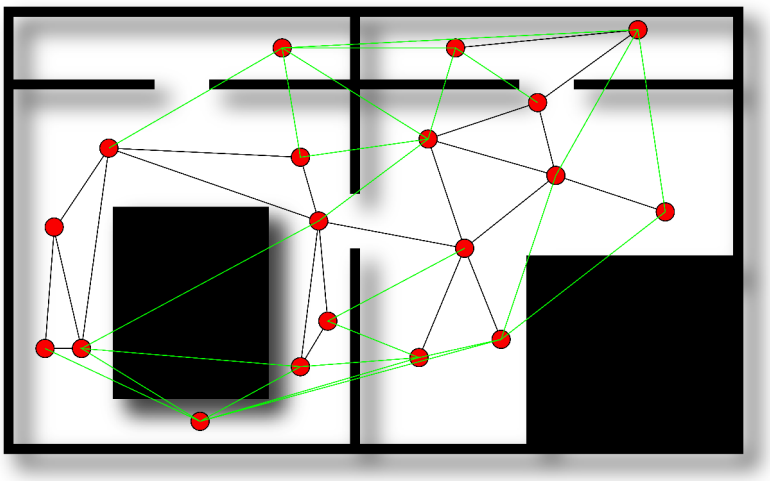

Figure 2.4 : Utilisation d'une triangulation de Delaunay filtrée avec un critère de visibilité.[19]

2.3 Les grilles de voisinage

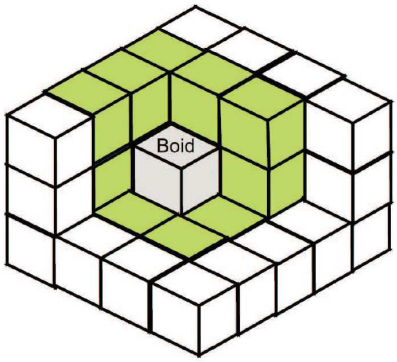

Figure 2.5 : Grille de voisinage avec, en vert, un voisinage de Moore de taille 1.[3]

Nous nous intéressons ici à une structure apparue dans [3] appelée grille de voisinage. Dans ces travaux, les auteurs se sont intéressés aux simulations 3D de foules gigantesques d'entités appelées « boids ».

La structure qu'ils utilisent, présentée sur la figure 2.5, est une grille dans laquelle chaque cellule contient une entité à simuler. Les entités sont triées dans la grille selon leur position dans l'espace à chaque pas de temps. Cette grille permet donc d'accéder facilement aux boids précédents et suivants dans toutes les directions autour d'un boid. Ce voisinage correspond à un voisinage de Moore sur cette grille.

Cette représentation a été utilisée pour réaliser des simulations de millions d'entités sur GPU. Pour atteindre de telles performances de simulations les auteurs ont fortement parallélisé les processus sur GPU avec CUDA. Le processus de tri n'est réalisé que partiellement à chaque pas de temps sur les indices pairs ou impairs de la grille.

Cette technique est donc impressionnante par ses performances, en permettant de simuler des foules gigantesques mais de manière imprécise. En effet, les techniques d'accélération peuvent amener à des erreurs de détection, notamment le fait de ne trier qu'une moitié de la grille à chaque pas de temps. De plus la notion de voisinage utilisée ici est très spécifique, et les auteurs considèrent uniquement les deux premiers voisins dans chaque direction. Cette limitation est loin de la réalité géométrique dans des situations de plusieurs zones de densité réparties.

2.4 Les grilles d'enregistrement régulières

Nous allons voir ici des décompositions régulières de l'espace à travers les grilles d'enregistrement régulières. Cette technique calque l'espace (en 3D) ou le plan (en 2D) sur une grille, dont les cases vont contenir des informations sur la zone de l'environnement correspondante. Ce type de structure non-adaptative demeure la même durant la simulation et facilite donc la représentation.

Dans de telles grilles, la notion de voisinage est souvent purement géométrique. Les voisins sont donc les entités présentes dans la même cellule de la grille mais aussi celles présentes dans les cellules voisines. Ici, selon la finesse de la grille et la visibilité que l'on désire donner aux entités, on choisira un voisinage de Moore d'ordre 1 ou 2.

Certaines optimisations telles que l'enregistrement des entités dans les cellules voisines et la détection des sorties de cellule permettent d'optimiser cette technique. En effet, dans ce genre de modèle, toutes les entités présentes dans une même cellule ont le même voisinage. À chaque pas de temps, chaque entité interroge donc la cellule dans laquelle elle se trouve pour récupérer son voisinage. Ainsi, cette information qui correspond à un parcours du voisinage de Moore choisi peut être enregistrée pour chaque cellule afin de ne pas répéter les parcours pour chaque entité présente dans une cellule. Avec cette optimisation les requêtes de proximité pour chaque entité de la simulation sont en temps constant et sont donc évaluées pour toute la simulation à $O(N_d)$.

La détection de sortie de cellule pour les entités permet de mettre à jour la structure moins souvent. Il est en effet moins coûteux de détecter les changements de cellule pour chaque entité que de les réenregistrer toutes. Si on a une grille de M cellules, la détection des changements est en $O(N_d)$ et la mise à jour de toutes les cellules est en $O(M * N_m)$. En pratique, selon la taille des cellules de la grille, très peu d'entités changent de cellule et la complexité maximale est donc rarement atteinte.

Dans [30], les auteurs utilisent une double structure. D'une part une structure hiérarchique de type quadtree pour effectuer la recherche de chemin, d'autre part une grille régulière pour les requêtes de proximité.

Ces méthodes sont de manière générale très efficaces, mais leur réussite dépend grandement de la résolution choisie pour la grille. Si les cellules de la grille sont trop grandes, trop d'entités sont considérées comme voisines et les calculs explosent. Si la grille est trop fine, les entités sont constamment en train de changer de cellule, ce qui augmente le coût des mises à jour, et diminue la visibilité des agents qui est intrinsèquement liée à cette taille de cellule.

Voyons plus en détail la méthode de la littérature la plus efficace à ce jour utilisant une grille régulière : le spatial hashing.

2.4.1 Le spatial hashing

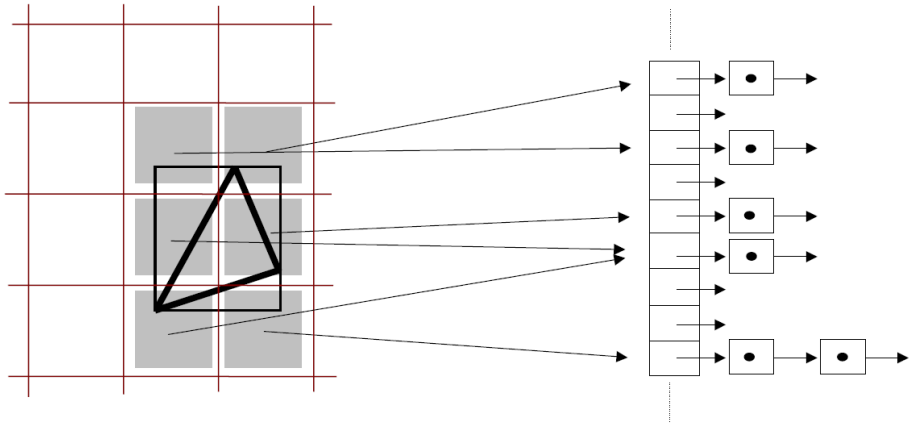

Figure 2.6 : La boîte AABB du triangle traverse des cellules liées à la table de hachage.[32]

Dans [32], Teschner et coll. présentent un algorithme de spatial hashing permettant de détecter en temps réel les collisions et auto-collisions d'objets représentés par des maillages tétraédriques. Cette méthode repose sur une r présentation simplifiée de l'environnement par une grille régulière potentiellement infinie, associée à une table de hachage.

L'algorithme se décompose en 3 étapes :

— Tout d'abord les sommets des tétraèdres sont enregistrés dans la table de hachage selon leur position et la taille L d'une cellule de la grille.
— Dans un second temps, on calcule les valeurs de hachage des cellules traversées par les boîtes AABB des tétraèdres pour récupérer les sommets enregistrés à ces indices.
— Les sommets récupérés n'appartenant pas au tétraèdre sont testés par rapport à la boîte AABB et s'ils sont à l'intérieur, on calcule leur coordonnées barycentriques dans ce tétraèdre pour la gestion de la collision.

La figure 2.6 présente la phase 2 de cet algorithme dans le cas 2D. On peut y trouver la boîte AABB d'un triangle et les cellules qu'elle recoupe grisées. Les indexes de ces cellules dans la table de hachage permettent de récupérer un ensemble de sommets.

Cette méthode s'appuie sur l'efficacité des tables de hachage en ce qui concerne leur capacité à emmagasiner des grandes quantités de données. Elle fonctionne donc très bien même dans des environnements vastes et s'applique autant en 2D qu'en 3D. Toutefois, elle souffre des mêmes problèmes de résolution que les autres techniques reposant sur des grilles régulières. De plus, la considération du voisinage par les cellules où est présente la boîte AABB d'un tétraèdre ne permet pas d'éviter les collisions mais plutôt de les gérer après qu'elles aient eu lieu.

Chapitre 3

Les cartes combinatoires

Sommaire

Nous présentons ici la structure de carte combinatoire, à mi-chemin entre les graphes et les grilles d'enregistrement, alliant les forces de ces deux structures que nous avons vues précédemment.

Nous verrons dans un premier temps la construction et les atouts de cette structure, avant de nous intéresser à ses extensions multirésolutions qui sont la base sur laquelle s'appuient nos travaux de thèse. Cette présentation, qui se veut concise, a été réalisée notamment à partir des travaux de P. Kraemer [18] pour les généralités sur les cartes, et ceux de L. Untereiner [35] pour l'aspect multirésolution.

Nous désirons utiliser cette structure comme représentation de l'environnement, pour optimiser nos requêtes de proximité. Nous conclurons donc ce chapitre par une description précise de l'utilisation de ce modèle dans notre problème de détection de collision.

3.1 Présentation des cartes combinatoires

Les cartes combinatoires sont un modèle de représentation topologique. Elles permettent de représenter un objet ou un environnement partitionné en cellules partageant des relations d'adjacence et d'incidence. Ainsi, un objet de dimension n est décomposé en un ensemble de k-cellules où $k \in [0, n]$ est la dimension de la cellule. Une 0-cellule est donc un sommet, une 1-cellule est une arête, une 2-cellule est une face, une 3-cellule est un volume, etc... Le bord d'une k-cellule est un ensemble de (k-1)-cellules avec $k \in [1, n]$, une 0-cellule n'ayant pas de bord.

Une cellule a est incidente à une cellule b si a fait partie du bord de b, ou si b fait partie du bord de a. Ces relations permettent une représentation purement topologique des objets sans aucune notion de géométrie. Le lien à la géométrie est rétabli via l'utilisation d'un modèle de plongement. Un plongement est une information quelconque associée à une cellule du partitionnement. De manière générale, les plongements sont utilisés pour associer une entité géométrique à son équivalent topologique. Par exemple, on peut associer un point de $\mathbb{R}_3$ avec ses coordonnées à une 0-cellule. Mais ces plongements peuvent également contenir des informations plus riches, comme des notions physiques sur les cellules ou encore des enregistrements d'entités plongées dans l'environnement.

Nous avons donc un modèle dans lequel on sépare franchement les notions topologique des informations géométriques. Voyons en détails la représentation de ces cartes combinatoires, et leur modèle de plongement.

3.1.1 Représentation

Les relations d'incidence entre cellules permettent une représentation par une graphe orienté appelé graphe d'incidence. Les noeuds du graphe sont les cellules, et les arcs sont les relations d'incidence entre deux cellules ayant une dimension d'écart. La figure 3.1 présente un graphe d'incidence contenant 2 faces, 6 arêtes et 5 sommets en A, et un plongement géométrique potentiel associé à ce graphe en B.

On utilise plus souvent la représentation B car elle donne une idée de la forme de l'objet modélisé. Cette représentation graphique du graphe d'incidence a évolué afin d'être plus lisible, moins ambiguë et enfin plus épurée.

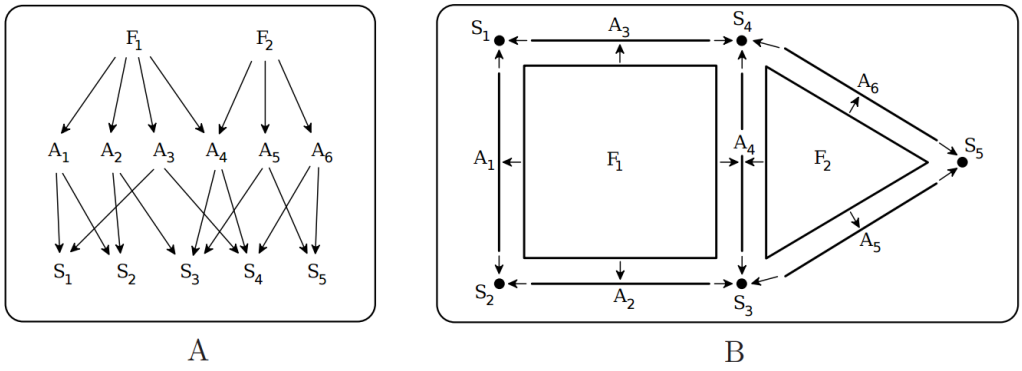

Figure 3.1 : Un graphe d'incidence et un plongement possible associé.[18]

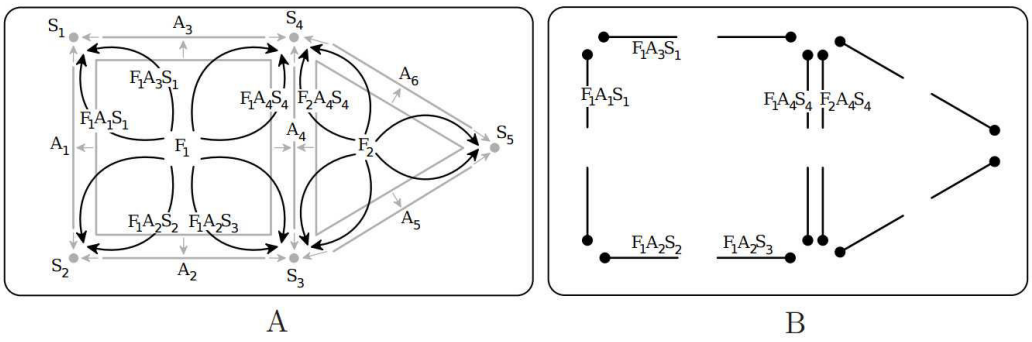

Figure 3.2 : La représentation par brins en B correspond aux chemins du graphe en A.[18]

Le modèle des cartes combinatoires généralisées ou G-cartes propose une représentation des quasi-variétés, orientables ou non, sans bord, à travers une entité unique : le « brin ». Un brin correspond à un chemin du graphe d'incidence allant d'une cellule de plus grande dimension jusqu'à une 0-cellule en suivant les relations d'incidence.

La figure 3.2 donne un exemple de représentation par brins en B avec les chemins du graphe correspondant en A. On définit ici que deux brins sont i-adjacents s'ils partagent les mêmes cellules à l'exception de celle de dimension i. Cette relation d'adjacence permet de définir une opération α_i permettant de trouver à partir d'un brin son brin i-adjacent. Il est important de noter que la relation d'i-adjacence lie les brins deux à deux, ce qui fait d'α_i une involution.

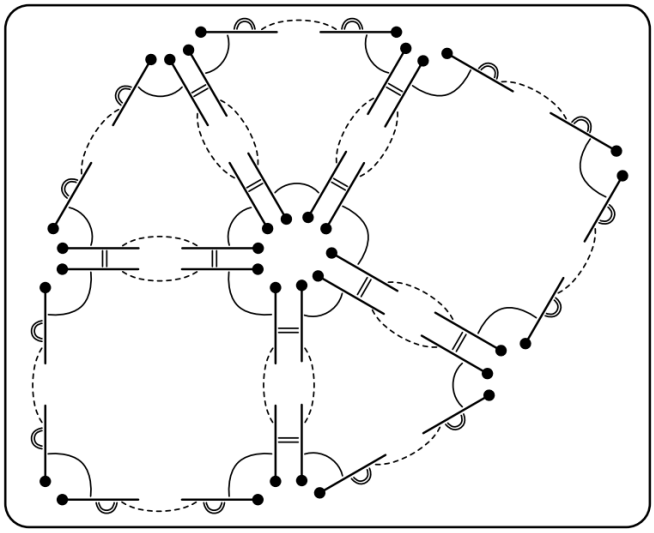

Figure 3.3 : Les relations α_i permettent de parcourir une G-carte.[18]

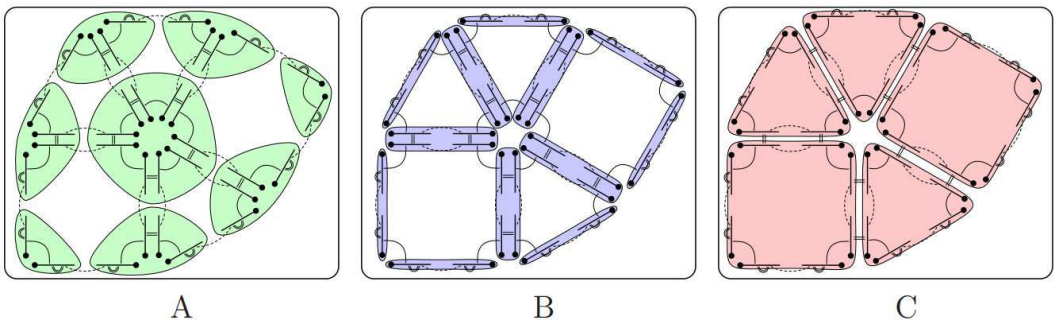

Figure 3.4 : Les k-cellules sont retrouvées en utilisant les relations α_i.[18]

La figure 3.3 représente une G-carte dont les arcs sont les relations α_0 en pointillés, α_1 en trait plein et α_2 en double trait. Ces relations permettent de parcourir intégralement la carte et de retrouver les k-cellules originelles comme on peut le constater sur la figure 3.4. En effet, on retrouve une i-cellule à partir d'un brin b en utilisant récursivement toutes les opérations α_j $\forall j \in [0...n]$ et $j \neq i$. Cet ensemble de brins qui contient une i-cellule est appelé i-orbite. Ainsi, on a en A les 0-orbites en vert contenant les sommets, en B les 1-orbites en bleu contenant les arêtes et en C les 2-orbites en rouge contenant les faces. Plus communément, on appelle une 0-orbite une orbite sommet, une 1-orbite une orbite arête, une 2-orbite une orbite face et une 3-orbite une orbite volume.

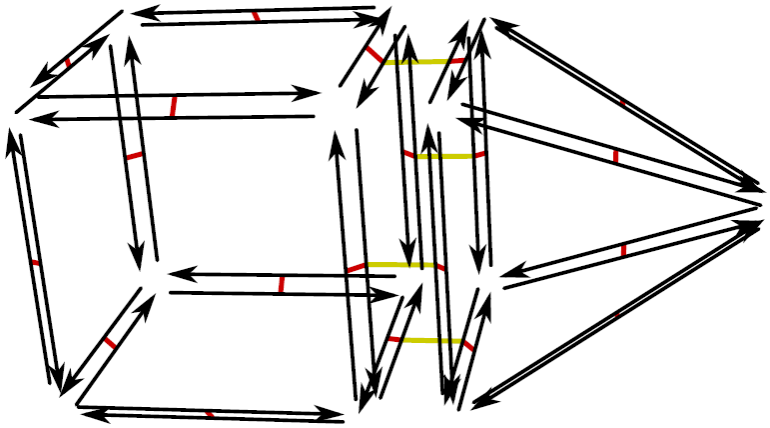

Figure 3.5 : La représentation finale d'une 3-carte.[35]

Enfin, cette représentation a été allégée en se limitant aux objets orientables sans bord dans les n-cartes combinatoires. Pour réaliser cette simplification il suffit de supprimer une des orientations présentes dans les G-cartes. Les deux orientations étant liées par la relation α_0 un nouvel opérateur de parcours est créé :

$$\forall i \in [1...n]\ \Phi_i = \alpha_i \circ \alpha_0.$$

Nous pouvons donc décrire formellement une n-carte comme le (n+1)-uplet $C = (B, \Phi_1, ..., \Phi_n)$ avec B un ensemble fini de brins, Φ_1 une permutation sur B et les Φ_i avec $2 \leq i \leq n$ sont des involutions sur B [18].

La figure 3.5 donne un exemple de la représentation d'une 3-carte. Dans cette représentation finale, les brins sont des flèches représentant leur relation Φ_1 avec le brin suivant dans une face. Les relations Φ_2 reliant les orbites faces d'une même orbite volume, et Φ_3 reliant les différents orbites volumes entres elles sont respectivement en rouge et jaune. Dans le reste de ce mémoire, c'est cette structure qui sera sous-jacente dans l'intégralité des maillages représentés. Sa proximité visuelle avec le maillage géométrique nous permet de les confondre la plupart du temps.

3.1.2 Plongements

Les plongements que nous allons présenter sont notamment la contribution des travaux de thèse de Sylvain Thery [34].

Une k-cellule étant représentée par un ensemble de brins, les plongements correspondants à cette cellule doivent être les mêmes pour tous ses brins. Un plongement a donc une dimension correspondant à la dimension de la cellule possédant les caractéristiques. De manière symétrique, étant donné qu'un brin représente $n+1$ cellules différentes, il peut posséder $n+1$ plongements différents. Un plongement correspond donc à des informations liant des brins d'une même orbite qui pointent vers la même adresse de plongement pour cette orbite. Lors de la définition de plongements, l'information associée à une adresse de plongement est modifiée.

Un exemple de plongement typique est le marquage, c'est à dire le plongement sur un booléen. Le marquage permet d'étendre la représentation des cartes combinatoires aux quasi-variétés possédant un bord, sans altérer les contraintes d'intégrité de la carte. Ainsi, les brins correspondants au bord de la carte sont « marqués » afin d'être reconnus dans les algorithmes de parcours.

De manière symétrique, on associe un brin à un plongement, ce qui permet de réaliser des parcours rapides de la carte appelés « traversées ». Si l'on souhaite parcourir les sommets de la carte, il suffit d'utiliser un traverseur qui va récupérer le brin associé à chaque plongement de type sommet. On ne parcourra ainsi que les brins traversés, c'est à dire un par orbite sommet.

D'autres outils sont disponibles en ce qui concerne les parcours des cartes et des divers plongements. Les opérations d'adjacence et d'incidence sont particulièrement optimisées avec des traverseurs d'adjacence ou d'incidence d'une k-cellule. Ainsi, on peut par exemple aisément récupérer les 3-cellules (volumes) 0-adjacents (adjacents par sommet) à un volume donné. La plateforme open-source CGoGN [6] regroupe notamment cet ensemble d'outils et permet une utilisation facilitée de ce modèle de représentation.

On a donc bien un modèle à mi-chemin entre grille d'enregistrement et graphe de voisinage. Les cellules d'enregistrement sont les faces ou les volumes de la carte selon la dimension. Le graphe de voisinage est fourni par les relations d'adjacence et d'incidence entre cellules. L'avantage que nous voulons exploiter est que la forme et la taille des cellules sont quelconques et peuvent être adaptées aux besoins de la simulation.

C'est sur cette base de représentation, d'opérations de parcours et de plongements que nous allons voir les extensions multirésolutions de ce modèle. Nous nous intéresserons ensuite de plus près à son utilisation pour notre problème d'enregistrement.

3.2 Les cartes multirésolutions

Les cartes multirésolution permettent de représenter des maillages adaptatifs. Sur un maillage adaptatif on peut réaliser les opérations de subdivision et de simplification sur les cellules localement pour en adapter la taille. Nous appellerons ces opérations des opérations de mise à l'échelle.

Les maillages adaptatifs classiques permettent d'accéder à l'état du maillage à un instant donné. Le maillage précédant une opération de mise à l'échelle est perdu et seule la version courante du maillage est accessible.

Le modèle de représentation multirésolution retient toutes les versions d'un maillage au fur et à mesure de ses changements. Chaque taille de subdivision correspond à un niveau de résolution du maillage et chaque niveau est accessible à tout moment, d'où le terme de multirésolution. Le principe du modèle des cartes combinatoires multirésolutions est d'ajouter de nouveaux brins au fur et à mesure de la modification de la carte, par des opérateurs élémentaires tels que la subdivision d'arête, la subdivision de face, etc. Ces opérateurs élémentaires garantissent que la carte résultante vérifie les contraintes d'intégrité d'une carte combinatoire.

La figure 3.6 montre l'imbrication des ensembles de brins dans une carte multirésolution. Les brins d'origine de la carte dans l'ensemble B^0 sont également présents dans l'ensemble B^1 des brins présents au niveau 1 où sont également apparus les brins oranges. Il en va de même pour le niveau 2 et les niveaux suivants.

Les brins ajoutés possèdent un niveau d'introduction et de nouvelles liaisons topologiques sont définies à chaque niveau. Les brins de niveau inférieur sont toujours présents dans un niveau supérieur. Seuls les liens topologiques, et donc les parcours, changent. Pour ne pas perdre les liens des niveaux inférieurs, on doit retenir l'information de liaison de chaque brin sur chaque niveau.

En ce qui concerne les plongements des brins, ils sont potentiellement différents à chaque niveau. Selon la représentation de la carte multirésolution, ils doivent être dupliqués ou non.

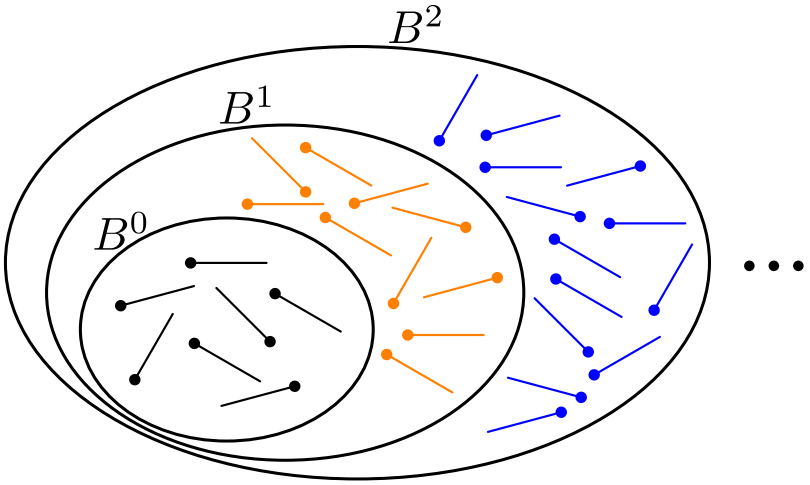

Figure 3.6 : Imbrication des ensembles de brins de différents niveaux dans une carte combinatoire multirésolution [18].

Afin de décrire les différentes représentations multirésolutions, nous allons définir qu'un niveau de résolution est dit « explicite » si tous les brins qui le composent possèdent des liens topologiques permettant de le parcourir directement.

La figure 3.7 présente une 2-carte multirésolution de niveau 2 subdivisée de manière adaptative. Les brins de niveau 0 sont noirs, les brins de niveau 1 sont violets et les brins de niveau 2 sont oranges. Le lien topologique par la relation Φ_1 est indiqué par la flèche sur la représentation des brins et le lien Φ_2 est un trait rouge. Si l'on omet les flèches vertes, on peut voir que le niveau 2 de résolution de cette carte est explicite, car tous les liens topologiques y sont directement présents. En revanche, si l'on se place au niveau 0 de la carte où seuls les brins noirs existent, aucun lien n'est présent. On peut rendre ce niveau 0 explicite pour la relation Φ_1 en y rajoutant les liens indiqués par les flèches vertes. Si l'on explicite également la relation Φ_2 au niveau 0, toutes les relations seront directement présentes dans ce niveau et il sera donc explicite.

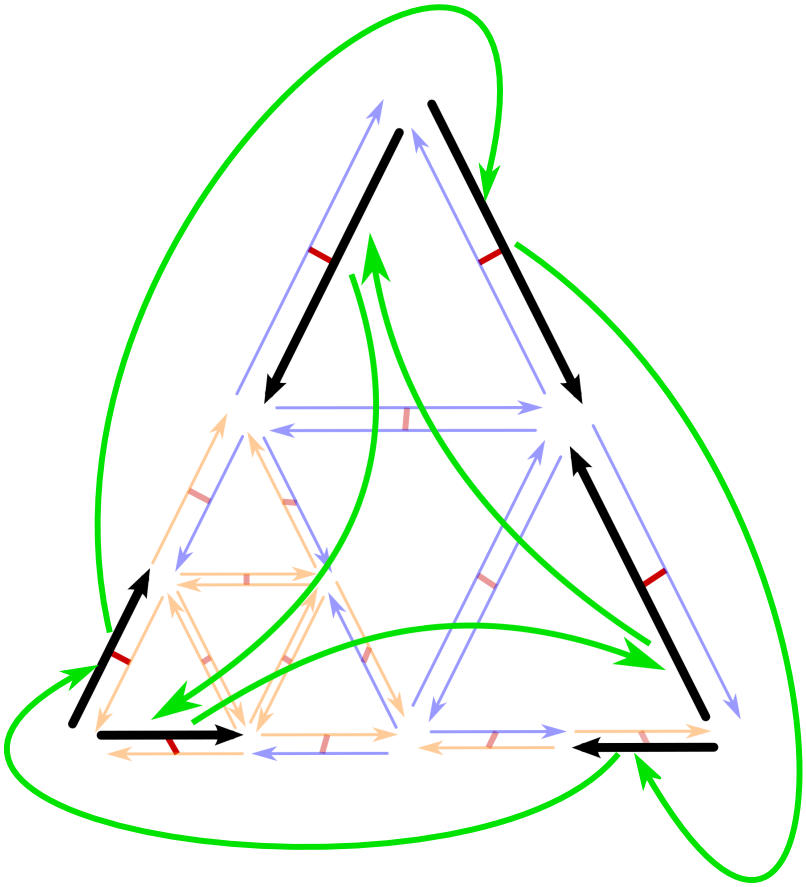

FIGURE 3.7 : Explicitation de la relation Φ_1 pour le niveau de résoluton 0 d'une 2-carte.

Deux versions de représentations ont été présentées dans [35] :
— Une première représentation dite explicite où l'on crée explicitement chaque niveau de résolution en dupliquant et en enrichissant les informations du niveau précédent.
— Une seconde version dite implicite où seul le niveau de résolution le plus fin est explicité et les autres niveaux sont reconstruits de manière implicite à la demande.

Ces représentations se distinguent notamment par les schémas de subdivision qu'elles supportent. Lorsqu'il s'agit de subdiviser des cellules, plusieurs approches sont concurrentes. De manière générale on les classifie en deux catégories appelées raffinement primal et raffinement dual.

Dans un **raffinement primal**, les anciens sommets sont conservés et les faces sont subdivisées, créant de nouveaux sommets dans les faces et/ou sur les arêtes. Dans un **raffinement dual**, les sommets et les arêtes sont éclatés pour créer des faces. La figure 3.8 présente ces deux types de schéma de subdivision appliqués au même maillage d'origine. On y trouve la subdivision primale en A avec conservation des sommets d'origine, et la subdivision duale en B éclatant les sommets. Certaines techniques sortent de ces catégories, comme le schéma de subdivision $\sqrt{3}$, qui subdivise depuis un point central en reliant ce point à chaque sommet et en faisant ensuite basculer les anciennes arêtes pour former les nouvelles faces.

Voyons désormais succinctement les possibilités offertes par ces représentations et leurs différences d'utilisation.

3.2.1 Multirésolution explicite

Une représentation multirésolution explicite correspond à l'implantation directe d'un modèle multirésolution. Chaque niveau est créé indépendamment des autres, même s'ils ont des brins en commun, l'information est dupliquée. On peut donc se placer à n'importe quel niveau de résolution et utiliser directement les relations entre les brins présentes à ce niveau pour le parcourir.

La figure 3.9 présente la gestion des relations topologiques pour deux brins $d1$ et $d2$ d'une carte multirésolution de dimension 3 possédant 3 niveaux de résolution. Les brins sont associés à un tuple liaisons/attributs qui détermine leurs voisins par les relations Φ_1, Φ_2 et Φ_3 et leurs attributs de sommet (S) et de volume(V) associés. On peut constater que le brin $d2$ a été introduit au niveau 1, et que pour chaque niveau, un brin pointe potentiellement vers un nouveau tuple de liaisons/attributs. Le brin d1 n'a pas eu de modification de ses liaisons et de ses plongements de sommet et de volume entre les niveaux 1 et 2, il pointe donc vers le même tuple.

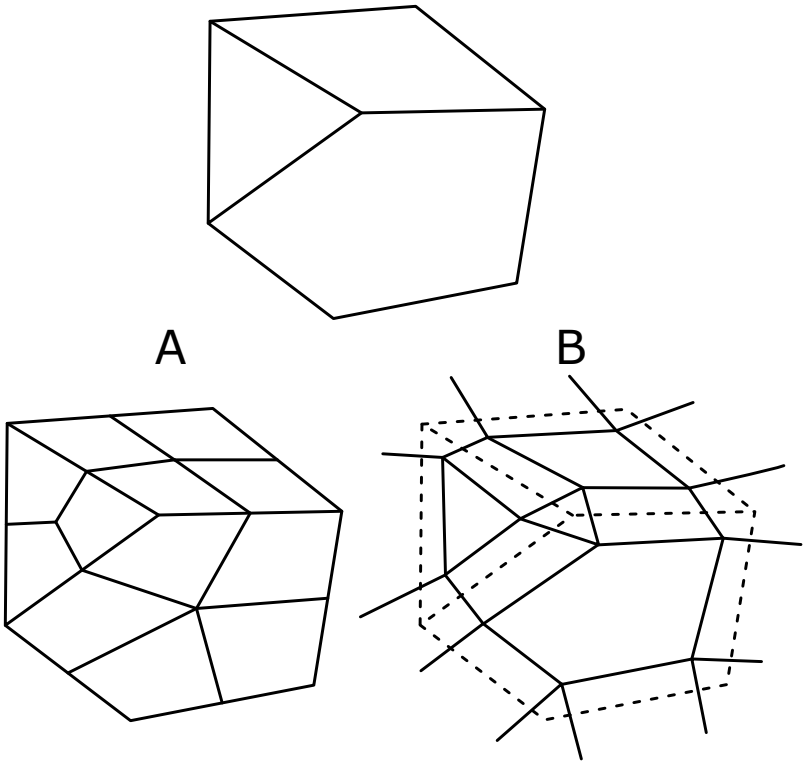

Figure 3.8 : Différents schémas de subdivision : le schéma primal en A et dual en B.

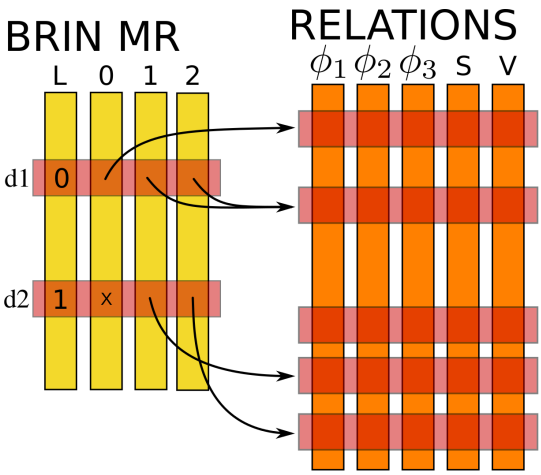

Figure 3.9 : Gestion des relations et plongements en multirésolution explicite.[35]

Cette représentation a un coût constant pour effectuer les requêtes de voisinage, quelque soit le niveau de résolution. Elle supporte tout type de raffinement de la topologie, que ce soit primal, dual, $\sqrt{3}$, etc. Son inconvénient est d'être plus coûteuse en mémoire que d'autres structures.

3.2.2 Multirésolution implicite

Dans un modèle de représentation multirésolution implicite, seul le niveau de résolution le plus fin est explicité. Ce type de représentation fonctionne uniquement sur les schémas de subdivisions primaux. Son principe est d'avoir accès uniquement au dernier niveau de résolution comme dans un modèle monorésolution afin d'éviter les redondances.

On simule ainsi de manière implicite les autres niveaux de résolution grâce à un système d'étiquetage de brins. Les étiquettes servent à retrouver les liens permettant de réaliser les parcours de plus haut niveau. La figure 3.10 présente cette technique sur une 2-carte possédant 3 niveaux de résolution. On retrouve en A le lien Φ_1^0, c'est à dire le lien Φ_1 au niveau de résolution 0, entre les brins $d1$ et $d2$ que l'on cherche à recréer de manière implicite en parcourant le niveau 2 comme illustré en B. Pour effectuer ce parcours, on suit les liens topologiques Φ_1 et Φ_2 au niveau 2 de résolution, pour retrouver le numéro d'étiquette de départ, ici 1. On suit cette étiquette jusqu'à trouver un brin de même niveau d'introduction que d1, donc un brin noir.

Comme on utilise un schéma de subdivision primal, les sommets ne changent pas et il est suffisant d'utiliser des étiquettes d'arêtes sur une 2-carte et des étiquettes d'arêtes et de faces sur une 3-carte. [35] a déterminé que le nombre d'étiquettes nécessaires est limité au nombre maximum d'arêtes adjacentes à un sommet introduit par le raffinement. Il suffit donc de 3 étiquettes différentes pour une subdivision de triangle, comme dans notre exemple, et de 2 étiquettes pour une quadrangulation de faces polygonales.

Nous avons donc une gestion des parcours grâce à ces étiquettes, voyons désormais la gestion des plongements dans de telles cartes. Comme les sommets sont fixes dans ce mode de représentation, leurs attributs peuvent être gérés aisément. Concernant les autres attributs, à savoir d'arête, de face ou de volume, la solution trouvée est de récupérer l'indice de plongement du dernier brin inséré dans l'orbite. Un brin de niveau 0 possédera toutes les informations de plongement de niveau 0. En revanche, à un niveau $k \neq 0$, pour trouver le plongement de la n-orbite il est nécessaire de parcourir cette orbite pour trouver un brin de niveau n et récupérer son indice de n-plongement.

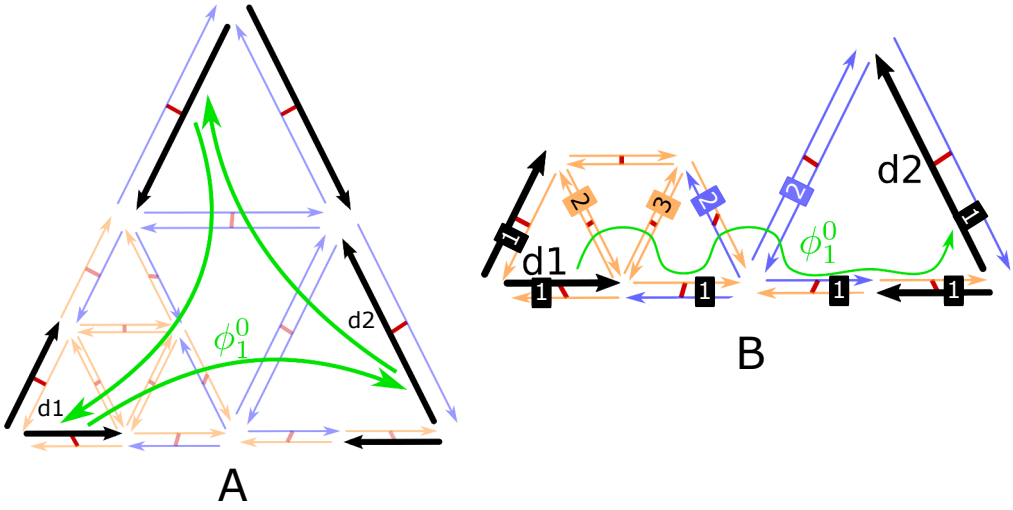

Figure 3.10 : On retrouve le parcours Φ_1 de d1 vers d2 au niveau 0 grâce à un parcours en suivant de manière implicite les bonnes étiquettes au niveau 2.[35]

[36] réalise la comparaison de performance entre les modèles explicites et implicites. Sa conclusion est que le modèle implicite coûte 15% de plus en terme de temps de parcours des différents niveaux de résolution, et coûte 33% et 14% de moins d'espace mémoire respectivement pour les cas 2D et 3D.

Cette représentation implicite a donc l'avantage d'avoir un coût en mémoire très faible pour une structure représentant une connectivité complète de chaque niveau de la hiérarchie. En revanche, elle est restreinte aux subdivisions primales et crée un surcoût des requêtes de voisinage à haut niveau.

3.3 Utilisation pour les requêtes de proximité

L'idée que nous défendons est d'utiliser une carte combinatoire pour représenter l'environnement de la simulation. Comme dans les grilles d'enregistrement classiques, nous désirons enregistrer des entités dans des cellules de la carte et comme dans les techniques utilisant des graphes, nous désirons utiliser les relations d'adjacence des cellules pour optimiser les requêtes de proximité.

Les enregistrements dans une carte combinatoire correspondent à des plongements complexes. Comme nous l'avons évoqués au début de cette section, les plongements permettent de lier une information quelconque à une k-cellule de la décomposition.

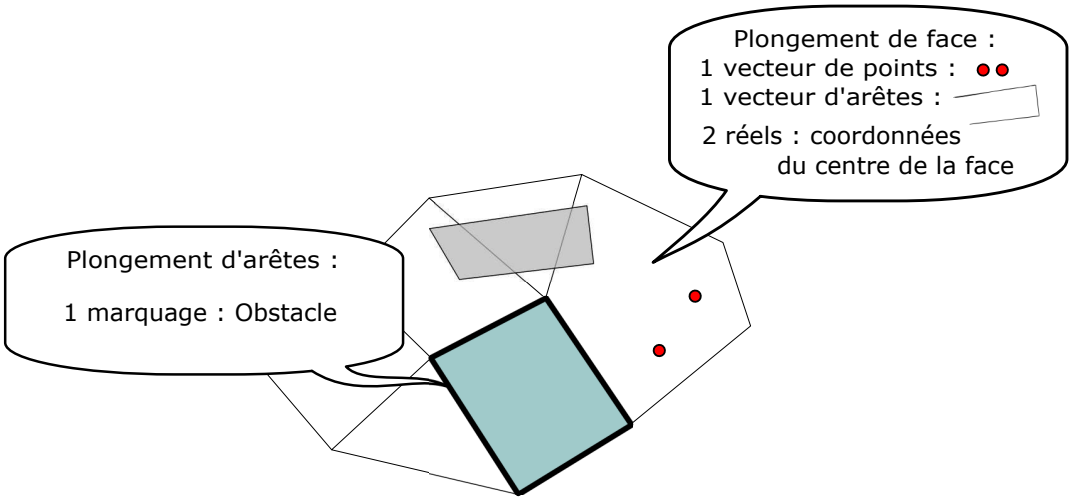

Figure 3.11 : Les enregistrements des entités sont une information de plongement parmi d'autres.

On peut représenter l'environnement par une carte combinatoire et se servir des plongements pour enregistrer les entités présentes dans une simulation. La figure 3.11 présente un exemple de simulation de foule avec diverses informations plongées dans l'environnement. Parmi ces informations, les entités de la simulation, ici les points et les arêtes, ont été plongés dans des vecteurs sur les faces de la carte représentant l'environnement. On peut également trouver d'autres informations, par exemple, des arêtes de la carte marquées comme obstacle ou encore des informations géométriques comme le centre de gravité d'une face.

Ces informations de plongement, associées aux opérations de parcours, permettent de recueillir rapidement l'ensemble des entités enregistrées dans le voisinage d'une cellule de l'environnement. Ici, la notion de voisinage est topologique et on va définir qu'une « cellule » de la carte est l'orbite de plus grande dimension dans la subdivision. Ainsi, une cellule d'une 2-carte sera une 2-orbite correspondant à une face, et une cellule d'une 3-carte sera une 3-orbite correspondant à un volume. Avec cette définition de cellule, on peut définir le voisinage d'une cellule :

Définition du voisinage d'une cellule C : :
Le voisinage d'une cellule C est l'ensemble des cellules qui lui sont adjacentes par sommet, c'est à dire 0-adjacente.

Cette définition correspond à un voisinage topologique de distance 1 par la 0-adjacence.

Nous avons donc une définition de voisinage et un modèle basé sur une représentation topologique de l'environnement, très efficace pour effectuer des requêtes de proximité. Ce modèle permet de représenter n'importe quelle quasi-variété, ce qui nous donne une grande liberté de choix pour notre environnement de simulation. Son modèle de plongement nous permet d'associer des information potentiellement de très haut niveau dans chaque cellule et même chaque orbite, grâce à un accès similaire pour toutes les entités du maillage (sommets, arêtes, faces et volumes). Ses performances et sa stabilité ont été prouvées mathématiquement dans divers travaux. Nous avons donc choisi ce modèle comme base de nos travaux de thèse.

Durant notre exploration des méthodes basées sur des grilles d'enregistrement de la section 2.4, nous avons déterminé que l'efficacité de la mise à jour des enregistrements dépend de la taille des cellules de la subdivision. Ainsi, dans les situations de forte densité d'entités, il convient de réduire le voisinage d'une entité et donc la taille des cellules pour faciliter ses calculs de collision. Dans des situations de faible densité d'entités, de grande cellules permettent de limiter les changements de cellule des entités et donc de ne pas avoir à mettre à jour les enregistrements. Nous avons donc choisi d'utiliser un modèle adaptatif pour réaliser nos simulations. La résolution du maillage est affinée selon les zones de densité d'entités plongées dans la carte.

La représentation implicite multirésolution semble plus adaptée à notre objectif et nous avons donc choisi de travailler exclusivement sur ce modèle. Pour le reste de ce mémoire, il sera donc question de modèle de multirésolution implicite. En effet, dans nos simulations, les recherches de voisinages se font toujours au niveau le plus fin de résolution, et on accède très rarement aux niveaux supérieurs. De plus, les schémas de subdivision primaux sont plus avantageux pour suivre nos enregistrements. En 2D, avec un schéma primal, lorsqu'une face est subdivisée, la forme des faces 0-adjacentes n'est pas modifiée. Cette propriété permet de changer uniquement les enregistrements de la face en question. Ce système de multirésolution est donc utilisé pour limiter la taille des voisinages et optimiser notre simulation.

Enfin, la complexité de ce modèle est la même que pour les grilles d'enregistrement régulières pour nos critères de comparaison, à savoir :

- $O(N_d)$ pour la mise à jour de la structure avec une détection des changements de cellules.
- $O(N_d)$ pour la récupération du voisinage des entités mobiles soit $O(1)$ pour chaque entité.

Avant de voir dans les parties suivantes nos travaux autour de ce modèle, nous allons nous intéresser à la détection du changement de cellules des entités avec un système de suivi de particules.

Chapitre 4

Suivi de particules

Sommaire

Ce chapitre présente de manière synthétique les travaux de thèse de Thomas Jund [15] sur le suivi de particules dans un environnement partitionné en cellules convexes. Nous présenterons ici le contexte d'application ainsi que les algorithmes de suivi de particules dans un environnement 3D.

Ces travaux, en association avec le modèle de cartes combinatoires multi-résolution, constituent la base sur laquelle s'établissent nos travaux de thèse et sont donc essentiels à la compréhension des chapitres suivants.

4.1 Contexte

Dans le contexte des simulations physiques, une particule est un point doté d'une vitesse et de propriétés physiques. Son déplacement est déterminé par le système d'interaction d'une simulation, qu'il soit automatique, avec des lois physiques et des objectifs, ou qu'il fasse intervenir un utilisateur. Ce système fournit une vitesse à appliquer à la particule pour le prochain pas de temps de la simulation, quelque soit la configuration de l'environnement alentour.

Les travaux de T. Jund proposent de suivre en temps réel des particules dans un environnement partitionné en cellules convexes et de détecter les collisions entre ces particules et les frontières de l'environnement.

L'efficacité du modèle que nous allons décrire est basée sur le fait que l'échantillonnage temporel est suffisamment important dans la plupart des simulations pour considérer des déplacements de tailles faibles. Ceci a pour conséquence de considérer la trajectoire entre deux positions successives comme linéaire et que le nombre de cellules du partitionnement traversées est réduit. De plus, le partitionnement permet d'exploiter la cohérence spatiale pour réaliser uniquement des tests locaux. Ainsi, cette technique est indépendante de la taille de l'environnement utilisé.

En partant d'un ensemble de prédicats géométriques et d'un modèle de particules adapté aux cartes combinatoires, nous définissons un ensemble d'algorithmes de navigation d'une particule dans une carte représentant l'environnement. Voyons plus en détails ces algorithmes avant de conclure sur ces travaux.

4.2 Algorithmes

En dimension 3, l'environnement est partitionné en volumes, faces, arêtes et sommets. La figure 4.1 représente les principaux prédicats géométriques utilisés en 3D.

En a) on définit les prédicats « à gauche », « à droite » et « sur » par rapport à 3 points non-colinéaire. Ici, p_8, p_9 et p_{10} sont « à gauche » de (p_1, p_2, p_3) et p_{10} est « à gauche » de (p_1, p_2, p_4), p_5 est « à droite » de (p_1, p_2, p_3) et p_5, p_6 et p_8 sont « à droite » de (p_1, p_2, p_4), p_6 est « sur » (p_1, p_2, p_3) et p_9 est « sur » (p_1, p_2, p_4). Ces prédicats dépendent de l'orientation et donc de l'ordre dans lequel on prend les 3 points.

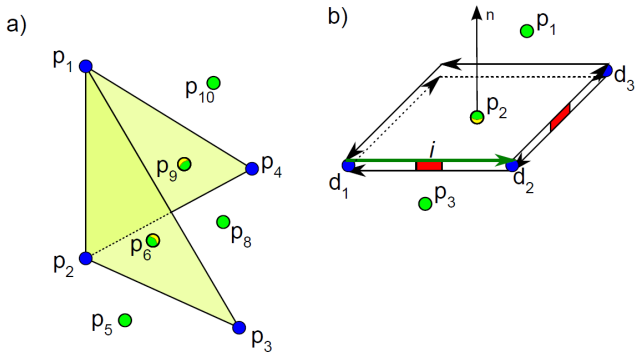

Figure 4.1 : Prédicats géométriques en 3D en a) « à gauche », « à droite » et « sur » et en b) « en dessus », « en dessous », et « sur ».[15]

En b) on définit les prédicats « au dessus », « au dessous », et « sur » par rapport à au plan formé par la face à laquelle appartient le brin i. Ici, p_1 est « au dessus » du plan, p_3 est « au dessous » du plan et p_2 est « sur » le plan. Ces prédicats dépendent de l'orientation et on se réfère donc à la normale de la face qui est orientée par les brins.

On définit également les 4 états possibles pour une particule :

— L'état « volume », où la particule est strictement à l'intérieur du volume défini par la cellule.
— L'état « face », où la particule est sur une face de la cellule.
— L'état « arête », où la particule est sur une arête de la cellule.
— L'état « sommet », où la particule est sur un sommet de la cellule.

Ces états permettent de faciliter les transitions entre cellules et limiter le nombre de tests. Afin de les atteindre en pratique, il convient de donner une épaisseur ϵ aux faces, arêtes et sommets. Nous considérons donc que ces états sont atteints si la particule est à une distance inférieure à ϵ de l'orbite en question.

En supplément, une particule contient une référence vers un brin de la cellule dans laquelle elle se trouve et qu'elle vise. Ce brin, appelé brin « de prédiction », permet à une particule de connaître à tout instant dans quelle cellule de l'environnement elle se situe, et donc d'optimiser son orientation. En 2D dans l'état face, ce brin et la position p de la particule forment un triangle dit triangle de prédiction car il correspond à l'orientation actuelle de l'agent.

Cette zone de prédiction permet donc d'avoir une idée de la zone de déplacement à venir d'une particule. En 3D, l'équivalent est un tétraèdre de prédiction dans l'état volume formé par p, le brin visé et un point de la face correspondant à ce brin.

Nous définissons la configuration S d'une particule P avant un déplacement comme le triplet $S(p, b, e)$ où p est la position dans l'espace, b son brin de prédiction et e son état dans la cellule.

L'objectif des algorithmes que nous allons décrire ici est de partir d'une configuration $S(p, b, e)$ d'une particule et, étant donnée une position à atteindre p_2, de réaliser l'orientation de la particule et son déplacement pour trouver la configuration $S(p', b', e')$ finale.

Notons que la particule arrive à la position p' qui n'est pas forcément la position objectif p_2 à cause des contraintes de l'environnement. Cette configuration est bien une configuration finale, la procédure peut passer par une série de configurations intermédiaire appelée « cheminement » avant d'y parvenir.

Les algorithmes que nous allons présenter permettent de faire le lien entre les différentes configurations possibles en terme d'états d'une particule.

Quatre fonctions de déplacement sont donc définies :

— Depuis l'état volume : OrientVolume(S).
— Depuis l'état face : OrientFace(S).
— Depuis l'état arête : OrientArete(S).
— Depuis l'état sommet : OrientSommet(S).

Pour la présentation de ces algorithmes, nous noterons p_1 la position de départ de la particule, i son brin de prédiction de départ, p_2 la position objectif, p'_1 la position atteinte et Δ le vecteur déplacement objectif (p_1, p_2). Les symboles Φ_k correspondent aux liens topologiques de parcours entre k-orbites présentés dans le chapitre 3. Voyons donc ces procédures par ordre décroissant de dimension de l'état en question.

4.2.1 Déplacement depuis l'état volume

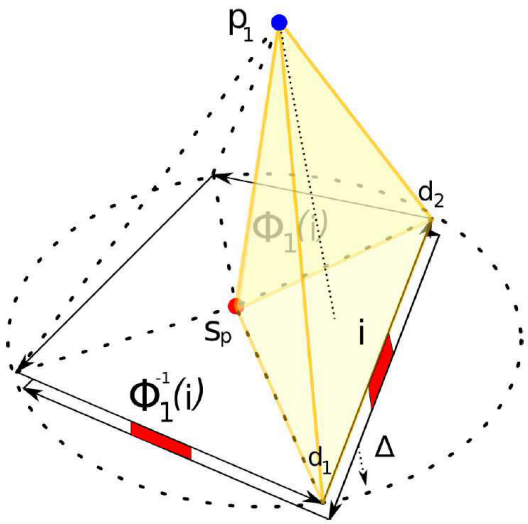

Figure 4.2 : Orientation depuis un volume.[15]

Comme pour chaque algorithme de déplacement que nous allons présenter, l'objectif est de trouver le bon brin de prédiction et la nouvelle position avant de passer à la procédure suivante ou de s'arrêter. Nous présentons ici l'algorithme de déplacement depuis un état volume. La particule se trouve donc strictement à l'intérieur du volume définit par une cellule k dont elle vise actuellement le brin i.

Ce brin définit, avec la position p_1 de la particule et un point s_p de la face associée à i, un tétraèdre de prédiction. Ce tétraèdre présenté sur la figure 4.2 en jaune nous fournit 4 plans de vérification de nos prédicats géométriques : (p_1, s_p, d_1) noté R, (p_1, d_2, s_p) noté L, (p_1, d_1, d_2) noté T et (s_p, d_2, d_1) noté B avec d_1 et d_2 les sommets de l'arête définie par i.

L'algorithme débute par une phase d'orientation où l'on cherche quel brin k défini le tétraèdre de prédiction intersecté par Δ.

Dans un premier temps, on tourne sur la face de i grâce aux relations Φ_1 et Φ_{-1} afin de trouver le dièdre formé par les plans R et L contenant p_2. On optimise la recherche en tournant dans un sens ou dans l'autre selon les résultats des prédicats « à gauche », « à droite » et « sur ».

Ensuite, on vérifie que p_2 est bien « en dessous » de ou « sur » T. Si ce n'est pas le cas, on change de face par l'opération Φ_2, on met donc à jour s_p

et on recommence l'orientation en tournant sur la face.
Si c'est le cas, on a trouvé le cône défini par R, L, T qui contient Δ. On peut donc réaliser la deuxième phase de l'algorithme : le déplacement.

On teste si $[p_1, p_2]$ coupe B, donc si p_2 est « en dessous » de ou « sur » B. S'il y a intersection, le déplacement passe par la face, l'arête ou le sommet. p'_1 est donc placé sur l'intersection et l'état de sortie e' est modifié en conséquence avant d'appeler la fonction correspondante à cet état pour continuer le déplacement. Sans intersection, p_2 est à l'intérieur de la cellule et on met fin au déplacement avec $p'_1 = p_2$ et $e' =$« volume ».

L'algorithme termine donc le déplacement si l'objectif est dans le volume en visant le brin définissant le tétraèdre qui contient cet objectif. S'il ne termine pas le déplacement, le prochain algorithme de déplacement est appelé selon l'état dans lequel on a conclu l'orientation. Cette situation de terminaison est vérifiée pour tous les algorithmes suivants.

4.2.2 Déplacement depuis l'état face

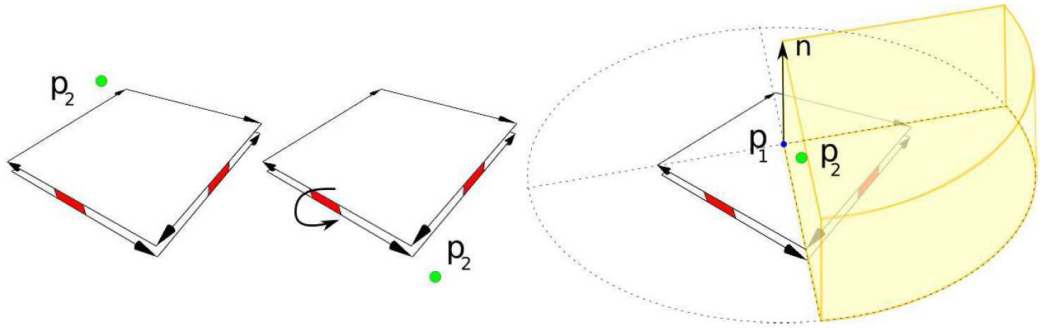

Figure 4.3 : Orientation depuis une face.[15]

Pour le déplacement depuis une face, on a p_1 « sur » la face définie par i. On veut s'orienter comme précédemment en tournant sur les brins de cette face, il nous faut donc un point en dehors de la face pour créer les plans de comparaisons. Le point choisi est p_n qui est la translation de p_1 le long de la normale à la face. On définit donc deux plans de comparaisons : $R = (p_1, d_1, p_n)$ et $L = (p_1, d_2, p_n)$, présentés à droite de la figure 4.3.

On voit sur cette même figure que l'on commence par tester si p_2 est « en dessus » ou « en dessous » de la face. Si c'est le cas, on appelle directement la fonction de déplacement correspondante en se plaçant dans le bon volume si nécessaire via l'opération Φ_3. Si ce n'est pas le cas, p_2 est « sur » la face et on tourne donc sur cette dernière avec les opérations Φ_1 et Φ_{-1} pour trouver le secteur angulaire défini par R et L qui contient Δ.

Une fois le bon secteur trouvé, on teste si $[p_1, p_2]$ intersecte i. S'il y a intersection, le déplacement passe soit par l'arête soit par le sommet. p'_1 est donc placé sur l'intersection et l'état de sortie e' est modifié en conséquence avant d'appeler la fonction correspondante à cet état pour continuer le déplacement. Sans intersection, p_2 est à l'intérieur de la face et on met fin au déplacement avec $p'_1 = p_2$ et $e' =$« face ».

4.2.3 Déplacement depuis l'état arête

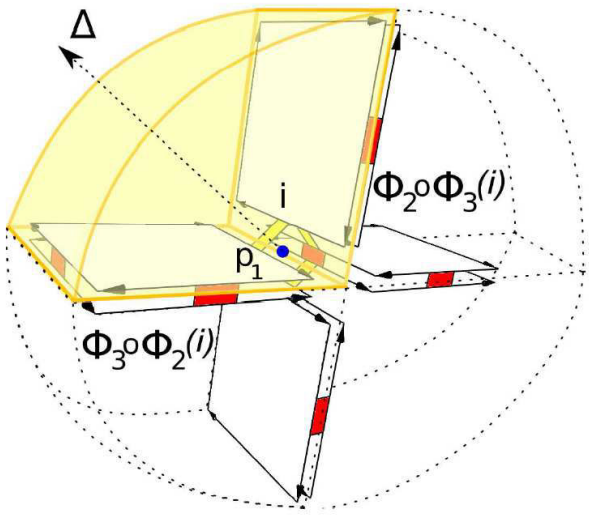

FIGURE 4.4 : Orientation depuis une arête.[15]

Le déplacement depuis une arête consiste en une recherche de secteur angulaire défini par les plans $R = (i, \Phi_1(i))$ et $L = (\Phi_2(i), \Phi_1 \circ \Phi_2(i))$ comme illustré sur la figure 4.4.

On recherche donc le secteur angulaire défini par R et L qui contient Δ. Pour cela on va tourner autour de l'arête par les opérations $\Phi_2 \circ \Phi_3$ ou $\Phi_3 \circ \Phi_2$ selon les résultats d'orientation par rapport à R et L.

Une fois le secteur trouvé, on vérifie si p_2 est sur la droite définie par i, sur R ou L ou dans le volume défini par ce secteur.

Si le point n'est pas sur la droite, on appelle l'orientation depuis un volume ou depuis une face en visant un brin de la bonne face.

Si le point est sur la droite, on vérifie s'il est dans $]d_1,d_2[$ et donc sur l'arête. Si le point est bien sur l'arête, le déplacement est terminé, sinon on appelle l'orientation depuis un sommet en se plaçant sur d_1 ou d_2 selon nos tests précédents.

4.2.4 Déplacement depuis l'état sommet

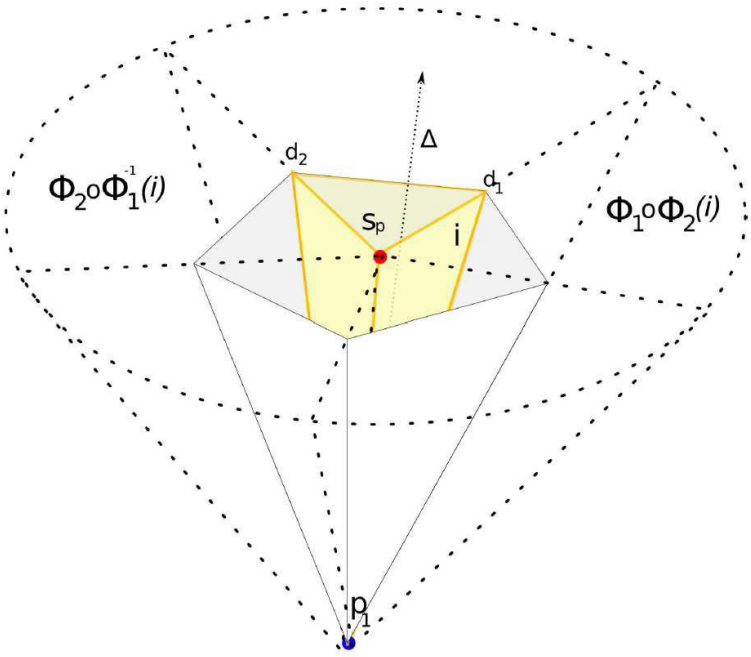

Figure 4.5 : Orientation depuis un sommet.[15]

Pour l'orientation depuis un sommet S, on va considérer les faces $f_1, ..., f_n$ adjacentes à ce sommet appartenant au même volume V_1. Ces faces forment une ombrelle ou un cône depuis S comme on peut le constater sur la figure 4.5. On exploite la convexité des volumes qui rend cette ombrelle convexe. Un point de référence s_p est placé sur la médiatrice du cône et on note $R{=}(i, s_p)$ et $L{=}(\Phi_1(i), s_p)$.

On commence par tester si p_2 est sur le sommet (donc qu'il n'y a pas de déplacement à ϵ près). C'est en effet le seul cas terminal pour le déplacement depuis un sommet.

On recherche ensuite dans quel secteur angulaire défini par R et L se trouve Δ. On change de secteur angulaire par les relations $\Phi_1 \circ \Phi_2$ ou $\Phi_2 \circ \Phi_{-1}$ selon le sens de rotation.

Une fois ce secteur trouvé, on compare p_2 avec le plan défini par la face de i soit $F{=}(i, \Phi_1(i))$. Si p_2 est « au dessus » de ce plan, on change de volume par l'opération Φ_3 et on recommence l'orientation. Sinon, Δ intersecte le tétraèdre (R, L, F) et on peut donc continuer le déplacement depuis l'état arête, face ou volume selon nos tests d'orientation.

4.3 Conclusion

Nous avons vu dans ce chapitre les travaux de T.Jund sur le déplacement de particules en environnement partitionné en cellules convexes. Des algorithmes atomiques sont fournis, décomposant le déplacement selon l'état de la particule défini par la dimension de l'orbite traversée. La figure 4.6 présente l'enchaînement de ces algorithmes de déplacement. Ils ont été conçus et optimisé pour être utilisés dans le paradigme des cartes combinatoires.

Ces travaux constituent le socle de nos travaux de thèse. Ils nous permettent de réaliser efficacement la détection des changements de cellules de particules plongées dans des cartes combinatoires. Ils présentent également une première approche pour une extension multirésolution qui a été exploitée dans [16].

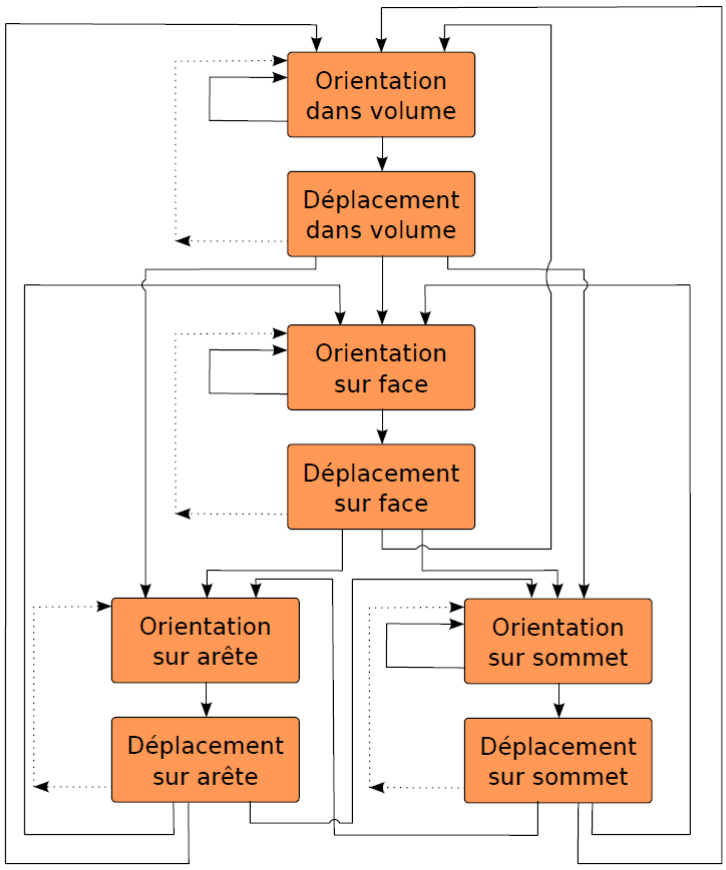

FIGURE 4.6 : Graphe de l'enchaînement des différents algorithmes de déplacement de particule.[15]

Deuxième partie

Contributions

Chapitre 5

Enregistrement des entités

Sommaire

Ce chapitre présente les diverses techniques utilisées pour réaliser l'enregistrement de particules, d'arêtes et de surfaces dans des maillages 2D et 3D pour la simulation temps réel.

Nous présenterons ici les diverses entités utilisées en détail, en expliquant nos choix de représentation. Les entités en déplacement évoluent dans un environnement associé à une carte combinatoire. Chaque entité est autonome et doit pouvoir accéder à l'ensemble de ses voisins pour effectuer la tâche de détection de collision. L'environnement est décomposé en cellules convexes qui sont le support des enregistrements. La plupart du temps, nous ne distinguerons pas les cas 2D et 3D. Ainsi, le terme « cellule » sera utilisé en 2D pour une orbite face et en 3D pour une orbite volume de la carte.

5.1 Enregistrement de particules

La particule est l'élément de base utilisé dans les autres représentations. Elle renferme une position dans l'espace et un brin visé de la carte. Ce brin visé permet à la particule de s'orienter dans l'espace lors de ses déplacement via les algorithmes vus précédemment (Chapitre 4).

Nous utilisons les atouts des cartes combinatoires pour avoir un accès direct aux voisinages et avoir la possibilité de stocker des attributs dans les cellules de l'environnement. Ainsi, chaque cellule possède un attribut de type vecteur $V_{PartPresent}$ où l'on stocke les particules qui y sont présentes. Les particules ayant accès à la cellule qu'elles visent, elles peuvent donc accéder aux autres particules présentes dans la même cellule qu'elles, c'est à dire leurs voisines.

Dans un environnement décomposé en cellules, le voisinage ne se restreint pas aux particules présentes dans la même cellule. Il faut également considérer les particules présentes dans les cellules proches car deux particules peuvent être très proches sans être dans la même cellule.

Il est relativement rapide de faire le tour du voisinage de la cellule où se trouve la particule afin de récupérer les particules présentes dans ce voisinage. Mais lorsque l'on simule des milliers de particules, il faut que chaque opération soit optimisée au maximum. Ainsi nous utilisons la notion de voisinage topologique pour accélérer les opérations.

Si l'on considère que toutes les particules présentes dans une même cellule ont les mêmes voisins, il est plus efficace de stocker directement dans chaque cellule un vecteur de particules voisines $V_{PartVoisin}$. Ainsi, chaque particule accède en temps constant à la liste des particules présentes dans sa cellule et dans les cellules voisines.

On peut donc formaliser les notions sur l'enregistrement de particules en notant C l'ensemble des cellules de la carte, P l'ensemble des particules et $Voisin(x)$ le voisinage d'une cellule x :

Particule bien orientée :
Une particule p est dite bien orientée BO si son brin de prédiction visé b fait partie de l'orbite de la cellule dans laquelle elle se situe.
$\forall p(b) \in P$, $BO(p) \Leftrightarrow \exists a \in C$, *p inside a and* $b \in a$.

Cellules voisines :
Deux cellules sont voisines si elles sont 0-adjacentes (adjacentes par sommet).
$\forall$ (a,b) $\in C/$ $a \neq b$, $a\ Adj_0\ b \Leftrightarrow b \in Voisin(a)$ et $a \in Voisin(b)$.

Particule bien enregistrée :
Une particule p est dite bien enregistrée si elle est bien orientée, qu' elle appartient au vecteur de particules présentes $V_{PartPresent}$ de l'unique cellule a dans laquelle elle se trouve, et si elle appartient aux vecteurs de particules voisines $V_{PartVoisin}$ de chaque cellule voisine de a.
1. $\forall a \in C, \forall p,\ p \in a \Leftrightarrow p \in V_{PartPresent}(a)$.
2. $\forall a \in C, \forall p,\ p \in a \Leftrightarrow \forall b \in C/\ b \neq a,\ p \notin b$.
3. $\forall a, b \in C, \forall p,\ p \in V_{PartPresent}(a) \wedge b \in Vois_a \Rightarrow p \in V_{PartVoisin}(b)$.
4. $BO(p)$.

Enregistrement correct :
Un enregistrement de C est dit correct si toute entité présente dans C est bien enregistrée.

La figure 5.1 présente un exemple d'enregistrement en 2D. Elle montre la notion de voisinage topologique utilisée, soit une distance topologique de 1 que nous désignerons comme le « One-Ring » (en bleu clair) d'une cellule (en vert clair). On peut noter l'importance de la considération de ce One-Ring pour la particule jaune qui est très proche de la particule verte mais pas dans la même cellule. Les vecteurs d'enregistrements de deux des cellules sont affichés pour montrer la portée de l'enregistrement.

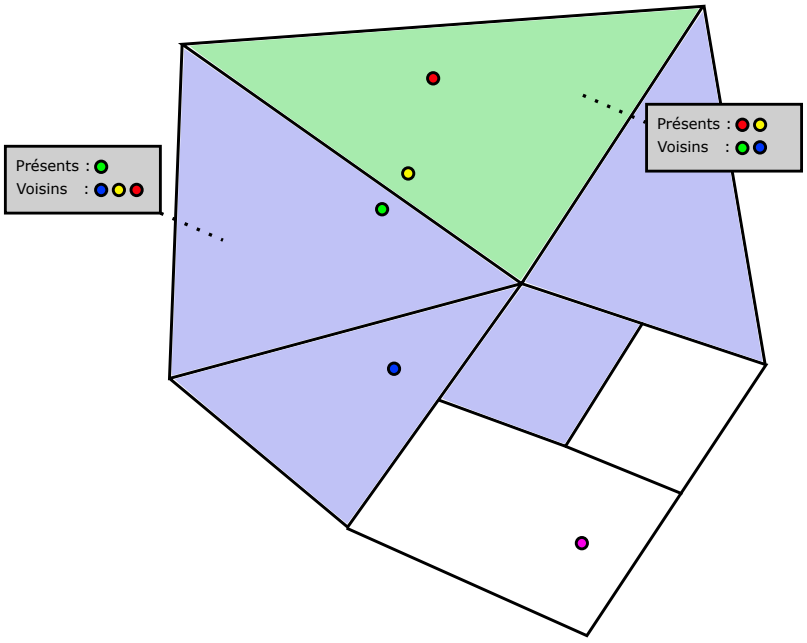

Figure 5.1 : Exemple d'enregistrement en 2D

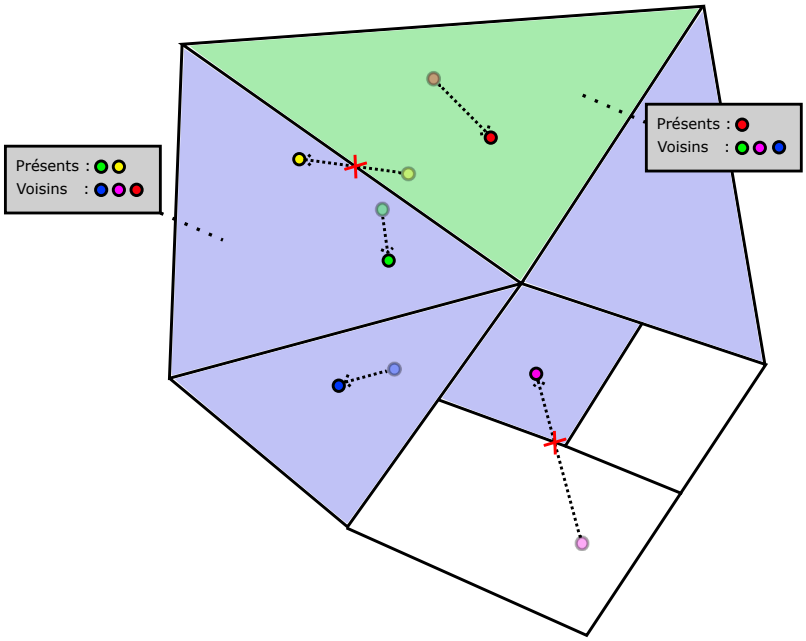

FIGURE 5.2 : Exemple de mise à jour d'enregistrement en 2D

5.2 Mise à jour de l'enregistrement des particules

Lors de la simulation, les particules sont amenées à bouger et potentiellement à changer de cellule. Ces particules pouvant être très nombreuses, il convient de ne pas ré-effectuer la procédure d'enregistrement à chaque pas de temps pour chaque particule, si l'on veut conserver l'aspect temps réel.

Nous exploitons les atouts du modèle de suivi de particules présenté précédemment afin de détecter efficacement les changements de cellule. Ainsi, les particules qui changent de cellule sont les seules à être mises à jour. Même si certaines cellules du voisinage demeurent les mêmes lors de la mise à jour, il s'avère être plus coûteux de détecter les similarités entre deux enregistrements plutôt que de désenregistrer complètement la particule pour ré-appliquer ensuite la procédure d'enregistrement. De plus, en ré-appliquant la procédure complète, on garantit de retomber sur un enregistrement correct.

La figure 5.2 présente un exemple de mise à jour de l'enregistrement en 2D. Les particules jaune et fuchsia détectent un changement de cellule ce qui modifie leurs enregistrements.

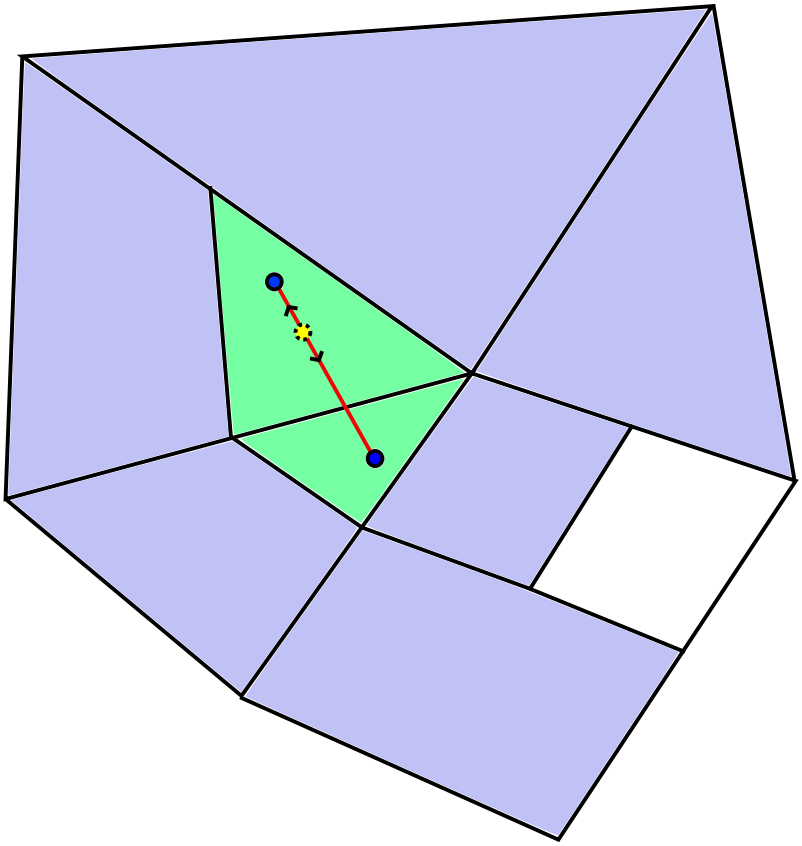

Figure 5.3 : Une arête en 2D

5.3 Enregistrement d'arêtes

Les arêtes permettent de représenter des entités filiformes ou des frontières d'entités de dimension 2. Afin d'utiliser au mieux les avantages de notre structure de représentation, nous représentons une arête par 2 particules et un ensemble de cellules traversées. Voyons en détails cette représentation choisie, illustrée sur la figure 5.3.

L'arête est composée de deux particules en bleu foncé, d'un ensemble de cellules traversées en vert, d'un ensemble de cellules voisines en bleu clair, et d'une particule virtuelle en jaune. Les deux particules aux extrémités servent à détecter en temps réel les changements de cellule, et à réinitialiser la particule virtuelle dans la bonne cellule.

Comme pour les particules, on stocke dans chaque cellule un vecteur d'arêtes présentes $V_{AretePresent}$ et voisines $V_{AreteVoisin}$. La particule virtuelle est une particule dite « mémo » car elle possède une mémoire.

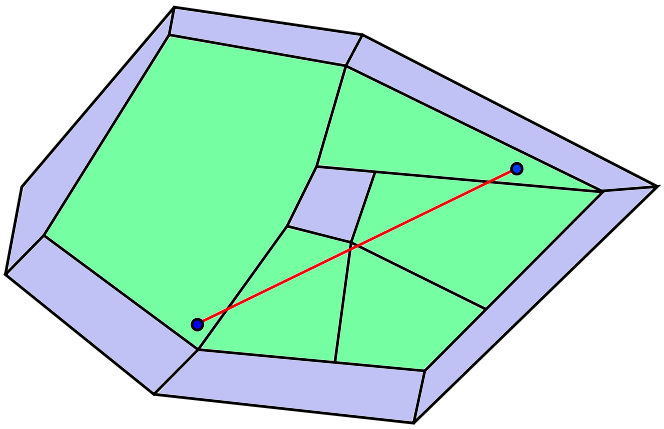

Figure 5.4 : Cas problématique pour la détection du voisinage

Elle est envoyée d'un sommet à l'autre et a la particularité de stocker l'ensemble des cellules qu'elle traverse. Cette particule est créée et lancée à chaque fois que l'on désire enregistrer l'arête. Elle permet d'obtenir l'ensemble des cellules où l'arête doit être enregistrée comme « présente » dans $V_{AretePresent}$. On peut noter que l'envoi de la particule n'est pas nécessaire si les deux particules de ses sommets sont dans la même cellule. En effet, la convexité de la représentation topologique des cellules de l'environnement nous permet de conclure que l'arête ne traverse aucune autre cellule.

On introduit ici la notion de cellule voisine à un ensemble de cellule :

Cellule voisine à un ensemble de cellules E :
Une cellule est dite voisine à un ensemble de cellules E, s'il existe une cellule a de E qui lui est 0-adjacente (adjacente par sommet).
$\exists\ a \in E/, b \notin E, a\ Adj_0\ b \Leftrightarrow b \in Voisin(E)$.

Une fois obtenu l'ensemble des cellules traversées, il faut d'obtenir l'ensemble des cellules voisines, ici en bleu clair. Diverses approches on été abordées pour résoudre ce problème en 2D.

Une première approche consiste à partir d'une cellule voisine et tourner autour de l'ensemble de cellules marquées vertes pour déterminer cet ensemble bleu. Ce tour est réalisé via les opérations topologiques de base, permettant de trouver efficacement l'ensemble bleu sans répétition. Malheureusement cet technique ne fonctionne que dans certains cas et n'est pas applicable en 3D. La figure 5.4 présente un exemple de cas problématique pour la détection du voisinage par cet algorithme. Ici, nous avons un cas ou le voisinage est séparé en deux composantes dues à la décomposition cellulaire de la carte.

Nous avons donc opté pour un algorithme moins efficace en terme de complexité, mais plus générique. Il peut être résumé ainsi : On prend une à une les cellules vertes et on fait le tour des cellules adjacentes par au moins un sommet. Si la cellule n'est pas verte ou bleue alors on la marque en bleu. Cet algorithme vérifie plusieurs fois les mêmes cellules bleues mais il garantit un résultat dans tous les cas même en 3D. On verra dans la partie résultat que les performances globales ne sont pas impactées par ce choix. On peut maintenant définir une arête bien enregistrée :

Arête bien enregistrée :
Une arête s est dite bien enregistrée si les particules qui lui sont associées sont bien orientées, si elle appartient au vecteur d'arêtes présentes $V_{AretePresent}$ de l'ensemble de cellules E dans lequel elle se trouve, et si elle appartient aux vecteurs d'arêtes voisines $V_{AreteVoisin}$ de chaque cellule voisine de E.
1. $\forall E \subset C, \forall s,\ s \in E \Leftrightarrow s \in V_{AretePresent}(E)$.
2. $\forall E \subset C, \forall s,\ s \in E \Leftrightarrow \forall B \subset C/\ B \neq E,\ s \notin B$.
3. $\forall E \subset C,\ \forall s,\ s \in V_{AretePresent}(E) \Rightarrow \forall b \in Vois(E),\ s \in V_{AreteVoisin}(b)$.
4. $\forall p$ *Assoc* s, $BO(p)$

5.4 Mise à jour de l'enregistrement des arêtes

La mise à jour de l'enregistrement d'une arête est nécessaire uniquement lorsque l'ensemble vert des cellules traversées doit être modifié. Un moyen simple de détecter ce changement est d'envoyer à nouveau une particule virtuelle d'un sommet à l'autre de l'arête puis de comparer la liste des cellules traversées par la particule et la liste actuelle de cellules en vert. Cette solution est très coûteuse si elle est effectuée à chaque pas de temps. Pour optimiser cette détection nous distinguons les différents cas à traiter, en considérant les données à notre disposition via les particules bleues des sommets. Ces particules permettent d'accéder à la cellule où elles se trouvent ainsi que de détecter lorsqu'elles changent de cellule.

La figure 5.5 présente les deux cas distingués. À gauche, le cas « monocellulaire » : les deux particules de l'arête sont dans la même cellule. À droite, le cas « pluricellulaire » : les deux particules sont dans des cellules différentes.

La problématique est de considérer un enregistrement à un instant t et de calculer sa mise à jour éventuelle à l'instant $t+1$. Pour cela nous allons considérer les situations de départ à l'instant t et d'arrivée à l'instant $t+1$ en terme de cas monocellulaire et pluricellulaire.

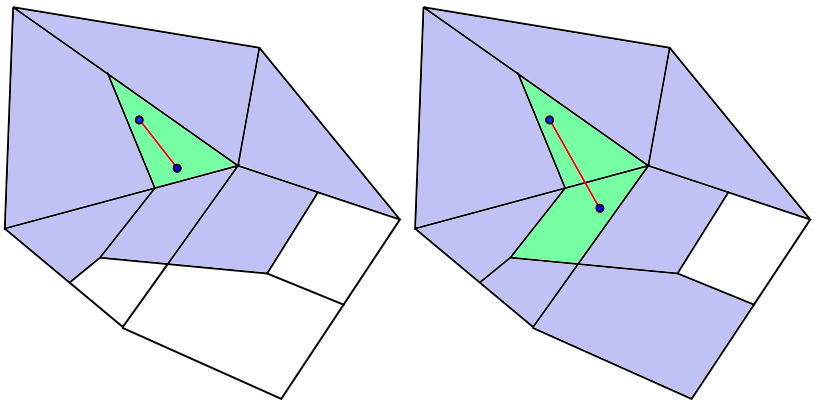

Figure 5.5 : Les deux cas possibles de disposition des particules d'une arête

Si à l'instant t les particules sont dans un cas monocellulaire et qu'elles ne détectent pas de changement de cellule à l'instant $t+1$, on peut déterminer grâce à la convexité des cellules que l'enregistrement ne doit pas être mis à jour. C'est l'unique cas où l'enregistrement d'arête ne nécessite aucun calcul. Nous verrons que notre approche multirésolution cherche à rendre ce cas le plus fréquent possible en conservant des cellules « aussi grandes » que possible.

Si en revanche des changements de cellules ont été détectés, on doit modifier l'enregistrement de l'arête en commençant par la désenregistrer pour la réenregistrer ensuite avec ou sans envoi de particule virtuelle. L'envoi de particule virtuelle étant nécessaire uniquement dans un cas pluricellulaire.

Si à l'instant t, les particules sont dans un cas pluricellulaire, il faut toujours vérifier l'enregistrement. En effet, comme le montre la figure 5.6, même si les particules ne détectent pas de changement de cellule, une mise à jour de l'enregistrement peut être nécessaire. Il faut donc de toujours envoyer une particule virtuelle et comparer les cellules traversées aux instants t et $t+1$ dans les cas pluricellulaires pour mettre à jour l'enregistrement.

On peut donc vérifier que l'arête reste bien enregistrée dans tous les cas :

— Un cas monocellulaire stable où l'enregistrement reste le même.
— Les autres cas où l'on envoie une particule récupérer l'ensemble E des cellules traversées par l'arête. Dans ces cas, on désenregistre complètement l'arête et on ré-applique la procédure standard d'enregistrement. Cet enregistrement est donc correct si et seulement si le déplacement de particules détecte le bon ensemble E.

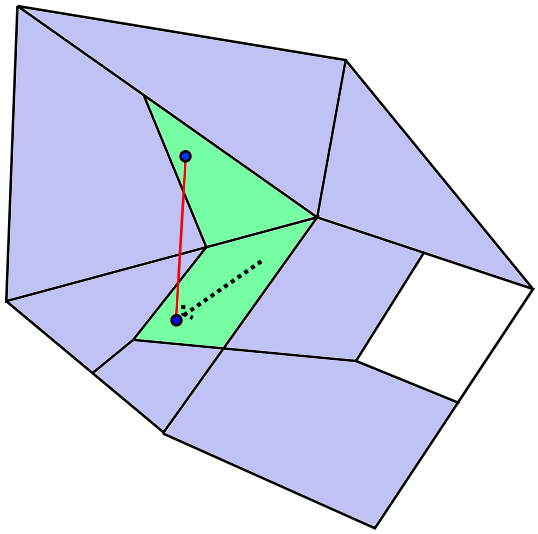

Figure 5.6 : Cas pluricellulaire sans changement de cellule nécessitant une mise à jour

5.5 Enregistrement de surfaces

Les surfaces de collision permettent de représenter des volumes modélisés par leur bord en 3D et d'interagir avec ce bord. Contrairement aux arêtes, il est difficile de balayer efficacement une surface afin de détecter les cellules qu'elle traverse.

Notre approche est donc de trianguler notre surface et de plonger les triangles qui la composent un par un. Nous représentons donc une surface par un ensemble de triangles et un ensemble de particules placées sur les sommets de ces triangles.

Chaque triangle est composé de 3 arêtes qui vont être enregistrées comme présenté dans la section 5.3. Cependant, cet enregistrement du bord du triangle n'est pas forcément suffisant en fonction du maillage 3D dans lequel il est plongé. Afin de bien comprendre la logique derrière cet enregistrement, nous rappelons que notre objectif est d'enregistrer les entités afin qu'elles puissent être visibles dans un certain voisinage pour effectuer la détection de collision.

Enregistrer les entités dans une zone étendue de cellules voisines permet d'augmenter la visibilité de celles-ci, et va aussi nous servir ici à prendre des libertés de représentation. En effet, si l'on garantit certaines contraintes sur les cellules de l'environnement, on peut considérer que l'enregistrement d'un triangle via les cellules traversées par les arêtes de son bord ainsi que leurs cellules voisines est suffisant à sa détection.

Critère de validité :
Aucune cellule (volume) de la carte représentant l'environnement ne peut être entièrement contenue dans une cellule de la subdivision d'une surface plongée dans cette carte.

Avec ce critère, il est impossible qu'une cellule du maillage soit à l'intérieur d'un triangle sans être marquée en tant que présente ou voisine (en vert ou en bleu).

Cette liberté de marquer éventuellement en bleu des cellules qui devraient être marquées en vert ne pose pas de problème pour la détection. En effet, elle ne concerne que des cas limitrophes où un triangle traverse une volume qui n'est pas traversé par ses arêtes. Dans ces cas, le fait de marquer en bleu les cellules réduit la portée de la visibilité de ce triangle sans trop altérer sa détection. Si l'on compare avec le cas 2D, cela revient à considérer l'enregistrement de la figure 5.6 comme étant correct. La différence fondamentale étant que dans le cas 3D, nous posons une contrainte sur le maillage empêchant les cas d'erreur. De plus notre contrainte peut aisément être respectée au moment de la triangulation de la surface.

En effet, en cas de changement de la taille des cellules dans une région de l'environnement, la triangulation de la surface plongée peut évoluer en vérifiant pour tous les triangles de cette région si la contrainte est toujours respectée.

On a donc une définition établie de surface triangulée bien enregistrée :

Surface triangulée bien enregistrée :
Une surface S possédant une triangulation T est dite bien enregistrée si les particules qui lui sont associées sont bien orientées, si chacun de ses triangles appartient au vecteur de triangles présents $V_{TrianPresent}$ de l'ensemble de cellules E dans lequel se trouvent ses arêtes a_1, a_2, a_3 et si chacun de ses triangles appartient aux vecteurs de triangles voisins de chaque cellule voisine de son ensemble E.
1. $\forall t = (a_1, a_2, a_3) \in T$, $\exists E \subset C$, $a_1 \in E$ *and* $a_2 \in E$ *and* $a_3 \in E \Leftrightarrow t \in V_{TrianPresent}(E)$.
2. $\forall t \in T$, $\exists E \subset C$, $t \in V_{TrianPresent}(E) \Leftrightarrow \forall B \subset C / \ B \neq E, \forall t = \in T$ $t \notin V_{TrianPresent}(B)$.
3. $\forall t \in T$, $\exists E \subset C$, $t \in V_{TrianPresent}(E) \Rightarrow \forall b \in Vois(E)$, $t \in V_{TrianVoisin}(b)$.
4. $\forall p$ *Assoc* S, $BO(p)$

On peut noter que notre technique fonctionne avec n'importe quelle subdivision de la surface tant que le critère de validité est respecté. Nous avons opté pour la triangulation qui peut se faire rapidement, possède un nombre de segments par cellules stable et se subdivise facilement. Cela nous permet également de considérer une entité unique à plonger dans les volumes de la carte.

5.6 Mise à jour de l'enregistrement des surfaces

Mettre à jour l'enregistrement d'une surface revient à mettre à jour de l'enregistrement des triangles qui la composent. Pour cette mise à jour, on repère les cas monocellulaires et pluricellulaires similaires à ceux définis pour une arête.

Si les trois particules d'un triangle sont dans le même volume, il s'agit d'un cas monocellulaire et nous allons pouvoir optimiser la mise à jour de l'enregistrement comme vu précédemment.

Si en revanche ces particules ne sont pas dans la même cellule, il s'agit d'un cas pluricellulaire et la mise à jour de l'enregistrement va être plus coûteuse et nécessiter des envois de particules virtuelles le long des arêtes du triangle.

De la même manière, on peut donc vérifier que l'enregistrement des surfaces reste correct si et seulement si le déplacement de particules détecte de manière correct les ensembles de cellules traversées.

5.7 Conclusion

Nous avons présenté ici un ensemble de méthodes permettant d'enregistrer et de maintenir enregistrées des entités quelconques dans un environnement 2D ou 3D en temps réel.

Notre méthode s'appuie sur les qualités des cartes combinatoires pour avoir un accès simplifié à des informations de localisation afin d'optimiser les calculs, notamment sur les environnements de grande taille. Ainsi, même dans un environnement très vaste, nous pouvons faire évoluer des entités ponctuelles, filiformes ou surfaciques, et leur donner une visibilité tout au long de la simulation, en temps réel.

Deux types d'entités seront globalement plongées dans l'environnement : des points et des arêtes. Nous ne considérons pas les surfaces, car elles sont enregistrées par leurs arêtes en respectant le critère présenté dans la section 5.5.

Si l'on note respectivement n_p et n_a le nombre de particules et d'arêtes à plonger dans l'environnement et c_m le nombre moyen de cellules traversées par une arête, on peut déduire les complexités temporelles suivantes :

— Pour la construction / mise à jour de la structure à chaque pas de temps : $O(n_a + n_p)$
— Pour l'accès d'une particule à son voisinage : $O(1)$
— Pour l'accès d'une arête à son voisinage : $O(c_m)$

Ainsi, le maintien de la structure d'enregistrement est efficace et l'accès en temps constant pour des entités ponctuelles. De plus, la mise à jour n'étant déclenchée que lors des changements de cellules pour les particules, cette complexité critique n'est que très rarement approchée.

Chapitre 6

Gestion de la multirésolution

Sommaire

Ce chapitre présente les outils nécessaires à l'utilisation des cartes combinatoires multirésolutions dans le cadre de la détection de collision.

Les cartes multirésolutions utilisées sont des cartes implicites présentées dans la section 3.2.2. Nous utilisons donc un raffinement primal qui maintient les attributs de sommet mais nécessite un traitement particulier pour les attributs d'arête, de face et de volume.

L'enjeu est de maintenir à jour les enregistrements des entités ponctuelles, filiformes et surfaciques plongées dans notre carte lors du processus d'adaptation de la résolution. Nous discuterons brièvement des enjeux avant de présenter les solutions proposées pour maintenir les enregistrements lors des deux opérations de la multirésolution : la subdivision et la simplification.

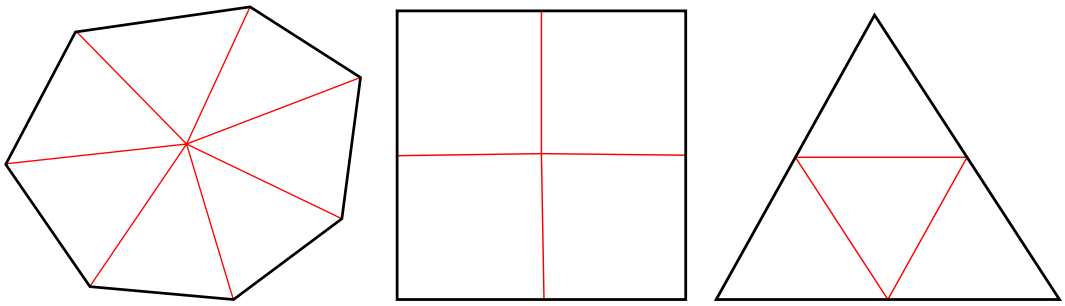

FIGURE 6.1 : Différents modèles de subdivisions

6.1 Introduction

La multirésolution nous permet d'adapter la taille des cellules de notre carte combinatoire en fonction d'un critère choisi, et en suivant un schéma de subdivision prédéterminé. Par exemple, lors d'une simulation de foule, un grand nombre d'agents se déplacent dans notre environnement. Afin de faciliter les calculs de collision sur le voisinage de chaque agent, on peut de limiter la taille d'un tel voisinage. Grâce à l'outil multirésolution, il est possible de subdiviser les cellules de l'envionnement dans les zones de forte densité d'agents et de les simplifier dans les zones vides. Dans cet exemple, le critère utilisé est donc la densité d'entités enregistrées dans une cellule.

Nos travaux sont les premiers à utiliser cet outil multirésolution pour cartes combinatoires implicites 3D, introduit par les travaux de L. Untereiner dans [36]. Le processus de multirésolution comprend 2 opérations :

— La subdivision qui découpe une cellule selon le schéma primal choisi.
— La simplification qui reconstruit une cellule à partir d'un ensemble de cellules en remontant le schéma primal.

Chaque opération crée ou détruit des cellules et nécessite donc une mise à jour de nos attributs de sommet, d'arête, de face et de volume.

6.2 La subdivision

La subdivision est la décomposition d'une cellule de niveau n en un ensemble de cellules de niveau $n+1$.

La figure 6.1 présente différents modèles de subdivision sur des cellules 2D. La cellule originelle de niveau n en noir, est décomposée en cellules de niveau $n+1$ en rouge. À gauche, la subdivision est réalisée depuis le barycentre géométrique vers les sommets. Au centre, le découpage est réalisé entre le barycentre géométrique et les milieux des arêtes. À droite, on subdivise en reliant les milieux des arêtes. Les schémas au centre et à droite ont l'avantage de conserver la forme des cellules après la subdivision. Lors de cette opération, il convient de mettre à jour nos enregistrements, à savoir :

— Les entités présentes dans la cellule à subdiviser « S »
— Les entités voisines de S qui sont donc présentes dans les cellules adjacentes.

La manière la plus efficace pour mettre à jour tous les enregistrements est une procédure en quatre temps.

Dans un premier temps, on désenregistre les entités présentes et on oublie les entités voisines dans la cellule S. Le fait de désenregistrer les entités présentes permet de les retirer en tant que voisines dans les cellules adjacentes à la cellule S et de les mémoriser pour la suite.

Dans un deuxième temps, on réalise la subdivision topologique de la cellule. Durant cette opération, de nouvelles cellules sont créées. Certains brins de la carte sont modifiés mais aucun n'est supprimé.

Troisièmement, on met à jour les attributs des nouvelles cellules et on récupère les voisins de chaque nouvelle cellule. La récupération des voisins est faite en réalisant un tour de chaque nouvelle cellule de niveau $n+1$ pour récupérer les entités présentes dans leurs cellules voisines. On utilise pour cela un traverseur d'adjacence par sommet optimisé pour parcourir les cartes combinatoires.

Enfin, on réenregistre les entités présentes retenues au départ avec la procédure d'enregistrement classique présentée dans le chapitre 5.

On peut vérifier que les propriété 1. 2. et 3. des entités bien enregistrées sont respectées au début et à la fin de la subdivision.

La propriété 1. que l'on peut nommer propriété d'appartenance est vérifiée car toutes les entités appartent aux $V_{XPresent}$ ont été complètement désenregistrées pour être réenregistrées dans la dernière phase. Les autres entités impactées ont seulement vu leur voisinage modifié.

La propriété 2. que l'on peut qualifier de propriété d'absence en dehors de l'appartenance est vérifiée car nos modifications prennent soin de complètement désenregistrer les entités qui vont être modifiées.

La propriété 3. que l'on peut définir comme propriété de présence dans le voisinage est vérifiée d'une part pour les entités présentes dans la cellule subdivisée et d'autre part pour les entités voisines impactées par la subdivision. Concernant les entités présentes, leur réenregistrement complet garantie la vérification de cette propriété. Concernant les entités voisines, leur enregistrement est à nouveau rendu correct par la phase de récupération des voisins.

Des précisions sur le respect de cette propriété 3. et les garanties concernant la propriété 4. ou propriété de bonne orientation des particules seront apportées dans les sous-section suivantes

La figure 6.2 présente ces diverses étapes avec des entités ponctuelles en 2D. Ces étapes sont toujours respectées, mais elles nécessitent toutefois des précisions. Voyons en détails ces précisions concernant chaque type d'entité traitée ici : particules, arêtes et triangles.

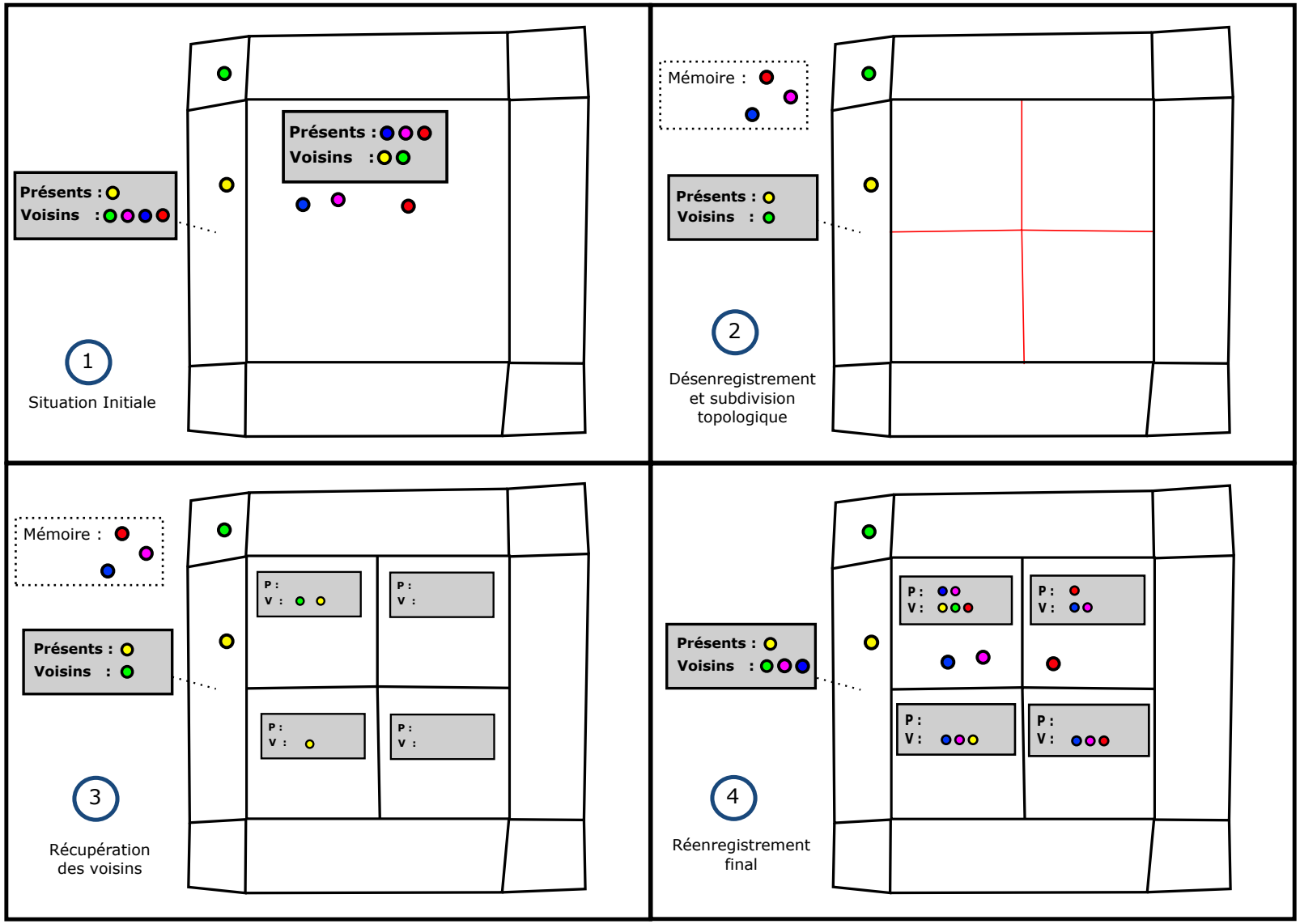

FIGURE 6.2 : Les étapes de la subdivision en 2D

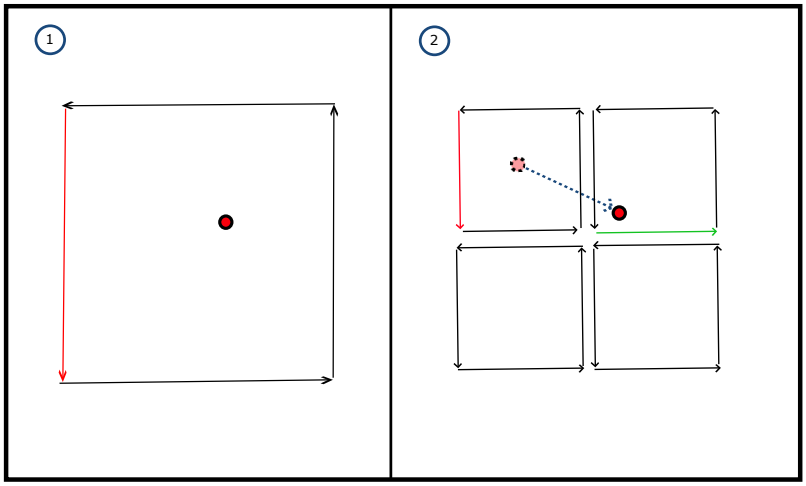

Figure 6.3 : Mise à jour des particules lors de la subdivision

6.2.1 Les particules

Concernant les entités ponctuelles et les particules en général, une mise à jour du brin visé est nécessaire pour que la particule soit bien orientée. Cette mise à jour permet donc de vérifier la 4ème propriété des entités bien enregistrées.

Or, lorsque l'on désenregistre les particules lors de la première phase du processus de subdivision, la particule vise un certain brin B. Lors du réenregistrement de la dernière phase, on ne peut pas simplement réintroduire la particule à sa position géométrique et la réenregistrer dans la cellule correspondant à ce brin.

Comme le montre la figure 6.3, le brin visé initialement peut ne pas correspondre à la cellule renfermant la position géométrique de la particule après la subdivision.

Notre solution à ce problème consiste à réintroduire la particule au centre de la nouvelle cellule correspondant au brin B, avant de déplacer cette particule jusqu'a sa réelle position géométrique afin qu'elle se réoriente d'elle-même.

Cette méthode présentée dans la figure 6.3 permet de conserver la cohérence géométrique avec le brin visé tout au long du processus. Le brin B est coloré en rouge et le brin visé à la fin du repositionnement est coloré en vert. Une fois que la particule est repositionnée, elle connaît le nouveau brin correspondant à sa nouvelle cellule et peut donc s'y enregistrer.

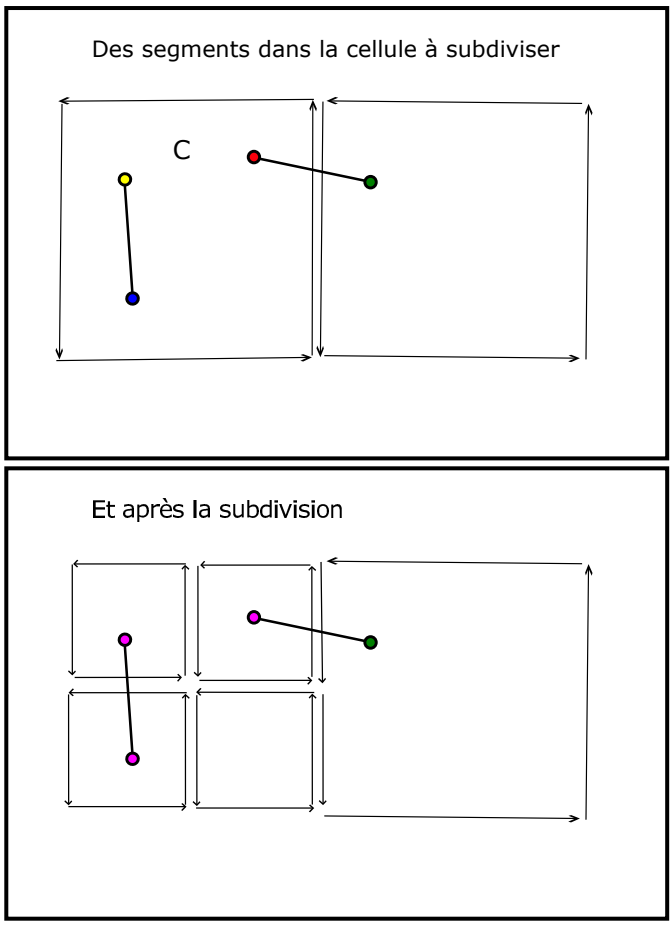

Figure 6.4 : Cas des segments à l'intérieur de la cellule

6.2.2 Les arêtes et les triangles

Les entités de dimension supérieure, telles que les arêtes et les triangles sont plus difficiles à mettre à jour.

Elles sont toutes basées sur des particules et nécessitent donc le même processus de repositionnement que celui présenté dans la section 6.2.1. Mais, contrairement aux entités ponctuelles, ces entités retiennent les cellules dans lesquelles elles sont enregistrées ainsi que les cellules voisines.

L'enregistrement de ces entités étant plus complexe que celui de simples particules, on doit appliquer diverses optimisations afin de limiter la complexité du processus de subdivision. Pour cela, il convient de distinguer les différents cas afin de les traiter correctement. La figure 6.4 montre l'évolution du cas de segments enregistrés dans une cellule à subdiviser.

L'intégralité de l'enregistrement de ces segments doit être mis à jour car il est plus complexe de détecter les changements que de tout réenregistrer. On peut en revanche noter que seules les particules marquées en rose dans la deuxième partie de la figure nécessitent d'être mises à jour.

La figure 6.5 présente l'évolution du cas de segment voisin d'une cellule à subdiviser. Les brins colorés sont les brins visés par les particules de la couleur correspondante. Une fois les particules repositionnées, l'entité peut être réenregistrée totalement lors de la dernière phase du processus.

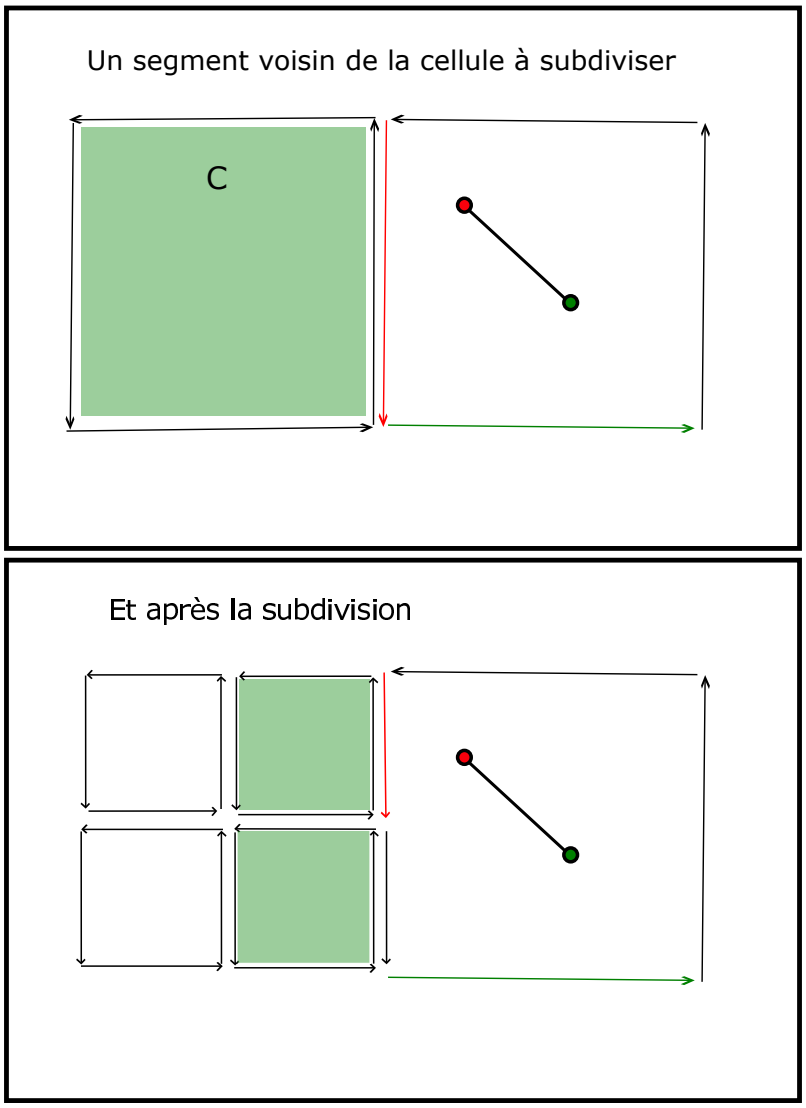

Figure 6.5 : Cas de segment voisin de la cellule

Ici, il n'est pas nécessaire de mettre à jour les brins visés et seule la cellule voisine en question doit être modifiée dans l'enregistrement. La propriété 4. reste donc vérifiée pour les entités voisines.

Nous désirons éviter les redondances dans la détection des cellules qui doivent être ajoutées dans les vecteurs des arêtes alentours à la place de la cellule voisine retirée. Pour cela, on laisse chaque nouvelle cellule de niveau $n+1$ détecter ses voisins et s'jouter à leur vecteur de cellules voisines.

Ainsi, durant la phase 3 du processus, lorsque l'on réalise le tour des nouvelles cellules, chaque cellule récupère ses voisins puis s'ajoute au vecteur des voisins de chaque entité de type arête ou triangle.

Le fait de ne réenregistrer les entités présentes dans la cellule qu'à la fin permet de ne pas les détecter lors de la recherche de voisins. Ces entités s'enregistreront d'elles-mêmes en tant que voisines par la suite.

Après toutes les phases, on peut vérifier la propriété 3. : les entités voisines sont correctement enregistrées en tant que voisines dans les nouvelles cellules et les entités présentes ont subi un réenregistrement total.

6.3 La simplification

La simplification est l'opération inverse de la subdivision. Elle consiste à regrouper un ensemble de cellules de niveau $n+1$ en leur cellule originelle de niveau n. Afin de conserver une cohérence dans notre structure hiérarchique topologique, seules des cellules issues de la même cellule de niveau inférieur peuvent être simplifiées.

Lors de cette opération, il convient de mettre à jour nos enregistrements. Les entités nécessitant une mise à jour sont celles plongées :

— En tant que présentes dans les cellules à simplifer $S_1, S_2, ..., S_n$.
— En tant que voisines dans $S_1, S_2, ..., S_n$ et qui ne font pas partie des entités présentes dans ces $S_1, S_2, ..., S_n$.

Cette mise à jour doit garantir les propriété 1., 2. et 3. des entités bien enregistrées. De plus, le processus de gestion de la simplification est plus complexe que la subdivision, car on y supprime des cellules et donc des brins potentiellement visés par des particules. Il faut donc porter une attention particulière à la propriété 4. dans ce processus.

Durant la première phase, on retient et on désenregistre les entités marquées comme présentes dans les cellules à simplifier puis on supprime les entités voisines. Cette opération est réalisée en effectuant deux tours successifs des cellules $S_1, ..., S_n$.

Durant le premier tour, on désenregistre les entités présentes en premier pour de ne pas faire d'erreur avec les entités voisines. En effet, une entité présente dans la cellule S_1 est également marquée en tant que voisine dans les autres cellules à simplifier. De plus, le désenregistrement d'une entité est appliqué partout où elle se trouve. Ainsi, chaque entité ne sera traitée qu'une seule fois même si elle est présente dans plus d'une des cellules $S_1, ..., S_n$.

Le deuxième tour permet de supprimer chaque cellule des vecteurs de cellules voisines où elle est encore présente. Par exemple, en traitant la cellule S_1, on va aller dans chaque entité encore plongée comme voisine dans S_1, et supprimer à chaque fois S_1 du vecteur de cellules voisines.

Durant la deuxième phase, on effectue la simplification topologique proprement dite. Cette opération supprime les brins du niveau $n+1$ pour ne laisser que les brins de niveau inférieur ou égal à n qui composent la cellule S.

Enfin, on fait un tour de la nouvelle cellule S pour récupérer les voisins avant de réenregistrer les cellules anciennement présentes ayant été mémorisées. Le fait de commencer par la récupération des voisins permet de ne pas avoir à éviter les entités présentes à la fois dans la cellule S et son voisinage. En effet, si on avait commencé par le réenregistrement de ces entités, elles auraient été à nouveau détectées lors du tour de récupération des voisins.

Comme pour la subdivision, le fait de désenregistrer complètement les entités présentes pour les réenregistrer ensuite nous permet de vérifier les propriétés 1., 2. et 3. pour ces entités. Les propriétés 1. et 2. sont également respectées pour les entités voisines de manière triviale. La propriété 3. est à nouveau respectée pour les entités voisines grâce à la phase de récupération des voisins. La vérifiaction de la propriété 4. sera détaillée dans la sous-section suivante.

La figure 6.6 présente ces diverses étapes avec des entités ponctuelles en 2D. Les étapes de désenregistrement et de réenregistrement sont très similaires aux étapes présentées dans la section 6.2. En revanche les particules en elles-mêmes nécessitent un traitement particulier.

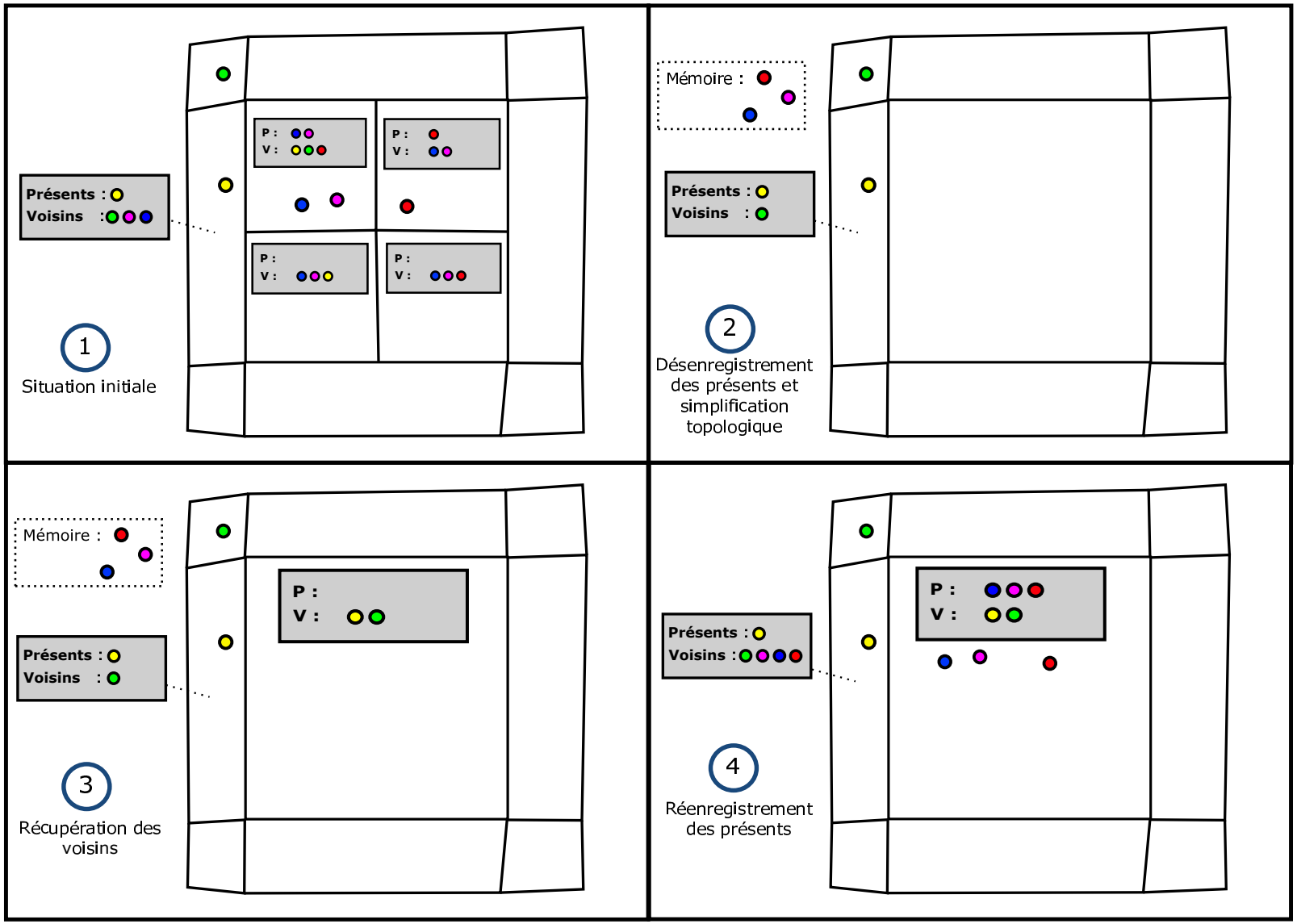

FIGURE 6.6 : Les étapes de la simplification en 2D

6.3.1 Cas des particules

Comme énoncé précédemment, l'opération de simplification supprime éventuellement des brins de la carte. Il y a donc un risque important de non respect de la propriété 4. aboutissant à des particules mal orientées.

La suppression de brins est déterminée par l'adjacence de cellules de niveau inférieur ou égal à n. En 3D, si un volume de niveau $n+1$ est voisin par face au volume que l'on veut simplifier, alors la face ne va pas être simplifiée. De même, une arête ne sera pas simplifiée si un volume adjacent par arête est de niveau $n+1$.

Afin de respecter la propriété 4., il faut vérifier qu'aucune particule ne vise un brin qui va être retiré .

— Pour les particules présentes dans les cellules à simplifier, on sait qu'elles seront toutes à l'intérieur de la cellule finale S. On peut donc les faire viser un même brin de niveau n de cette cellule.
— Pour les particules présentes dans des cellules voisines de S, le cas est plus complexe et nécessite une vérification.

De ce fait, une étape supplémentaire doit être effectuée au début du processus afin de réaliser cette vérification.

La figure 6.7 présente en rouge les brins des cellules S_1, S_2, S_3 et S_4 qui vont être supprimés par la simplification. Les brins en vert ne vont pas être supprimés car la cellule voisine est elle-même subdivisée.

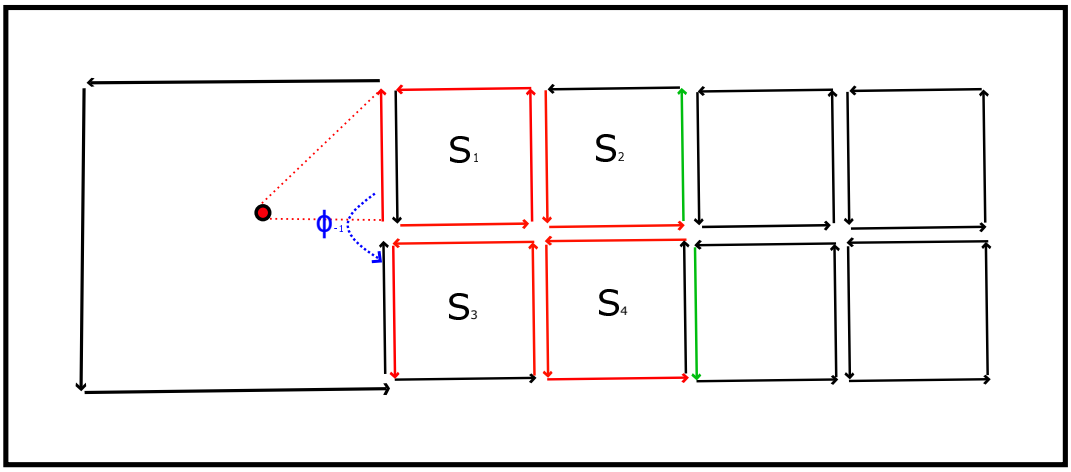

Figure 6.7 : Brins supprimés et mise à jour nécessaire des particules

Concernant le cas 2D, il suffit de regarder si le brin de niveau $n+1$ est visé dans chaque cellule voisine par une arête. Si c'est le cas, on change le brin b visé par le brin $\Phi_1^{-1}(b)$ comme montré dans la figure 6.7.

Pour le cas 3D, beaucoup plus de brins sont supprimés. Les volumes, qui contiennent des entités susceptibles de viser un brin amené à disparaître, sont les volumes de niveau inférieur ou égal à n, qui sont adjacents par arête à chaque face traitée. On doit donc de faire le tour de chacun de ces volumes et vérifier si les particules ne visent pas un brin de niveau $n+1$. Si c'est le cas, on change le brin visé b par par le brin $\Phi_1^{-1}(b)$ jusqu'à tomber sur un brin de niveau inférieur ou égal à n. Ainsi, on est sûr de viser un brin correct, que ce soit dans le cas d'un brin intérieur à une face ou sur une arête.

6.4 Conclusion

Le processus de multirésolution nous permet d'adapter la taille des cellules et donc d'un voisinage. Cette adaptabilité permet souvent d'optimiser les calculs ou d'obtenir localement une plus grand précision.

Nous avons vu que le processus de mise à jour n'est pas gratuit et nécessite une attention particulière pour ne pas créer des incohérences ou détériorer les performances du processus. Avec les algorithmes présentés, nous arrivons à un coût du processus de multirésolution négligeable par rapport aux autres coûts de simulation. Nous aborderons plus en détails les avantages apportés par cet outil et nous chiffrerons les performances dans la partie III.

Chapitre 7

Suivi de particules dans des maillages non-convexes

Sommaire

Ce chapitre présente notre extension des travaux de T.Jund concernant le suivi de particules dans des maillages respectant la contrainte BC, sans considération de la multirésolution qui a été traitée dans le chapitre 6.

Les travaux de T. Jund sont très efficaces pour le suivi de particules dans des maillages décomposés en cellules élémentaires convexes, telles que des tétraèdres en 3D. Toutefois, lors de nos travaux sur des maillages hexaédriques déformables, notamment en simulation médicale, les cellules subissent de légères déformations qui mettent à l'épreuve la robustesse des calculs par plans de séparation décrits dans ces travaux.

Nous rappelons que les concepts de base et les algorithmes concernant le suivi de particules présentés dans le chapitre 4 sont nécessaires à la compréhension de ce chapitre.

Voyons dans un premier temps les cas problématiques avant de présenter la solution apportée.

7.1 Cas problématiques

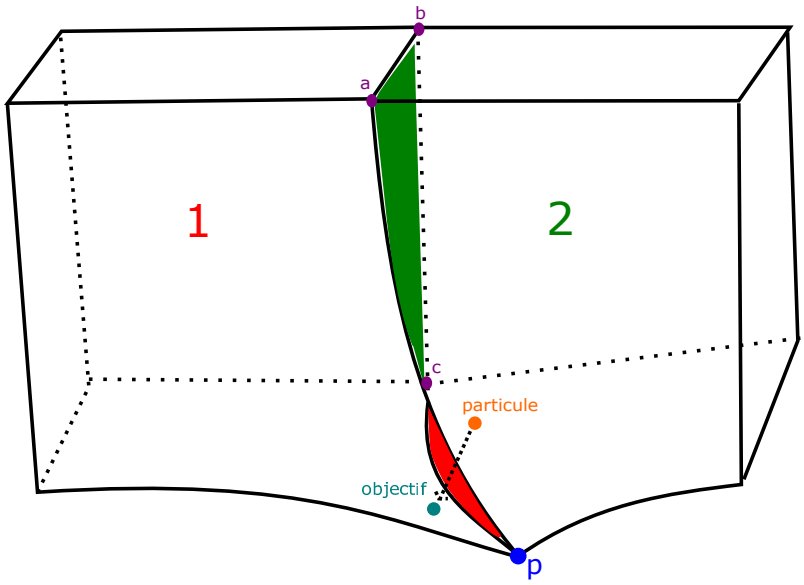

FIGURE 7.1 : Cas de déformation d'un hexaèdre entrainant une défaillance des algorithmes basés sur la convexité des cellules.

Les cas problématiques ont été rencontré dans le cadre particulier de cellules hexaédriques déformables. En effet, pour ce genre de cellules, le fait qu'un sommet du maillage bouge par rapport aux autres remet en cause la contrainte de convexité du maillage. Notons ici que ce n'est pas le cas pour des cellules tétraédriques qui restent toujours convexes.

Dans ces cas particuliers problématiques, les algorithmes prévus pour l'orientation des particules échouent car les faces ne sont plus des « faces » au sens géométrique mais plutôt des « surfaces » car elles sont formées de 4 points non-coplanaires. Comme l'orientation est réalisée en effectuant les calculs par rapport a un plan formé par 3 points d'une face, selon le choix de ces 3 points, le plan considéré ne sera pas le même.

Par exemple, dans le cadre de l'orientation depuis un volume, si le cheminement de l'algorithme fait que la particule traverse une face déformée pour se retrouver dans un autre volume mais a l'intérieur de la courbure de la face, on se retrouve dans un cas soit de boucle infinie de l'algorithme soit d'erreur sur le brin visé.

Ce cas est présenté dans la figure 7.1 où la particule en orange doit se déplacer à la position turquoise marquée « objectif », en passant du volume 2 au volume 1. Si l'on prend les points A, B et C pour définir le plan de la

face, le test d'orientation déterminera que la particule arrive a destination sans changer de volume.

Si l'on prend n'importe quelle combinaison qui inclut le point déplacé P (en bleu), l'algorithme effectue le changement de volume et depuis le volume n°1 va réeffectuer le test. Or on ne peut pas garantir que le test sera effectué sur les mêmes 3 sommets depuis le volume 1. Si dans ce second test on utilise d'autres combinaisons de sommets, le test peut ainsi nous renvoyer au volume 2, qui lui-même nous renverra au volume 1, etc.

Ces cas de déplacements courts n'étant pas rares, il convient d'essayer de les contenir en évitant les cas d'échec critique de l'algorithme d'orientation (boucle infinie) ainsi que de les rendre les plus corrects possibles par rapport à la géométrie (mauvais volume visé).

7.2 Solution

La solution évoquée par T.Jund dans ses travaux consiste en un remaillage local pour retrouver la configuration de convexité des cellules. Cette solution est cependant relativement lourde à appliquer si ces petites déformations arrivent fréquemment, et la mise en oeuvre d'un remaillage efficace et robuste reste non-triviale.

Notre solution consiste en un retour à des faces planes au sens géométrique du terme. Pour cela, nous utilisons une approximation de la face non plane en introduisant un point fictif en son barycentre et en effectuant une triangulation fictive de la face déformée. Cela nous permet de réaliser l'orientation par rapport à ces triangles. Nous avons donc une approximation en triangles d'une face non plane depuis son barycentre.

Cette notion de point central permettant une orientation plus précise a été étendue aux volumes. On ne s'oriente plus par rapport à la position de la particule en question, mais par rapport à un point m depuis lequel tous les points du volume sont visibles. Nous désignerons ce point comme le centre de visibilité.

On définit ainsi notre contrainte BC sur les cellules :

Cellule BC :
Une cellule C est dite Bien Centrée (BC) s'il existe un point m tel que pour tout point p appartenant à C, le segment (m, p) est dans cette cellule.
C *est BC* $\Leftrightarrow \exists m \in C, \forall p \in C\ [m, p] \in C$.

Avec ces considérations, nous pouvons noter que la position de la particule n'est plus utile pour réaliser l'orientation. Notre nouvel objectif est de partir d'un brin visé et d'un état de la particule et de trouver le nouveau brin visé et le nouvel état pour la position donnée comme objectif. Cette position est finalement toujours atteinte en tant que nouvelle position pour la particule en l'absence de collisions. Cette prise de recul par rapport à la position réelle de la particule nous permet également d'éviter les erreurs dûes aux calculs d'orientation dans les cas de valeurs trop petites. Le centre de visibilité restant relativement éloigné des faces du bord, les plans d'orientation sont toujours bien distincts. Ces erreurs d'orientations représentaient la majeure partie des instabilités des travaux antérieurs. Cette absence de considération de la position nous amène à ne plus calculer d'intersections en cas de collision. Comme les déplacements sont très petits, le déplacement entraînant une collision peu simplement être ignoré.

Nous pouvons également noter que cette liberté de représentation a un coût : il faut pouvoir récupérer à tout moment, et éventuellement un grand nombre de fois les barycentres des faces et le centre de visibilité des volumes. Nous nous appuyons sur la force de la structure de données proposée dans CGoGN [6] pour rendre ce coût négligeable. En effet, les centres des faces peuvent être stockés et mis à jour dans un attribut de face, et les centres de visibilité peuvent être stockés dans un attribut de volume. Comme expliqué dans le chapitre 3, l'accès à ces attributs est réalisé en temps constant. Il suffit donc d'accéder à ces attributs pour récupérer les points désirés lors de l'orientation dans la carte.

Voyons désormais l'évolution des algorithmes d'orientation des particules pour les 4 cas d'état possibles : volume, face, arête et sommet.

7.2.1 Orientation depuis un volume

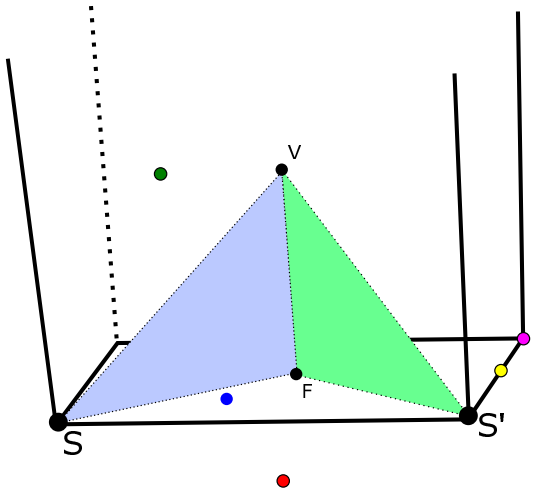

Figure 7.2 : Orientation depuis un volume

L'orientation depuis un volume est le cas le plus courant et donc celui qui doit être le plus efficace. Comme pour chaque cas que nous allons aborder, nous supposons que la particule se trouve dans l'état du cas concerné (ici, volume) et que nous connaissons la position dans l'espace de son objectif.

Grâce aux attributs stockant les centres des volumes et des faces, nous n'avons plus besoin de la position de la particule qui importe peu car elle doit finir par arriver à la même position que son objectif. Il faut toutefois noter que si l'objectif est de l'autre coté d'un obstacle, l'objectif ne sera pas atteint. Ce problème est généralement résolu en amont par les algorithmes de recherche de chemin dans les applications ayant des obstacles.

La figure 7.2 présente la situation ou l'on cherche à s'orienter dans un volume pour atteindre un des objectifs colorés. Les points colorés représentent les différents objectifs possibles. On note V le centre du volume, F le centre de la face visée, S le sommet visé, S' le sommet suivant par Φ_1.

Tous les cas possibles pour un objectif sont présentés ici, l'objectif est :

— En vert : dans le volume.
— En bleu : sur une face du volume.
— En jaune : sur une arête du volume.
— En rose : sur un sommet du volume.
— En rouge : à l'extérieur du volume.

Les 3 premiers cas doivent être des cas d'arrêt de l'algorithme, qui doit renvoyer le nouveau brin visé, la nouvelle position de la particule et le nouvel état de celle-ci. Le cas rouge doit poursuivre le cheminement en appelant les cas face, sommet ou arête selon l'orientation trouvée. Avec ces notations, on peut décrire simplement l'algorithme d'orientation depuis un volume.

Le principe est de rechercher si l'objectif est dans le tetraèdre VSS'F. Si oui, on s'arrête dans l'état courant qui est le bon. Si non, selon la face du tetraèdre la plus proche du point, on va tourner, c'est à dire changer le brin courant :

— en changeant de sommet visé sans changer de face visée si l'objectif est au dela d'un des plans VFS ou VFS'
— en changeant de face visée si l'objectif est au dela du plan VSS'
— en passant la main à l'orientation de sommet, d'arête ou de face si l'objectif est au dela du plan FSS'

Comme dans les travaux de T. Jund, on optimise les rotations en tournant soit vers la droite soit vers la gauche selon l'orientation initiale.

On ne présente pas ici la fonction « OrientationPointPlan » qui renvoie OVER, UNDER ou ON selon la position du point par rapport au plan. Nous rappelons également que Φ_{-1} est l'opération inverse de Φ_1 qui sont des opérations de changement d'arête au sein d'une même face ; Φ_2 permet de changer de face dans un même volume et Φ_3 permet de changer de volume.

Pour plus de clarté, l'algorithme a été séparé en deux parties. L'algorithme 1 permet de récupérer le bon cône défini par les plans VSS', VFS et VFS' contenant l'objectif. L'algorithme 2 vérifie enfin l'orientation par rapport au plan FSS' pour conclure sur l'orientation depuis un volume.

Algorithme 1 Trouver la bonne face et la bonne arête dans l'orientation depuis un volume

fonction PLACEONRIGHTFACEANDRIGHTEDGE(objectif , S , V , F)
  $aDroite \leftarrow faux$ ▷ On retient si on a deja tourné vers la droite
  $aGauche \leftarrow faux$ ▷ et vers la gauche
  $casOn \leftarrow faux$ ▷ si l'on se trouve exactement sur un plan
  $enDessous \leftarrow faux$ ▷ si on est au dessous de VSS'
  répéter
    $orient \leftarrow$ ORIENTATIONPOINTPLAN$(PlanVFS, objectif)$
    si orient est OVER **alors**
      $aGauche \leftarrow vrai$
      $S \leftarrow \Phi_{-1}(S)$
    sinon si orient est UNDER **alors**
      $aDroite \leftarrow vrai$
      si $aGauche \neq vrai$ **alors** ▷ ici on tourne uniquement si on est
        $S \leftarrow \Phi_1(S)$ ▷ encore jamais allé vers la gauche
      fin si
    sinon ▷ si orient est ON
      si aGauche est vrai ou aDroite est vrai **alors**
        $casOn \leftarrow vrai$
      sinon
        $S \leftarrow \Phi_1(S)$
      fin si
    fin si
  jusqu'à (aDroite est vrai et aGauche est vrai) ou (casOn est vrai) ou (on a fait un tour)
  $orient \leftarrow$ ORIENTATIONPOINTPLAN$(PlanVSS', objectif)$
  si orient est OVER **alors** ▷ si on est au dessus, il faut changer de côté de la cellule
    $S \leftarrow \Phi_2(S)$
    $F \leftarrow CentreFace(S)$
    renvoyer PLACEONRIGHTFACEANDRIGHTEDGE(objectif , S , V , F)
  sinon si orient est UNDER **alors** ▷ ici on sort si on est ON ou UNDER, enDessous sert a différencier les cas
    $enDessous \leftarrow vrai$
  fin si
  renvoyer enDessous , casOn , S , F
fin fonction

Algorithme 2 Orientation depuis un volume

```
fonction OrientationVolume(objectif , S , V , F)
    PlaceOnRightFaceAndRightEdge(objectif , S , V , F)
    orient ← OrientationPointPlan(PlanFSS', objectif)
    si orient est OVER alors
        Objectif est dans le tétraèdre, on est arrivé
        état d'arrivée : Volume
    sinon si orient est ON alors
        Objectif est sur la face, on est arrivé
        si enDessous est vrai alors
            état d'arrivée : Face
        sinon si casOn est vrai alors
            état d'arrivée : Sommet
        sinon
            état d'arrivée : Arête
        fin si
    sinon                    ▷ orient est UNDER, on est pas dans le bon volume
        S ← Φ3(S)
        F ← CentreFace(S)
        V ← CentreVolume(S)
        OrientationVolume(objectif,S,V,F)
    fin si
fin fonction
```

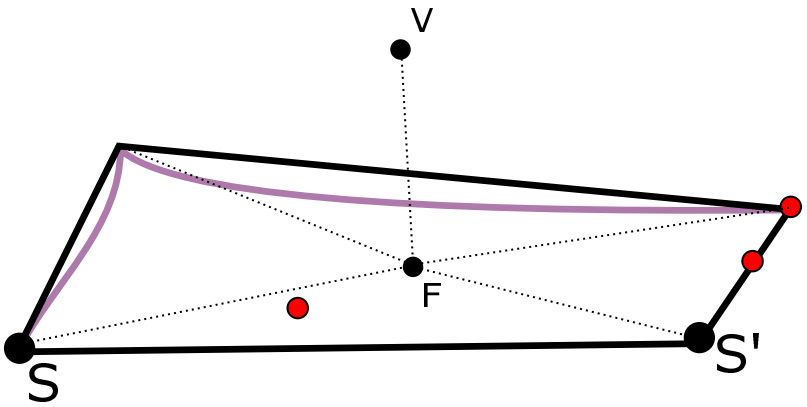

Figure 7.3 : Orientation depuis une face

7.2.2 Orientation depuis une « face »

Les « faces » n'étant pas des faces au sens géométriques, nous nous y référerons par le terme surface. Les faces que nous utilisons sont des approximations obtenues en décomposant virtuellement les surfaces réelles en facettes triangulaires depuis leur centre de gravité.

La figure 7.3 présente la situation où l'on cherche à s'orienter dans la face pour atteindre un des objectifs colorés. Nous reprenons les mêmes notations que pour l'orientation depuis un volume. Ici, nous avons en couleur la surface géométrique réelle, et en en contour noir l'approximation topologique. Les facettes triangulaires virtuelles utilisées pour l'orientation sont représentées en pointillés. Les positions rouges représentent des cas d'arrêt de l'algorithme, c'est à dire quand l'objectif est sur la surface. Les différents points montrent les 3 états possibles de FACE, SOMMET et ARÊTE. Si l'objectif n'est pas sur la surface, l'algorithme doit continuer l'orientation depuis un nouvel état.

L'orientation depuis une surface varie très peu des algorithmes présentés par T. Jund. Pour rappel, cette opération consiste en deux étapes clés. Définir si on est bien sur la face, puis si c'est le cas, rechercher le secteur de la face qui contient l'objectif.

La différence notable avec nos travaux réside dans l'ordre des opérations et dans la création des plans d'orientation. En effet, comme la surface n'est pas composée de points coplanaires, on ne peut pas commencer par chercher si le point est sur la surface, car cette vérification n'est plus triviale. Ainsi, on commence par trouver le secteur de l'espace, défini par une facette de la surface, qui contient l'objectif. Ensuite, on teste si on est sur le même plan que cette facette.

De plus, la non-coplanarité nous empêche d'utiliser la normale à la face pour créer les plans d'orientation. Ainsi nous allons utiliser le centre du volume comme point de référence pour comparer de manière cohérente les orientations au fur et à mesure de la recherche du secteur.

Pour trouver le secteur, on effectue les tests d'orientation de l'objectif par rapport aux plans VFS et VFS'. L'algorithme 3 présente la démarche complète pour s'orienter depuis une surface.

Algorithme 3 Orientation depuis une surface

```
fonction OrientationSurface(objectif , S , V , F)
    aDroite ← faux                    ▷ On retient si on a déjà tourné vers la droite
    aGauche ← faux                                          ▷ et vers la gauche
    casOn ← faux                      ▷ si on se trouve exactement sur un plan
    répéter
        orient ← OrientationPointPlan(PlanVFS, objectif)
        si orient est OVER alors
            aGauche ← vrai
            S ← Φ_{-1}(S)
        sinon si orient est UNDER alors
            aDroite ← vrai
            si aGauche ≠ vrai alors   ▷ ici on tourne uniquement si on n'est
                S ← Φ_1(S)            ▷ encore jamais allé vers la gauche
            fin si
        sinon                                               ▷ si orient est ON
            si aGauche est vrai ou aDroite est vrai alors
                casOn ← vrai
            sinon
                S ← Φ_1(S)
            fin si
        fin si
    jusqu'à (aDroite est vrai et aGauche est vrai) ou (casOn est vrai) ou
(on a fait un tour)
    orient ← OrientationPointPlan(PlanFSS', objectif)
    si orient est OVER alors
        S ← Φ_3(S)
        V ← CentreVolume(S)
        OrientationVolume(objectif , S , V , F)
    sinon si orient est UNDER alors
        OrientationVolume(objectif , S , V , F)
    sinon                             ▷ on est bien sur la facette en question
        orient ← OrientationPointPlan(PlanVSS', objectif)
        si orient est OVER alors      ▷ on est de l'autre côté de l'arête
            OrientationEdge(objectif , S , V , F)
        sinon si orient est UNDER alors                 ▷ on est sur la facette
            Objectif est sur la surface, on est arrivé
            état d'arrivée : Face
        sinon                                           ▷ on est sur l'edge
            Objectif est sur la surface, on est arrivé
            Si casOn est vrai : état d'arrivée : Sommet
            Si casOn est faux : état d'arrivée : Arête
        fin si
    fin si
fin fonction
```

7.2.3 Orientations depuis une arête et un sommet

Les orientations depuis une arête ou un sommet varient très peu de leur version précédente. Dans chaque cas, le principal changement réside en l'utilisation de la facette liée au brin visé pour les calculs de l'orientation, à la place d'utiliser la face qui n'est pas forcément plane.

Orientation depuis une arête

Ici, le but est de partir d'une arête visée et de déterminer si l'on passe la main à l'orientation depuis un volume adjacent, à une face adjacente ou si l'on conclut avec l'objectif sur l'arête ou sur un sommet de l'arête.

L'algorithme consiste à savoir si l'objectif est inclut dans la section d'espace définie par deux facettes d'un même volume, qui sont incidentes à l'arête visée. Si l'objectif n'est pas dans cette section, on tourne autour de l'arête pour changer de volume via l'opération topologique $\Phi_2 \circ \Phi_3$. Les résultats d'orientation sur les deux facettes en question permettent de déterminer s'il faut :

— passer la main à l'orientation depuis une face si l'on est sur une facette uniquement.
— passer la main à l'orientation depuis un volume si on est à l'intérieur mais sur aucune des facettes.
— conclure si on est sur les deux facettes donc sur l'arête.

Dans ce dernier cas, on effectue deux derniers tests d'orientation pour déterminer si l'on se trouve bien sur l'arête ou sur l'un ou l'autre des sommets.

Orientation depuis un sommet

Le but est de partir d'un sommet visé et de déterminer si on l'a quitté. Si c'est le cas, on fait appel à l'orientation depuis la bonne entité.

Ce cas est très peu utilisé car il est uniquement appelé lorsqu'une particule démarre son déplacement depuis un sommet. En effet, les autres algorithmes d'orientation sont capables de conclure directement si l'objectif se trouve sur un sommet ou non. Cet algorithme est donc utilisé pour redémarrer une orientation de manière optimisée, lorsque la particule est dans le cas ambigu entre plusieurs volumes où elle pourrait changer en permanence l'orientation.

L'algorithme consiste à savoir si l'objectif est inclut dans le cône défini par les facettes du volume incidentes au sommet visé. Si l'objectif n'est pas dans cette section, on tourne autour du sommet pour changer de volume via l'opération topologique $\Phi_1 \circ \Phi_3$.

7.3 Cohérence avec les enregistrements

Dans les parties précédentes, nous nous sommes souvent appuyé sur la convexité des cellules notamment pour optimiser les cas monocellulaires d'enregistrement des arêtes. L'extension aux cellules BC ne nous permet plus de réaliser ces optimisations et nous mène à deux choix possibles.

On peut considérer que dans nos applications, les déformations sont petites. Dans ce cadre, la déformation des cellules engendrée peut faire qu'une arête en cas monocellulaire traverse également une cellule adjacente. Cette cellule fait alors partie du voisinage direct, notre arête est donc bien perceptible dans cette cellule. L'incohérence se trouve dans son enregistrement car elle sera dans le vecteur $V_{AreteVoisin}$ au lieu de $V_{AretePresent}$. Cette incohérence ne pose pas de problème pour nos applications qui n'ont que de faibles déformations. On peut donc ignorer ces petits défauts d'enregistrement qui n'ont pas d'impact sur le voisinage en lui-même.

L'autre choix possible est de vouloir une cohérence parfaite entre l'enregistrement et la réalité géométrique. Cette cohérence prend tout son sens en cas de très fortes déformations. Dans ces cas, il convient de supprimer le cas monocellulaire et donc de vérifier en permanence l'enregistrement des arêtes.

On est donc comme toujours face à un choix entre précision et rapidité, dont la solution dépend uniquement de l'application visée.

7.4 Conclusion

Nous avons donc une série d'algorithmes basés sur l'approximation d'un maillage, dont les faces « déformées » ne sont pas planes, en un maillage dont les faces on été triangulées depuis leur centre de gravité. Ces algorithmes sont suffisamment efficaces pour conserver l'aspect temps réel nécessaire à la simulation de milliers de particules évoluant dans des maillages qui se déforment.

Il faut toutefois noter que notre approximation topologique a un coût en terme de précision. En effet, plus la déformation est grande, plus l'approximation topologique sera éloignée de la réalité géométrique du maillage. Cela mène à des incohérences entre la position géométrique et le brin visé dans la topologie.

Par exemple, une particule peut se trouver géométriquement dans un volume et viser un brin appartenant au volume d'à côté. Ces erreurs n'ont pas de répercution importante dans nos applications, où nous nous intéressons à un voisinage dans son ensemble. En revanche, elles pourraient être gênantes si l'on recherche une cohérence parfaite entre la géométrie et la topologie au niveau des particules.

Troisième partie

Applications

Chapitre 8

Application 2D : la simulation de foule

Sommaire

Dans ce chapitre, nous allons présenter une application de nos travaux de recherche dans le domaine de la simulation de foule. Tout d'abord nous présenterons l'application et les challenges qu'elle représente. Puis nous réaliserons un rapide état de l'art des travaux sur le comportement d'agents autonomes dans les simulations de foule. Nous présenterons enfin les solutions apportées dans cette application, ainsi que quelques résultats pratiques, avant de conclure.

8.1 Présentation du sujet

La simulation de foule est un outil important dans la production de mondes virtuels pour les jeux vidéos, l'architecture ou encore l'urbanisme. Nous rappelons que nous traitons ici de simulation discrète (voir chapitre 1). Dans une simulation de foule, des entités autonomes appelées « agents » évoluent dans un environnement virtuel. Chaque agent a un objectif ou une série d'objectifs dans l'espace virtuel qu'il vise à atteindre. Cette autonomie est considérée comme nécessaire pour produire des comportements réalistes dans les simulations de foule [33]. Tout en suivant leur chemin vers cet objectif, les agents virtuels doivent adapter leur trajectoire afin d'éviter à la fois les obstacles fixes de l'environnement et les autres agents de la simulation.

La plupart des simulations actuelles réduisent les agents à leur boule englobante pour faciliter la détection de collisions. Dans [16] est présentée la première utilisation de notre structure de cartes combinatoires appliquée à la simulation de foule. Ces travaux portent sur la simulation d'agents dits « ponctuels », car assimilés à des points ou des cylindres de taille négligeable par rapport à la simulation. Cette réduction permet de représenter des foules d'agents de même type de manière efficace, en utilisant la notion d'enregistrement de points présentée dans la partie 5.1.

À chaque agent est ainsi associée une particule qui vise toujours un brin du maillage représentant l'environnement. En 2D, ce brin définit un triangle de prédiction pour le mouvement de la particule, comme décrit dans le chapitre 4. Les agents sont enregistrés dans les cellules de la carte, et lorsqu'un agent change de cellule, sa particule le détecte et déclenche la mise à jour de son enregistrement. Pour optimiser la recherche de voisins, l'enregistrement est dupliqué. Une cellule contient ainsi deux vecteurs : un premier contenant les agents présents dans la cellule et un second contenant les agents présents dans les cellules voisines.

Nos travaux permettent d'étendre le domaine de représentation aux agents non-ponctuels et d'affaiblir la contrainte de convexité des cellules de l'environnement. Les agents non-ponctuels sont modélisés par un maillage surfacique avec une forme et une topologie arbitraires (figure 8.1).

À chaque agent est associée une cage polyédrique qui l'englobe et qui est utilisée pour contrôler son animation. Les empreintes au sol des cages sur la surface de l'environnement forment des polygones éventuellement concaves qui sont utilisés pour représenter l'agent au niveau des enregistrements de détection de collision. Le terme *agent polygonal* référera par la suite à un tel polygone.

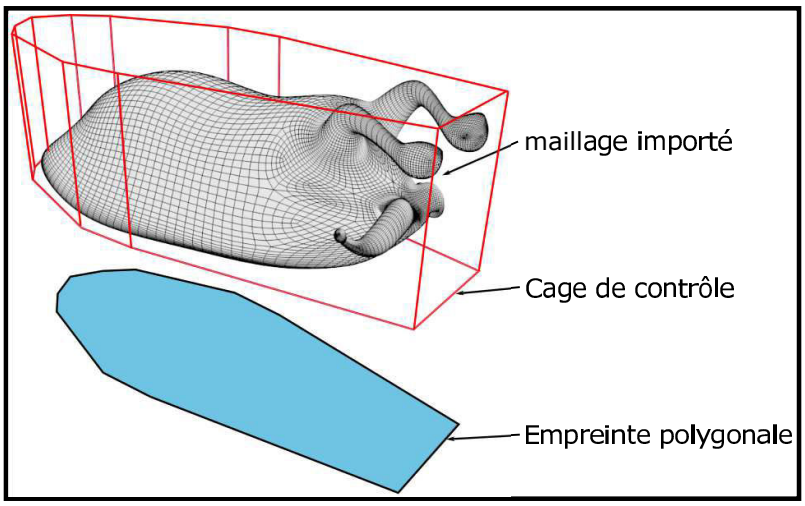

Figure 8.1 : Modèle de représentation d'un agent non-ponctuel

L'objectif principal de cette application concerne la gestion des interactions entre des agents ponctuels et polygonaux se déplaçant dans des environnements dynamiques. Ces derniers peuvent être des environnements urbains, complexes et non-planaires, qui contiennent des obstacles fixes (comme des bâtiments, des ponts, des routes, des arbres, ...). Le travail présenté est une extension de [16] qui a introduit une structure unifiée pour la simulation de foule. Cette structure est un modèle de cartes combinatoires multirésolutions utilisées à la fois pour la représentation et le rendu de l'environnement, le suivi des agents et les requêtes de proximité.

La figure 8.2 représente un exemple de scénario que nous désirons simuler. L'environnement est représenté grâce à une décomposition cellulaire dont les éléments (faces ou arêtes) peuvent être marqués en tant qu'obstacles. Les agents ponctuels et polygonaux suivent leur chemin tout en évitant les autres agents et les obstacles de l'environnement. Les travaux sur cette application ont été présentés à la conférence *Computer Animation and Social Agents* (CASA) en 2014 et ont donné lieu à la publication de l'article [26].

La principale contribution de cet article est l'enregistrement adaptatif d'agents ponctuels et polygonaux. Cet enregistrement borne le nombre d'agents considérés en interaction à chaque pas de temps. La multirésolution permet à chaque agent de parcourir les agents et obstacles voisins en temps constant, même en cas de forte densité. Elle permet ,d'une part, d'effectuer des requêtes très localisées, indispensables pour la simulation temps réel de milliers d'agents ponctuels. D'autre part, elle facilite l'anticipation longue distance des grands agents polygonaux qui peuvent avoir une grande vitesse de déplacement.

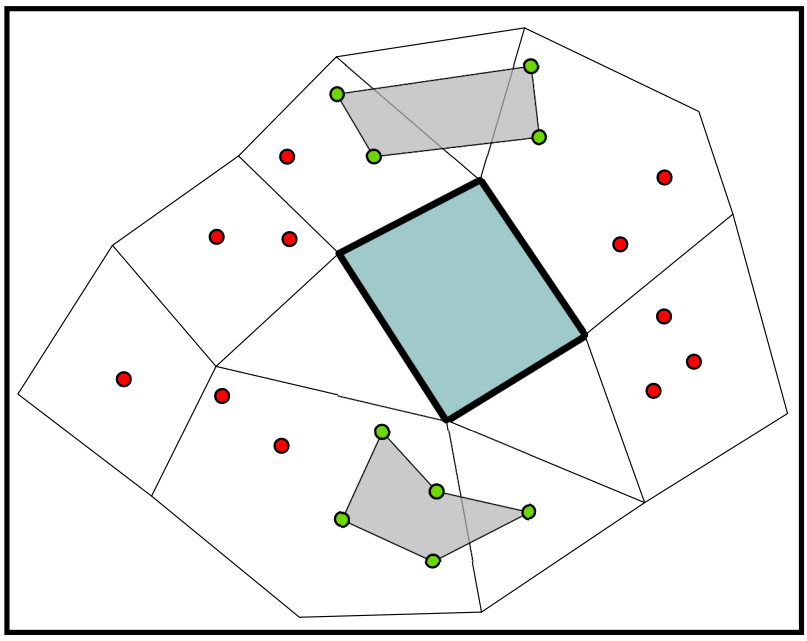

Figure 8.2 : Exemple de scénario de simulation de foule à représenter.

Nos travaux ont donc couvert la partie de la détection de collision portant sur les requêtes de proximités. Intéressons en premier à la seconde tâche de la détection de collision, qui concerne le comportement des agents résultant de ces requêtes.

8.2 Le Comportement

Le comportement des agents peut être décomposé en deux parties.

La première implique les forces qu'exercent les agents les uns par rapport aux autres, c'est à dire les forces externes. Nous étudierons les différentes solutions proposées par la communauté pour répondre à ce problème d'interaction.

La seconde concerne la gestion de la réception de ces forces, tout particulièrement dans le cas des agents polygonaux. En effet, la réponse d'agents ponctuels est triviale, alors que les agents polygonaux doivent posséder un système de forces interne. Ce système permet à la fois leur déformation éventuelle et la conservation de leur forme originelle. le problème de gestion des forces internes est résolu de manière élégante et efficace par la technique du « shapematching », présentée par M. Müller dans [23].

Voyons maintenant un bref état de l'art concernant le problème de gestion des forces externes des agents dans les simulations de foules. Ensuite, nous détaillerons les solutions que nous avons retenues et les optimisations que nous avons proposées.

8.2.1 État de l'art sur le comportement externe

De nombreux travaux s'intéressent au comportement externe d'agents dans les simulations de foules. Ces travaux se distinguent par leurs objectifs variés en termes de résultats comportementaux. Si certains s'intéressent particulièrement au réalisme de la simulation en essayant de reproduire au mieux des comportements humains, d'autres visent à optimiser les trajets, ou encore éviter les congestions.

Le réalisme : Les comportements humains recherchés vont des interactions, telles que des mouvements de panique, aux organisations naturelles de la foule, telles que des rangées de circulation ou des attroupements. [2] et [33] proposent un aperçu accessible et synthétique des méthodes existantes.

Dans [5], les auteurs évaluent l'impact de différents comportements sur le scénario d'évacuation d'un bâtiment. Les agents sont considérés comme ayant des caractéristiques personnelles qui vont les faire se regrouper avec les agents partageant les mêmes caractéristiques lors de l'évacuation.

L'optimisation : De manière générale, il existe deux niveaux de programmation pour un agent.

Un niveau global permet de calculer le chemin optimal que doit suivre l'agent dans l'environnement pour atteindre son objectif. Ce niveau ne prend en compte que les contraintes environnementales sans prendre en compte les autres entités mobiles.

Un second niveau local est utilisé pour gérer les collisions éventuelles le long du chemin suivi.

Afin d'optimiser les trajets ou d'éviter les congestions, des considérations plus globales sont parfois requises. Dans [14], les auteurs utilisent un niveau supplémentaire qui considère des groupes d'agents ayant une même direction. Ces groupes sont utilisés pour être évités comme des entités de taille supérieure. Cela permet d'éviter qu'un agent en contresens ne traverse un tel groupe, créant ainsi des ralentissements.

L'ensemble de ces travaux utilisent des techniques permettant la réponse en cas de collision. Ces techniques peuvent être classées en trois catégories :

— Les règles d'évitement
— L'évitement selon la vitesse
— Les champs de forces

Les méthodes basées sur des règles d'évitement

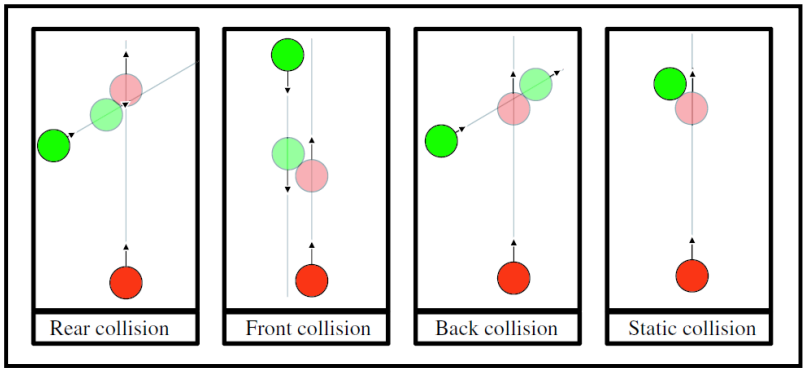

Figure 8.3 : Différentes collisions possibles.[19]

Les méthodes basées sur des règles d'évitement sont utilisées dans beaucoup d'applications visant à représenter des humains virtuels. Leur principe est de lister un ensemble de situations possibles de collision et, pour chaque situation, de proposer une réponse d'évitement. Ainsi, lorsqu'une collision est imminente, on va pouvoir déterminer, via une base de règles, l'évitement à adopter.

Ces méthodes provoquent des comportements considérés comme proches de la réalité, ce qui est intéressant pour les applications en recherche de réalisme. La base de règles de comportement permet d'ajuster simplement les paramètres d'évitement, ce qui rend intuitif le contrôle de la simulation. Leur défaut est intrinsèque à la base de règles utilisée. Cette base est finie et comme la même base est utilisée pour toutes les entités, des artefacts peuvent apparaître avec des situations de blocage. De plus, plus la base est riche, plus les situations vont être difficiles à prédire. Cette technique est utilisée directement dans [19] et [30]. La figure 8.3 présente les quatre situations de collision envisagées dans [19].

Dans [21], une grille régulière au sol est utilisée pour stocker les différents chemins et comportements à adopter lorsqu'un agent atteint une cellule.

Les méthodes basées sur la vitesse

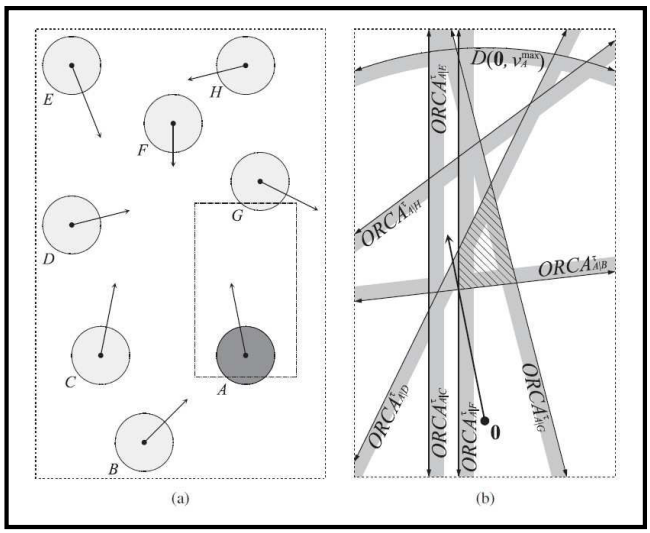

Figure 8.4 : Le calcul des ORCA lines et la zone sûre résultante.[37]

Les méthodes basées sur la vitesse sont apparues dans le domaine de la robotique et ont été popularisées par l'article [8]. Cette méthode robuste permet des calculs précis et rapides pour le déplacement sans collision d'un robot dans un environnement comprenant des obstacles mobiles. Le principe est de calculer et d'éviter les zones de l'espace qui amènent à une collision. Ce calcul est réalisé en prenant en compte la vitesse actuelle du robot et la vitesse des obstacles proches. À chaque pas de temps, le vecteur vitesse du robot est mis à jour afin d'éviter les obstacles.

La méthode a ensuite été étendue au cas multi-agent dans [12]. La notion de « Reciprocal Velocity Obstacle » (RVO) encore très utilisée aujourd'hui y a été introduite. [37] reprend la méthode et la rend encore plus robuste. L'amélioration consiste à calculer des lignes d'évitement optimal (ORCA lines). Ces lignes découpent le plan pour ne laisser qu'une zone de déplacement sûre autour de chaque robot. Le vecteur vitesse du robot pour le pas de temps suivant est ensuite sélectionné dans cette zone sûre.

La figure 8.4 présente le calcul des ORCA lines pour un robot entouré de 7 autres robots. À gauche, la situation met en scène 8 robots et leur vecteur vitesse respectif. À droite, les ORCA lines entre le robot A et les autres robots découpent le plan pour ne laisser que la zone rayée comme zone sûre pour le déplacement du robot A.

Cette méthode permet de garantir une simulation sans collision, à condition que chaque entité mobile utilise le même système de calcul d'évitement. Elle permet de gérer correctement les situations même en cas de forte densité. La représentation est toutefois limitée à des agents circulaires et ne permet pas de gérer différents types d'agents de manière aisée.

Les méthodes basées sur des champs de forces

Les méthodes basées sur des champs de forces ou champs de potentiel appliquent des forces d'attraction et de répulsion sur les agents.

Dans [11], ce type de méthode est présenté pour la première fois dans le cadre de la simulation de foule. On associe à chaque agent un champ de force qui l'entoure et qui repousse les autres agents lorsqu'ils y pénètrent. De la même manière, on associe un champ de répulsion aux obstacles de l'environnement et chaque agent est attiré par son objectif. Il suffit de faire la somme des forces appliquées à l'agent, et de réaliser une intégration d'Euler, pour obtenir la nouvelle position de l'agent et le faire se déplacer vers son objectif en évitant les collisions. La méthode est souvent couplée avec un « path planning » qui décompose la trajectoire à suivre en une série d'objectifs à atteindre. Cela permet le déplacement dans des environnements qui n'ont pas de chemin en ligne directe, comme des labyrinthes.

[13] améliore les trajectoires des agents grâce au Principe du Moindre Effort, ce qui permet d'avoir des résultats plus réalistes. Les résultats présentent des calculs rapides pour un grand nombre d'agents et des comportements émergents, tels que la formation de lignes de circulation et la formation d'attroupements. Il faut toutefois se méfier des équilibres possibles des forces qui peuvent apparaître au cours de la simulation, notamment en cas de forte densité. On introduit de petites perturbations aléatoires de la trajectoire qui suffisent à débloquer ces situations.

8.2.2 Solutions retenues et optimisations

Bulles de potentiel

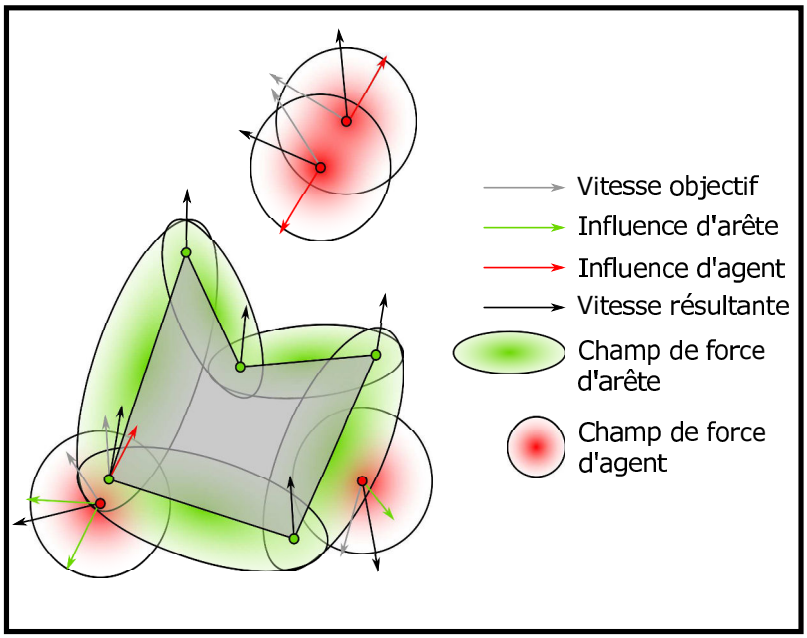

Figure 8.5 : Les forces appliquées aux agents.

Nous avons choisi de reprendre la méthode basée sur des champs de forces pour sa rapidité et son extension aisée à des agents non-ponctuels. La seconde contribution présentée dans [26] est donc la définition de champs de forces sur des formes polygonales utilisées pour calculer les interactions entre agents.

La figure 8.5 présente un exemple de forces appliquées à des agents ponctuels (en rouge) et polygonaux (en gris). Nous avons étendu à des arêtes les champs de potentiel originellement appliqués à des entités ponctuelles ou circulaires. La forme du champ résultant pour les arêtes est arbitraire, mais nous avons choisi des ellipses. Ces dernières permettent de calculer les forces par une simple somme de deux distances point à point et sont donc très peu coûteuses en terme de calcul. Les forces résultantes contraignent les agents à s'écarter les uns des autres et à s'éloigner des obstacles fixes.

Nous avons choisi, par volonté de réalisme, de créer une dissymétrie entre agents ponctuels et agents polygonaux. Ainsi, les agents ponctuels sont considérés comme trop petits pour influencer les agents polygonaux.

En revanche, nous avons permis les interactions entre agents polygonaux, notamment dans le cas d'agents déformables. La force d'interaction appliquée a un agent situé à une distance d est une force de type élastique dont voici l'équation :

$$F = k(R - d)^2 - z\dot{d} \tag{8.1}$$

$\dot{d}$ est la dérivée première de la distance, k est la raideur de la répulsion, z est l'amortissement et R le rayon d'influence. Aucune force n'est appliquée au delà de R.

Nous proposons une évaluation adaptative de ces forces, compatible avec les scénarios denses et les simulations interactives. Chaque agent ponctuel ou polygonal prend en compte les forces générées par ses voisins (d'autres agents ou obstacles). Notre structure sélectionne naturellement les arêtes des agents polygonaux qui contribuent dans ces champs de forces, évitant des calculs inutiles. En effet, les voisins dont on va calculer l'influence, sont dans le One-Ring des cellules qui contiennent un agent.

N'importe quel autre modèle physique peut être utilisé, à condition que les forces de répulsion restent cohérentes avec notre système d'enregistrement.

Shapematching

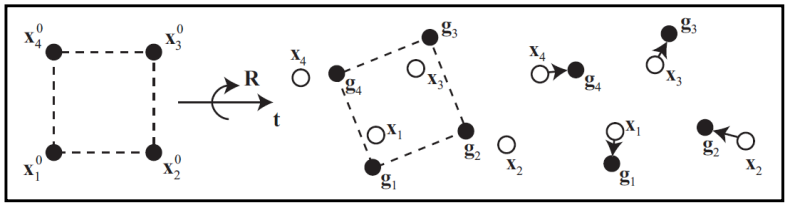

Figure 8.6 : La procédure du shapematching.[23]

Concernant la gestion de la déformation des agents, nous avons retenu la méthode du Shapematching présentée par M.Müller dans [23]. Cette méthode permet de maintenir la forme d'un ensemble de point qui subit des déformations durant une simulation. Les algorithmes proposés permettent de gérer l'élasticité et la plasticité avec laquelle la forme d'origine est maintenue. Le fait de considérer uniquement des points permet de s'extraire du concept de maillage et de simplifier le modèle.

La figure 8.6 présente l'algorithme du shapematching. En partant d'un ensemble de points au repos $x^0_{1...4}$ et de ce même ensemble déformé $x_{1...4}$, on calcule la rotation R de la forme de repos qui correspond le plus à la forme actuelle. À partir de cette rotation optimale rigide, on peut appliquer un facteur de plasticité β qui va déterminer un ensemble d'objectifs $g_{1...4}$ a atteindre.

Un second paramètre α d'élasticité nous permet de définir à quelle vitesse les points $x_{1...4}$ vont converger vers leurs objectifs respectifs $g_{1...4}$. On termine ainsi avec un vecteur vitesse à appliquer à chaque point pour retourner vers sa forme d'origine.

Optimisation : anticipation des objets rapides

Lors de nos tests, nous avons remarqué que lors de situations de forte densité, il arrivait que les agents ponctuels n'aient pas le temps d'éviter les agents polygonaux trop rapides. Cela est dû à la corrélation directe entre enregistrement et visibilité. Étant donné qu'un agent ne « voit » que dans son One-Ring et que l'environnement est subdivisé en fonction de la densité, les situations de forte densité font que les agents anticipent peu car leur One-Ring est réduit.

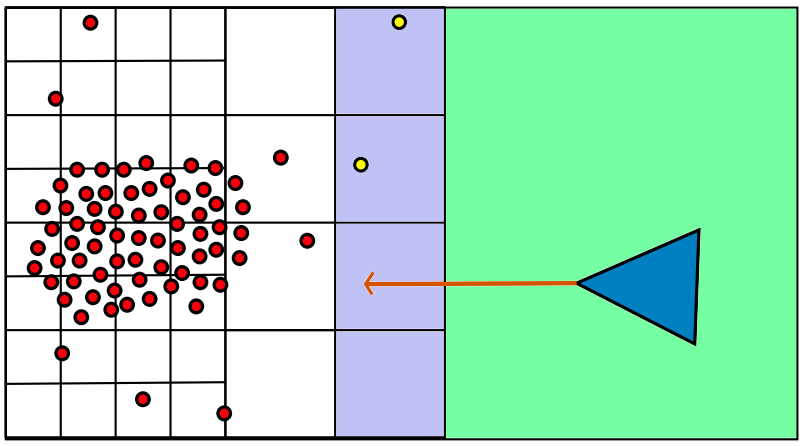

Figure 8.7 : Un agent polygonal se dirige rapidement vers une foule compacte.

Lorsque l'on est au centre d'une foule compacte, on ne voit pas à distance et on ne peut donc pas anticiper. Seuls les agents en bordure de la masse compacte peuvent initier un mouvement de foule. Ce phénomène peut être considéré comme réaliste. Cela dit, les agents en périphérie ont souvent une visibilité trop restreinte pour créer les mouvements de foule nécessaire au passage de l'agent polygonal.

La figure 8.7 présente une de ces situations où un agent polygonal (en bleu) se dirige rapidement (vecteur vitesse en orange) vers une foule d'agents ponctuels compacte. Les cellules sur fond vert contiennent l'agent polygonal et celle sur fond bleu constituent son One-Ring. Les agents en rouge ne détectent pas l'agent polygonal et ne le prendront pas en compte pour leur calcul de trajectoire. Les agents en jaune détectent l'agent et peuvent se préparer à l'éviter.

Notre solution consiste à étendre l'enregistrement d'un agent polygonal en fonction de sa vitesse. Nous enregistrons donc pour chaque agent polygonal une arête virtuelle supplémentaire, dont les extrémités sont son barycentre (ou le barycentre de sa « tête ») d'une part et la pointe de son vecteur vitesse d'autre part.

La figure 8.8 présente l'extension de l'enregistrement. On peut voir que les agents perçoivent l'agent polygonal de manière plus anticipée. Les agents en rose sont trop proches de l'arête virtuelle et sont donc directement sur la route de l'agent polygonal. Nous leur avons ajouté un effet de panique qui donne priorité à l'évitement de cet agent et leur permet une accélération soudaine.

Cette solution est rapide à mettre en place et assez intuitive. Comme l'arête virtuelle possède une bulle de potentiel comme les autres arêtes, elle permet

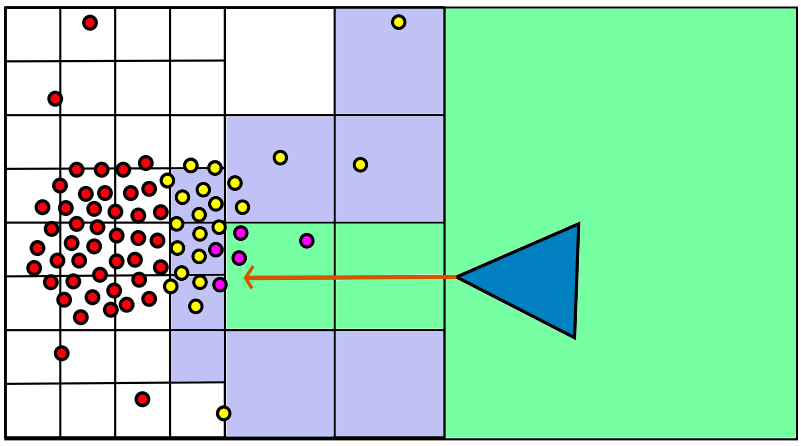

FIGURE 8.8 : Enregistrement étendu de l'agent polygonal.

de créer une percée dans la masse compacte d'agents.

Dans le cas d'agents polygonaux très larges dans la direction orthogonale à leur déplacement, ce système ne suffit pas car la percée créée est trop étroite. Pour de tels agents, une solution est d'enregistrer 2 arêtes pour former une pointe permettant une percée suffisamment large.

8.3 Expérimentations et résultats

Nous ne détaillerons pas ici les résultats présentés dans [16] qui comparent notre structure de représentation aux autres structures existantes, telles que les kd-tree et le spatial hashing [32].

Ces résultats demeurent les mêmes pour les agents ponctuels. Concernant les agents polygonaux, il n'existe pas à notre connaissance de simulation d'agents polygonaux, en dehors de travaux sur la robotique, qui ne consiste pas à réduire ces agents à une boule englobante. Ces travaux de robotique ne sont pas utilisables dans notre cadre car il s'agit de calcul très précis sur quelques robots, voire un seul comme dans [10]. Nous allons donc nous intéresser aux changements qu'apportent les agents polygonaux en terme de complexité aux résultats présentés dans [16], puis nous verrons quelques scénarios de tests que nous avons réalisé pour valider nos travaux.

8.3.1 Complexité temporelle

Évaluons la complexité temporelle pour la construction, la mise à jour et l'accès à notre structure de données. Pour cela, notons :

— n_a le nombre d'agents ponctuels.
— n_p le nombre total d'arêtes d'agents polygonaux.
— n_f le nombre total d'arêtes d'obstacles fixes.
— n_e le nombre moyen de cellules où est enregistré un agent polygonal.
— n_n le nombre moyen de voisins visibles par un agent (lié à la densité).

Le nombre d'entités à enregistrer est $n = n_a + n_p + n_f$. Nous comparons notre approche multirésolution (MR) avec les méthodes utilisant des arbres (kd-trees, BSP),des diagrammes de Voronoï et des grilles d'enregistrement régulières (spatial hashing) présentées dans la partie 2. La construction ou la mise à jour de la structure à chaque pas de temps a une complexité de :

— $O(n.log(n))$ pour les arbres et les diagrammes de Voronoï.
— $O(n_a + n_p)$ pour les grilles d'enregistrement dont MR.

En pratique, peu d'agents changent de cellule à chaque pas de temps. Ainsi, moins de 10% des agents nécessitent une mise à jour à chaque pas de temps. La mise à jour de la liste d'agents d'une cellule s'effectue en $O(n_n)$. Les requêtes de proximité permettant de récupérer les voisins pour un agent s'effectuent en :

— $O(n_n.log(n))$ pour les arbres.
— $O(n_n)$ pour les diagrammes de Voronoï.
— $O(1)$ pour les grilles d'enregistrement dont MR.

La complexité temporelle globale de notre méthode et des méthodes basées sur des grilles d'enregistrement est en $O(n)$ contre $O(n.log(n))$ pour les méthodes classiques. Nous nous démarquons de ces grilles par notre liberté de représentation et notre adaptabilité par rapport aux situations denses. De plus, en cas de forte densité, les autres méthodes nécessitent de filtrer les agents les plus proches qui seront sélectionnés pour l'évitement. Cette opération s'effectue en $O(n_n)$ pour chaque agent. Notre structure adaptative effectue naturellement ce filtrage, permettant de considérer seulement les agents du voisinage dans l'évitement.

8.3.2 Scénarios

Nous avons évalué nos travaux sur divers scénarios de simulation de foule. Dans ces scénarios, des agents ponctuels évoluent avec des agents polygonaux qui peuvent être des agents simples rigides (voitures rouges 8.9, 8.10), des agents clusterisés en sous-sections polygonales rigides (serpents, trains 8.12) et des agents déformables (limaces jaunes 8.13, 8.14, 8.15). Chaque scénario présente un intérêt particulier, que ce soit en terme de taille, de complexité de l'environnement ou de densité d'agents.

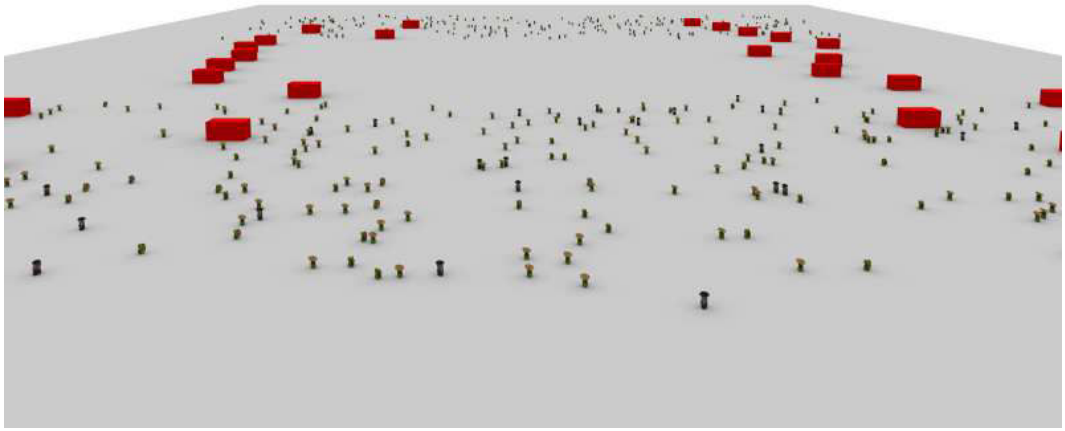

Figure 8.9 : Les différents scénarios de tests : Le couloir.

La figure 8.9 présente le scénario du « couloir » où des agents ponctuels sont divisés en deux groupes de part et d'autre d'une carte rectangulaire. Chaque groupe d'agents doit traverser la carte pour se rendre au côté opposé. Des agents polygonaux rigides en rouge sont divisés en deux groupes et effectuent des va-et-vient dans la largeur pour gêner la traversée des agents ponctuels.

Ce scénario permet de tester le comportement d'agents ponctuels confrontés à des obstacles mobiles, tout en créant une zone de forte densité au milieu de la carte lorsque les deux groupes d'agents se rencontrent.

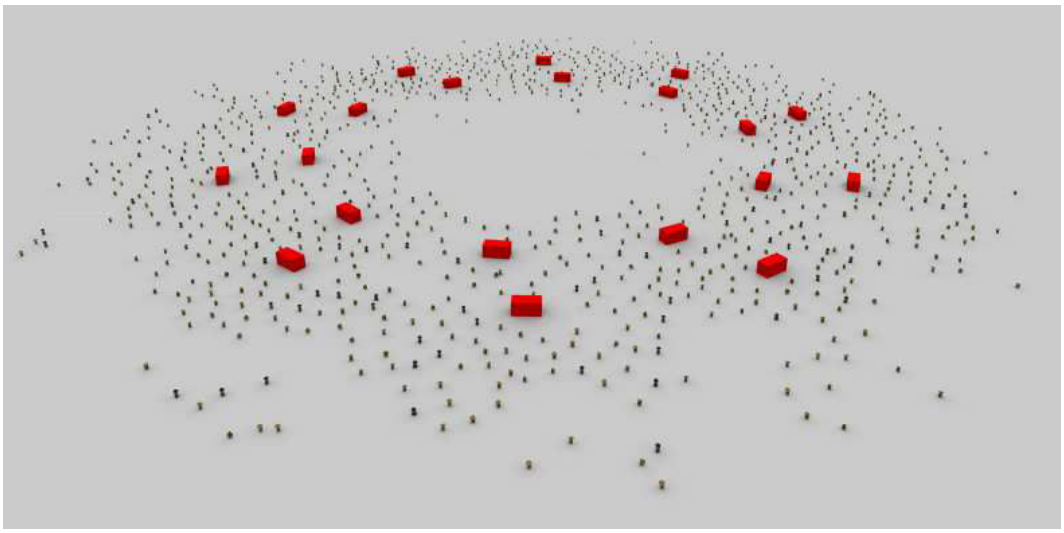

Figure 8.10 : Les différents scénarios de tests : Le cercle.

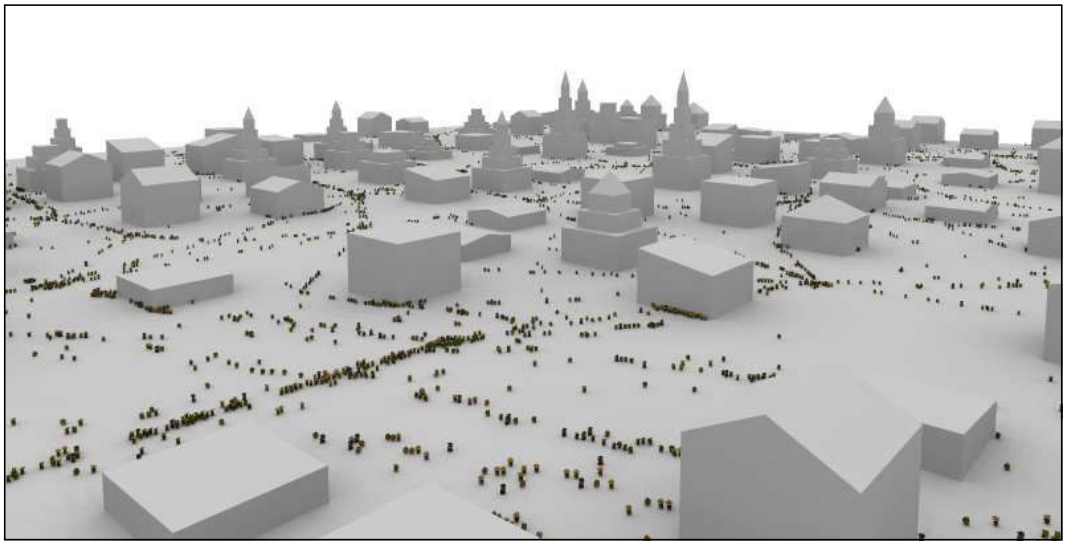

Figure 8.11 : Les différents scénarios de tests : La ville.

La figure 8.10 présente le scénario du « cercle », où des agents ponctuels sont répartis le long d'un très grand cercle. Chaque agent ponctuel doit traverser l'environnement pour aller au point diamétralement opposé. Des agents polygonaux rigides en rouge sont divisés en deux groupes, formant deux cercles concentriques de différentes tailles . Chaque agent polygonal tourne le long du cercle sur lequel il se trouve pour gêner la traversée des agents ponctuels. Les deux cercles d'agents polygonaux tournent dans des sens opposés.

Ce scénario permet de tester le comportement des agents ponctuels confrontés à des obstacles mobiles tout en créant une zone de forte de densité au milieu de la carte lorsque tous les agents ponctuels se croisent. Ici, la densité varie énormément durant la simulation. La présence de véhicules crée des flots de circulation d'agents ponctuels qui se faufilent entre les agents polygonaux.

La figure 8.11 présente un scénario de « la ville » où des agents ponctuels sont répartis dans une ville. Chaque agent a une liste d'objectifs à atteindre tirés au hasard sur la carte parmi une liste de points d'intérêts.

Ce scénario permet de tester le comportement des agents ponctuels confrontés à des obstacles fixes, et de vérifier la non-dépendance de nos algorithmes à la taille de l'environnement. D'autres scénarios urbains ont été créés en utilisant OpenStreetMap [24]. Une fois que les zones de circulation possible ont été calculées pour les piétons et les voitures, ce genre d'environnement nous permet d'obtenir un rendu réaliste d'une foule évoluant dans une ville réelle.

La figure 8.12 présente le scénario du « serpent », où des agents ponctuels en rouge sont amassés dans une zone de forte densité au centre de la carte. D'autres agents clusterisés en sous-sections polygonales rigides suivent une trajectoire qui leur fait traverser la foule en faisant varier leur vitesse. Ce scénario nous a permis de tester l'ajout comportemental d'anticipation que nous avons mis en place, ainsi que la réaction d'agents ponctuels face à de très grands agents polygonaux potentiellement rapides.

Les figures 8.13, 8.14 et 8.15 présentent respectivement les scénarios de la planète, du noeud torique et des patatoïdes. Dans ces scénarios, des agents ponctuels et des agents déformables (limaces) évoluent dans des environnements non-planaires comportant des bâtiments fixes. Ces scénarios mettent en avant la liberté de représentation fournie par notre structure et valident le suivi de particule en 2D surfacique. Tous les scénarios avec obstacles fixes et trous dans la carte nécessitent d'utiliser des algorithmes de recherche de chemin pour les trajectoires des agents.

FIGURE 8.12 : Les différents scénarios de tests : Le serpent.

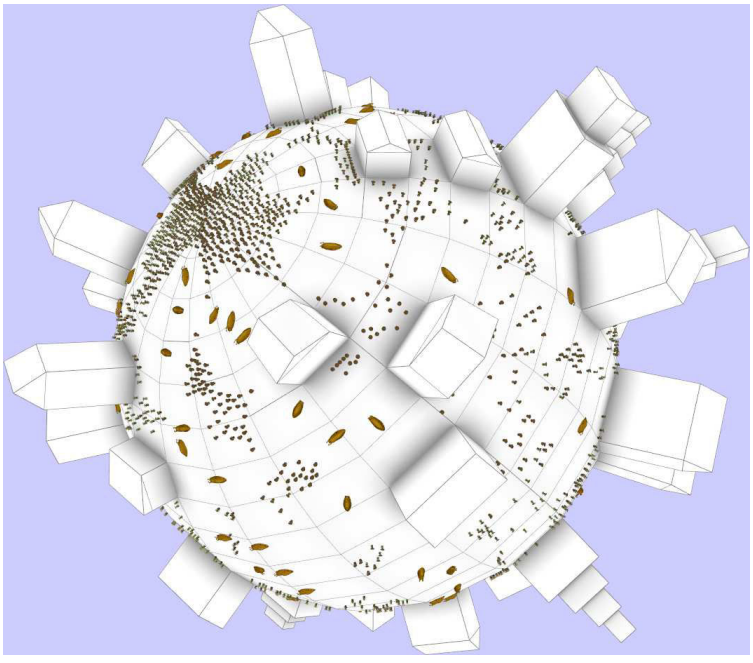

FIGURE 8.13 : Les différents scénarios de tests : La planète.

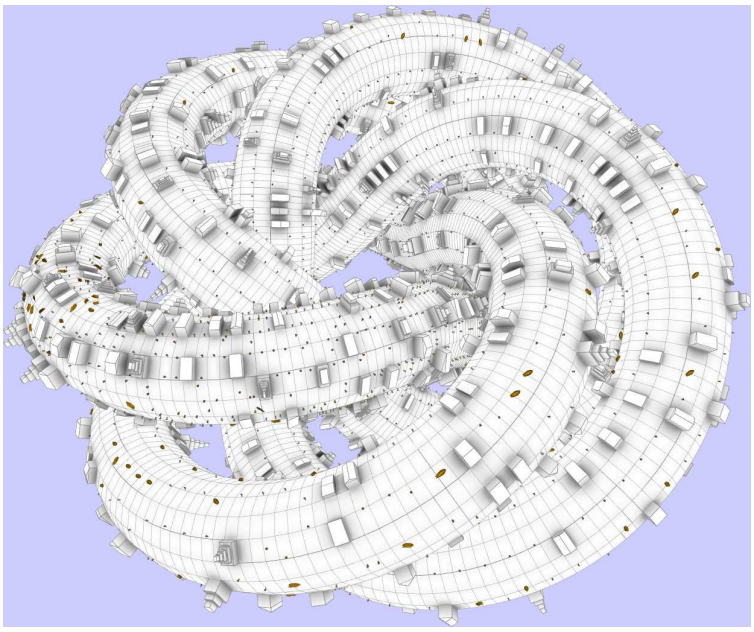

Figure 8.14 : Les différents scénarios de tests : Le noeud torique.

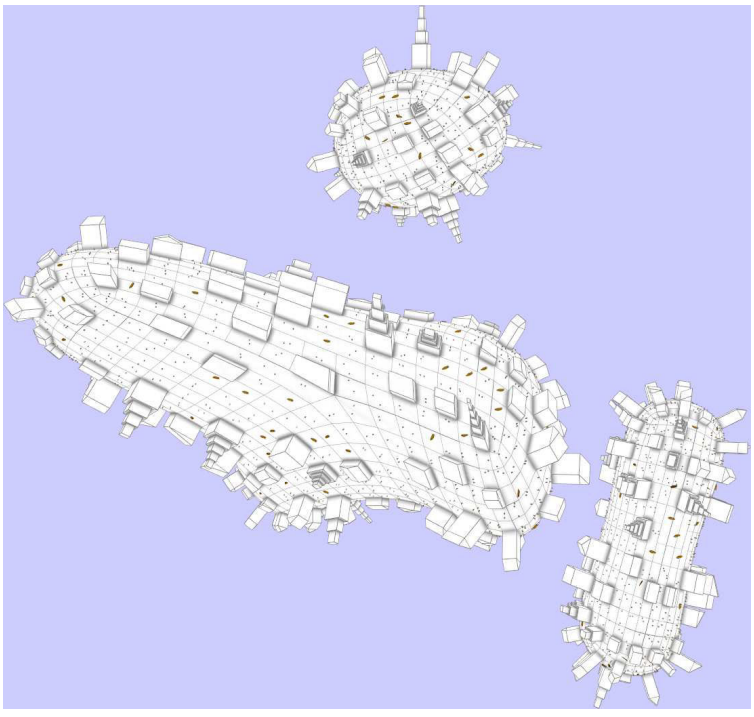

Figure 8.15 : Les différents scénarios de tests : « Les patatoïdes ».

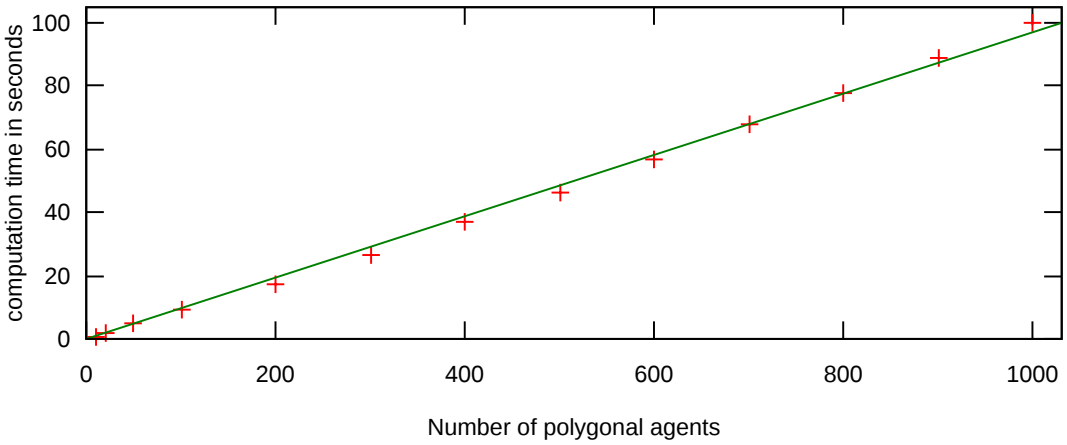

Figure 8.16 : Évolution du temps de calcul en fonction du nombre d'agents polygonaux sur le scénario du couloir.

8.3.3 Résultats

Nous présentons ici un ensemble de résultats recueillis sur nos scénarios de tests afin d'évaluer l'impact des agents polygonaux sur les performances du dispositif présenté dans [16].

La figure 8.16 présente le temps nécessaire à l'exécution de 10 000 pas de temps sur le scénario du couloir en fonction du nombre d'agents polygonaux présents. Le nombre d'agents ponctuels est fixé à 1 000 afin de constater uniquement l'impact des agents polygonaux. On peut en déduire que le temps de calcul augmente linéairement avec le nombre d'agents polygonaux présents.

En moyenne, les requêtes de proximité sont donc exécutées en temps constant même si plus d'agents sont présents dans la scène. Cela confirme l'efficacité de notre approche multirésolution pour borner le nombre de voisins localement.

La figure 8.17 présente le coût des différentes tâches de la simulation en fonction du nombre d'agents sur le scénario du cercle. La section de gauche est la section témoin avec 500 agents ponctuels et 50 agents polygonaux. Dans la section du centre, on a doublé le nombre d'agents ponctuels par rapport au témoin. Dans la section de droite, on a doublé le nombre d'agents polygonaux par rapport au témoin.

On remarque tout d'abord que la gestion de la multirésolution en rouge a un coût négligeable face aux autres tâches.

Dans ce scénario, le coût du comportement en turquoise concerne unique-

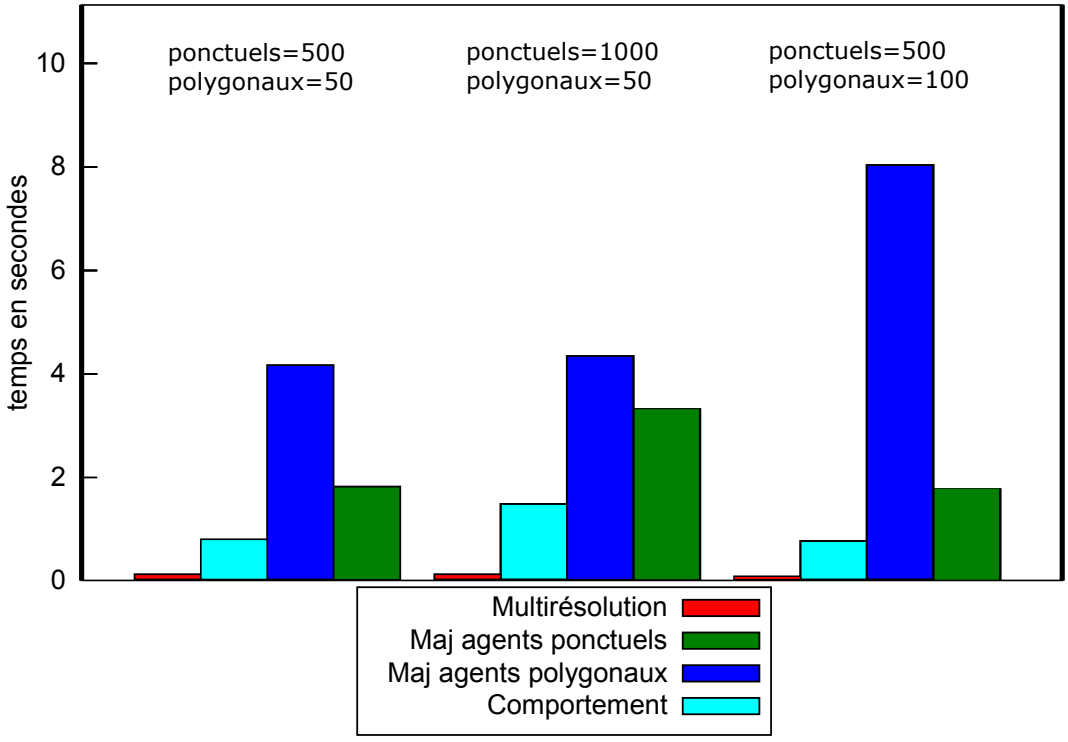

FIGURE 8.17 : Coût des différentes tâches en variant le nombre d'agents sur le scénario du cercle.

ment les agents ponctuels, car les agents polygonaux ne font que tourner en rond sans jamais se croiser. Il est donc normal que ce coût suive l'évolution du nombre d'agents ponctuels, en doublant dans la section du centre. Dans la section de droite, il n'augmente pas avec le nombre d'agents polygonaux et a même légèrement baissé. En effet, comme plus d'agents polygonaux sont à éviter sur le chemin du centre du cercle, les agents ponctuels traversent au goutte à goutte et moins d'agents se retrouvent en même temps au centre de la scène. La densité étant réduite, les coûts en calculs le sont aussi.

Concernant les coûts de mise à jour des agents, on observe de nouveau la linéarité avec le nombre d'agents : lorsque le nombre double, le coût double aussi. Ces coûts représentent les enregistrements des agents mais aussi leurs calculs de voisinage. Ces calculs de voisinage consistent à sélectionner les voisins les plus importants qui seront utilisés pour le calcul du comportement.

On peut également déduire que la gestion d'un agent polygonal rigide de type pavé revient au même coût que la gestion de 20 agents ponctuels. Ces agents polygonaux possèdent 4 particules et 5 arêtes, dont une arête virtuelle d'anticipation. Le surcoût réside principalement dans l'enregistrement et le suivi d'arêtes.

Évaluons maintenant l'impact sur notre dispositif multirésolution.

La figure 8.18 présente le coût des différentes tâches selon la décomposition de l'environnement utilisée sur le scénario du couloir, avec 1000 agents ponctuels et 50 agents polygonaux.

Les deux premières sections ont été réalisées avec des grilles régulières. La première a un maillage grossier, c'est à dire des grandes cellules. La seconde a un maillage fin, soit des petites cellules.

La dernière section (à droite) représente les résultats avec nos maillages multirésolutions. Dans cette section, la taille des cellules peut varier entre la grande taille utilisée pour la première section et la petite taille utilisée pour la deuxième section. Cette variation s'effectue en fonction de la densité d'agents ponctuels.

Avec des cellules grossières, le coût concernant la mise à jour des agents ponctuels est très grand car chaque agent a beaucoup de voisins à vérifier pour effectuer ses calculs de voisinage. Le coût concernant la mise à jour des agents polygonaux est très faible car, comme nous l'avons vu, il réside principalement dans l'enregistrement. Les cellules étant grandes, les agents n'ont pas à mettre à jour souvent leur enregistrement.

Dans un scénario avec des petites cellules, le coût des agents ponctuels est grandement réduit, car leur voisinage est très limité. Par contre, le coût des agents polygonaux est augmenté car ils doivent sans arrêt vérifier leur enregistrement comme ils sont dans des cas pluricellulaires (cf section 5.4). On note

également que le coût du comportement est diminué, car les agents ponctuels considèrent moins de voisins.

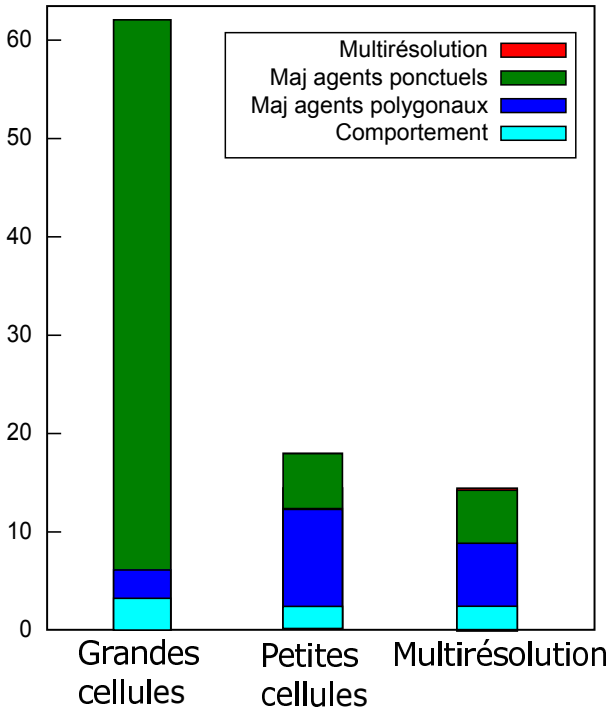

Figure 8.18 : Coût des différentes tâches selon la décomposition de l'environnement utilisée sur le scénario du couloir.

Notre dispositif multirésolution est plus performant ici, avec un coût négligeable pour la multirésolution en elle-même. Cet outil permet de profiter des avantages des deux autres modèles. Dans les zones où peu d'agents ponctuels sont présents, les cellules grossissent, ce qui réduit le coût associé aux agents polygonaux. Dans les zones de forte densité, les cellules se subdivisent, réduisant le coût associé aux agents ponctuels.

On remarque également lors des expériences qu'au moment où un agent polygonal arrive sur une zone de forte densité, sa capacité à repousser les agents ponctuels réduit localement la densité. Il ramène alors les cellules à une taille favorable à son propre enregistrement.

Cette variété de tailles de cellules adaptatives permet d'être optimal dans toutes les situations : Là où il y a trop d'agents, on diminue la charge de calcul et là où il y a peu d'agents, ceux-ci ne nécessitent pas d'être mis à jour régulièrement.

8.4 Conclusion

Nous avons réalisé un dispositif qui gère à la fois des agents ponctuels et des agents polygonaux en temps réel. Ces agents polygonaux sont créés à partir de formes arbitraires, et peuvent se déformer durant la simulation via leurs interactions avec l'environnement. Nous pouvons donc simuler des véhicules, des objets articulés, ainsi que des objets déformables en utilisant une cage 3D de simplification (souvent utilisée pour réaliser l'animation de personnages).

Nos simulations contiennent des milliers d'agents ponctuels et des centaines d'agents polygonaux, qui interagissent entre eux et avec l'environnement en temps réel. Ces interactions sont possibles grâce aux outils fournis par les cartes combinatoires multirésolutions dont nous avons démontré l'efficacité. Le processus multirésolution est contrôlé par la densité de la foule, et permet de réduire la complexité des requêtes de proximité quand la densité augmente tout en réduisant la fréquence de mise à jour des enregistrement quand la densité diminue.

Nous avons utilisé des champs de potentiel locaux pour réaliser l'évitement entre les agents, et adapté le modèle existant à nos agents polygonaux éventuellement concaves. Toutefois, notre modèle de représentation permet d'utiliser n'importe quelle autre technique d'évitement.

Une multitude d'ajouts comportementaux peut être envisagée, et la communauté scientifique continue de s'intéresser à ce problème complexe du comportement d'agents virtuels. Par exemple, dans nos simulations, les agents ont une perception à 360°. Dans [3], les agents possèdent un cône de vision limité en face d'eux afin de correspondre à la réalité et créer des comportements plus crédibles.

Dans notre modèle, le brin visé par un agent ponctuel définit naturellement un triangle de prédiction qui correspond à un cône de visibilité. Il serait facile de limiter la prise en compte des voisins aux faces adjacentes à ce triangle. L'enregistrement des agents ponctuels pourrait alors être placé sur les arêtes et non plus sur les faces de la carte. Ce changement pourrait accélérer les requêtes en récupérant facilement les agents allant dans la même direction.

[28] propose un critère d'évaluation intéressant pour la qualité en terme de réalisme d'une simulation de foule : le « crowd stalling » que l'on peut traduire par « paralysie de la foule ». Ce critère semble pertinent à explorer, afin de pouvoir mesurer le réalisme d'une simulation qui reste assez subjectif.

Chapitre 9

Applications 3D

Sommaire

Dans ce chapitre, nous allons présenter les extensions 3D de nos travaux. D'une part, notre exploration de la simulation de foule 3D. D'autre part, nos travaux réalisés en collaboration avec l'équipe MIMESIS d'Inria sur des simulations physiques pour la médecine.

9.1 Le challenge de la 3D

Nous allons discuter ici du challenge que représente le passage à la troisième dimension dans le contexte de la détection de collision.

Pour cela, nous allons nous placer du point de vue applicatif et examiner le problème à travers deux applications, qui sont similaires en terme de détection de collisions mais différentes en termes d'objectifs recherchés. Notre première application est une simulation de foule volumique très semblable à notre application 2D surfacique présentée dans le chapitre 8. Notre deuxième application, dans ce cadre d'extension volumique, est une simulation d'insertion d'aiguille pour des opérations de la jambe.

Voyons en détails ces deux applications dans leur cadre particulier.

9.1.1 La simulation de foule 3D : exploration

La simulation de foule 3D a pour but de simuler des agents dont le déplacement n'est pas contraint à un plan ou une surface. Les foules simulées peuvent être des oiseaux, des poissons ou encore des flux de particules servant à modéliser des fluides.

Ici, notre objectif est d'évaluer la viabilité de notre modèle pour réaliser de la simulation volumique de foule. Pour cela, nous avons étendu en 3D le scénario du cercle en répartissant des agents ponctuels uniformément sur une sphère. Diverses techniques sont présentées dans [17] pour réaliser cette répartition. La solution que nous avons choisie est d'affecter des coordonnées aléatoires aux particules sur une sphère, puis de leur appliquer des forces de Coulomb jusqu'à obtenir une situation proche de l'équilibre.

On peut voir cet état d'équilibre qui constitue la situation de départ de notre simulation de 500 agents ponctuels à gauche de figure 9.1. Sur cette figure, les agents sont colorés en fonction de leur vitesse maximale autorisée pour la simulation. Plus leur couleur tend vers le rouge, plus ils sont rapides. Comme pour le scénario du cercle, chaque agent doit se rendre à la position diamétralement opposée. L'espace est représenté par une carte multirésolution cubique qui se subdivise et se simplifie en fonction de la densité d'agents présents. Cette carte est affichée à droite de la figure 9.1,

Grâce aux enregistrements, les agents peuvent se détecter et s'éviter avec leurs bulles de potentiel. La figure 9.2 présente cet évitement dans la région centrale de la sphère. On peut voir que les agents les plus rapides sortent en premier de la foule d'agents centrale.

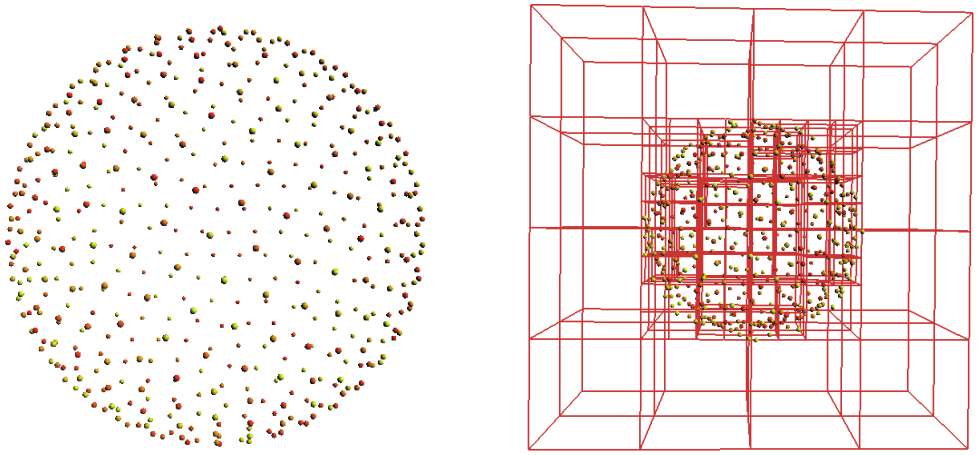

Figure 9.1 : À gauche : 500 Agents ponctuels répartis équitablement sur une sphère. À droite : La carte 3D se raffine et se simplifie en fonction de la densité d'agents.

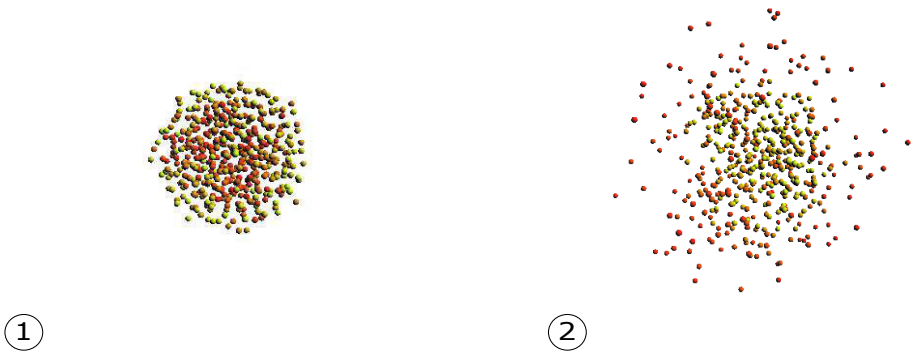

Figure 9.2 : Évitement des agents au centre de la sphère.

L'évitement et les enregistrements fonctionnent très bien en 3D, mais les performances ne sont pas satisfaisantes. En effet, les cartes multirésolutions volumiques sont encore en développement, et de nombreuses améliorations sont nécessaires afin de pouvoir supporter ce genre de simulations comportant un grand nombre d'entités très dynamiques.

Cela est dû d'une part au fait que les voisinages et le nombre de brins sont beaucoup plus grands en 3D qu'en 2D et d'autre part au fait que le processus de multirésolution est beaucoup plus coûteux. En effet, avec les subdivisions, le nombre de cellules voisines atteint rapidement un seuil critique en 3D, et il en va de même pour le nombre de brins au sein d'une cellule. Si en 2D les cellules comportaient en moyenne 4 à 8 brins, en 3D elles peuvent atteindre 10 fois ce nombre.

Or, si le nombre de brins augmente, chaque parcours de la carte, même local, devient plus coûteux. Le problème intrinsèque à ce genre d'applications, qui comportent beaucoup d'entités, est l'équilibre entre les parcours et la mise à jour. En effet, deux solution sont possibles pour accéder à une information : effectuer un parcours de la carte pour l'obtenir, ou bien l'enregistrer dans chaque cellule pour la récupérer directement. Mais ces enregistrements nécessitent d'être mis à jour par des parcours de la carte.

Il faut donc, pour choisir une stratégie optimale, correctement étudier le nombre d'accès à une information, le coût et la fréquence de sa mise à jour en cas d'enregistrement.

Il reste du chemin à parcourir en terme d'optimisation pour que notre structure multirésolution puisse être viable en 3D dans ce type de simulations qui mettent à jour le maillage souvent et de manière non localisée.

Concernant le comportement de foules en 3D, la plupart des recherches se réfèrent aux travaux de Reynolds parus pour la première fois dans [27]. Ces travaux visent à simuler des groupes d'animaux, appelés boids (pour bird-oïd), ayant un comportement similaire. Ils exposent les paramètres à mettre en place pour obtenir une simulation réaliste de ce type d'individus.

La seconde application que nous avons traitée concerne l'insertion d'aiguille.

9.1.2 L'insertion d'aiguille

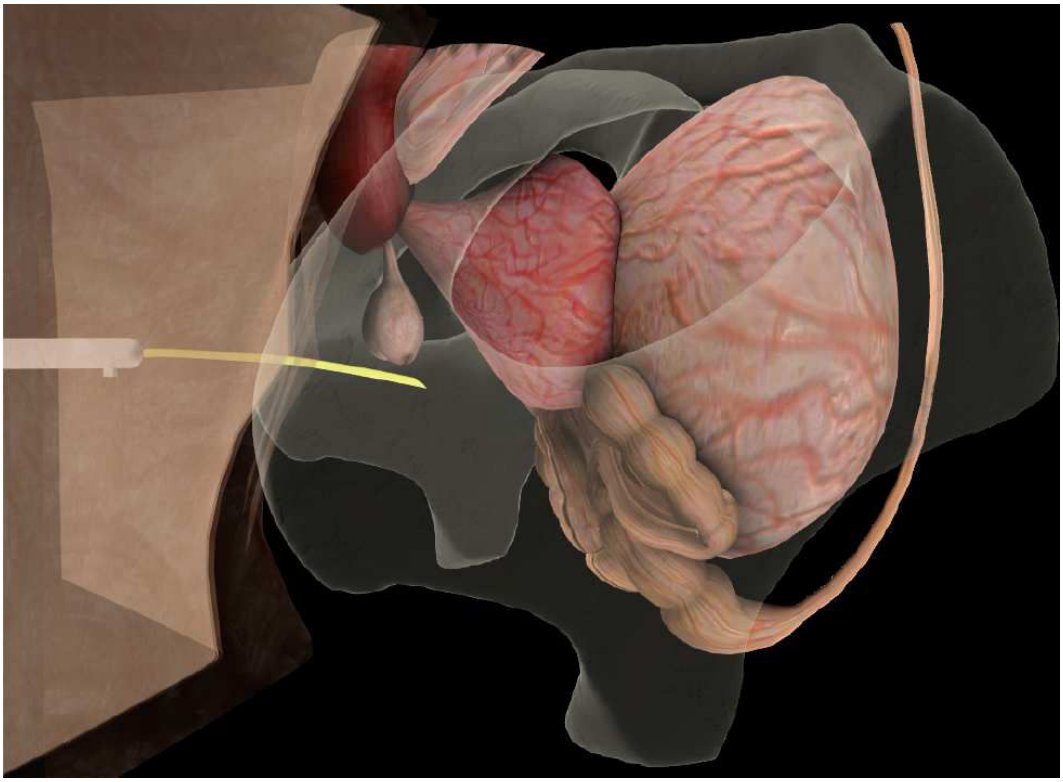

Figure 9.3 : Simulation d'insertion d'aiguille pour l'opération de la prostate.[7]

L'insertion d'aiguille fait partie du domaine des simulations physiques médicales. Ces simulations ont la particularité de représenter un environnement déformable dynamique avec de grandes différences d'échelles. Par exemple, lors de la simulation de l'opération d'une jambe, on représente une jambe entière avec ses différents modèles pour la peau, les muscles, les os, le réseau sanguin, etc. Dans cet environnement grand et détaillé, on va simuler l'insertion d'un aiguille fine dont la pointe est très petite. Afin de modéliser convenablement les interactions entre la pointe de l'aiguille et les tissus, la précision doit être assez haute pour pouvoir détecter les micro-fractures créées dans les tissus.

Cette différence d'échelle entre la jambe, voire le corps entier, et la micro-fracture représente un réel défi de simulation.

La plupart des simulateurs chirurgicaux utilisent une décomposition en éléments finis afin de gérer les interactions physiques. Cette méthode venant des mathématiques appliquées a été largement étudiée et optimisée pour les simulations médicales au fil des années.

[7], dont est tiré la figure 9.3, est un bon exemple de travaux récents sur le sujet. Ces travaux traitent de l'insertion d'aiguille en temps interactif pour la chirurgie, en réalisant un remaillage le long de la trajectoire de l'aiguille. La trajectoire est calculée pour que l'insertion de l'aiguille soit réalisable par un robot. Leur efficacité permet l'exécution en temps interactif mais le remaillage forcé autour de l'aiguille pose une contrainte de représentation.

SOFA (Simulation Open Framework Architecture) [1] est une plateforme de simulations physiques open-source notamment utilisée pour des applications médicales. L'équipe MIMESIS d'Inria [31] est spécialisée dans les simulations médicales en temps réel pour l'entraînement et l'assistance des médecins. Elle travaille actuellement sur l'intégration de notre modèle de cartes combinatoires (CGoGN) dans SOFA.

L'objectif de notre collaboration est, d'une part, d'optimiser la détection de collision dans SOFA, et d'autre part, d'utiliser les cartes multirésolutions pour résoudre le problème d'échelle des simulations médicales. Nous travaillons sur un projet d'insertion d'aiguille dans la jambe pour réaliser l'anesthésie locale du système nerveux. Celui-ci forme des ramifications qui permettent de toucher une zone précise en anesthésiant la « branche » correspondante. Nous devons donc modéliser une aiguille qui évolue dans un maillage déformable d'organe dans lequel sont plongées diverses entités pouvant être :

— filiformes, pour les vaisseaux sanguins très fin et le système nerveux.
— surfaciques, pour les membranes à traverser et pour la limite des organes internes.

Les opérations peuvent utiliser des aiguilles rigides, pour une trajectoire rectiligne, ou souples avec une pointe biseautée. Avec une aiguille souple, la trajectoire est modifiée selon l'angle de son biseau. Cela permet des trajectoires beaucoup plus évoluées à l'intérieur du corps du patient. Si l'on insère de manière rectiligne une aiguille biseautée, elle va suivre une trajectoire courbe. De nombreux travaux tels que [22] ont étudié ce phénomène qui fonctionne sur le modèle de l'unicycle. On peut ainsi donner une trajectoire rectiligne à l'aiguille en ajustant la période de rotation en fonction de la vitesse de pénétration.

Dans SOFA, lors de l'introduction de cette aiguille, des contraintes physiques sont créées. Elles permettent d'effectuer les calculs des forces et de représenter le déplacement restreint de l'aiguille, une fois qu'elle a pénétré dans le corps. Nous désirons subdiviser le maillage le long du trajet de l'aiguille et tout particulièrement à sa pointe afin d'obtenir la précision nécessaire aux calculs des micro-fractures. Enfin, nous devons détecter les collisions et nous repérer par rapport à un objectif dans l'espace.

Environnement de tests et premiers résultats

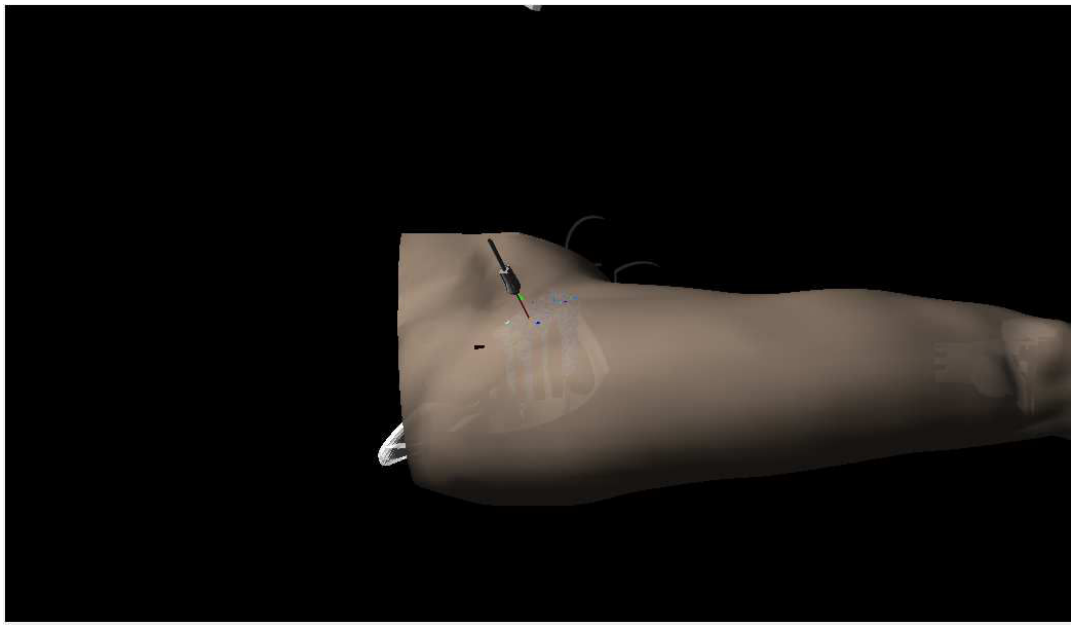

Figure 9.4 : L'application d'insertion d'aiguille de SOFA.

Nous avons réalisé nos premiers tests sur la plateforme de CGoGN afin de valider nos représentations avant de les importer dans SOFA.

Le première entité à représenter est l'aiguille. Dans un premier temps, la détection du premier contact entre l'aiguille et le corps est laissée au moteur de SOFA. Afin de réaliser cette détection avec nos travaux, il conviendrait de modéliser l'air autour du corps afin de créer une grande carte englobant tout l'environnement.

La partie que nous modélisons est donc l'aiguille à partir du moment où elle pénètre dans le corps. Une fois l'aiguille enfoncée d'une certaine distance, un point représentant une contrainte physique doit être placé.

Nous avons choisi de représenter l'aiguille par un ensemble de segments. Au départ elle n'est constituée que d'un seul segment de taille variable avec une particule à chaque extrémité. Une fois que ce segment atteint une taille suffisante, une particule est déposée au bout de l'aiguille pour « verrouiller » le segment actif, et un nouveau segment à taille variable est créé.

La particule déposée permet à l'aiguille d'avoir une forme plus souple, en la segmentant au fur et à mesure de son insertion. De plus, cette particule va pouvoir être « activée » en tant que contrainte physique pour la simulation. En

effet, la création de contrainte est facilitée par le fait que la particule connaisse la cellule dans laquelle elle se trouve.

La figure 9.5 décrit notre représentation d'une aiguille souple. L'aiguille est sectionnée en segments bleus durant sa trajectoire même temps que les particules bleues sont semées. Elle est plongée dans une carte cubique multi-résolution qui se subdivise autour de sa pointe et se simplifie ailleurs, comme on peut le constater entre les images 2 et 3.

Dans ce scénario de test, le déplacement de l'aiguille est réalisé interactivement à la souris. On peut donc vérifier l'aspect temps réel de la simulation en fonction du nombre de segments d'aiguille enregistrés.

Nous avons ensuite introduit d'autres entités dans notre cube de test afin de mettre à l'épreuve la détection de collision. La figure 9.6 présente cette détection pour des entités plongées filiformes et surfaciques, ainsi que pour l'auto-collision. Plus une entité est proche de la pointe de l'aiguille, plus elle est colorée en rouge. Comme chaque segment de l'aiguille est lui-même plongé, on peut détecter les auto-collisions.

Nous pouvons donc valider nos objectifs dans un environnement de test, et vérifier l'interactivité de l'exécution. La prochaine étape consiste à intégrer notre travail dans SOFA afin d'optimiser les applications d'insertion d'aiguille comme celle présentée sur la figure 9.4.

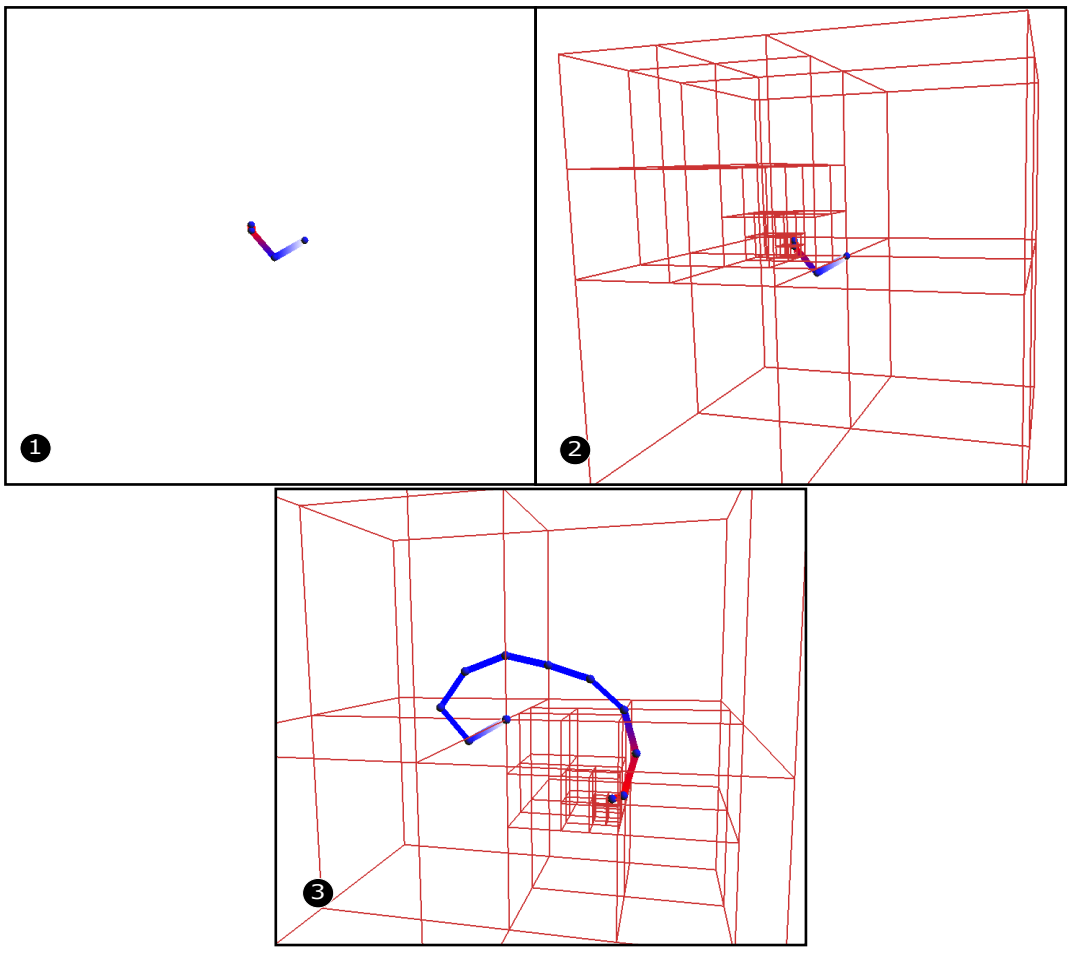

FIGURE 9.5 : La représentation d'une aiguille souple avec raffinement au niveau de la pointe.

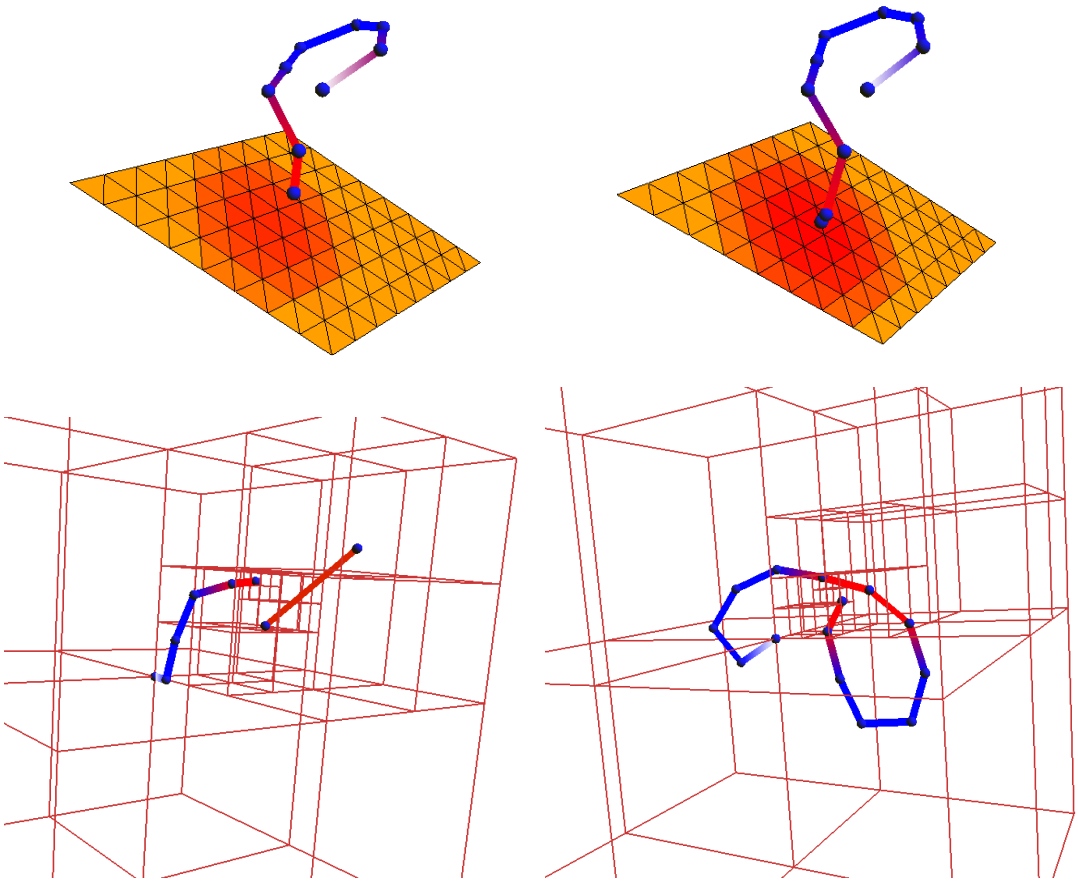

Figure 9.6 : Détection de l'auto-collision et de la collision avec des arêtes et des surfaces plongées.

Intégration dans SOFA

Principalement dédiée à la recherche, la plateforme SOFA est développée par différents laboratoires d'Inria et du CNRS. Elle fonctionne avec une librairie de plugins qui sont utilisés comme des « briques » pour mettre en place une simulation. L'utilisateur charge l'ensemble des plugins dont il a besoin via une interface XML qui lui permet également de régler les paramètres de sa simulation. La librairie actuelle contient un ensemble de solveurs permettant de réaliser des interactions physiques détaillées. On y trouve :

— Des modèles déformables utilisant la méthode masse-ressort ou la méthode des éléments finis.
— Des modèles rigides et articulés.
— Des modèles de fluides.
— Des modèles de représentation et de détection de collision.

Elle est également optimisée pour paralléliser au maximum avec la possibilité d'utiliser l'interface de CUDA pour coder sur GPU. SOFA gère pour le moment une représentation classique pour ses maillages 3D. L'équipe MIMESIS travaille sur un plugin permettant d'utiliser les cartes combinatoires de CGoGN afin de changer cette représentation. Nous avons créé en parallèle un plugin pour la détection de collision et la subdivision adaptative. Ce plugin, couplé au plugin de représentation CGoGN, permettra à l'utilisateur de réaliser un ensemble de tâches :

D'une part, la possibilité de créer aisément des entités (particules, arêtes, triangles) et de les utiliser pour récupérer des informations de localité. Par exemple, pour l'application d'insertion d'aiguille, quelques lignes de code suffisent à créer une aiguille et à récupérer à tout moment la cellule dans laquelle se trouve la pointe.

D'autre part, la possibilité de paramétrer et d'utiliser de manière simple la partie multirésolution du plugin. L'utilisateur peut demander la subdivision ou la simplification d'une cellule facilement ou de manière automatique en respectant un certain critère.

Avec de telles fonctionnalités, notre travail est accessible à un utilisateur de SOFA. Il reste toutefois à gérer le problème de l'intégration aux autres plugins. En effet, les ingénieurs de l'équipe travaillent encore sur la gestion de la multirésolution dans leur système de calcul de forces avec la méthode des éléments finis (FEM).

Un exemple de problèmes rencontrés est la création de « jonctions en T » lors des opérations de subdivision adaptative. Ces jonctions sont créées à la frontière entre deux cellules de niveaux différents et ne sont pour l'instant pas gérées par la représentation en éléments finis utilisée. Ce problème non-trivial est encore à l'étude mais les premières avancées sont prometteuses.

9.2 Conclusion et remarques

Nous avons étendu avec succès nos travaux dans le cadre des cartes combinatoires multirésolutions volumiques. Cette extension a été testée sur deux applications aux objectifs très différents, qui nous ont permis de mesurer le challenge que constitue la 3D.

D'une part, une application de simulation de foule 3D où un grand nombre d'entités évoluent de manière dynamique dans l'environnement. D'autre part, une application à visée médicale où un petit nombre d'entités évoluent dans un environnement complexe, et nécessitent des interactions précises. Cette seconde application nous a également permis de valider l'enregistrement de surfaces.

Nous insistons sur le fait que les cartes combinatoires multirésolutions volumiques sont encore en cours d'optimisation. Ainsi, nous ne pouvons pas, à l'heure actuelle, conclure sur nos premiers résultats. Toutefois, ces résultats montrent la difficulté de gestion des scénarios très dynamiques comme ceux de la simulation de foule. En revanche, dans les scénarios à interactions localisées et précises, nos travaux permettent de résoudre avec efficacité les problèmes d'échelle et d'obtenir des résultats en temps interactif.

Leur intégration à la plateforme SOFA est en cours et permettra leur utilisation par l'ensemble de la communauté SOFA.

Conclusion

Les travaux présentés dans ce mémoire concernent le traitement des interactions entre des objets en mouvement dans un environnement complexe partitionné. L'environnement est une quasi-variété de dimension 2 ou 3, modélisée par une carte combinatoire multirésolution. Les objets en mouvement sont soit de simples particules, soit des objets déformables plus complexes représentés sous forme de polygones ou de surfaces triangulées.

Ces travaux ont été validés dans le cadre d'une application de simulation de foule 2D surfacique. Ils sont également en cours d'intégration dans la plateforme de simulations physiques pour la médecine SOFA.

Nous rappelons ici les différentes contributions de ce mémoire, avant de proposer quelques perspectives de recherche concernant ces travaux.

1 Rappel des contributions

Enregistrement de trajectoires dans un environnement complexe : Notre première contribution consiste en un ensemble d'outils permettant le suivi d'entités déformables dans un environnement représenté par une carte combinatoire.

Enregistrement multirésolution et densité de la simulation : Notre seconde contribution est l'extension de ces outils à des cartes multirésolutions. Elle permet d'adapter la taille des cellules de l'environnement en fonction de la densité d'objets. Cela nous permet d'atteindre le temps interactif sur des simulations comprenant un grand nombre d'entités inégalement réparties.

Enregistrement dans des cellules non-convexes : Notre troisième contribution est l'extension du modèle de suivi de particules de T. Jund à des cellules potentiellement non-convexes. Nous avons affaibli la contrainte de convexité à la contrainte géométrique « BC ». Cette approximation permet de gérer des cas de faible déformation, dans des maillages volumiques qui nécessitent plus de robustesse.

Revenons sur les points forts de nos contributions autour de trois notions fondamentales : la généricité, la cohérence et l'efficacité.

1.1 Généricité

La généricité de nos travaux se retrouve selon plusieurs critères.

D'une part, au niveau de la structure de représentation de l'environnement et des entités plongées. En effet, notre structure, offre un large spectre d'environnements de simulation possibles, en couvrant les quasi-variétés de dimensions 2 et 3 déformables. Nous nous limitons aux environnements partitionnés mais le système de partitionnement demeure libre tant qu'il respecte le critère de cellules BC. Le caractère multirésolution est lié à un critère de subdivision qui est au choix de l'utilisateur. Dans ce mémoire, nous nous sommes focalisés sur un critère de densité, mais d'autres critères sont possibles. Le fait d'avoir défini des enregistrements sur des entités de bases, telles que des sommets et des arêtes, nous permet également de représenter n'importe quelle entité déformable construite à partir de ces bases.

D'autre part, au niveau de la détection de collision, le système d'évitement est indépendant de notre méthode de requêtes de proximité. En effet, nous fournissons uniquement la notion de voisinage associée à notre structure et laissons libre la méthode de gestion des interactions. Cette notion de voisinage elle-même peut être altérée en jouant sur le critère de multirésolution.

1.2 Cohérence

On peut noter une cohérence forte entre notre représentation théorique des entités et leur réalité géométrique. En effet, contrairement à certaines représentations telles que les grilles de voisinage, les entités considérées comme voisines dans notre structure topologique sont également voisines géométriquement.

Concernant l'aspect multirésolution, sa cohérence au fur et à mesure de l'évolution du maillage à été démontrée dans [35]. Ainsi, nous pouvons gérer divers problèmes de manière fine car notre structure représente chacune des cellules élémentaires que sont les sommets, les arêtes, les faces et les volumes. Par exemple, l'apparition des jonctions en T lors des opérations de subdivision peut être traitée, car on a facilement accès aux entités concernées.

1.3 Efficacité

L'efficacité de nos travaux à été prouvée dans nos applications 2D et 2D surfacique pour la simulation de foule. Nous avons pu simuler en temps réel des milliers d'agents ponctuels et des centaines d'agents polygonaux évoluant dans des environnements complexes et dynamiques. L'efficacité de notre solution est indépendante de la taille de l'environnement. Cela est possible grâce à des accès et des calculs localisés.

De plus, notre efficacité est garantie même en cas de scènes présentant de fortes densités, grâce à la subdivision adaptative des cartes multirésolutions. Le critère de densité nous permet de borner le nombre d'entités en interaction dans un voisinage, et donc de limiter les calculs de collisions.

Nos premiers tests 3D ont montré des capacités intéressantes pour la modélisation d'objets complexes 3D. De plus, nous avons établi la robustesse de notre suivi de particules face à des déformations locales en brisant la contrainte de convexité des cellules de nos maillages.

2 Perspectives

Les perspectives de recherche s'articulent autour de 3 axes : l'optimisation des algorithmes existants, les extensions comportementales et les autres solutions de représentation.

2.1 L'optimisation des algorithmes existants

Les optimisations prévues s'appuient sur une parallélisation des calculs et sur une amélioration de la représentation.

La parallélisation des calculs pourrait grandement optimiser les performances des applications mettant en jeu des milliers d'entités ,comme la simulation de foule. On peut imaginer assigner un thread à chaque cellule multirésolution de bas niveau et que ce thread réalise les calculs d'évitement pour toutes les entités présentes dans les sous-cellules de niveau supérieur. Si la parallélisation est utilisée après les requêtes de proximité, on peut éviter les problèmes de recollement entre des cellules de bas niveau.

Certaines optimisations restent à mettre en place en ce qui concerne le compromis accès/enregistrement. Si l'on doit accéder un certain nombre de fois à la même information, il peut être intéressant de l'enregistrer dans la structure via le modèle de plongement. Il faut cependant mesurer l'impact des mises à jour de cet enregistrement par rapport à un parcours de la carte réalisé pour récupérer l'information. C'est ce même compromis qui nous a , par exemple, amené à enregistrer les entités voisines dans chaque cellule.

Pour le moment, tous les enregistrements se font au niveau de résolution le plus fin. Grâce aux dernières avancées en matière de cartes multirésolutions implicites, on peut optimiser notre modèle en utilisant des enregistrements à différents niveaux de résolution. On peut ainsi imaginer que les agents de plus grosse taille seraient enregistrés dans un autre niveau. Cela permettrait d'optimiser leur enregistrement et d'augmenter leur visibilité, sans avoir à recourir à l'anticipation que nous avons mis en place.

2.2 Les extensions comportementales

Les extensions comportementales concernent notamment le domaine de la simulation de foule. Nous avons tout juste abordé les possibilités de ce domaine de recherche qui est très riche et très étudié. La diversité des informations que l'on peut récupérer dans un voisinage permet l'implémentation de comportements évolués pour les agents. Nos extensions multirésolutions pourraient être exploitées pleinement, en utilisant des notions de haut niveau plongées dans les différentes strates. Par exemple, si l'on peut accéder à des informations sur la densité de la foule dans de grandes zones, les algorithmes de recherche de chemins pourraient être plus efficaces. [38] présente une optimisation similaire grâce un graphe pour éviter les zones de congestion de la foule.

On pourrait également considérer les agents à deux niveaux : individuellement, comme c'est déjà le cas, et par groupes d'agents ayant localement les mêmes objectifs. Cette distinction a été réalisée habilement dans [29] et pourrait être optimisée par les travaux sur l'évitement de groupes d'agents présentés dans [14].

2.3 Les autres solutions de représentation

Les autres représentions que nous avons envisagées utilisent différemment le suivi de particule. Concernant le suivi des arêtes dans une partition de cellules convexes, on pourrait imaginer une alternative au lancer de particule virtuelle.

Par exemple, il serait possible de suivre de manière détaillée les arêtes de l'environnement coupées par la surface d'un objet, en créant une particule à chacune de leurs intersections. De telles particules se déplaceraient le long des intersections durant la simulation. Elles fusionneraient en arrivant sur les sommets et se dupliqueraient en les quittant. Grâce à la propriété de convexité, on pourrait déduire des changements de cellules même dans des cas d'arêtes pluricellulaires, sans envoyer de particule virtuelle. Les enregistrements seraient donc mis à jour localement, sans avoir à réaliser des parcours fréquents.

Il serait intéressant de tester l'efficacité d'une telle méthode qui suit un ensemble de particules plutôt que d'envoyer fréquemment une particule virtuelle et de réaliser des parcours de la carte.

Table des figures

Bibliographie

[1] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette et L. Grisoni : Sofa - an open source framework for medical simulation. *In MMVR*, vol. 125, p. 13–18, 2007. [124]

[2] N. Badler, J. Allbeck et N. Pelechano : *Virtual Crowds: Methods, Simulation, and Control (Synthesis Lectures on Computer Graphics and Animation)*. Morgan and Claypool Publishers, 2008. [99]

[3] E. Baptista Passos, M. Joselli, M. Zamith, E. Walter Gonzales Clua, A. Montenegro, A. Conci et B. Feijo : A bidimensional data structure and spatial optimization for supermassive crowd simulation on gpu. *Computers in Entertainment (CIE)*, 7(4):60, 2009. [21, 118, 136]

[4] J. Barnes et P. Hut : A hierarchical o(n log n) force-calculation algorithm. *Nature)*, (324):446–449, 1986. [17]

[5] A. Braun, S. Musse, L. de Oliveira et B. Bodmann : Modeling individual behaviors in crowd simulation. *In CASA*, p. 143, 2003. [99]

[6] CGoGN : Combinatorial and Geometric modeling with Generic N-d maps. [30, 82]

[7] N. Chentanez, R. Alterovitz, D. Ritchie, L. Cho, K. Hauser,

K. Goldberg, J. Schewchuk et J. O'Brien : Interactive simulation of surgical needle insertion and steering. *In SIGGRAPH*, vol. 28, 2009. [123, 124, 138]

[8] P. Fiorini et Z. Shiller : Motion planning in dynamic environments using velocity obstacles. *Int. Journal of Robotics Research*, 17:760–772, 1998. [101]

[9] J. Friedman, J. L. Bentley et R. Finkel : An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977. [17]

[10] K. Fujimura et H. Samet : Time-minimal paths among moving obstacles. *In Robotics and Automation*, vol. 2, p. 1110 – 1115, 1989. [107]

[11] S. Goldenstein, M. Karavelas, D. Metaxas, L. Guibas, E. Aaron et A. Goswami : Scalable nonlinear dynamical systems for agent steering and crowd simulation. *Computers & Graphics*, 25(6):983–998, 2001. [102]

[12] S. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha et P. Dubey : Clearpath: highly parallel collision avoidance for multi-agent simulation. *In EG Symposium on Computer Animation*, p. 177–187, 2009. [18, 101]

[13] S. J. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin et D. Manocha : PLEdestrians: a least-effort approach to crowd simulation. *In EG Symposium on Computer Animation*, p. 119–128, 2010. [102]

[14] J. He, L. van der Berg : Meso-scale planning for multi-agent navigation. *In IEEE International Conference on Robotics and Automation (ICRA)*, p. 2839 – 2844, 2013. [99, 135]

[15] T. Jund : *Détection de collision dans des subdivisions volumiques*. Thèse de doctorat, Université de Strasbourg, Septembre 2010. [41, 43, 45, 47, 48, 49, 50, 137]

[16] T. Jund, P. Kraemer et D. Cazier : A unified structure for crowd simulation. *Computer Animation and Virtual Worlds*, 23(3):311–320, 2012. [50, 96, 97, 107, 114]

[17] A. Katanforoush et M. Shahshahani : Distributing points on the sphere, i. *Experimental Mathematics*, 2(12):199–209, 2012. [120]

[18] P. Kraemer : *Modèles topologiques pour la multirésolution*. Thèse de doctorat, Université Louis Pasteur - Strasbourg I, Novembre 2008. [25, 27, 28, 29, 32, 136]

[19] F. Lamarche et S. Donikian : Crowd of virtual humans: a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*, 23(3):509–518, 2004. [19, 20, 100, 136, 137]

[20] G. Leach : Improving worst-case optimal delaunay triangulation algorithms. 1992. [19]

[21] C. Loscos, D. Marchal et A. Meyer : Intuitive crowd behaviour in dense urban environments using local laws. *In TPCG*, p. 122, 2003. [100]

[22] D. Minhas, J. Engh, M. Fenske et C. Riviere : Modeling of needle steering via duty-cycled spinning. *In IEEE EMBS*, p. 27562759, 2007. [124]

[23] M. Müller, B. Heidelberger, M. Teschner et M. Gross : Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, 2005. [98, 105, 137]

[24] OpenStreetMap : Map data ©openstreetmap contributors, cc-by-sa. [111]

[25] M. Pharr et R. Fernando : *GPU Gems2: Programming Techniques for High-Performance Graphics and General- Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005. [18, 136]

[26] T. Pitiot, D. Cazier, T. Jund, A. Habibi et P. Kraemer : Deformable polygonal agents in crowd simulation. *Computer Animation and Virtual Worlds*, 25:343–352, 2014. [97, 103]

[27] C. W. Reynolds : Flocks, herds, and schools: A distributed behavioral model. *In Computer graphics and interactive techniques*, p. 25–34, 1987. [122]

[28] B. Ricks et P. Egbert : A whole surface approach to crowd simulation on arbitrary topologies. *In Visualization and Computer Graphics*, vol. 20, p. 159 – 171, 2013. [118]

[29] J. Sewall, D. Wilkie et M. C. Lin : Interactive hybrid simulation of large-scale traffic. *ACM Transaction on Graphics (Proceedings of SIGGRAPH Asia)*, 30(6), 2011. [135]

[30] W. Shao et D. Terzopoulos : Autonomous pedestrians. *In EG Symposium on Computer Animation*, p. 19–28, 2005. [22, 100]

[31] M. Team : http://mimesis.inria.fr/. [124]

[32] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes et M. H. Gross : Optimized spatial hashing for collision detection of deformable objects. *In Vision, Modeling and Visualization*, p. 47–54, 2003. [23, 107, 136]

[33] D. Thalmann, C. O'Sullivan, P. Ciechomski et S. Dobbyn : Populating virtual environments with crowds. *In Eurographics 2006 Tutorial Notes*, 2006. [96, 99]

[34] S. Thery : *Plongements de complexes cellulaires.* Thèse de doctorat, Université Louis Pasteur, Strasbourg, 2000. [30]

[35] L. Untereiner : *Représentation des maillages multirésolutions : application aux volumes de subdivision.* Thèse de doctorat, Université de Strasbourg, Novembre 2013. [25, 29, 34, 35, 36, 37, 133, 136]

[36] L. Untereiner, P. Kraemer, D. Cazier et D. Bechmann : Cph: a compact representation for hierarchical meshes generated by primal refinement. *Computer Graphics Forum*, 2015. 5-Year Impact Factor : 1,920. [37, 66]

[37] J. van den Berg, S. Guy, M. Lin et D. Manocha : Reciprocal n-body collision avoidance. *In Robotics Research*, vol. 70 de *Springer Tracts in Advanced Robotics*, p. 3–19. Springer, 2011. [101, 137]

[38] G. v. T. Wouter, F. C. I. Atlas et G. Roland : Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds*, 23(1): 59–69, 2012. [135]

Résumé

Outils multirésolutions pour la gestion des interactions en simulation temps réel

La plupart des simulations interactives ont besoin d'un modèle de détection de collisions. Cette détection nécessite d'une part d'effectuer des requêtes de proximité entre les entités concernées et d'autre part de calculer un comportement à appliquer. Afin d'effectuer ces requêtes, les entités présentes dans une scène sont soit hiérarchisées dans un arbre ou dans un graphe de proximité, soit plongées dans une grille d'enregistrement.

Nous présentons un nouveau modèle de détection de collisions s'appuyant sur deux piliers : une représentation de l'environnement par des cartes combinatoires multirésolutions et un suivi en temps réel de particules plongées dans ces cartes. Ce modèle nous permet de représenter des environnements complexes tout en suivant en temps réel les entités évoluant dans cet environnement. Nous présentons des outils d'enregistrement et de maintien de l'enregistrement de particules, d'arêtes et de surfaces dans des cartes combinatoires volumiques multirésolutions.

Mots clés : Simulation temps réel, Détection de collisions, Suivi de particules, Maillages multirésolution adaptatifs

Abstract

A multiresolution framework for real-time simulation interactions

Most interactive simulations need a collision detection system. First, this system requires the querying of the proximity between the objects and then the computing of the behaviour to be applied. In order to perform these queries, the objects present in a scene are either classified in a tree, in a proximity graph, or embedded inside a registration grid.

Our work present a new collision detection model based on two main concepts : representing the environment with a combinatorial multiresolution map, and tracking in real-time particles embedded inside this map. This model allows us to simulate complex environments while following in real-time the entities that are evolving within it. We present our framework used to register and update the registration of particles, edges and surfaces in volumetric combinatorial multiresolution maps.

Results have been validated first in 2D with a crowd simulation application and then in 3D, in the medical field, with a percutaneous surgery simulation.

Keywords: Real-Time Simulation, Collision Detection, Particle Tracking, Multiresolution Adaptive Meshes