N° d'ordre: 725

UNIVERSITÉ DE STRASBOURG

THÈSE

présentée pour obtenir le grade de

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

par

Thomas Jund

Détection de collision dans des subdivisions volumiques

Soutenue publiquement le **30 septembre 2010**

Membres du jury :

M. Stéphane COTIN ..... *Rapporteur*
Directeur de recherche à l'INRIA Lille

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

M. Hervé DELINGETTE ..... *Président du jury*
Directeur de recherche à l'INRIA Sophia-Antipolis

M. David CAZIER ..... *Examinateur*
Maître de conférences à l'Université de Strasbourg

M. Jean-François DUFOURD ..... *Directeur de thèse*
Professeur à l'Université de Strasbourg

Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection UMR CNRS-UdS 7005

Remerciements

Je remercie tout d'abord David Cazier de son encadrement depuis mon stage de Master 2 jusqu'à la fin de ma thèse, il a su me transmettre sa rigueur et sa passion pour la recherche.

Je remercie également Jean-François Dufourd, mon directeur de thèse, qui malgré la distance à son domaine de prédilection a accepté de m'encadrer.

Merci à Stéphane Cotin, directeur de recherche à l'INRIA Lille, et Philippe Meseure, professeur à l'Université de Poitiers, d'avoir remplis la lourde tâche d'évaluer ce mémoire. Merci également à Hervé Delingette qui a accepté de participer à mon jury de soutenance.

Un grand merci à tous les thésards avec qui j'ai partagé ces 3 ans, notamment Guillaume, Jonathan, Lucas, Olivier, Simon et tout particulièrement à Manu avec qui j'ai partagé mon bureau. Ils m'ont permis de rendre plus amusantes – bien que parfois dans la douleur– mes journées de travail.

Je ne peux évidemment pas oublier le reste de l'équipe IGG de Strasbourg, plus particulièrement Basile, Cyril, Pierre, Olivier et Sylvain pour des raisons qu'ils connaîtront.

Je remercie également l'ensemble de mes amis et de ma famille de leur soutien. Plus particulièrement mes parents, mon frère, Michael, Arnaud, Christophe, Fred, Jeremy et Nico. La liste pourrait être bien plus longue mais je m'arrêterai là en espérant que les autres m'excuseront.

Enfin, je ne peux oublier de remercier Sophie qui est ma compagne depuis quelques années déjà. Elle m'a poussé à tenter cette aventure qu'est la thèse et en commence une à son tour.

Table des matières

Introduction

1 Contexte et objectifs

La simulation du monde réel est au cœur d'un nombre de plus en plus important d'applications. Elle relie des disciplines comme la physique, la mécanique et l'informatique. Elle intervient dans des domaines variés allant de la CAO[1], à la réalité virtuelle en passant par les jeux vidéos ou le cinéma. Ces applications de simulations ont été développées pour répondre à de nombreux besoins, tels que la planification, l'apprentissage, la compréhension, ou plus simplement la visualisation de certains phénomènes physiques ou naturels.

De telles simulations nécessitent la mise en place de modèles géométriques, de modèles physiques et de systèmes d'interaction. En effet, lorsque plusieurs objets coexistent dans une même simulation, il faut être capable de calculer leurs interactions, c'est à dire de détecter les collisions et les contacts entre eux, ce qui permet aux modèles physiques d'estimer les frottements, les rebonds, etc.

Dans le cadre d'application en réalité virtuelle, où un utilisateur interagit avec la simulation, il faut également être capable de détecter les collisions entre les outils manipulés par l'utilisateur et les objets constituant la scène.

[1] Conception Assistée par Ordinateur

Une autre contrainte, liée à la réalité virtuelle, concerne les temps de réponses. Pour assurer une interactivité correcte, l'ensemble des processus mis en place doit s'exécuter en temps-réel

Si, historiquement, les premières simulations se contentaient d'objets rigides, les besoins applicatifs actuels nous poussent à considérer des objets déformables. La gestion physique de ces derniers est plus coûteuse, en termes de temps de calcul, et contraint d'autant plus les temps de réponses de la détection de collision.

Nous nous plaçons dans ce cadre et souhaitons proposer des algorithmes de détection de collision permettant d'atteindre des réponses temps-réel pour des scènes déformables complexes, c'est à dire des scènes composées de plusieurs centaines de milliers de polygones.

1.1 Applications médicales

Le besoin en simulation est particulièrement présent dans le domaine des applications médicales. Il intervient par exemple lors de la planification d'opérations chirurgicales ou pour le développement de simulateurs permettant l'apprentissage de gestes techniques de plus en plus complexes. Ce domaine d'application passionnant impose de nouvelles contraintes. Les scènes considérées sont très complexes et font intervenir, entre autre, des organes, des vaisseaux sanguins et des outils souples (comme dans le cadre de l'angioscopie). Ces objets sont fortement déformables, leurs zones de contact peuvent être importantes et les organes ou outils sont parfois inclus les uns dans les autres. Cette situation survient notamment pour la simulation d'opérations micro-invasives ou d'insertions de cathéters dans un réseau vasculaire. Ce type d'opération doit être précis et supporte mal les approximations au niveau du calcul des contacts, comme par exemple l'apparition d'interpénétrations entre les outils chirurgicaux et les organes.

Un simulateur chirurgical doit permettre au praticien d'effectuer des découpes, des ligatures ou encore de bruler des tissus pour leur ablation. Cela impose, de gérer les changements topologiques au niveau des modèles de représentations des objets virtuels.

Même si notre approche se veut générale, nous nous plaçons dans le cadre d'applications médicales. Les méthodes que nous proposons doivent prendre en compte les déformations topologiques et supporter efficacement l'inclusion de certains objets dans d'autres. Nous veillons également à contrôler les inter-

pénétrations entre les différents objets simulés.

1.2 Détection de collision

Le processus de détection de collision est étudié depuis de nombreuses années. Ces études ont vu émerger trois grandes familles de méthodes.

Les méthodes les plus répandues utilisent des structures accélératrices souvent basées sur des hiérarchies de volumes englobants. Elles exploitent des algorithmes du type diviser pour régner. Ces méthodes ont montré leur efficacité dans de nombreux domaines, mais supportent encore mal la gestion d'objets déformables. Cela provient de la nécessité de mettre à jour les structures accélératrices utilisées lors de déformations et de changements topologiques. Ces méthodes supportent rarement l'inclusion complète d'objets dans d'autres et sont plus adaptées à la gestion d'objets totalement séparés.

Les méthodes stochastiques utilisent un échantillonnage des objets qui les rendent moins sensible aux déformations. Mais de par leur nature, elles autorisent des interpénétrations qui peuvent être mal maîtrisées et se révèlent inadéquates pour nos applications.

Récemment, de nombreuses méthodes basées sur l'exploitation des cartes graphiques ont vu le jour. Elles montrent des performances séduisantes et offrent des perspectives intéressantes. Mais elles sont encore limitées par la mémoire disponible et la résolution de l'espace image utilisée implique des approximations gênantes au niveau des détails dans les zones de contact.

Quelques travaux portant sur la simulation d'angioscopies existent. Ces travaux atteignent de bonnes performances sur ce type d'opérations. Mais, les méthodes employées sont fortement liés au type d'applications visées.

Nous voulons pallier ces limitations et proposer une approche générique les réduisant.

2 Contributions

Nous proposons une approche innovante basée sur le suivi de mobiles se déplaçant dans un environnement complexe. L'espace environnant, dans lequel les mobiles se déplacent, est décomposé en cellules convexes. Il modélise la scène

et plus précisément, dans le cadre d'applications chirurgicales, le corps humain et ses organes. Les mobiles sont échantillonnés par des particules placées aux sommets des maillages qui les modélisent. Notre approche consiste à suivre les trajectoires de ces particules à l'intérieur de la partition de l'environnement.

La décomposition de l'espace permet de limiter les tests d'intersections à des zones restreintes, bornées par les cellules adjacentes à celle contenant l'élément testé. Cette exploitation de la cohérence spatiale de la scène, nous permet d'éviter l'utilisation d'une structure accélératrice globale. Nous obtenons ainsi un algorithme de détection de collision dont la complexité ne dépend pas de la taille globale de la scène. Les temps de calcul sont directement liés à la taille des mobiles et plus précisément à la densité de leur échantillonnage.

Le fait de partitionner l'environnement et de ne pas utiliser de structure hiérarchique globale, facilite également la gestion des déformations et des changements topologiques. Lorsque l'environnement est déformé ou que sa topologie est modifiée, seules des mises à jour locales de ses cellules sont nécessaires. En ce qui concerne les mobiles, ils peuvent être déformés ou découpés sans calculs supplémentaires.

Nous obtenons ainsi un système de détection de collision supportant des environnements complexes et déformables particulièrement efficace. Nous avons conduit différentes expérimentations validant les propriétés théoriques de notre approche et l'avons comparé à un système similaire bien connu [Bul]. De plus, notre système est suffisamment générique pour exploiter des accélérations classiques comme l'élimination des faces arrières.

Notre approche est générique dans le sens où son utilisation n'est pas restreinte à un modèle physique particulier. Tout modèle physique possédant une information de surface portée par des sommets peut être utilisé. L'intégration de mobiles possédant une structure interne est parfaitement envisageable dans le cadre de notre système. Des structures volumiques sont, par exemple, utilisées pour des modèles de type masse-ressort ou utilisant des méthodes par éléments-finis. La détection de collision est alors effectuée sur les sommets à la surface des objets en déplacement, la structure interne n'est exploitée que pour les calculs physiques.

Pour le montrer, nous proposons différentes simulations de nature très différentes dans lesquelles nous intégrons deux modèles physiques distincts. La première est une simulation de l'insertion d'un cathéter dans un réseau vasculaire utilisant un système masse-ressort. La seconde est la simulation d'objets déformables utilisant la technique du *shape-matching*.

Enfin, les informations de contact obtenues sont précises et fiables. Notre approche permet la détection de toutes les collisions et empêche les interpénétrations des sommets et des arêtes des mobiles dans l'environnement. Ce dernier point est crucial dans le cadre d'applications médicales pour assurer le réalisme de simulation et la perception subjective qu'en à l'utilisateur.

2.1 Robustesse numérique

La question des approximations est une question importante pour le calcul numérique. Nous faisons une parenthèse sur ce point dans cette introduction et ne revenons pas dessus par la suite.

Les approximations numériques, les erreurs d'arrondis ou d'égalités influencent les réponses des algorithmes présentés dans ce mémoire. Ces problèmes n'étant pas le point central que nous traitons, l'ensemble des explications qui suivent est donné en considérant que tous les calculs géométriques d'orientations et d'intersections sont fiables et déterministes.

Pour les questions d'implantation, nous orientons le lecteur vers des travaux ayant déjà traités cette question selon diverses approches. Comme l'arithmétique d'intervalles de flottants [GSS89] ou de rationnels [Mil95] ou encore l'arithmétique multi-précision [For95, She97].

Dans le cadre des méthodes proposées nous considérons, lorsqu'il y a des doutes concernant des tests d'appartenance à un sommet, une arête, une face ou un volume, que l'appartenance est définie par l'entité de dimension la plus fine. Par exemple si un point appartient à une arête selon un critère mais qu'il appartient à une face selon un autre, nous considérons qu'il appartient à l'arête.

3 Plan du mémoire

Ce mémoire est organisé de la manière suivante :

Le premier chapitre est consacré à la définition précise de la détection de collision, elle définit les différents cadres existant dans des applications de simulations. Nous décrivons les méthodes existantes tout en définissant leurs cadres d'applications, leurs avantages ainsi que leurs limites.

Le deuxième chapitre précise le contexte applicatif dans lequel nous nous positionnons, en l'occurrence la gestion d'environnements partitionnés. Un tour d'horizon de la détection de collision, appliqué à ce cas, est alors donné. Nous exposons dans ce même chapitre le modèle de données que nous utilisons pour représenter nos objets. Il s'agit du modèle de représentation à base topologique des cartes combinatoires.

Le troisième chapitre est consacré à la présentation d'algorithmes répondant à la problématique du déplacement de particules de manière continue dans un environnement partitionné. Cette présentation décrit de manière précise les conditions d'applications de nos algorithmes. Les algorithmes de déplacement unitaire dans le plan, puis dans l'espace sont présentés successivement. Une extension de ces algorithmes pour des environnements déformables est donnée.

Le quatrième chapitre présente le cas du déplacement d'arêtes dans ce même cadre. De la même façon que pour les particules, une explication progressive dans le plan étendue à l'espace est donnée. Deux méthodes sont proposées. La première est une extension directe du déplacement de particules, se base sur certaines approximations et autorise des interpénétrations limitées. La deuxième, plus générale, concerne le déplacement d'arêtes de manière continue.

Le cinquième chapitre expose deux modèles physiques existant pour effectuer des simulations chirurgicales ou physiques. Nous indiquons également ici comment effectuer l'intégration d'un modèle physique avec le système de détection de collision que nous proposons.

Le sixième chapitre consiste en l'analyse à la fois théorique et pratique de l'ensemble des algorithmes de détection de collision mis en place. Nous y donnons les complexités théoriques des différents algorithmes proposés, avant de fournir des analyses statistiques de mises en applications. Une comparaison de performance à une autre méthode de détection de collision est également présentée.

Enfin, nous terminons ce mémoire par une conclusion résumant l'ensemble des apports et limitations de nos méthodes. Ce chapitre nous permet également de présenter diverses perspectives tant pour le développement de nouvelles fonctionnalités au niveau de la détection de collision qu'au niveau des applications auquel notre système peut s'intégrer.

Chapitre 1

Détection de collision : cas général

Sommaire

Ce chapitre débute par une introduction au concept de modélisation avant de définir la détection de collision. Les différents cadres dans lesquels s'intègre son utilisation sont présentés. Une fois le contexte défini, nous présentons les travaux antérieurs effectués et les algorithmes développés dans le domaine pour répondre à cette problématique.

1.1 Introduction

Une simulation physique met en jeu trois processus : le rendu, la simulation physique et la détection de collision. Le processus de rendu vise à fournir une représentation visuelle de la simulation la plus réaliste possible. Le processus de simulation physique gère les déplacements d'objets selon les lois de la physique et gère l'interaction. Le processus de détection de collision, objet de notre étude, indique la validité ou l'invalidité de déplacements selon des informations données par la simulation physique.

Dans ce cadre, la notion de détection de collision consiste à déterminer si, à un instant donné, plusieurs entités partagent une même partie de l'espace. Cette notion est nécessaire afin de permettre l'implantation d'applications de simulations. Une application de simulation doit permettre de reproduire la réalité de la manière la plus précise possible. Cela implique de respecter au mieux les réactions physiques des objets réels. Sans la notion de détection de collision, les objets se déplaceraient en traversant tout types d'obstacles.

Une collision entre plusieurs objets donne lieu à une réponse physique qui dépend des caractéristiques des objets et du domaine d'application. Dans ce chapitre, nous laissons de coté les questions relatives à la gestion de la physique et présentons l'ensemble des grandes familles d'algorithmes que l'on peut rencontrer pour répondre à la détection de collision. La question des modèles physiques est abordée de manière plus précise dans le chapitre 5.

Des modèles de représentation variés sont utilisés pour modéliser les objets en déplacement. Une brève introduction à la modélisation est donnée à la suite de cette section.

Le problème majeur concernant la détection de collision est le suivant : comment trouver les éléments en collision dans un temps minimum ?

Des calculs précis et coûteux en terme de temps de calcul ne sont pas une contrainte lors de simulations hors-ligne (*i.e.* non-interactives). Ces dernières n'ayant pas de contraintes de temps particulières, l'accent est principalement mis sur la justesse des résultats. Toutefois, nous nous plaçons dans le cadre de la simulation interactive voire temps-réel. Ce type de simulation est plus exigeant car une contrainte de temps d'exécution est imposée. Cela implique que l'ensemble des opérations nécessaires à la simulation est effectué en un temps limité. La contrainte de justesse reste présente dans la plupart des simulateurs. Lorsqu'elle ne l'est pas, on désire avoir une approximation de la réalité suffisante pour la tâche à effectuer.

La contrainte temporelle dans le cadre de la simulation interactive est fortement dépendante du type d'application visé. Si l'application met en place des systèmes d'interactions haptiques, le taux de rafraîchissement est fixé entre 500Hz et 1000Hz. Pour des applications plus classiques au niveau de l'interaction, on peut considérer des taux de rafraîchissement 10 fois moins importants.

Bien que l'état de l'art que nous présentons dans ce chapitre vise à aborder l'ensemble des techniques mises en œuvre pour l'accélération des calculs de la détection de collision, il n'est pas exhaustif. Pour chacune des grandes familles présentées, une sélection des articles que nous considérons comme les plus représentatifs est donnée.

Des états de l'art complétant celui qui est présenté dans ce chapitre peuvent être trouvés dans la littérature. Nous conseillons la consultation des travaux publiés dans [LM04, TKZ+04]. L'étude de Lin et Manocha [LM04] porte sur la simulation d'objets rigides. L'étude de Teschner *et al.* [TKZ+04] se concentre sur la gestion de la détection de collision d'objets déformables.

1.1.1 Modélisation de l'environnement

La modélisation géométrique consiste à donner une représentation manipulable par un programme aux objets intervenant dans une simulation. Dans le cas de la détection de collision, les environnements de simulation sont potentiellement complexes et de grandes tailles. Ces modèles géométriques doivent être les plus compacts possibles tout en conservant une précision suffisante pour être proche de situations réelles.

Afin de représenter des objets géométriques, il existe de nombreux modèles de représentation. Parmi ceux-ci, on trouve deux grandes familles : la représentation implicite et la représentation explicite. La représentation implicite repose sur l'utilisation d'équations pour représenter les objets. Un plan en 3D est, par exemple, représenté par son équation. Il est alors possible, de manière théorique, de visionner chacun de ses points. De manière pratique, pour visionner ce même plan, un échantillonnage est effectué avec une précision donnée à partir de cette équation.

La représentation explicite repose sur l'énumération, en nombre fini, des éléments composant les objets. Ces derniers sont alors *discrétisés* par un ensemble de sommets, d'arêtes, de faces ou de volumes comme illustré par la figure 1.1.

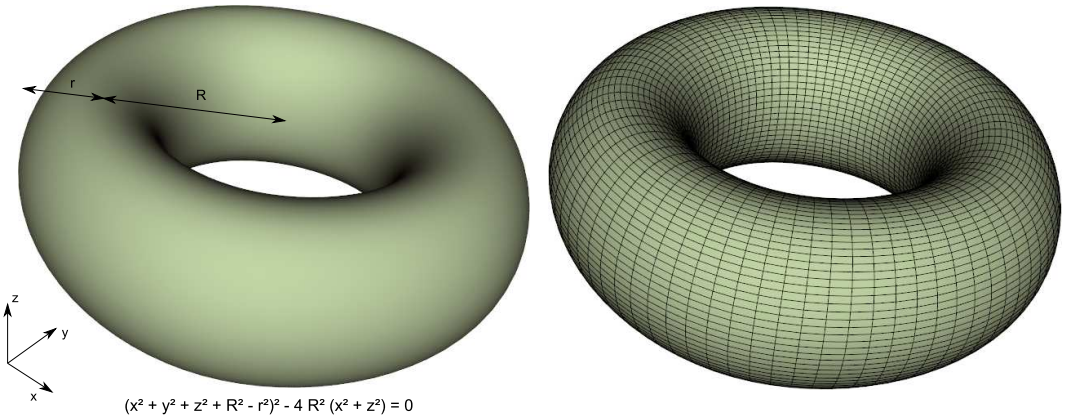

Figure 1.1 : Gauche : Représentation surfacique continue d'un tore par son équation ; droite : représentation surfacique discrétisée d'un tore par un ensemble de sommets, arêtes et faces.

Dans l'ensemble de ce mémoire, nous appelons *objet*, une entité virtuelle représentée par sa surface, son volume. Par la suite, nous nommons également *élément* ou *primitive* une partie d'un objet représenté explicitement (*i.e.* un sommet, une arête, une face ou un volume composant un objet).

Dans le cadre de la simulation, bien que certains travaux aient été effectués sur les calculs d'intersections d'objets représentés de manière implicite, il est plus commun de considérer une représentation explicite. Ce type de représentation est courant dans le sens où les systèmes d'acquisitions actuels sont basés eux même sur une discrétisation du monde réel en primitives élémentaires (sommets dans le cas surfacique, *voxels*[1] dans le cas volumique).

Ces modèles sont également nécessaires aux méthodes numériques utilisées pour résoudre les équations physiques ou mécaniques. C'est le cas des méthodes par éléments finis ou des modèles masse-ressort utilisant les différences finies.

Par la suite, tous les objets que nous considérons sont donc représentés explicitement et sont finis.

1.2 Types de détection de collision

Nous commençons par définir les environnements de simulation et les réponses nécessaires à quelques types de simulations. Nous définissons également dans cette section les notions propres à la détection de collision, utilisées dans

[1] élément volumique équivalent à un pixel en dimension 3

ce manuscrit.

La détection de collision peut être effectuée pour : un seul objet, une paire d'objets, ou un ensemble de n objets.

La détection de collision sur un seul et même objet est communément appelée le cas de l'auto-collision. Ce type de détection de collision correspond à tester les intersections entre diverses primitives d'un même objet, ce type de collision intervient par exemple lorsqu'un tissu se replie sur lui-même. Diverses méthodes ont été proposées afin de résoudre de manière optimale ce problème. Les méthodes mises en œuvre rejoignent pour certaines les idées qui sont détaillées au cours de ce chapitre. De nombreux articles récents traitent ce problème [GKJ+05, CTM08, TCYM09]. Nous choisissons cependant de nous concentrer ici sur le cas de la détection de collision entre des objets disjoints. Le cas de l'auto-collision peut être traité avec les méthodes présentées dans les articles cités ci-dessus.

La détection de collision entre une paire d'objets et n objets est généralement traitée de manière similaire. Lorsque l'on considère un ensemble de n objets, cet ensemble est décomposé en sous-ensembles de paires.

Nous considérons que deux objets sont en collision s'il existe une intersection non vide entre ces derniers, sinon on considère qu'ils sont séparés. Nous disons qu'une collision est manquée lorsqu'une intersection n'a pas été décelée.

Les réponses fournies par les algorithmes de détection de collision peuvent être multiples. On a par exemple, des réponses de type :

- booléenne : on souhaite uniquement savoir s'il y a une intersection entre plusieurs objets, c'est-à-dire s'ils ont au moins un élément en commun. Ce type de réponse permet par exemple de déterminer si un déplacement est autorisé ou non ;
- énumération : on souhaite avoir l'ensemble des éléments en commun entre plusieurs objets. Ce type de réponse peut être utilisé pour sélectionner des primitives d'un objet ;
- approximant : selon une précision variable ou prédéfinie, on souhaite avoir une estimation d'une collision. Ce type de réponse est utilisé pour définir une direction et une zone (potentiellement un volume) de collision à prendre en compte pour la réponse physique.

En plus des différents types de réponses, on distingue deux types de détection de collision, possédant chacune leurs caractéristiques : la détection de collision effectuée de manière discrète et la détection de collision effectuée de

manière continue.

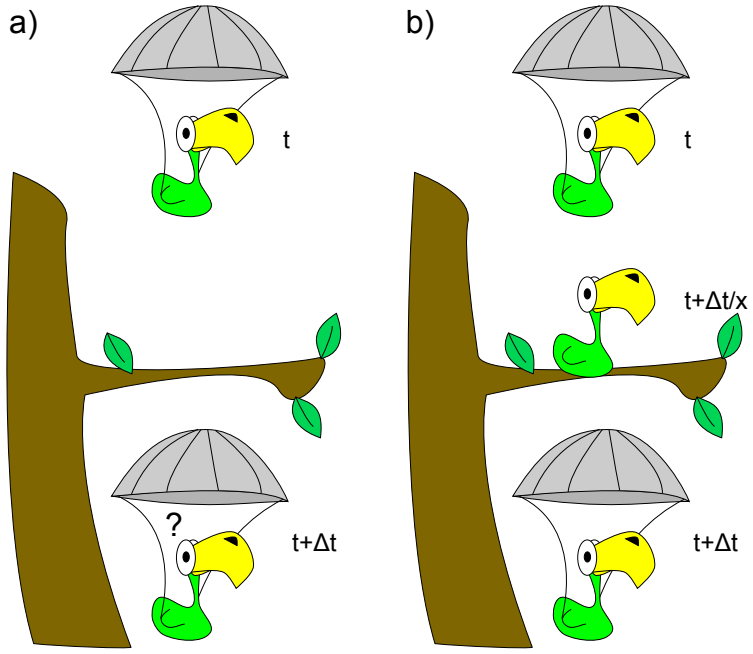

Figure 1.2 : a) déplacement discret : collision manquée. b) déplacement continu : collision détectée.

La détection de collision discrète considère des objets figés, à une position et un instant donnés, afin de déterminer si oui ou non ils sont en collision. La détection de collision continue considère quant à elle l'espace balayé par chaque objet lorsque ceux-ci sont déplacés. Il s'agit alors de déterminer si ce déplacement entraîne une collision et, si c'est le cas, à quel moment du déplacement elle s'est produite.

L'avantage de la détection de collision continue se situe dans l'impossibilité de manquer une collision à cause d'un échantillonnage de temps trop large tel qu'illustré par la figure 1.2. Toutefois, le fait de ne pas évaluer l'intégralité d'un déplacement pour la détection de collision effectuée de manière discrète permet de diminuer la complexité des tests à effectuer et donc de gagner du temps de calcul.

La détection de collision continue consiste à considérer le volume balayé lors du déplacement des objets au cours du temps et de calculer l'intersection entre ces volumes comme illustré par la figure 1.3. Cependant ces volumes peuvent être en intersection sans que les objets ne rentrent en contact pendant leur déplacement. Il est donc nécessaire de considérer le déplacement en dimension 4, afin de vérifier que ces objets se retrouvent dans une configuration où leurs positions à un temps donné se trouvent en intersection.

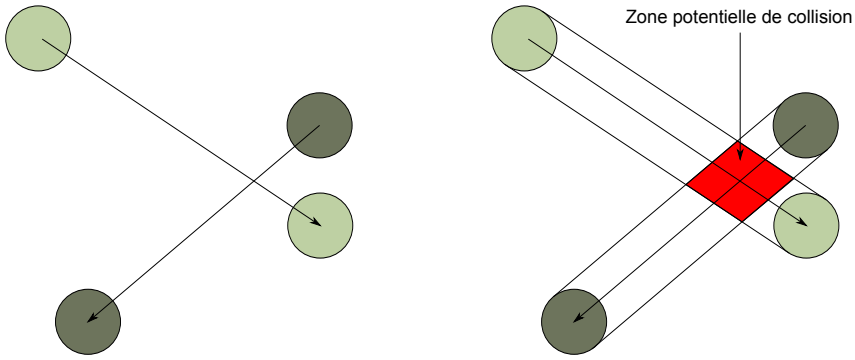

Figure 1.3 : Considération des volumes balayés : une analyse selon le temps permet de définir s'il y a eu ou non collision.

Certaines des méthodes que nous présentons concernant la détection de collision sont basées sur des informations précalculées. Ces méthodes sont particulièrement appropriées pour la simulation d'objets rigides, car aucune modification des informations précalculées n'est à effectuer au cours du temps. Le cas de la simulation d'objets déformables nécessite quant à lui d'effectuer des mises à jours régulières de ces informations. Pour cette raison, nous faisons une distinction entre les divers types d'objets manipulés.

Nous considérons la classification des objets de la manière suivante :
- objet statique : un objet dont la position n'est pas modifiée ;
- objet dynamique : un objet se déplaçant au cours du temps ;
- objet rigide : un objet ne subissant pas de modification de forme au cours du temps (ex : un dé) ;
- objet déformable : un objet subissant des déformations (ex : une peluche) ;
- objet déformable topologiquement : un objet subissant des opérations de couture, d'incision, de perforation, etc. (ex : un organe durant une opération chirurgicale).

Lorsque deux objets sont dynamiques, la plupart des méthodes se ramènent à tester un objet statique avec un objet dynamique. En effet, il est possible d'exprimer le déplacement d'un objet par rapport au déplacement d'un l'autre. Cette transformation réduit alors la dimension du problème.

La détection de collision est également considérée en plusieurs étapes clés :
- l'étape grossière : détermine si des objets sont potentiellement en collision (selon une certaine proximité) ;
- l'étape exacte : détermine de manière exacte l'ensemble des éléments en intersection ;
- l'étape de réponse : détermine les points, les normales et potentiellement les volumes de collision.

Les stratégies que nous allons présenter pour l'accélération de la détection de collision consistent principalement à [Ebe04] :
- effectuer des tests simples pour éviter des calculs coûteux ;
- se baser sur les résultats de tests précédents ;
- essayer de réduire la dimension du problème.

Enfin, les grandes familles de stratégies peuvent être classifiées de la manière suivante : les calculs d'intersections entre volumes convexes, les hiérarchies de volumes englobants, les méthodes approximantes, les regroupements de tests exacts et les accélérations matérielles. Nous définissons et détaillons l'ensemble de ces méthodes dans les parties correspondantes.

1.3 Intersection d'objets convexes

Parmi les optimisations proposées en détection de collision on trouve les algorithmes de recherche de proximité entre deux objets rigides convexes. 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. Ces méthodes se basent sur cette propriété pour orienter la recherche de distance minimale entre deux objets. Cette distance, lorsqu'elle est nulle (ou négative pour une distance orientée), indique la présence d'une intersection entre ces deux objets.

Nous présentons cette famille d'algorithmes car elle est fréquemment utilisée. Son utilisation provient du fait que les intersections d'objets concaves (*i.e.* non convexes) peuvent être ramenées à des ensembles de tests d'intersections d'objets convexes qui sont moins coûteux à effectuer.

Parmi les méthodes d'intersections d'objets convexes, les plus répandues sont celles de :
- Gilbert, Johnson et Keerthi ;
- Dobkin et Kirkpatrick ;
- Lin et Canny.

A partir de ces méthodes, de nombreuses améliorations et extensions ont été proposées. Nous présentons ici les méthodes de base et quelques extensions de la littérature.

1.3.1 Gilbert, Johnson et Keerthi

L'algorithme de Gilbert, Johnson et Keerthi [GJK88], plus communément connu sous le nom de GJK, vise à calculer la distance entre deux objets convexes séparés, ou à donner une approximation de la distance d'interpénétration pour deux objets en collision.

Cette estimation de la distance est basée sur la différence de Minkowski. La distance entre deux polygones convexes est exprimée directement selon cette différence. Le calcul de cette distance est ramené au calcul de la distance minimum à l'origine de la différence de Minkowski entre les objets à tester.

La différence de Minkowski n'est pas calculée intégralement mais de manière incrémentale pour les tests. La complexité de cette méthode est donc en $m+n$, m et n étant le nombre de sommets des objets à tester.

Une version améliorée de l'algorithme consiste à réutiliser en permanence les points les plus proches de la dernière itération [Cam97]. Lorsque les déplacements sont faibles, les nouveaux éléments les plus proches sont proches des anciens, les trouver nécessite donc moins de calculs.

Les travaux de Vlack *et al.* [VT01] proposent une extension de l'algorithme GJK à la détection de collision de manière continue. Ils consistent à considérer deux polyèdres convexes en déplacement et à identifier, en balayant l'espace parcouru par les polyèdres, si des collisions surviennent.

1.3.2 Dobkin-Kirkpatrick

L'algorithme de Dobkin-Kirkpatrick [DK90] consiste à précalculer une représentation hiérarchique des polyèdres de manière incrémentale en un temps $m + n$, m et n correspondant au nombre de sommets des polyèdres que l'on souhaite tester. Le plus bas niveau de la hiérarchie est le polyèdre original. Le niveau le plus élevé correspond à un tétraèdre. Pour construire un niveau de la hiérarchie, des sommets non-adjacents sont supprimés et associés à des faces.

A partir du niveau le plus élevé de la hiérarchie, les éléments les plus proches sont recherchés. Une fois ces éléments trouvés, seule la partie de la hiérarchie contenant ces éléments est raffinée : les polyèdres étant convexes, les éléments les plus proches au niveau inférieur sont soit ceux déjà trouvés, soit appartenant à la partie qui vient d'être raffinée.

La complexité de cet algorithme de recherche est, dans le pire des cas, en $log(m) \times log(n)$, si les éléments les plus proches se trouvent au niveau d'une feuille de la hiérarchie.

1.3.3 Lin-Canny

L'algorithme de Lin-Canny [LC91] calcule la distance entre deux polyèdres convexes de manière incrémentale. A chaque élément des polyèdres (faces, arêtes et sommets) est associée la zone de l'espace contenant les primitives dont il est le plus proche. Ces zones définissent des *régions de Voronoi* comme montré par la figure 1.4.

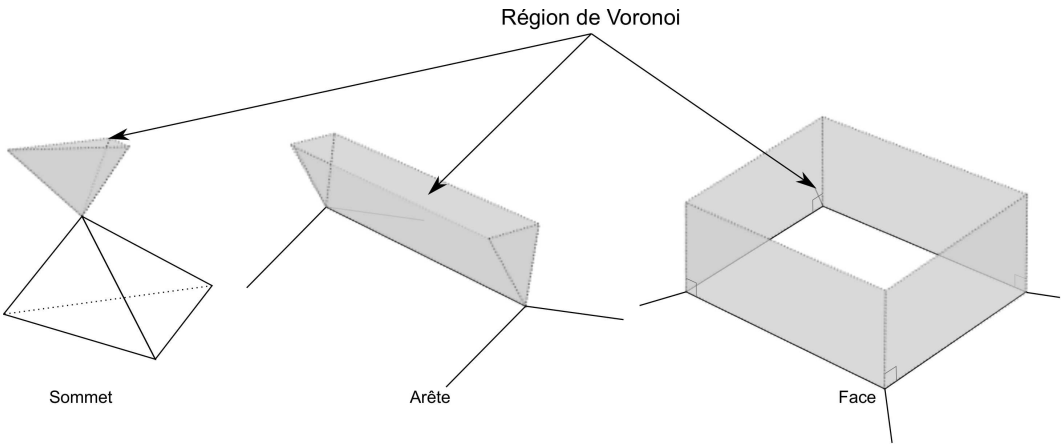

Figure 1.4 : Région de Voronoi d'un sommet, d'une arête et d'une face d'un polyèdre convexe.

Le principe sur lequel se base cet algorithme est le suivant : pour deux objets P et Q, auxquels appartiennent respectivement deux éléments x et y, si x appartient à la zone de Voronoi de y et y appartient à la région de Voronoi de x alors x et y sont les éléments les plus proches entre P et Q. Les éléments x et y contiennent donc les deux points les plus proches entre P et Q. Ces deux éléments permettent de calculer la distance entre les deux objets et de définir s'il y a ou non une collision.

L'algorithme est initialisé avec un élément de chaque objet et progresse de manière itérative sur la surface des deux objets afin de trouver les deux éléments les plus proches en se basant sur les régions de Voronoi. La complexité d'une itération complète est linéaire en fonction du nombre de sommets des polyèdres dans le pire des cas. Le cas où les éléments sélectionnés à l'initialisation sont les éléments les plus proches, ou voisins des éléments les plus proches,

donne un résultat en temps quasiment constant, car la progression itérative à effectuer sur les surfaces est nulle ou faible.

Un algorithme robuste se basant sur ce même principe a été implanté par Mirtich [Mir98] sous le nom V-Clip (pour Voronoi-Clip). V-Clip se base également sur une association de chaque élément du maillage à une région de Voronoi, il corrige certaines situations problématiques comme la génération de boucles infinies lors de la collision de deux objets.

1.4 Partitionnement de l'espace

Les méthodes de partitionnement de l'espace consistent à subdiviser l'espace autour des objets en sous-ensembles disjoints. Ce partitionnement permet de n'avoir à tester des intersections de primitives que lorsque ces dernières sont incluses dans le même sous-ensemble.

Le partitionnement de l'espace par des grilles de *voxels* [Lev66, Tur90] est une approche suivant ce principe. Deux difficultés sont à noter pour cette approche. La première est le fait de devoir mettre à jour constamment la notion d'appartenance d'une primitive à un ou plusieurs voxels. La deuxième concerne le choix de la taille des voxels. Les voxels doivent être suffisamment fins pour pouvoir être sélectifs et diminuer le nombre de primitives à tester. Ils doivent également être suffisamment grossiers pour qu'il n'y ait pas trop de duplications de tests d'intersections. Chaque fois qu'une primitive est incluse dans plusieurs cellules elle est associée à chacune d'elles. Si plusieurs primitives proches sont dupliquées dans de nombreux voxels, le nombre de tests d'intersections à effectuer augmente.

L'utilisation d'une table de hachage sur des grilles régulières de voxels permet d'accélérer les tests d'intersections [THM+03, EL07]. Nous détaillons son utilisation. Les objets sont représentés par des ensembles de tétraèdres. Chacun des sommets et des tétraèdres de la scène est associé à la grille de voxels par le biais d'une table de hachage. Lorsqu'un sommet appartient à un tétraèdre de la scène, une collision a lieu. L'utilisation d'une table de hachage permet ici d'effectuer des tests d'appartenance uniquement pour les sommets et les tétraèdres partageant la même entrée dans la table de hachage. La limite de cette méthode provient du fait qu'elle se base uniquement sur des tests d'appartenance de sommets à des tétraèdres, les collisions d'arêtes ne sont pas détectées. La question du choix de la taille de la grille est fait ici selon la longueur moyenne des arêtes formant les tétraèdres.

Le problème principal du partitionnement spatial par des grilles régulières est l'incapacité à traiter de manière optimale les objets distribués de manière non uniforme [Mou04]. C'est pourquoi des grilles irrégulières ont été mises en place.

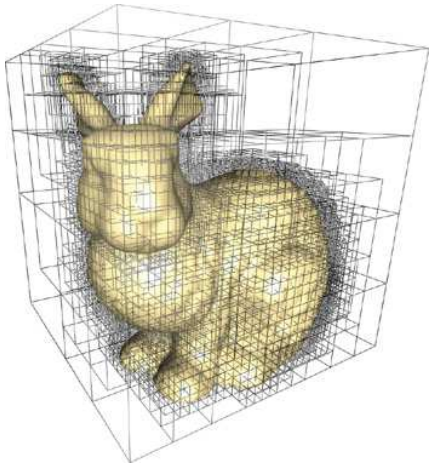

Figure 1.5 : Exemple d'octree. Image de [PF05].

Une méthode de partitionnement par grille régulière répandue est celle des octrees [JT80]. Cette méthode consiste à subdiviser l'ensemble de l'espace en huit sous-espaces. Quand trop de primitives se trouvent dans un même sous-espace, il est alors lui-même subdivisé en huit selon les axes médians. Ainsi la structure des octrees est une hiérarchie de volumes raffinée uniquement aux endroits où il y a une forte concentration de primitives à tester, comme le montre la figure 1.5. Le nombre de subdivisions est déterminé par le nombre de primitives incluses dans les sous-espaces ou par une profondeur fixe [BT95].

Dans le même esprit que les octrees, on trouve également les arbres binaires constitués de plans plus connus sous le nom de *Binary Space Partitionning trees* (BSP-trees) [SBGS]. Les arbres BSP consistent en la définition de plans orientés permettant de définir, par l'intersection de demi-espaces des zones d'inclusions et d'exclusions. L'intersection de ces demi-espaces permet de définir les ensembles d'objets potentiellement en collision [NAT90]. De la même façon, les k-d trees [FBF77] partagent l'espace par le biais d'un arbre binaire de plans et peuvent être utilisés pour la détection de collision. La seule différence ici est que l'ensemble des plans créés est aligné sur les axes afin de simplifier les calculs d'orientations.

Ces dernières structures sont coûteuses à mettre à jour lors de la gestion d'objets dynamiques. Dans ce cas, il est nécessaire de reconstruire l'ensemble

ou une partie des plans formant l'arbre afin qu'ils suivent les objets déplacés [LCF05].

1.5 Hiérarchie de volumes englobants

Afin de détecter s'il y a, ou non, une intersection entre deux volumes, l'utilisation de volumes englobants permet de limiter le nombre de tests géométriques lorsque les objets sont séparés. Un volume englobant est un volume fermé associé à un ou à une partie d'un objet de manière à le contenir entièrement. La limitation du nombre de tests géométriques provient du fait que les volumes englobants sont construits avec peu de primitives comparativement aux objets que l'on souhaite tester. Si l'on considère deux objets dans deux volumes englobants, s'il n'y a pas d'intersection de leurs volumes englobants alors les objets ne peuvent pas être en collision.

Une hiérarchie de volumes englobants, plus connue sous le nom de *Bounding Volume Hierarchy* (BVH) consiste à associer une arborescence de volumes englobants à un objet. Chaque nœud de l'arborescence est associé à un sous-ensemble de nœuds jusqu'à atteindre les feuilles. Les nœuds sont associés à des volumes englobants, plus ou moins complexe. Les feuilles de l'arborescence contiennent des primitives de l'objet géométrique englobé. Chaque nœud père englobe l'ensemble des volumes de ces nœuds fils de manière à ce que l'ensemble des primitives des feuilles de l'arbre soit inclus. L'idée de cette méthode est que les tests sur les volumes des nœuds de l'arbre permettent d'élaguer rapidement des ensembles de tests exacts d'intersections de primitives, plus coûteux en temps de calcul.

Divers types de volumes englobants ont été proposés dans la littérature, chacun ayant des caractéristiques propres. Nous effectuons un classement de ceux-ci selon la simplicité de leur forme. Pour chacune de ces formes, diverses implantations ont été mises en œuvre : on trouve par exemple des hiérarchies de sphères [Hub93], de tétraèdres [JFSO06, JS08], de boîtes isothétiques (AABB : *Axis Align Bounding Box*) [CLMP95], de boîtes englobantes orientées (OBB : *Oriented Bounding Boxes*) [GLM96, RKC02a], de volumes balayés par des sphères (SSV : *Swept Sphere Volumes*) [LGLM99], de polytopes à orientation discrète formés de k plans : les k-DOP (*Discrete Oriented Polytope*) [KHM+98], d'enveloppes sphériques [KPLM98] et d'enveloppes convexes. La figure 1.6 illustre l'application de certains de ces volumes englobants sur un même objet dans le plan.

Plus le volume englobant est simple, plus les tests d'intersections sont rapi-

des. Ainsi pour une sphère seule la distance entre deux points est à mesurer. Cependant lorsque le volume englobant n'est pas adapté à l'objet qu'il approxime, la quantité d'espace vide donnant une réponse positive à une potentielle intersection peut être importante.

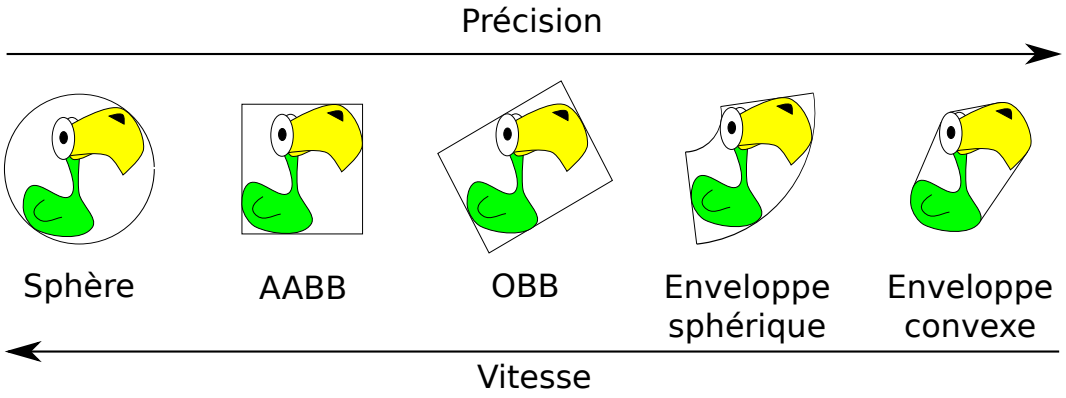

Figure 1.6 : Ensemble des BVH ordonnés selon leur complexité. Plus un volume englobant est simple : plus un test d'inclusion est rapide ; Plus un volume englobant est complexe : plus la précision du test d'inclusion est élevée.

Les méthodes classiques de BVH placent des volumes englobants autour des primitives composant l'objet. Des extensions volumiques ont été proposées pour pallier le problème de l'inclusion complète d'un objet dans un autre. En effet, lorsqu'un objet se retrouve totalement inclus, aucun test de feuilles de la méthode classique ne permet de déterminer la présence d'une collision. Les travaux de Liu *et al.* [LqWhX07] et de Weller *et al.* [WZ09a, WZ09b] proposent des solutions de recouvrement de l'intérieur des objets par des volumes englobants considérés comme contenant de la matière. Ils proposent respectivement des extensions nommées *Inner Space Bounding Volume Hierarchy* et *Inner Sphere Tree*. Cette dernière méthode vise à évaluer le volume d'interpénétration efficacement, en donnant une estimation correcte du volume de collision.

Le choix d'un type de volume englobant doit être fait en prenant en compte divers critères [Ebe04] que nous allons énumérer. Parmi ces critères, la concordance de la forme est assez importante. Un volume englobant doit être adapté à la forme des objets à tester, dans le cas contraire des tests d'inclusions donneront de fausses réponses positives de nombreuses fois. Par exemple, pour un objet avec une dimension grandement supérieure aux autres (*e.g.* une hache), un volume englobant de type sphères contient un grand espace vide, alors qu'une boite (de type AABB ou OBB par exemple) est plus proche de l'objet à tester.

Le coût d'un test d'intersection entre deux hiérarchies de volumes englobants entre également dans les critères à prendre en compte. Ce coût peut être évalué selon l'équation 1.1 [WHG84, GLM96].

$$T = N_v \times C_v + N_p \times C_p + N_u \times C_u \tag{1.1}$$

Dans cette équation nous avons :
- T : le coût total de la fonction de test
- N_v : le nombre de boites englobantes testées
- C_v : le coût du test de collision entre deux boites englobantes
- N_p : le nombre de paires de primitives testées
- C_p : le coût du test de collision entre deux primitives
- N_u : le nombre de paires de volumes englobants à mettre à jour
- C_u : le coût d'une mise à jour d'un volume englobant

La notion de mise à jour apparaissant ici, avec le paramètre $N_u \times C_u$, concerne les mouvements de rotation. L'ensemble des volumes englobants est invariant par translation. Par contre, les volumes englobants de type AABB ou k-dop ne sont pas invariants par rotation. Pour ces deux types de volumes englobants, la notion d'alignement sur les axes est importante, il n'est pas possible d'appliquer une rotation de l'objet sans avoir à prendre en compte une modification du volume englobant.

De manière plus générale, on considère que le nombre de tests à effectuer pour la détection entre une primitive et une hiérarchie de volume englobant pour un arbre de profondeur n est de l'ordre de $log(n)$.

Les tests d'intersections de volumes englobants composés de plans s'effectuent facilement. Il suffit de vérifier que les projections des boites englobantes selon les différents axes définis par les plans indiquent qu'il existe au moins un plan séparant les deux objets. Ces tests sont basés sur le théorème de l'axe séparateur décrit ci-dessous et illustré par la figure 1.7.

Théorème 1 (Théorème de l'axe séparateur)**.** *Pour deux polyèdres convexes A et B, disjoints, il existe un axe sur lequel la projection des polyèdres A et B forment deux intervalles disjoints. Un tel axe est appelé axe séparateur. Si les polyèdres sont disjoints, il existe un axe séparateur orthogonal à :*
- *une face de A*
- *une face de B*
- *une arête de chacun des deux polyèdres*

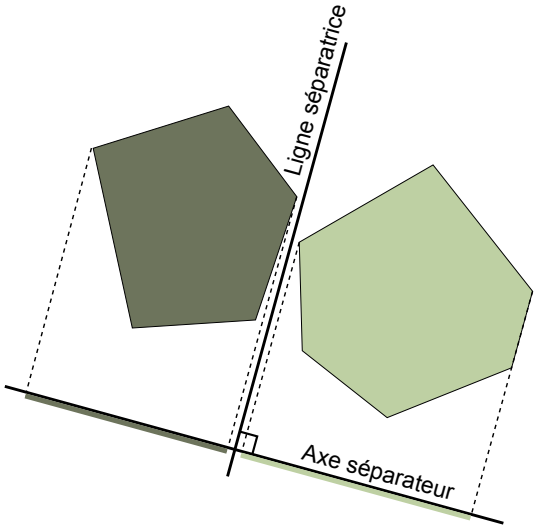

Figure 1.7 : Exemple d'application du théorème de l'axe séparateur dans le plan : les deux polygones convexes sont disjoints, il existe un axe orthogonal à l'une des faces tel que leur projections sur cet axe ne s'intersectent pas.

L'intérêt de l'utilisation des hiérarchies de AABB ou de k-dop se situe dans la connaissance préalable des différents axes de séparation possibles.

Construction de hiérarchies de volumes englobants

Avant de pouvoir effectuer des tests d'intersections sur les différentes hiérarchies, un ensemble de pré-calculs est nécessaire pour leurs constructions. Nous faisons un premier point sur les méthodes de construction de hiérarchies de volumes englobants afin de revenir sur le cas de la gestion des objets déformables. Il existe deux méthodes principales de construction de hiérarchies de volumes englobants : de haut en bas et de bas en haut.

La méthode de construction de haut en bas consiste à partitionner l'ensemble des primitives de l'objet en deux (ou plus) sous-ensembles. Ces sous-ensembles sont ensuite associés à des volumes englobants. Ce partitionnement est répété jusqu'à atteindre un nombre acceptable de primitives dans les feuilles de l'arbre.

La méthode de construction de bas en haut débute par la création de l'ensemble des feuilles et regroupe ensuite les feuilles par ensemble de deux (ou plus) pour former des nouveaux nœuds. Ces nœuds sont alors eux même regroupés de manière itérative jusqu'à obtenir un seul et unique nœud père.

La construction d'une hiérarchie de volumes englobants comportant k primitives est effectué en $\Theta(k\ log(k))$.

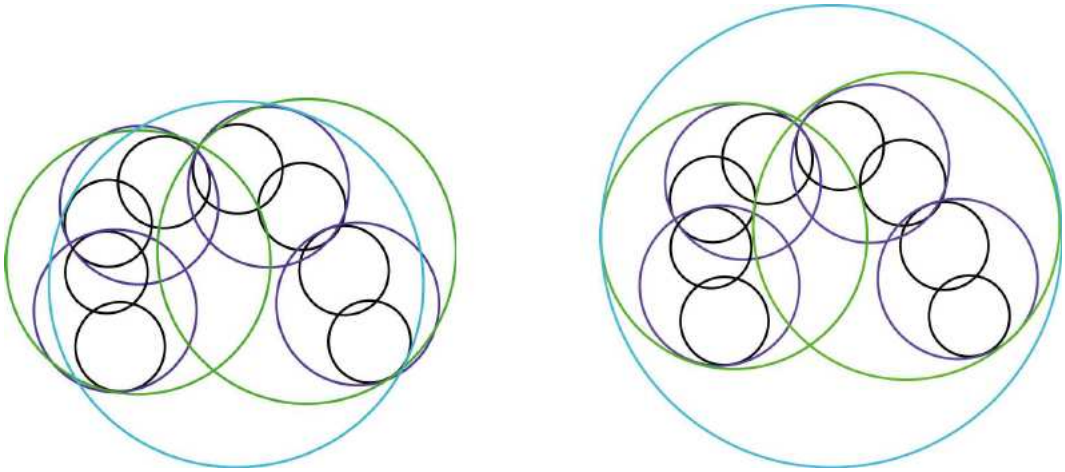

Figure 1.8 : Hiérarchies de sphères : *wrapped* (à gauche) et *layered* (à droite). Image de [GNRZ02].

La méthode de création d'une hiérarchie de volumes englobants est essentielle afin de permettre l'élimination de faux positifs. Comme on le voit sur la figure 1.8 illustrant les travaux de Guibas *et al.* [GNRZ02], le simple fait de créer des volumes englobants par couches superposées (*layered hierarchy*) ou basés sur les primitives contenues dans la hiérarchie (*wrapped hierarchy*) peut modifier de manière conséquente les performances de hiérarchies de mêmes profondeurs. La *layered hierarchy* construit les nœuds pères de manière à englober les volumes englobants de leurs nœuds fils. La *wrapped hierarchy* construit les nœuds pères de manière à englober les primitives contenues dans les feuilles. Sur l'exemple donné par la figure 1.8, on voit que la *wrapped hierarchy* est plus compacte, elle est cependant plus coûteuse à construire.

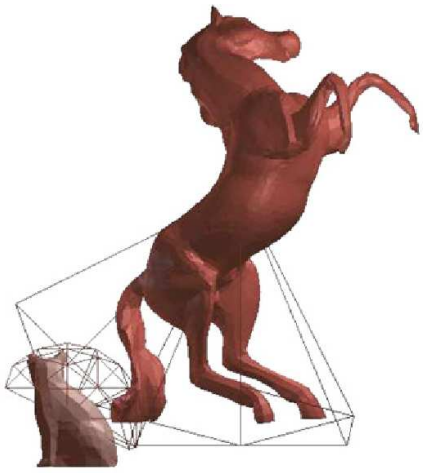

Figure 1.9 : Résultat d'un test d'intersection de tetra-trees. Image de [JS08].

La méthode de volumes englobants composés de tétraèdres [JFSO06, JS08], illustrée par la figure 1.9, utilise une construction différente. Il s'agit, pour cette hiérarchie, de créer des angles de recouvrements formés par des tétraèdres. Un nombre de niveaux est défini. L'environnement est découpé hiérarchiquement par des tétraèdres selon ces niveaux. Puis, les polygones des objets à tester sont associés à ces tétraèdres. Une dernière phase permet de placer la face manquante du tétraèdre afin de donner une borne de distance par rapport à l'objet.

Mise à jour pour les déformations

Lorsque les objets pris en compte subissent des déformations, il est nécessaire de mettre à jour l'ensemble des volumes englobants à chaque pas de temps afin qu'ils soient toujours valides.

Les travaux de van den Bergen [vdB97] démontrent que la mise à jour d'une hiérarchie de boites englobantes peut être jusqu'à dix fois plus rapide qu'une construction complète sur certaines de ces expérimentations. En effet, [LAM06] a montré que la mise à jour d'une hiérarchie de volumes englobants peut être effectuée en temps linéaire selon le nombre de nœuds, là où la construction complète est effectuée en temps sur-linéaire.

Larsson *et al.* [LAM01] proposent une approche hybride pour des objets continuellement déformés. Cette approche met continuellement à jour de bas en haut les $n/2$ premiers niveaux de l'arbre et mémorise si les volumes englobants

se situant en dessous de ce niveau ont été mis à jour. Si un volume non à jour est atteint, il est alors recalculé. Cette technique se base sur l'observation suivante : les volumes englobants occupant un plus grand espace –c'est-à-dire ceux se situant dans la partie haute de la hiérarchie– sont plus fréquemment testés que les autres, il est alors possible d'économiser du temps de calcul sur les volumes englobants les plus fins. Cette méthode est cependant plus gourmande en mémoire car les éléments contenus dans les feuilles doivent être mémorisés dans l'ensemble des nœuds internes à la hiérarchie.

Une autre approche, proposée par Mezger *et al.* [MKE03], consiste à diminuer le temps nécessaire aux mises à jours des arbres en ne les effectuant pas durant quelques pas de temps. Pour ce faire, les volumes englobants sont dilatés dans le sens du déplacement des objets.

Une méthode intéressante a été proposée par James *et al.* [JP04] sur une hiérarchie de sphères. Cette méthode est inspirée d'une méthode adaptée au *morphing* [LAM03]. Elle consiste à ne recalculer les volumes de la hiérarchie que de manière approximée. Il s'agit de déduire à partir de la déformation subie par le modèle, l'augmentation potentielle de la taille de chaque sphère englobante de la hiérarchie. Le problème de cette méthode est qu'elle tend à augmenter la taille des volumes englobants de la hiérarchie.

Yoon *et al.* [YCM07] proposent de baser la reconstruction d'une partie d'un BVH selon différentes métriques. Cette méthode a été appliquée au lancer de rayons pour des scènes dynamiques mais convient parfaitement à la détection de collision dans le cadre d'une application de simulation. Ces métriques permettent d'évaluer la capacité d'élimination d'un volume englobant (selon une heuristique d'aire ; SAH : *Surface-area heuristic*) et donnent une estimation du bénéfice apporté par la restructuration de certaines parties d'un BVH. Cependant, ces métriques ne garantissent pas l'augmentation de la performance dans tous les cas. Le fait de calculer ces métriques peut donc impliquer, dans certains cas, un surcoût comparé aux méthodes de mises à jour sans heuristique.

Otaduy *et al.* [OCSG07] se sont penchés sur le problème de la restructuration de hiérarchies pour des objets subissant des fractures. Leur méthode entraîne une perte de qualité de la hiérarchie de volume englobant mais permet d'accélérer les temps de calcul d'un ordre de magnitude par rapport à une reconstruction complète. Leur méthode est basée sur le rééquilibrage de l'arbre hiérarchique lors de modifications dues à des fractures de l'objet.

Cas de la détection de collision de manière continue

Dans le cas de la détection de collision de manière continue, il est nécessaire de construire les volumes englobants de manière à ce qu'ils incluent intégralement l'ensemble du déplacement effectué par les objets en déplacement. Redon *et al.* [RKC02a] ont traité le cas de hiérarchie de OBB pour la détection de manière continue d'objets rigides. Bridson *et al.* [BFA05] utilisent cette technique pour la simulation d'étoffes.

1.6 Approximation

Certains systèmes de détection de collision effectuent des approximations afin de fournir une estimation des directions d'impact. Ces approches proviennent de deux notions importantes. La première est liée au fait que la perception de réalisme dans une application de simulation est souvent inexacte. Les utilisateurs ne sont généralement pas capables de déterminer quand une réponse à une collision correspond à une réalité physique ou à une approximation. En se basant sur cette idée, la seconde notion, qui est la notion de temps-critique prend son sens. On souhaite généralement pouvoir définir une borne maximum de temps pour l'exécution d'un cycle de simulation (détection de collision, réponse physique et visualisation). Cette borne peut être présente afin de répondre à des requêtes d'information des outils d'interactions, tel que les systèmes à retour d'effort, ou simplement parce que l'on souhaite une animation fluide.

1.6.1 Collision et perception

L'article de O'Sullivan et Dingliana [OD01] traite de la perception des réponses aux collisions. Ces travaux donnent des pistes intéressantes concernant l'exploration des réponses approximatives. Cet article se concentre sur la notion de réalisme d'une réponse à une collision afin de déterminer les situations pour lesquelles une réponse approximative est suffisante. Ils démontrent qu'il est possible de simplifier les calculs au niveau de la détection de collision (influençant donc la réponse) sans détériorer le réalisme tel que perçu subjectivement par l'utilisateur. Il est démontré que les collisions erronées en périphérie du point d'attention sont moins détectées, il en va de même lorsque des éléments distrayants occupent l'espace environnant la collision. Il est également apparu que le temps de réponse à une collision est une notion importante

afin d'identifier la causalité d'événements physiques.

1.6.2 Temps critique

Le concept de temps-critique est lié au fait que l'on souhaite borner le temps de calcul de l'ensemble des fonctions pour assurer un taux de rafraîchissement suffisant [Hub95]. Selon la résolution des objets à prendre en compte il n'est parfois pas possible d'effectuer des calculs précis pour la détection de collision dans le temps imparti. Divers travaux proposent des solutions pour pallier ces problèmes. Les approches présentées dans la suite de cette section sont également liées à cette notion de temps-critique. Nous présentons ici les notions d'interruption, de discrétisation et de réponse probabiliste avant d'aborder les autres approches.

Interruption

La méthode proposée par Gissler *et al.* [GST09] consiste à identifier les zones potentielles de collision en utilisant l'algorithme basé sur la table de hachage spatiale présentée précédemment. Cette étape est considérée comme terminable dans le temps imparti et n'est pas sujette à interruption. Une fois que les zones de collisions sont repérées, l'algorithme calcule l'ensemble des intersections de manière exacte pour déterminer la profondeur d'interpénétration. C'est à ce moment là que les interruptions peuvent avoir lieu. Si ce processus est interrompu, la force de contact est extrapolée selon les derniers pas de temps. Les calculs exacts, qui ont été interrompus, sont reportés au pas de temps suivant.

Discrétisation

Certaines méthodes d'approximations se basent sur une discrétisation des réponses aux collisions. L'ensemble de ces méthodes consiste à donner une approximation de la direction d'une collision et de la force de réponse à fournir selon un degré de précision souhaité.

Ces méthodes se basent sur une résolution plus grossière des objets en déplacement. Une représentation approximative est parfois suffisante pour les besoins d'une simulation. McNeely *et al.*, ainsi que l'ensemble des méthodes

dérivant de celle présentée dans [MPT99], utilisent un échantillonnage en voxels des objets afin de permettre un rendu haptique en temps-réel.

La méthode de Dingliana et O'Sullivan [DO00] consiste, dans le cadre du temps-critique, à estimer des collisions uniquement à partir de sphères. Ces estimations sont utilisées lorsque la contrainte temporelle ne permet pas d'effectuer des tests exacts d'intersections. Une implantation de cette méthode pour les objets déformables a été effectuée [MO06], la mise à jour de la hiérarchie de sphères est basée sur le même principe que celle de [JP04] évoquée précédemment. La méthode de [DO00] a également été améliorée par Giang et O'Sullivan [GO05], en associant à des parties de la sphère les normales des polygones les plus proches dans une phase de précalcul. Cette association permet de diminuer les approximations des réponses basées uniquement sur des sphères.

La méthode de James *et al.* [JP04] calcule les champs de déformations des objets selon les sphères de la hiérarchie utilisée lorsqu'une précision suffisamment fine est atteinte. Il est également possible d'effectuer des évaluations des tests exacts d'intersections de primitives en se basant uniquement sur des calculs d'intersections de sphères approximantes [SJC05].

Probabilité de collisions

Une solution originale proposée par Hsu et Keyser [HK09], consiste à précalculer des tendances statistiques de comportements d'objets en réponse à des collisions. Une première phase de pré-calculs simule la chute d'objets sous de multiples configurations pendant 6 à 8 heures. La détection de collision est ensuite uniquement effectuée sur des sphères. Les statistiques sont alors utilisées pour déduire la réponse la plus statistiquement plausible. Cette méthode est cependant limitée aux objets rigides à topologie fixe.

La technique développée par Klein et Zachmann [KZ03] nommée ADB-Tree (Average-Distribution Tree) est basée sur la notion de probabilité de collision dans un arbre de volumes englobants. A chaque intersection, entre deux nœuds des BVH des objets à tester, est associée une probabilité de collision réelle. Cette probabilité est calculée selon la quantité de primitives contenues dans les différents nœuds de l'arbre. Lorsque la probabilité de collision est suffisamment importante, une collision est signalée.

1.6.3 Multirésolution

Un modèle de représentation multirésolution permet d'accéder à différentes versions des objets correspondant à différents niveaux de détails. Dans les applications de détection de collision, cela permet d'accéder à des échelles de précisions différentes. Autrement dit, un objet a donc à tout moment une représentation grossière et une représentation fine. Parfois, des représentations intermédiaires sont également accessibles. Les modèles multirésolutions permettent d'évaluer des zones d'intérêt de manière localisée. Il s'agit donc d'identifier les zones à élaguer à un niveau grossier, lorsque l'on a déterminé qu'elles ne prennent pas part à une collision, et d'effectuer des tests plus précis dans les autres zones.

SWIFT

La librairie SWIFT [EL00] applique le principe de multirésolution à la détection de collision. Les différentes résolutions sont construites à partir d'une modification de la hiérarchie de Dobkin-Kirkpatrick. La modification principale de cette hiérarchie est la suivante : chaque niveau construit est mis à l'échelle afin d'englober le niveau plus fin à partir duquel il a été créé. A partir de cette hiérarchie l'algorithme de Lin-Canny est utilisé pour se déplacer sur les surfaces des objets. Si, à un niveau de la hiérarchie, une distance suffisante est mesurée entre les objets, l'algorithme s'arrête. Cette librairie ne gère que les objets convexes. Une extension de cette librairie existe sous le nom de SWIFT++ [EL01]. Cette extension décompose au préalable les objets concaves en ensemble d'objets convexes et est enrichie d'une hiérarchie de volumes englobants.

CLOD

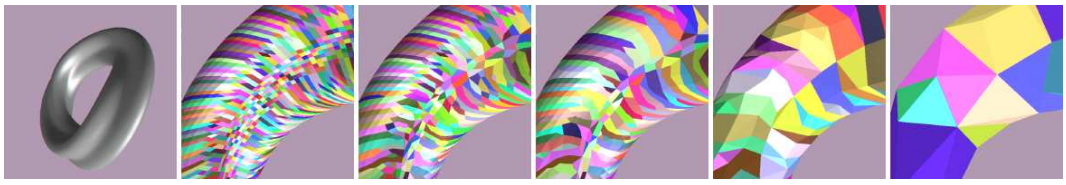

Figure 1.10 : De gauche à droite : le maillage original et les différentes résolutions du modèle. Image de [OL03].

La méthode CLOD (Contact Levels Of Detail) [OL03] se base sur une

représentation multirésolution, illustrée par la figure 1.10. La représentation multirésolution est utilisée en tant que BVH mais aussi pour approximer les réponses aux collisions. La résolution testée est raffinée tant que l'erreur mesurée est supérieure à une borne donnée. Cette erreur est basée sur : la taille concernant le contact, la vitesse des objets en collision et la distance au point de vue de l'utilisateur. L'algorithme supporte également la gestion du temps-critique en mettant la priorité sur les zones ayant un rapport erreur/borne d'erreur important.

Tout comme les BVH, la gestion des objets déformables requiert la mise à jour de l'ensemble des résolutions.

1.6.4 Méthode stochastique

Lorsque la précision de la détection des collisions peut être réduite, les méthodes stochastiques offrent un bonne solution. Elles permettent de borner les temps de calcul tout en donnant une estimation relativement précise des zones de contact.

La plupart des approches stochastiques se basent sur un échantillonnage des objets par des particules. Ces particules peuvent se déplacer sur la surface des objets afin de trouver les zones où la distance est minimale.

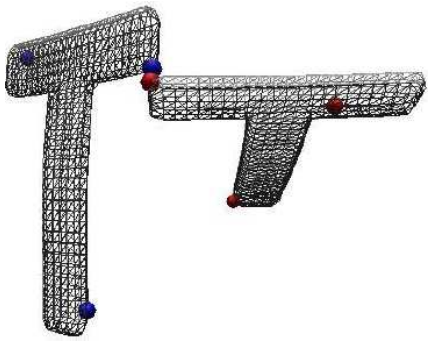

Figure 1.11 : Méthode d'échantillonnage des objets par des particules. Image de [SKSH03].

La technique proposée par Senin *et al.* [SKSH03], illustré par la figure 1.11, définit un système de particules évoluant sur deux objets. Les particules possèdent des propriétés physiques d'attraction et de répulsion. A chaque itération, les particules sont attirées vers les particules de l'objet voisin à tester. Les forces de répulsion sont appliquées pour éviter un agglutinement des particules sur une même zone, qui empêcherait d'identifier d'éventuelles collisions

simultanées en des positions distantes. Des résultats concluant ont été obtenus pour détecter les collisions entre des surfaces composées de 10000 triangles échantillonnées avec 5 particules. Cette méthode fonctionne efficacement sur des objets déformables car les particules suivent les arêtes définissant l'objet quelle que soit sa forme.

Quelques améliorations ont été apportées à cette méthode. Ainsi [SK06] propose de coupler cette méthode avec des zones de déplacement permettant d'améliorer la gestion de collision multi-objets. En définissant des zones de déplacement, il est possible de s'assurer d'une meilleure uniformité de l'échantillonnage sur la surface de l'objet. La création des zones sur un objet est effectuée en calculant, au préalable, le nombre d'objets maximal qui peut entrer en collision avec celui-ci. Lorsqu'un objet est déformé, et qu'une zone devient trop grande, des particules sont ajoutées automatiquement.

L'algorithme présenté dans [KNF04] combine l'utilisation d'un BVH avec un échantillonnage stochastique à base de particules. L'échantillonnage est effectué dans les feuilles de la hiérarchie des volumes englobants potentiellement en collision. En effet, les autres feuilles de la hiérarchie ne peuvent être en collision et ne nécessitent pas de vérifications. Le fait d'utiliser des tests stochastiques dans les feuilles permet d'accélérer les calculs exacts d'intersections de primitives habituellement utilisés. Les paires de particules testées sont déplacées de manière répétée sur les objets, afin de minimiser la distance les séparant. Lorsque la distance passe en dessous d'un certain seuil, une collision est signalée.

La méthode présentée par Joussemet *et al.* [JCA06] aborde le problème des tests stochastiques d'un point de vue de l'intelligence artificielle, des opérateurs de mutations et de croisements entre des particules placées à la surface des objets permettent de sélectionner une population de particules minimisant la distance entre des objets. Les opérateurs de mutations permettent ici d'éviter le blocage dans des minimums locaux.

La méthode présentée dans [WLW+06] est à cheval entre les méthodes de [KNF04] et [JCA06]. Des essaims de particules, basés sur des techniques d'intelligence artificielle, sont générés dans les feuilles d'arbres hiérarchiques afin d'indiquer des collisions potentielles.

Une méthode de détection de collision basée sur un algorithme de type Monte-Carlo a été proposée par Guy *et al.* [GD04], illustré par la figure 1.12. Cette méthode crée de manière aléatoire des paires d'éléments entre les objets à comparer. Ces paires sont alors évaluées selon les paires déjà existantes, seules celles minimisant la distance entre les deux objets sont conservées. Une marche

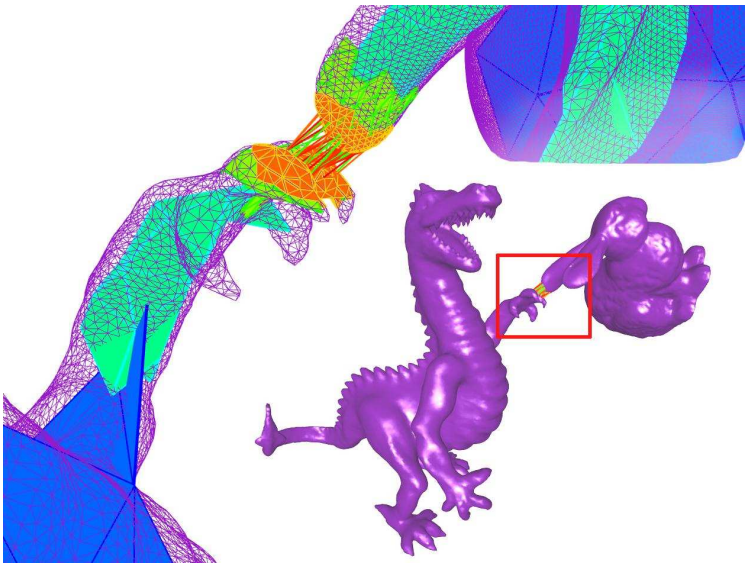

Figure 1.12 : Détection de collision de Monte-Carlo entre deux maillages complexes et déformables en temps réel : on voit l'ensemble des paires de points détectées en proximité, les couleurs indiquent les différentes résolutions. Image de [GD04].

sur la surface, du type Lin-Canny, est effectuée sur les paires proches afin de s'assurer d'identifier les distances minimales localement. Cette méthode est couplée à une représentation multirésolution permettant de n'avoir à évaluer qu'une représentation grossière lorsque les objets sont éloignés.

Les méthodes stochastiques sont efficaces pour limiter les calculs à effectuer pour la détection de collision. Cependant, elles ne peuvent garantir de trouver l'ensemble des collisions, à moins d'un échantillonnage particulièrement fin des objets à tester. De plus, la plupart de ces méthodes ne sont pas capables de gérer les changements de topologies des objets et ne peuvent pas identifier les collisions d'arêtes. Les méthodes stochastiques basées sur des structures de type BVH ou multirésolution souffrent des mêmes limites que ces dernières lorsque les objets à gérer sont déformables.

1.6.5 Adaptivité aux déplacements

Certaines méthodes tentent de prendre en compte les déplacements des objets à tester. Il s'agit ici par exemple d'identifier si la cohérence temporelle entre deux pas de temps est suffisante pour une recherche locale sur les informations précédentes, ou encore de prendre en compte la vitesse relative de déplacement de deux objets.

H-Walk

Guibal *et al.* ont proposé une méthode nommée H-Walk [GHZ99]. Cette méthode consiste à utiliser l'algorithme de Dobkin-Kirkpatrick tout en permettant de parcourir la hiérarchie dans les deux sens.

Nous décrivons le principe de cette méthode. A un temps t, les deux éléments les plus proches ont été trouvés, au temps $t+1$, la recherche des nouveaux éléments les plus proches repart de ces éléments à une résolution i. Le choix de la résolution i est effectué selon la cohérence du déplacement des éléments de la scène.

Lorsque le niveau i n'est pas le plus fin et qu'un nombre borné de déplacement sur la surface permet de trouver les éléments les plus proches, l'algorithme poursuit à un niveau plus fin. Si ce n'est pas le cas, l'algorithme poursuit sa recherche dans un niveau plus grossier de la hiérarchie de Dobkin-Kirkpatrick. Quand les éléments les plus proches sont trouvés pour une résolution grossière, la recherche est alors effectuée en évoluant dans la hiérarchie vers les niveaux plus fins, jusqu'à atteindre le niveau le plus fin.

Lorsque le déplacement relatif des objets est faible, peu de déplacements sur la surface sont à effectuer. Quand le déplacement relatif des objets est important, le fait de passer dans des niveaux plus grossiers de la hiérarchie permet de se déplacer plus rapidement vers la solution optimale.

Critère cinétique

Certains tests de détection de collision se basent sur des critères cinétiques. Ces critères sont utilisés afin de diminuer le nombre de tests à effectuer ou encore afin de limiter le nombre de mises à jour à effectuer sur une hiérarchie de volumes englobants dans le cadre d'objets déformables.

La méthode de Vanecek [Van94] est basée sur le principe de l'élimination des faces cachées (*culling*) utilisé traditionnellement en rendu. Cette méthode est utilisée lors de la phase exacte de la détection de collision. Lorsque l'angle entre la vitesse relative des objets et la normale des faces est supérieur à un angle de 90°, les polygones sont ignorés pour les tests de collision. La figure 1.13 illustre cette méthode. Redon *et al.* [RKC02b] fournissent une extension de cet algorithme appliquée aux hiérarchies de volumes englobants.

Zachmann et Weller [ZW06] quant à eux, proposent un algorithme pour

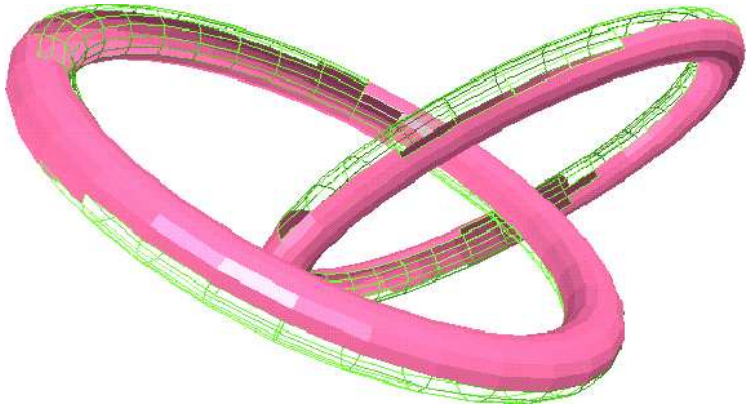

Figure 1.13 : Élimination des faces selon leur déplacement : les faces en fils de fer ne sont pas testées pour la collision. Image de [Van94].

optimiser le maintien de la validité de BVH. Les mises à jour des BVH sont basées sur les événements internes à ceux-ci, c'est-à-dire que l'échantillonnage temporel n'est pas fixé mais déterminé par l'échec de certaines conditions. Ces conditions incluent, entre autres, le fait qu'une primitive associée à un volume englobant sorte de ce dernier ou encore que l'un des fils d'un nœud ne soit plus totalement inclus dans ce dernier. Ils ont également étendu cette méthode à la gestion de la détection de collision entre plusieurs BVH et aux auto-collisions [WZ06] selon le même principe.

La méthode de Liu *et al.* [LSGR05] couple la notion d'unique niveau de détail possédant plusieurs résolutions simultanément et les propriétés de cohérence temporelle que l'on peut trouver dans les applications à interface haptique. Ce couplage s'effectue par une structure nommée SPM (*Space Partitioned Multi-resolution*).

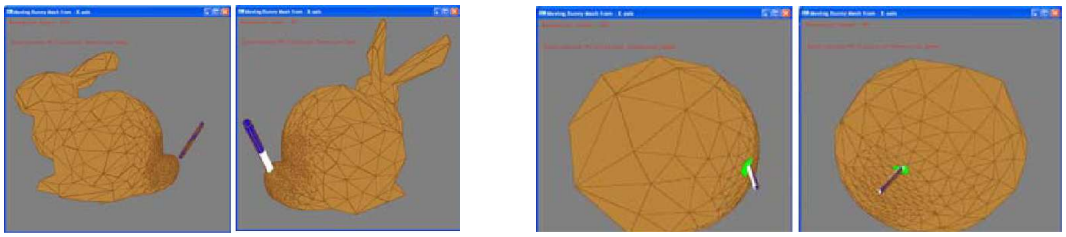

Figure 1.14 : La structure SPM se raffine selon la proximité de l'outil d'interaction. Image de [LSGR05].

Cette structure permet pour un seul et même objet d'avoir plusieurs résolutions différentes qui cohabitent. La proximité de l'outil haptique manipulé entraîne une résolution locale fine de l'objet, une résolution grossière est utilisée sinon. Les différentes résolutions évoluent selon le déplacement de l'outil haptique pour ne raffiner que les zones d'intérêt comme l'illustre la figure 1.14.

Cohérence temporelle

La technique proposée par Li et Chen [LC98] consiste à mémoriser, à chaque pas de temps, la liste des volumes englobants testés qui ne se recouvrent pas, alors que leurs parents s'intersectent. Chaque test entre deux BVH repart alors de la liste mémorisée au pas de temps précédent. En partant de la supposition que la cohérence temporelle est suffisamment élevée, cette méthode permet d'économiser des tests d'intersections sur l'ensemble des parents des nœuds de la liste courante. Lorsque les objets sont en train de s'éloigner, la procédure de test des arbres hiérarchiques standard est alors reprise.

Tropp *et al.* [TTSD06] proposent non seulement de mémoriser l'ensemble des volumes, mais également les axes séparateurs trouvés sur les volumes englobants. En effet la probabilité que l'axe séparateur entre deux objets en mouvement reste similaire entre deux pas de temps est assez élevée.

Ces méthodes permettent d'atteindre, de manière périodique, une complexité amortie constante malgré l'utilisation d'une hiérarchie de volumes englobants, contrairement à la méthode classique qui à une complexité logarithmique.

1.7 Suppression des redondances

Lors de la phase exacte de détection de collision, le test le plus répandu est celui de l'intersection entre deux triangles. Afin de tester l'intersection entre deux triangles, 15 tests sont à effectuer : 6 tests correspondant à l'intersection d'un sommet d'un triangle dans la face de l'autre et 9 tests correspondant à des tests d'intersections d'arêtes [Pro97].

Lorsqu'aucune structure n'est utilisée pour représenter les objets à considérer, l'ensemble des tests est effectivement à opérer. La présence d'une structure topologique au niveau des objets, c'est-à-dire la présence de relations d'adjacences et d'incidences entre les triangles d'un même objet, permet certaines optimisations. En effet, la plupart du temps lorsque deux objets sont proches, des ensembles de triangles adjacents sont à tester de manière exacte. Dans ce cadre, il est alors possible d'effectuer des regroupements de tests d'arêtes et de sommets qui sont communs aux triangles adjacents entre eux.

La figure 1.15 illustre le cas d'un test d'un ensemble de triangles adjacents entre eux par rapport à un autre triangle indépendant. Sur cette figure,

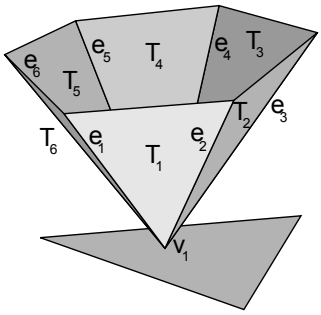

Figure 1.15 : Les tests élémentaires d'intersection triangle/triangle d'un ensemble de triangles adjacents permet le regroupement de tests : le sommet v_1 appartient à l'ensemble des triangles T_i, chaque arête e_i appartient à deux triangles.

l'ensemble des triangles adjacents possèdent des éléments en commun : le sommet v_1 est partagé par tous les triangles et chacun d'entre eux partage deux arêtes avec ses voisins. Il est alors possible de ne tester qu'une fois l'intersection du sommet v_1 avec le triangle indépendant, et de diviser par deux le nombre de tests d'intersections d'arêtes requis.

Nous ne cherchons pas ici à approfondir plus le problème. Les articles [WB06, CTM08, TYM08], se penchent sur l'élimination des redondances dans l'ensemble de ces tests. L'idée principale est de permettre la mémorisation de l'ensemble des éléments formant un triangle qui ont déjà été testés ou qui seront testés par un autre triangle afin d'éviter la duplication de ces tests.

1.8 Accélération matérielle

L'évolution actuelle du matériel, tant des cartes graphiques que des processeurs, tend vers des architectures parallèles. Les algorithmes présentés dans ce chapitre sont pour la plupart parallélisables. Certains travaux se sont penchés sur la question de l'optimisation de ces algorithmes pour prendre en compte les contraintes inhérentes aux architectures parallèles. La parallélisation, pour être performante, doit : minimiser les temps de transfert de données et minimiser les zones bloquantes des algorithmes afin que les ressources puissent être exploitées à leur maximum. Nous décrivons ici certains des travaux portant sur une parallélisation adaptée aux architectures matérielles actuelles. Nous abordons la question de l'exécution parallèle de tests sur des hiérarchies de volumes englobants, puis nous présentons certains des travaux concernant l'utilisation de cartes graphiques. Cette partie nous permet également d'aborder la notion de détection de collision basée sur des techniques de rendu.

1.8.1 Parallélisation de la comparaison de deux BVH

Un ensemble de méthodes vise la parallélisation de l'exécution des tests d'intersections entre des hiérarchies de volumes englobants [HFSQ01, KHH+09, TMT10]. En effet, les comparaisons entre volumes englobants sont répétées un grand nombre de fois et se prêtent particulièrement à la parallélisation.

La parallélisation devient rentable lorsque suffisamment de tests d'intersections doivent être effectués et lorsque ces tests sont correctement répartis sur les différents systèmes de calculs, qu'ils soient effectués sur des architectures multi-cœurs ou sur des cartes graphiques.

Une contrainte supplémentaire ,liée à l'utilisation de cartes graphiques, provient de la taille limitée de la mémoire disponible pour stocker l'ensemble des informations. Il est nécessaire d'opter pour des solutions qui limitent le nombre de transferts d'information des processeurs vers la carte graphique [ZK07].

Lorsque peu de calculs sont effectués en parallèle, le temps nécessaire au transfert des données peut être supérieur au gain de temps apporté par la parallélisation. Gress *et al.* [GGK06] proposent de restreindre l'exécution parallèle de l'algorithme aux situations pour lesquelles le degré de parallélisme est suffisant. Ils parallélisent ainsi les tests d'intersection de volumes englobants uniquement lorsque le nombre de tests à effectuer atteint le nombre de 256.

1.8.2 Utilisation des cartes graphiques

L'architecture des cartes graphiques est particulière et permet certains types d'optimisation que nous évoquons ici. Les cartes graphiques ou GPU (*Graphics Processing Unit*) sont construites selon une architecture multi-cœurs possédant leur propre mémoire. Elles ont été développées afin de permettre l'accélération de certains types de calculs nécessaire au rendu d'images. Elles sont donc efficaces pour effectuer des calculs concernant l'analyse de la géométrie de scènes en 3D.

Après la présentation des optimisations sur GPU, nous revenons dans un dernier point sur l'ensemble des limitations que présentent l'utilisation de cartes graphiques pour les méthodes de détection de collision.

Intersection basée image

Les cartes graphiques sont spécialisées pour le rendu de scènes 3D comportant de multiples objets. Récemment, un ensemble de méthodes a été proposé pour exploiter cette propriété afin d'accélérer les tests de détection de collision.

Parmi d'autres, Hoff *et al.* utilisent une première phase grossière à partir d'une hiérarchie de volumes englobants dans l'espace objet (sur CPU : *Central Processing Unit*), une deuxième phase est effectuée sur carte graphique pour les calculs de proximités entre des régions proches [HZLM01, HIZ+02]. Les calculs d'intersections se basent sur les techniques de rendus standards : la carte graphique permet d'identifier, selon un point de vue donné, si deux éléments possèdent les mêmes coordonnées géométriques et permet donc d'indiquer la présence d'une intersection.

La méthode de Govindaraju *et al.* [GKJ+05] utilise également une accélération sur carte graphique. L'ensemble des tests d'intersections de primitives est effectué sur carte graphique. Cette méthode a également été étendue dans [GKLM07] afin d'effectuer l'ensemble des tests grossiers de volumes englobants sur carte graphique. L'ensemble des tests grossiers est effectué par projection sur les différents axes du repère par rendu d'image.

L'ensemble de ces méthodes est limité par la résolution choisie pour les tests d'intersections entre primitives et ne permet pas la gestion des contacts. L'augmentation de la résolution du rendu image peut diminuer les problèmes d'approximation dus à la discrétisation des primitives, cependant, quelle que soit l'augmentation de la résolution, des approximations sont toujours présentes. Le problème de la gestion des contacts vient de la représentation discrète des données géométriques. La frontière entre deux objets est vue, elle aussi, selon la résolution choisie pour le rendu.

Les méthodes de Govindaraju *et al.* et de Jan *et al.* [GRLM03, JH08] évitent ce problème d'approximation des approches basées images en renvoyant l'ensemble des primitives vues comme en collision au CPU. Les calculs exacts sont alors effectués dans l'espace objet et ne contiennent plus d'erreurs liées à la résolution. Ce système ne prend pas en compte le fait que certaines collisions peuvent simplement ne pas avoir été identifiées dans l'espace image.

Pour pallier le problème de la résolution, Govindaraju *et al.* [GLM04] ont également proposé une méthode basée sur l'ajout de boites englobantes de type OBB autour de chaque primitive, de manière à ce que la discrétisation en voxels lors du rendu image ne manque pas de collisions.

Solveur sur carte graphique

Wong et Baciu ont porté un solveur d'équations polynomiales ainsi qu'un algorithme de calcul de la plus courte distance entre un point et un triangle sur carte graphique [WB05]. Ces implantations sur cartes graphiques se sont avérées plus performantes, malgré les temps de transfert nécessaires, que des implantations sur processeurs. Elles ont ainsi été utilisées pour des algorithmes de détection de collision sur des modèles déformables.

Élimination de paires d'objets

Nous avons traité dans l'ensemble de ce chapitre du cas de la collision entre deux objets. Sud *et al.* ont proposé une méthode d'optimisation basée sur des calculs géométriques effectués sur carte graphique permettant de trouver l'ensemble des paires d'objets à évaluer parmi un ensemble de n objets.

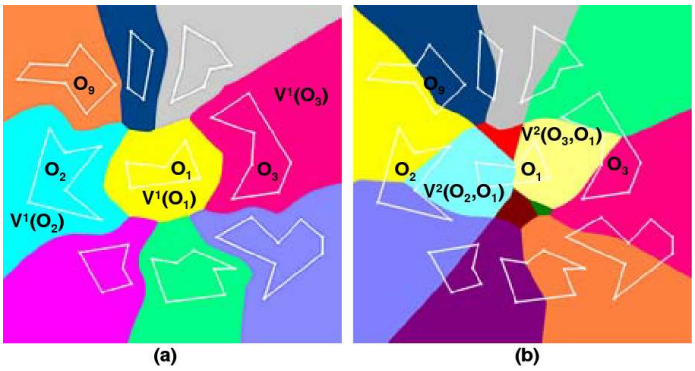

Figure 1.16 : Diagramme de Voronoi du premier et deuxième ordre pour 9 polygones. a) Diagramme de Voronoi du premier ordre : chaque couleur représente l'ensemble des points les plus proches d'un polygone, le polygone O_1 à 8 voisins. b) Diagramme de Voronoi du deuxième ordre : chaque couleur représente les régions proches entre deux polygones, le polygone O_1 a alors uniquement 2 voisins. Image de [SGG+06].

La méthode de Sud *et al.* [SGG+06] construit un diagramme de Voronoi de deuxième ordre sur carte graphique. Ce diagramme de Voronoi permet de réduire, comme l'illustre la figure 1.16, le nombre de paires d'objets à tester parmi n objets. Sans utiliser de diagramme de Voronoi, tester les potentielles intersections entre n objets, nécessite n^2 tests. Son utilisation couplée à des tests sur des AABB permet de limiter ce nombre de tests. Une diminution des temps de calcul d'un facteur 30 à 50 a été observée par rapport à des tests sur des AABB seuls pour des environnements complexes déformables.

Limitations

Les limitations actuelles des cartes graphiques concernent différents points. La latence induite par les temps de transfert d'information des processeurs vers les cartes graphiques est un problème important à prendre en considération. De plus, la taille mémoire des cartes graphiques est à l'heure actuelle plus restreinte que celle disponible pour les processeurs. Il n'est donc pas possible, pour des scènes complexes, de transférer l'intégralité d'une simulation sur celles-ci. La précision des nombres représentés sur carte graphique est également plus faible que celle disponible sur CPU et entraîne donc des erreurs d'arrondis dans certaines situations. Ces erreurs doivent alors être prises en compte afin de gérer au mieux des simulations nécessitant des calculs précis.

1.9 Conclusion

Nous avons présenté dans ce chapitre un panel représentatif des méthodes de la littérature pour la détection de collision. Certaines de ces méthodes sont spécialisées à la détection de collision d'objets rigides de manière discrète, d'autres gèrent les objets déformables géométriquement et/ou topologiquement, parfois de manière continue.

Certaines des méthodes que nous avons présentées sont limitées à des cas précis, comme la détection de collision entre deux polyèdres convexes, et ne conviennent qu'à certains types de simulations. L'interactivité au niveau des calculs des tests de détection de collision est un des buts majeurs à atteindre.

Nous avons volontairement omis la notion de champs de distance parfois utilisée pour effectuer des tests de détection de collision pour son point de vue non interactif dans le cadre d'environnement déformable.

La famille de méthodes majoritairement utilisée est celle des hiérarchies de volumes englobants. Elle permet l'élimination d'ensembles de primitives composant les objets dans le cadre de tests d'intersections exacts. La complexité est néanmoins liée à la structure d'arbre utilisée. Dans le cadre d'objets statiques, cette structure peut être précalculée efficacement. La simulation d'objets déformables soulève d'autres problèmes. Il s'agit alors d'être capable de fournir des algorithmes permettant le maintien de la validité de la structure en un temps minimum. Ce temps, lié à cette structure hiérarchique, implique la mise à jour de nombreux volumes car les objets à tester sont considérés de manière globale.

Une autre famille de méthodes utilise des approximations pour les réponses aux collisions. Elles permettent d'assurer l'interactivité des applications malgré la complexité des scènes. Ces approximations sont mises en place afin de pallier l'impossibilité d'effectuer des calculs lourds dans un temps d'exécution donné et dans le cadre de la simulation d'objets déformables. Dans certaines applications, comme la simulation chirurgicale, les approximations doivent être limitée au maximum afin que les informations renvoyées par le simulateur soient les plus proches possible de la réalité. Ce besoin de réalisme est impératif afin d'assurer la planification de certaines opérations.

Les méthodes permettant de vérifier l'auto-collision et les optimisations des tests exacts entre primitives liées à ce cadre n'ont pas été abordées dans cet état de l'art.

Les procédés d'optimisations se basant sur les accélérations matérielles sont de plus en plus développés. L'architecture du matériel, ainsi que ces performances, sont en constante évolution. Nous avons donné ici un survol de l'ensemble des caractéristiques de ces méthodes sans entrer dans le cœur du problème qui concerne la synchronisation des informations et la minimisation des temps de transfert. Ces méthodes sont principalement des adaptations des méthodes présentées au cours du chapitre.

Chapitre 2

Détection de collision dans un environnement partitionné

Sommaire

2.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, la plupart des algorithmes de détection de collision opèrent entre deux ou plusieurs objets distincts potentiellement séparables par un plan lorsqu'ils ne sont pas en collision. La détection de collision dans le cas où certains objets en mouvement sont entièrement inclus dans un autre est plus rare. Cependant les simulations d'angioscopie, d'endoscopie ou encore de chirurgie micro-invasives se situent dans ce cadre.

Comme évoqué dans le chapitre précédent, les environnements de simulation chirurgicale nécessitent d'être proche de la réalité afin que l'utilisateur bénéficie d'un réel apprentissage ou d'un retour suffisant. Cela implique de gérer des environnements très complexes tant en terme de taille qu'en terme de degré de précision. La gestion de la physique de ces environnements est particulièrement coûteuse en temps de calcul et nécessite également de pouvoir gérer des objets déformables géométriquement, ou encore modifiables topologique-

ment (*e.g.* création de trous, découpage d'éléments,...). Notre objectif est de fournir un ensemble de méthodes permettant de gérer la détection de collision sous ce type de contraintes.

De la même façon que les méthodes hiérarchiques, nous partitionnons la scène en sous-ensembles. Ainsi les objets naviguant (*e.g.* des cathéters, scalpels ou autres instruments chirurgicaux) à l'intérieur d'une scène (*e.g.* le corps humain) ne peuvent intersecter qu'un sous-ensemble des régions de cette partition de l'environnement. Toutefois, nous souhaitons nous abstraire de la notion de hiérarchie pour limiter les coûts de maintien lors de la simulation d'objets déformables.

Nous décrivons les travaux existants sur le déplacement d'objets à l'intérieur d'un partitionnement de l'espace. Nous décrivons ensuite la structure sur laquelle nous nous basons afin de permettre le développement de méthodes de détection de collision adaptées à des environnements complexes, déformables et modifiables topologiquement.

2.1.1 État de l'art

Les méthodes standards de détection de collision, nous le rappelons, considèrent des objets disjoints. Le cadre dans lequel se situe notre problématique n'est pas exactement similaire, étant donnée l'inclusion complète et permanente d'un de nos objets dans un autre.

Par la suite nous appelons *environnement* l'objet englobant pour permettre la différenciation avec les mobiles se déplaçant à l'intérieur.

Des travaux précédents ont été effectués pour ce type de configuration. Nous les décrivons ici et discutons de leurs avantages et de leurs limites.

Une méthode naïve pour tester la collision entre un élément en déplacement et son environnement consiste à tester l'intersection de celui-ci avec l'ensemble des primitives formant le bord de l'environnement, ce qui est coûteux. Les méthodes de hiérarchisation permettent d'éliminer une partie des tests à effectuer. Cependant, lorsque les environnements sont sinueux et tortueux, elles n'arrivent pas à fournir un élagage suffisant. En décomposant l'environnement en cellules (des faces en dimensions 2 et des volumes en dimension 3), il est possible de considérer le problème de manière plus locale. En effet, l'objet ne se déplace que dans un sous-ensemble de cellules de l'environnement. Le fait d'identifier ce sous-ensemble permet de limiter le nombre de cellules à tester

et donc de limiter le nombre de tests d'intersections à effectuer.

Lorsqu'un objet, ou un élément d'un objet, se trouve inclus dans un volume d'une partition, tout déplacement de cet objet ne nécessite que des tests sur ce volume afin de détecter s'il y a, ou non, une collision causée par ce déplacement. Étant donné le fait que nous considérons des environnements amenés à être composés de plusieurs milliers de volumes, le fait de ne tester qu'un nombre limité de volumes réduit de manière conséquente le nombre de tests à effectuer.

Parmi les méthodes de la littérature, nous présentons deux articles utilisant ce principe de partitionnement de l'environnement pour résoudre le problème de la simulation d'angioscopie. Le premier article que nous présentons [LCDN06] décompose l'espace de déplacement avec un arbre composé de volumes. Le deuxième [Gei00] quant à lui décompose l'espace en éléments simpliciaux : des tétraèdres.

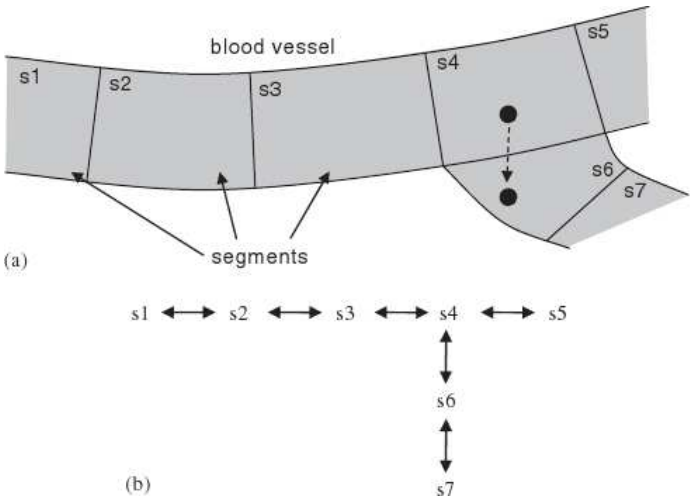

Figure 2.1 : Décomposition d'un vaisseau sanguin en segments et représentation de l'arbre créé. Image de [LCDN06].

La méthode présentée dans [LCDN06] représente un réseau veineux par un arbre de volumes. Chacune des branches de cet arbre représente une partie du vaisseau. Cette approche est particulièrement adaptée au cadre de la navigation dans des vaisseaux sanguins, la structure tubulaire du réseau vasculaire s'adaptant particulièrement bien à une représentation en arbre. Chacun des nœuds de l'arbre est lié à un ensemble de triangles décrivant la surface locale du vaisseau sanguin. L'algorithme de collision est basé sur une recherche exploitant la cohérence temporelle afin de diminuer l'espace de recherche.

Le cathéter est découpé en un ensemble de nœuds. Dès qu'un nœud du cathéter se déplace à l'extérieur d'un des segments volumiques définis (fig-

ure 2.1), l'algorithme recherche le nouveau segment auquel il appartient. La méthode consiste à vérifier, à chaque déplacement, qu'un nœud reste dans son segment. Si ce n'est pas le cas, il s'agit de chercher le nœud de l'arbre contenant sa nouvelle position. Lorsqu'un nœud du cathéter se retrouve en dehors de l'arbre, le segment de l'arbre sélectionné est celui qui, par rapport au segment précédent et au déplacement, est le plus proche en terme de distance.

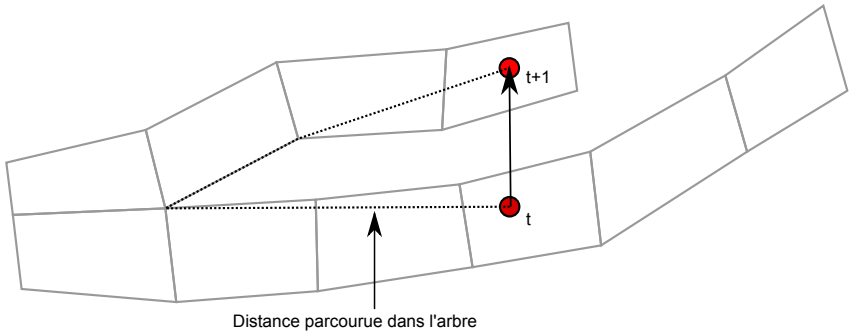

Figure 2.2 : Un déplacement faible dans l'environnement peut impliquer une traversée trop importante dans l'arbre : cela indique qu'une collision a eu lieu.

Si une distance est considérée comme trop importante pour être le résultat d'un seul déplacement, on considère qu'une paroi a été franchie, comme l'illustre la figure 2.2.

Une limite de cette méthode vient du fait que les modifications topologiques et géométriques de l'environnement ne sont pas prises en compte. Ainsi, si deux branches du vaisseau sanguin sont déformées et s'auto-intersectent, la simulation physique l'ignore.

Geiger [Gei00] propose de décomposer l'environnement de base, composé de triangles, en un ensemble de tétraèdres. L'objet navigant est considéré comme un nuage de points. Les tétraèdres de l'environnement contenant l'ensemble des points sont cherchés. Suite à un déplacement, l'utilisation de la cohérence temporelle et spatiale permet de rechercher, à partir de l'ancienne configuration et pour un coût réduit, les nouveaux tétraèdres contenant l'ensemble des points du nuage.

Deux stratégies sont utilisées pour effectuer cette recherche. Si le déplacement de l'objet est grand, la cohérence spatiale est utilisée : une fois qu'un point de l'objet est localisé à l'intérieur d'un tétraèdre, les tétraèdres contenant les autres points de l'objet sont recherchés à partir de celui-ci. Lorsque le déplacement est plus faible, la cohérence temporelle est utilisée : à partir de l'ensemble des tétraèdres contenant les points de l'objet, mémorisés au pas de temps précédent, l'algorithme recherche l'ensemble des tétraèdres contenant

Figure 2.3 : Surface d'une trachée tétraédrisée ; seule une couche est montrée. Image de [Gei00].

l'ensemble des nouvelles positions des sommets de l'objet. En effet, lors d'un déplacement faible, il est probable qu'une grande partie des sommets de l'objet n'a pas changé de tétraèdre.

Une limite de cette méthode vient du fait qu'elle ne permet de tester les collisions qu'entre un nuage de points et une surface ce qui implique donc des approximations d'intersections. Il est donc nécessaire d'augmenter l'échantillonnage de points sur la surface de l'objet navigant afin d'augmenter la précision.

Il faut également prendre en compte le fait que la méthode ne gère pas les déformations de l'environnement et exprime de manière explicite l'ensemble des tétraèdres partitionnant l'environnement. Nous proposons dans notre approche de ne pas nous limiter à un ensemble de tétraèdres mais de gérer toute partition de l'espace composée de volumes convexes et de permettre à cette partition d'être déformable.

2.2 Définition du modèle de représentation

De la même manière que les travaux présentés, nous utilisons un partitionnement de l'espace. Afin que ce partitionnement puisse être utilisé de manière optimale, nous avons besoin de connaître à tout moment les relations d'incidence et d'adjacence des éléments le constituant. La connaissance de

ces relations topologiques permet également de gérer le cas d'environnements déformables géométriquement (*e.g.* déplacement de sommets) et déformables topologiquement (*e.g.* découpage de faces, création de trous,...).

Pour une question de minimisation des coûts de calcul, cette partition est composée d'éléments convexes. Nous définissons ici le partitionnement convexe souhaité et donnons les prémices de l'explication justifiant cette contrainte de convexité. Ensuite, nous donnons la définition du modèle de représentation pour lequel nous avons opté : les cartes combinatoires. La définition formelle de ce modèle de représentation nous permet de nous assurer de la robustesse topologique de nos algorithmes de détection de collision.

Bien que les simulations s'effectuent, dans le cadre général, en dimension 3, nous présentons et détaillons nos méthodes en dimension 2 dans un premier temps afin de faciliter la compréhension du raisonnement présenté.

2.2.1 Subdivision convexe de l'espace

La contrainte de convexité est largement répandue dans le domaine de la détection de collision. Elle permet de vérifier l'appartenance d'un point à une cellule par de simples tests d'orientation point/arête ou point/face selon la dimension. Les figures 2.4 et 2.5 donnent respectivement des exemples en dimension 2 et 3 d'appartenance d'un point à une face et à un volume. La contrainte de convexité associée à la notion d'orientation permet ici de simplifier les calculs d'appartenance d'un point à une cellule.

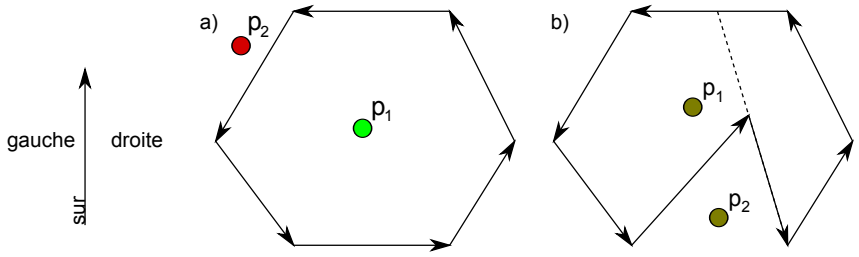

Figure 2.4 : Gauche : Notion d'orientation gauche/droite par rapport à une arête orientée ; Droite : p_1 appartient à la face, p_2 n'appartient pas à la face. Dans le schéma a) la face étant convexe l'appartenance d'un point à une face est vérifiée si ce point est toujours à gauche (ou toujours à droite selon l'orientation de la face) de chacune des arêtes de la face ; dans le schéma b) cette condition n'est pas suffisante pour tester l'appartenance.

Il est également commun de limiter les modèles à des ensembles de triangles ou de tétraèdres pour réduire les calculs. La méthode que nous mettons en place

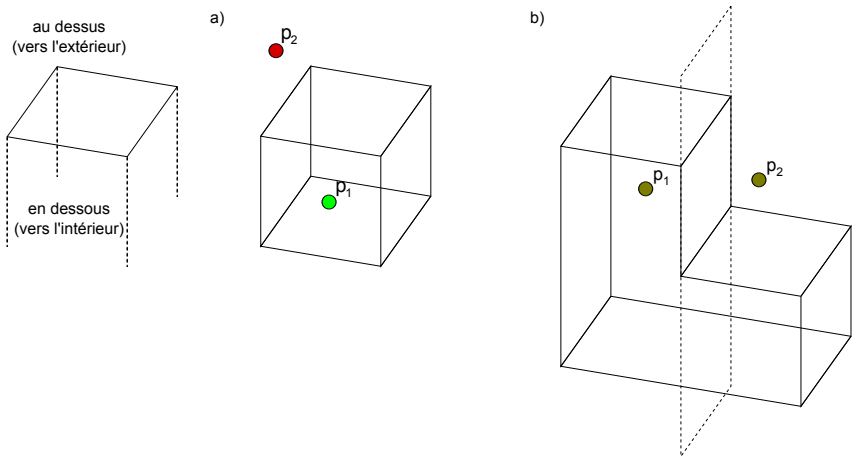

Figure 2.5 : Gauche : Notion d'orientation au-dessus/en-dessous par rapport à un plan orienté ; Droite : p_1 appartient au volume, p_2 n'appartient pas au volume. Dans le schéma a) le volume étant convexe l'appartenance d'un point à un volume est vérifiée si ce point est toujours en dessous de chacune des faces du volume ; dans le schéma b) cette condition n'est pas suffisante pour tester l'appartenance.

se veut plus générale et s'applique à tout type de subdivision de l'espace en polyèdres convexes, sans arête nulle ni face dégénérée.

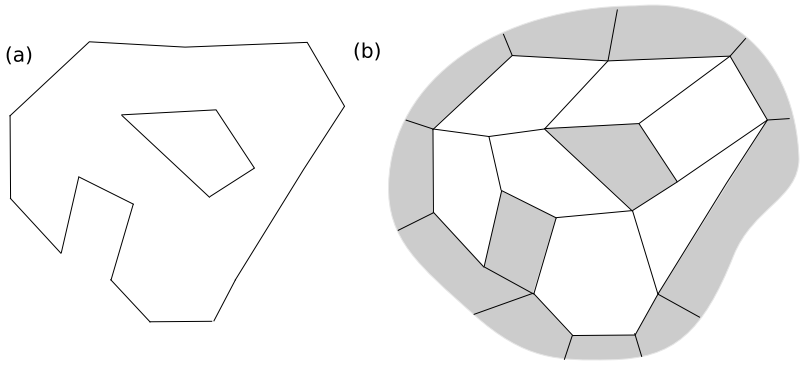

Figure 2.6 : Un environnement 2D subdivisé en polygones convexes. Les faces blanches sont *libres*, les faces grises sont *inaccessibles*.

Les environnements que nous considérons sont subdivisés en cellules convexes. En dimension 2, il s'agit d'un partitionnement du plan en un ensemble de polygones convexes comme illustré par la figure 2.6.b. En dimension 3, il s'agit d'un partitionnement de l'espace en un ensemble de polyèdres convexes dont les faces sont également convexes comme illustré par la figure 2.7.

Les objets inclus dans l'environnement se déplacent dans un ensemble de cellules *libres*. Les cellules à l'extérieur de notre environnement ou représentant des obstacles internes sont *infranchissables*. Des marqueurs associés aux arêtes

et faces inaccessibles nous permettent d'identifier ces zones comme illustré par la figure 2.6.b.

La face extérieure, ou le volume extérieur, selon la dimension, ne sont jamais convexes. Afin d'éviter la gestion de cas particuliers nous ajoutons des arêtes et faces afin d'assurer la convexité de toutes les cellules dites libres. L'ajout de ces éléments nous permet de rendre les zones infranchissables convexes comme le montre le schéma 2.6.b en dimension 2.

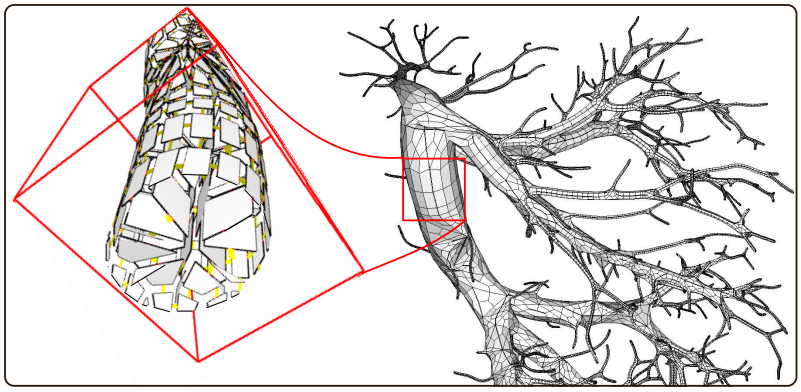

Figure 2.7 : Un réseau de polyèdres ; seul les polyèdres internes sont représentés.

En dimension 3, il est possible d'englober la scène dans un volume plus large contenant toute la scène. La décomposition de ce volume en cellules convexes et son rattachement à l'environnement de base assure la convexité de toutes les zones traversées.

2.2.2 Cartes combinatoires

Il existe dans le domaine de l'informatique graphique de nombreuses structures pour représenter un objet subdivisé en cellules dotées de relations d'adjacences [Bri93]. Parmi celles-ci nous optons pour le modèle des cartes combinatoires qui est une formalisation de ces structures [Lie91]. Une carte combinatoire représente une subdivision orientée de l'espace et est donc adaptée à notre problématique. L'intérêt des cartes combinatoires a été démontré dans de nombreuses publications [EG86, GHPT89, CD99, LSM08]. On note également qu'un modèle multirésolution est déjà existant sur un tel modèle [KCB09]. Cette extension à la multirésolution a été mise en place pour une question de visualisation ou d'adaptativité de précision et peut parfaitement se coupler à un système de détection de collision comme nous l'avons évoqué lors de l'état de l'art général.

Les cartes combinatoires possèdent une définition formelle et mathématique qui nous permet également de nous assurer de la robustesse topologique de nos algorithmes et de veiller à gérer les cas particuliers. Des modèles topologiques similaires ont déjà été implantés dans le cadre de simulation physique afin de traiter des opérations particulières telles que des sutures ou encore des coupures [BGTG03, LT07].

Définitions

Nous débutons par quelques rappels mathématiques.

Définition 1. *Pour E un ensemble fini, on a :*
- *une permutation sur E est une fonction bijective de E dans E (comme E est un ensemble fini, toute permutation sur E peut être représentée par un cycle) ;*
- *une involution est une permutation dans E dont le cycle ne comprend que deux éléments ;*
- *un point fixe dans E, pour une permutation α, est un élément x vérifiant $\alpha(x) = x$.*

Nous définissons également la notion d'orbite. Pour un élément $x \in E$, une orbite définit un sous-ensemble d'éléments de E accessibles, depuis x, par l'application d'une succession de permutations. Nous donnons la définition d'une orbite par une ou deux permutations. Une généralisation peut être effectuée pour n permutations selon le même principe.

Définition 2. *Soient E un ensemble fini, α une permutation dans E et x un élément de E. L'orbite de x, notée $< \alpha > (x)$, est définie par :*

$$< \alpha > (x) = \{x, \alpha(x), \alpha^2(x), \ldots, \alpha^{k-1}(x)\}$$

où k est le plus petit entier naturel tel que $\alpha^k(x) = x$.

L'orbite de x par rapport à deux permutations α et β est notée $< \alpha, \beta > (x)$. Cette orbite correspond à l'ensemble des éléments de E atteignables, depuis x, par l'application des permutations α et β.

Ces notations nous permettent de présenter les cartes combinatoires.

Définition 3. *Une carte combinatoire de dimension n, ou n-carte, est une structure algébrique* $M = (B, \phi_1, \ldots, \phi_n)$ *où :*
- B *est un ensemble fini de brins ;*
- ϕ_1 *est une permutation sans point fixe ;*
- $\phi_2, \ldots, \phi_n$ *sont des involutions avec la propriété suivante : pour tout* $i \geq 1$ *et* $j > i + 1$*, toute composition* $\phi_i \circ \phi_j$ *est une involution.*

La définition formelle des cartes permet de définir la notion de *cellule* topologique, grâce à la notion d'orbite. Dans une carte combinatoire, de dimension 2, la face topologique d'un brin $x \in B$ est définie par l'ensemble des brins de l'orbite $< \phi_1 > (x)$. Par la suite, nous notons i-cellule une cellule topologique de dimension i. Ainsi une 0-cellule est un sommet, une 1-cellule est une arête, etc.

La contrainte sur la composition de permutations de la définition des cartes combinatoires permet d'assurer que toute subdivision construite forme une partition correcte et fermée d'une quasi-variété de dimension n [Lie94].

Dimension 2 et 3

Dans l'ensemble de ce mémoire, nous tâchons de présenter les algorithmes en dimension 2, avant de les présenter en dimension 3 pour une question de clarté. Nous donnons ici la présentation formelle ainsi que des exemples de cartes combinatoires en dimension 2 et 3. Nous précisons également l'ensemble des permutations utilisées pour définir leurs i-cellules.

Définition 4. *Une 2-carte est un triplet* (B, ϕ_1, ϕ_2) *où :*
- B *est un ensemble fini de brins ;*
- ϕ_1 *est une permutation sans point fixe ;*
- ϕ_2 *est une involution.*

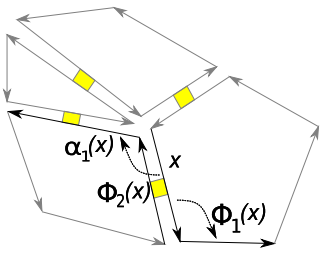

Figure 2.8 : Detail d'une 2-carte. Un cycle de brins forme une face. Les faces adjacentes sont cousues par ϕ_2. Les relations ϕ_2 sont mises en évidence par un rectangle jaune.

Un brin d'une 2-carte appartient à la fois à un sommet, une arête et une face.

La figure 2.8 donne un exemple d'une 2-carte comportant 4 faces. Bien que ce schéma donne une représentation géométrique des arêtes et des faces, nous n'abordons cette représentation que d'un point de vue topologique pour l'instant. La face d'un brin x est définie par l'orbite $< \phi_1 > (x)$. L'arête est définie par l'orbite $\phi_2(x)$. Enfin, le sommet est défini par l'orbite $< \alpha_1 > (x) =< \phi_1 \circ \phi_2 > (x)$.

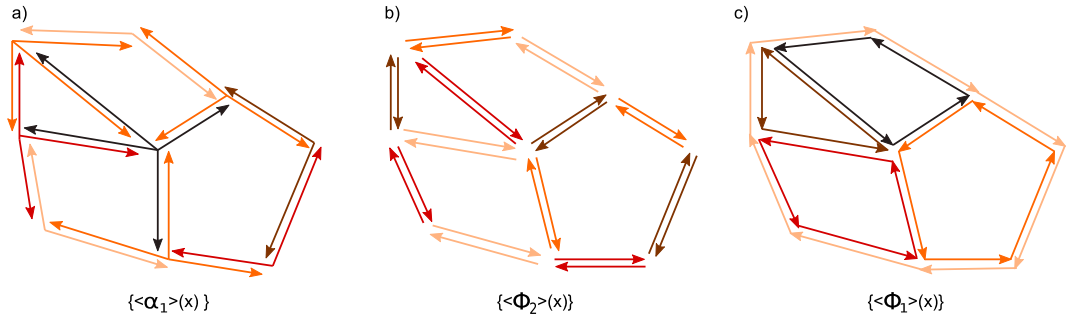

Figure 2.9 : Une 2-carte éclatée selon les orbites. Les brins proches de couleurs identiques correspondent à une i-cellule. Les représentations a), b) et c) de la même carte identifient respectivement les 0-cellules (orbites de sommets), les 1-cellules (orbites d'arêtes) et les 2-cellules (orbites de faces).

La figure 2.9 donne une représentation graphique des différentes orbites.

Comme on le voit sur la figure, une face externe entoure la partition. Cette face externe permet d'assurer la validité des orbites de manière uniforme sur l'ensemble des cellules même lorsque l'environnement est fini. En dimension 3, cette face externe est alors un volume externe.

Définition 5. *Une* 3-carte *est un quadruplet* $(B, \phi_1, \phi_2, \phi_3)$ *où :*
- B *est un ensemble fini de brins ;*
- ϕ_1 *est une permutation sans point fixe ;*
- ϕ_2 *et* ϕ_3 *sont des involutions.*

Pour une 3-carte, la définition des cellules est similaire. Un ensemble de faces cousues par des involutions ϕ_2 forme une surface orientée et fermée qui définit une 3-cellule, ou un volume orienté.

Ces volumes orientés sont cousus deux à deux par des involutions ϕ_3 pour construire une subdivision tel qu'illustré par la figure 2.10. La contrainte de composition de permutations définie sur les n-cartes, précisant que $\phi_1 \circ \phi_3$ doit

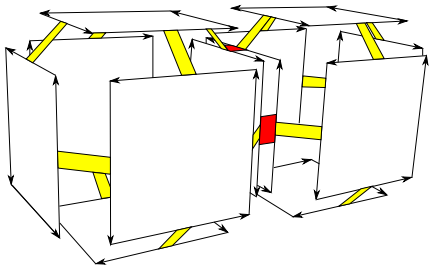

Figure 2.10 : Détail d'une 3-carte comportant deux cubes cousus en ϕ_3 : un rectangle jaune illustre une liaison ϕ_2 ; un rectangle rouge illustre une liaison ϕ_3.

être une involution, implique ici que les volumes soient cousus entièrement le long de leur face commune.

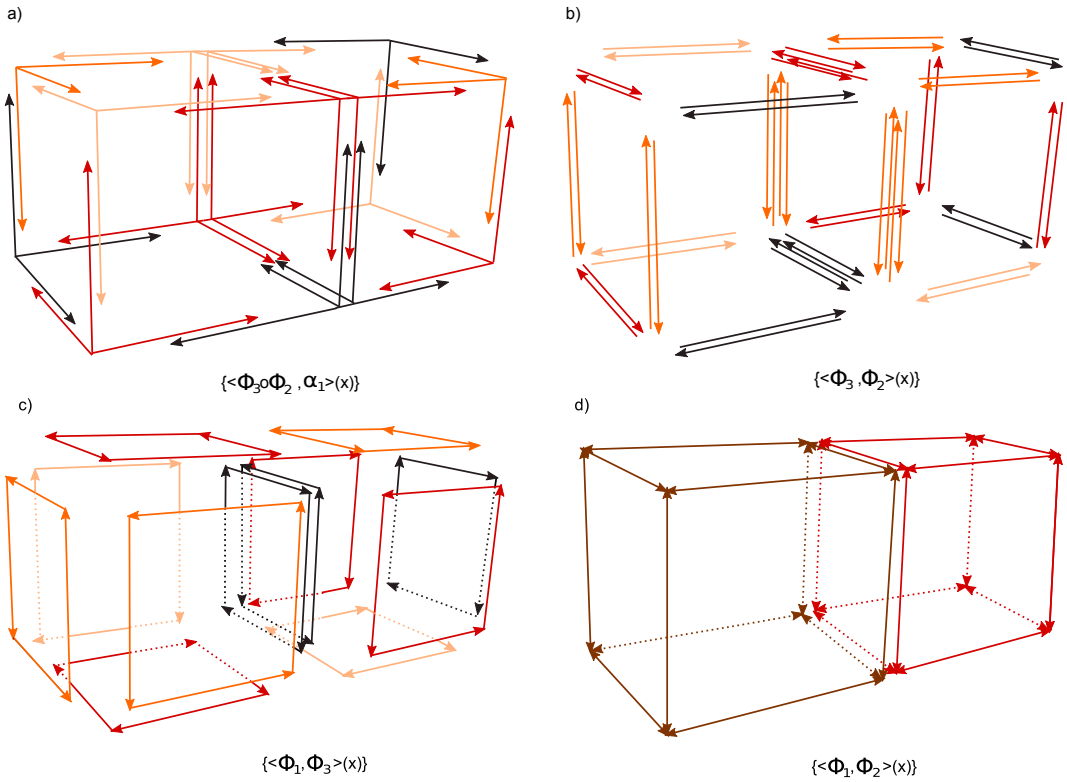

Figure 2.11 : Une 3-carte éclatée selon les orbites. Les brins proches de couleurs identiques correspondent à une i-cellule. Les représentations a), b), c) et d) de la même carte identifient respectivement les 0-cellules (orbites de sommets), les 1-cellules (orbites d'arêtes), les 2-cellules (orbites de faces) et les 3-cellules (orbites de volumes).

La figure 2.11 donne une représentation graphique des différentes orbites pour les 3-cartes. Pour une question de visibilité des points fixes sont laissés. L'orbite $< \alpha_1 > (x)$ correspond à un sommet d'un seul et unique volume. La

définition complète du sommet est donnée par l'orbite $< \phi_3 \circ \phi_2, \alpha_1 > (x)$. L'arête d'un brin x est définie par l'orbite $< \phi_3, \phi_2 > (x)$. La face est définie par l'orbite $< \phi_1, \phi_3 > (x)$. Le volume d'un brin x est défini par l'orbite $< \phi_1, \phi_2 > (x)$.

Plongement

La présentation des cartes n'a été jusqu'ici que topologique. Dans le but de représenter des objets géométriques dans l'espace, il est nécessaire de compléter cette définition par des plongements. Les plongements que nous considérons sont des informations géométriques permettant le placement des cellules dans le plan ou dans l'espace.

Nous rappelons que chaque brin x appartenant à une n-carte représente $n+1$ cellules ; i.e. un brin d'une 2-carte appartient à la fois à un sommet, une arête et une face, un brin d'une 3-carte appartient à la fois à un sommet, une arête, une face et un volume.

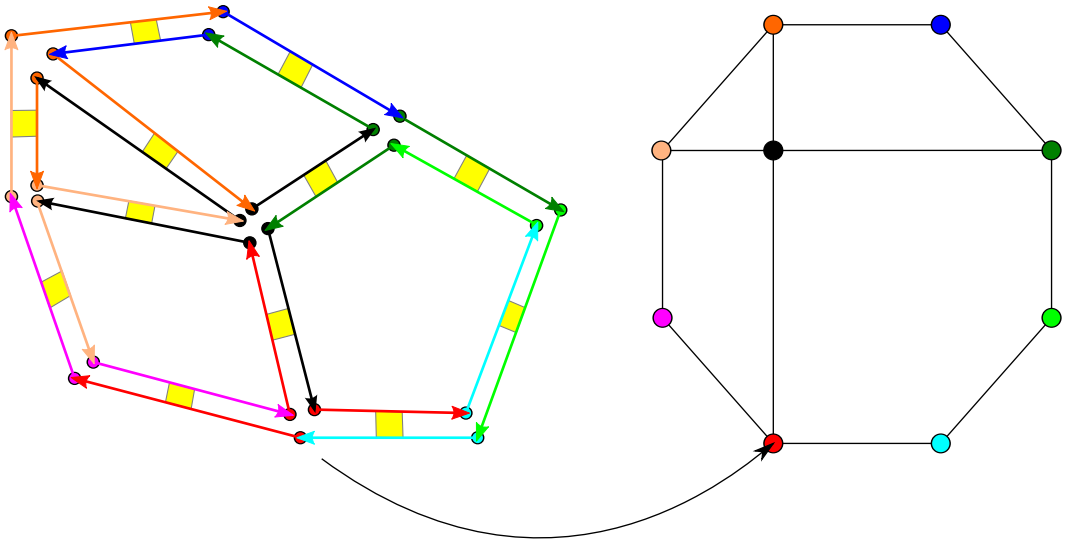

Figure 2.12 : Exemple de plongement d'une 2-carte. Tous les brins de couleur identique possèdent le même plongement de sommet. Tel qu'on le voit sur la droite, les arêtes et les faces sont définies comme une interpolation linéaire de ces sommets.

Nous définissons une fonction *plgt* qui associe à un brin x et pour une dimension i un plongement géométrique. On note $plgt(x, i)$ le i-plongement du brin x. Pour qu'une carte soit bien plongée, il faut que pour tout brin appartenant à une même i-cellule, cette fonction renvoie le même plongement. Pour une 2-carte cela signifie que tous les 0-plongements (i.e. les sommets)

géométriques des brins de l'orbite $< \alpha_1 >$ doivent être identiques comme illustré par la figure 2.12 et précisé par la condition 2.1.

$$si\ plgt(x,0) = a\ alors\ \forall y \in < \alpha_1 > (x)\ on\ a\ plgt(y,0) = a \qquad (2.1)$$

Nous associons ainsi à chaque sommet topologique un plongement géométrique. Un plongement est, selon la dimension choisie, un élément de $\mathbb{R}^2$ ou de $\mathbb{R}^3$. Les 1-plongements (i.e. les arêtes), les 2-plongements (i.e. les faces) et les 3-plongements pour les 3-cartes (i.e. les volumes) sont calculés respectivement comme des interpolations linéaires des 0-plongements, 1-plongements et 2-plongements. Toutes les faces que nous considérons sont planaires. Lorsqu'une face est non planaire, nous la découpons jusqu'à ce que cette contrainte soit respectée.

Avec cette notion de plongement, on peut assimiler un brin à une arête orientée. Cette orientation est implicite au niveau de la topologie de part la notion de prédécesseur et successeur des permutations. Nous veillons à ce que cette orientation reste cohérente du point de vue de la géométrie. Ainsi pour un brin x, l'arête $[plgt(x,0), plgt(\phi_2(x),0)]$ est orientée de x vers $\phi_2(x)$. L'orientation des faces est héritée de cette orientation. Elle nous permet de définir la notion d'intérieur et d'extérieur de manière uniforme sur l'ensemble d'une partition. Nous considérons que les faces sont créées dans le sens trigonométrique. L'orientation des volumes est alors héritée du sens de rotation des faces afin de définir l'intérieur et l'extérieur.

En sus du plongement géométrique, nous complétons les plongements par une autre information. Nous utilisons un marqueur afin de préciser si une cellule est considérée comme franchissable ou non pour la détection de collision. Ce marquage est effectué par face en dimension 2, et par volume en dimension 3. Ainsi pour un brin x de la carte, nous définissons une fonction *obstacle* nous indiquant si le volume auquel appartient x est, ou non, un obstacle.

2.3 Conclusion

Nous avons défini dans ce chapitre le modèle de représentation que nous allons exploiter afin de détecter les collisions entre des objets naviguant à l'intérieur d'environnements partitionnés.

Le modèle, en dimension 2 ou 3, permet de représenter une partition de l'espace en faces ou volumes convexes. Cette contrainte de convexité est utilisée pour simplifier les critères à évaluer pour les tests d'appartenance.

Les cartes combinatoires, en plus de fournir des relations d'adjacence aboutissant à des requêtes de voisinage optimales, définissent un cadre rendant possible une certaine formalisation des algorithmes. Cette formalisation nous permet d'assurer la robustesse et la validité de ces derniers.

L'uniformité de la définition en dimension n des cartes combinatoires, nous permet également, lors du passage de la dimension 2 à la dimension 3, une unification des notations utilisées pour les algorithmes que nous allons présenter dans le plan et dans l'espace.

Chapitre 3

Déplacement de particules

Sommaire

Ce chapitre introduit la problématique du déplacement de particules dans un environnement partitionné. Nous présentons ensuite des algorithmes répondant à ce problème dans le plan, puis dans l'espace. La dernière partie est consacrée à la gestion des environnements déformables et des changements topologiques.

3.1 Introduction

Nous présentons dans ce chapitre l'approche mise en œuvre pour répondre à la problématique de la détection de collision entre des particules et l'environnement à l'intérieur duquel elles se déplacent. Une particule est une masse ponctuelle représentée par un point, doté d'une vitesse et de propriétés physiques.

Le déplacement des particules est régi par un système de simulation physique et/ou par l'interaction d'un utilisateur. Ce système externe fournit des positions à atteindre indépendamment de la présence d'obstacles dans l'environnement des particules. Il est donc nécessaire de vérifier qu'elles n'entrent pas en collision avec un obstacle de l'environnement. S'il n'y a pas de collisions, leurs déplacements sont validés, sinon les informations de collisions ou de contact sont reportées au simulateur afin qu'une réponse appropriée soit fournie.

Nous traitons ici uniquement la question de la détection de collision en laissant la notion de simulation physique pour le chapitre 5.

Le problème que l'on cherche à résoudre est donc de savoir si une particule p entre en collision avec le bord de l'environnement lorsqu'elle passe d'une position p_t à une position p_{t+1} entre deux pas de temps. Quand une collision survient, nous souhaitons avoir des informations exactes concernant la collision, précisément : le point de collision, le temps auquel a lieu l'impact et la cellule où elle a eu lieu ainsi que son type (*i.e.* une face, une arête ou un sommet).

Comme nous l'avons vu dans le chapitre précédent, nous nous plaçons dans le cas où l'environnement est partitionné en cellules. Ainsi, en connaissant à tout instant la cellule contenant p, nous limitons l'ensemble des tests à ces cellules voisines dans l'environnement partitionné.

Un point important, que nous exploitons, est le fait que l'échantillonnage temporel utilisé dans la plupart des simulateurs est suffisamment important pour considérer des *déplacements de taille faible*. Ceci a deux conséquences. La première est que la trajectoire parcourue lors d'un déplacement d'une position p_t à une position p_{t+1} est, ou est proche, d'une trajectoire linéaire. La seconde est que le nombre de cellules de l'environnement traversées par un tel déplacement est réduit.

L'algorithme de détection de collision proposé ici est optimisé, en termes de tests géométriques. L'utilisation d'une décomposition cellulaire de l'environnement permet de n'avoir à faire que des tests locaux à chaque cellule et donc de rester indépendant de la taille ou de la complexité de l'environnement. La localité des tests et l'exploitation de la notion de cohérence temporelle permet d'atteindre un coût calculatoire, pour chaque déplacement de particule, ne dépendant que du sous-ensemble de cellules de l'environnement contenant le déplacement.

Nous revenons sur ce dernier point avec une analyse statistique du nombre de tests d'intersections nécessaires dans le chapitre 6.

3.2 Déplacement de particules dans le plan

Par souci de pédagogie, nous commençons par présenter l'ensemble des algorithmes utilisés pour le déplacement de particules dans le plan. Dans un premier temps, nous donnons l'ensemble des algorithmes de tests d'intersections entre une cellule et une particule en déplacement. Ensuite, nous abordons le point de vue plus général concernant le déplacement complet dans un ensemble de cellules. L'extension en dimension 3 est donnée dans la section suivante.

3.2.1 Tests d'intersections : cellules/déplacement

En dimension 2, l'environnement est partitionné en faces, arêtes et sommets. Nous supposons qu'au départ, la particule se trouve à une position p_1, appartient à un type de cellule k (*i.e.* une face, une arête ou un sommet) et est associée à un brin i de cette cellule. On appelle S_1 le triplet (p_1,k,i) correspondant à l'état initial de la particule. La particule est déplacée vers une position p_2. La question à laquelle nous souhaitons répondre est de savoir si ce déplacement entraîne : un changement de cellule, une collision ou rien, tel que montré sur la figure 3.1.

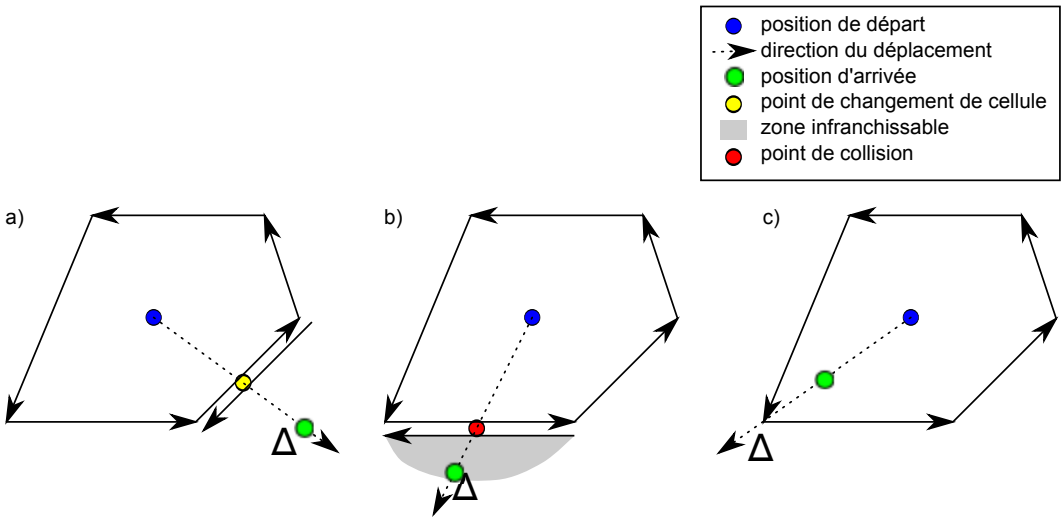

Figure 3.1 : Un déplacement peut a) faire changer de cellule ; b) entraîner une collision ; c) rester dans la même cellule.

Afin de décrire les algorithmes de manière formelle, nous définissons un ensemble de prédicats géométriques dans le tableau 3.1 permettant de préciser les tests géométriques effectués. Ces prédicats sont utilisés dans les algorithmes

de recherche des informations pour la détection de collision et nous permettent d'assurer la validité de l'enchaînement des opérations.

Ils définissent notamment la position d'un point par rapport à des demi-droites ou à des brins. Pour une question de lisibilité, nous utilisons respectivement les termes *gauche*, *droite* et *sur* lorsque l'angle entre un point et une demi-droite appartient aux ensembles $]0, \pi[$, $]\pi, 2\pi[$ et $\{0, \pi\}$ comme illustré par la figure 3.2. Cette notion de positionnement nous permet alors de définir par exemple l'appartenance d'un point à un secteur angulaire, à une face ou encore au bord d'une face. La propriété de convexité des cellules nous permet la simplification du test d'appartenance à une face.

De plus les prédicats permettent de définir de manière précise si un déplacement de particule se fait dans un ensemble de cellules non marquées comme infranchissables, jusqu'à atteindre un brin marqué comme infranchissable ou au travers de cellules marquées comme infranchissables.

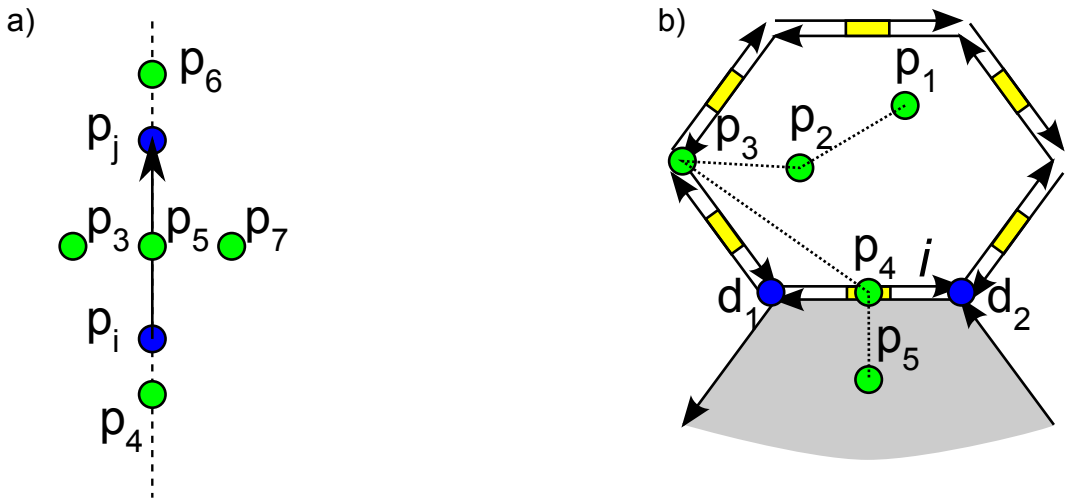

Figure 3.2 : Représentation graphique des prédicats 2D. a) prédicats à gauche, à droite et sur ; p_3 est à gauche de $[p_i, p_j)$, p_7 est à droite de $[p_i, p_j)$, p_4, p_5 et p_6 sont sur $[p_i, p_j)$. b) prédicats d'appartenances et de collisions ; p_1 et $p_2 \in face(i)$, p_3 et $p_4 \in bord(face(i))$; les déplacement définis par p_1, p_2 et p_2, p_3 sont "libre", le déplacement défini par p_3, p_4 est en "contact", le déplacement défini par p_4, p_5 est en "collision".

Notations :

p_1 : position de départ d'un déplacement
p_2 : position à atteindre
$p, p_i, \ldots$: points dans le plan
i : brin de la carte
$plgt(i,0)$: plongement de sommet du brin i
$obstacle(i)$: booléen indiquant si un brin i est franchissable
det : déterminant
"libre","contact","collision" : état d'un déplacement
$< k > (i)$: orbite d'une k-cellule

Prédicats :

Prédicats d'orientation sur des coordonnées :
p *à gauche de* $[p_i, p_j) := det(p, p_i, p_j) > 0$
p *à droite de* $[p_i, p_j) := det(p, p_i, p_j) < 0$
p *sur* $[p_i, p_j) := det(p, p_i, p_j) = 0$

Prédicats sur des brins :
p *à gauche de* $i := p$ *à gauche de* $[plgt(i,0), plgt(\phi_2(i),0))$
p *à droite de* $i := p$ *à droite de* $[plgt(i,0), plgt(\phi_2(i),0))$
p *sur* $i := p$ *sur* $(plgt(i,0), plgt(\phi_2(i),0))$

Prédicats d'appartenance :
$p \in face(i) := \forall d \in < \phi_1 > (i),\ p$ *à gauche de* d
$p \in bord(face(i)) := \neg(p \in face(i)) \wedge \forall d \in < \phi_1 > (i), p$ *à gauche de* $d \vee$
p *sur* d

Prédicats d'états :
$collision(i) := p_2$ *à droite de* $i \wedge (obstacle(i) \vee obstacle(\phi_2(i)))$
$contact(i) := p_2 \in [plgt(i,0), plgt(\phi_2(i),0)] \wedge (obstacle(i) \vee obstacle(\phi_2(i)))$
$libre(i) := \forall d \in < k > (i),\ \neg collision(d) \wedge\ \neg contact(d)$
$libre/collision(i)$:= *si* $collision(i)$ *alors* "collision" *sinon* "libre"
$libre/contact(i)$:= *si* $contact(i)$ *alors* "contact" *sinon* "libre"

Tableau 3.1 : Ensemble des prédicats utilisés en dimension 2.

Si un changement de cellule ou une collision intervient lors du déplacement, cela se fait obligatoirement par l'arête de la cellule courante coupée par la demi-droite $[p_1,p_2)$, que nous notons Δ par la suite. Suite à un déplacement,

nous cherchons d'abord à identifier cette arête car elle localise une intersection potentielle.

Indépendamment de la dimension de la cellule k, évoquée précédemment dans le triplet S_1, tous les algorithmes présentés se déroulent en deux étapes : une phase d'orientation autour de p_1, pour trouver l'arête de la cellule courante potentiellement intersectée, qui définit ce que l'on appelle le brin de *prédiction*, et une phase de déplacement vers celle-ci. Au cours de ces phases plusieurs brins sont testés. On nomme i le brin courant testé et d_1 et d_2 les deux sommets plongeant le brin i.

L'algorithme démarre avec un état S_1=(p_1,k,i) et une destination p_2 à atteindre. A partir de ces informations, on cherche le point p'_1 appartenant à la cellule k le plus proche de p_2, la dimension de la cellule et un brin de la cellule vers lequel le déplacement est effectué et une information sur la validité du déplacement. Cette information correspond au quadruplet (p'_1,k',i',e) où k' est la dimension de la cellule atteinte, i' un brin de celle-ci, et e indique si le déplacement entraîne une collision, un contact ou se fait de manière libre.

Pour chacune des valeurs possibles de l'état k nous formulons la recherche du quadruplet (p'_1,k',i',e) de manière optimisée. Nous détaillons cette recherche dans les sous-sections qui suivent.

Déplacement depuis une face

Quand p_1 appartient à une face, le brin de prédiction définit un triangle formé par p_1 et les sommets plongeant ce brin (figure 3.4).

La phase d'orientation consiste à tourner dans la face à partir du brin i d'origine autour du point p_1, en utilisant la relation topologique ϕ_1. Cette phase continue jusqu'à ce que l'arête coupée par la droite Δ soit atteinte (figure 3.4), c'est-à-dire jusqu'à ce que le brin de prédiction soit trouvé. Chaque brin de la face, associé au point p_1, permet de définir un intervalle angulaire. L'ensemble des arêtes d'une face convexe permet de couvrir l'ensemble des espaces angulaires dans le plan. Nous effectuons des tests d'orientation afin d'identifier si le brin de prédiction a été trouvé. Le brin est trouvé lorsque p_2 est à droite de [p_1,d_2) et à gauche de [p_1,d_1). Sur la figure 3.4 il s'agit de l'intervalle en couleur.

Pour réduire le nombre de tests géométriques, la première orientation détermine si la relation ϕ_1 (sens trigonométrique) ou son inverse ϕ_1^{-1} (sens horaire) est utilisée. Tant que l'intervalle final n'est pas atteint, i est remplacé par $\phi_1(i)$

ou $\phi_1^{-1}(i)$.

La phase de déplacement consiste à vérifier s'il y a, ou non, intersection entre $[p_1,p_2]$ et i. Trois cas sont possibles, p_2 se trouve à gauche du brin i et appartient à la face, à droite du brin et donc change de cellule ou bien sur le brin défini par i ce qui correspond également à un changement de cellule.

Dans le cas où p_1 appartient à une face, p'_1 peut donc appartenir : à la face si p_2 est dans celle-ci (p'_1 est alors égal à p_2), à une arête ou un sommet du bord de cette face s'il y a un changement de cellule et correspond alors à l'intersection entre Δ et i. Dans ce dernier cas, nous vérifions si ce changement de cellule se fait par un des brins considérés comme un obstacle.

L'algorithme 1 décrit l'intégralité de la phase de prédiction dans l'état face. Nous complétons cet algorithme par la définition des préconditions et des postconditions de ce dernier. La figure 3.3 schématise l'organisation des appels de fonctions de cet algorithme.

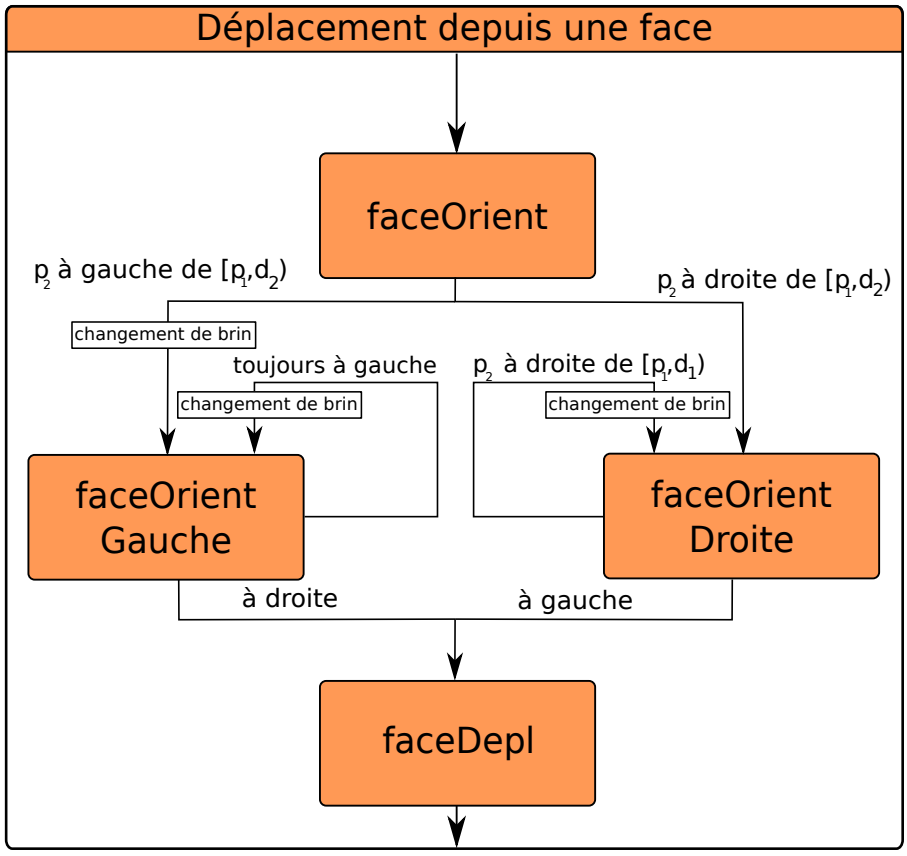

Figure 3.3 : Schéma représentant l'enchainement d'appels de fonctions de l'algorithme 1.

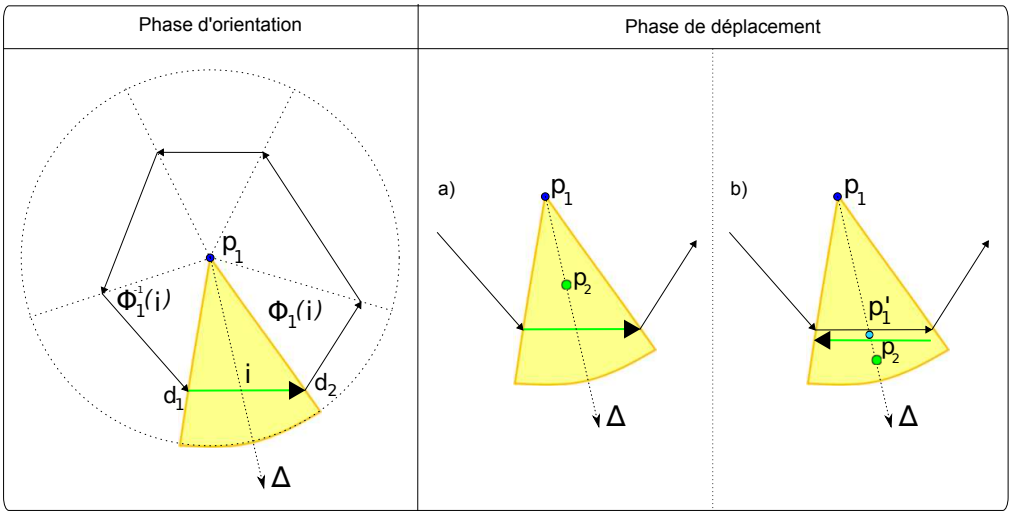

Figure 3.4 : A gauche : Orientation sur une face. Le cycle d'arêtes de la face définit une partition en secteurs angulaires. A droite : Phase de déplacement de l'état face.

Algorithme 1 Déplacement depuis une face

– *faceOrient* (p_1,i,p_2) =
  si p_2 à gauche ou sur $[p_1,d_2)$ **alors**
    faceOrientGauche $(p_1, \phi_1(i),p_2)$
  sinon
    faceOrientDroite (p_1,i,p_2)
  fin si

–*faceOrientGauche* (p_1, i,p_2) =
  si p_2 à gauche ou sur $[p_1,d_2)$ **alors**
    faceOrientGauche $(p_1, \phi_1(i),p_2)$
  sinon
    faceDepl (p_1, i,p_2)
  fin si

–*faceDepl* (p_1, i,p_2)
  si p_2 *à gauche de* i **alors**
    $(p_2, face, i,$"libre"$)$
  sinon si p_2 est sur i **alors**
    si $d_1 \in \Delta$ **alors**
      $(d_1, sommet, i,libre/contact(i))$
    sinon
      $(p_2, arete, i,libre/contact(i))$
    fin si
  sinon
    si $d_1 \in \Delta$ **alors**

      $(d_1, sommet, i,libre/collision(i))$
    sinon
      $((intersection(\Delta, i), arete, i,$
      $libre/collision(i))$
    fin si
  fin si

–*faceOrientDroite* (p_1, i,p_2) =
  si p_2 à droite de $[p_1,d_1)$ **alors**
    faceOrientDroite $(p_1, \phi_1^{-1}(i),p_2)$
  sinon
    faceDepl (p_1, i,p_2)
  fin si

Déplacement depuis une face
Entrée : (p_1,i,face,p_2) Sortie : (p'_1,k',i',e) Préconditions : $p_1 \neq p_2 \wedge (p_1 \in face(i) \vee p_1 \in bord(face(i))$ Postconditions : p_2 *à gauche ou sur* $[p_1,d_1) \wedge p_2$ *à droite de* $[p_1,d_2)$ *et l'une des possibilités suivantes :* – p_2 *à gauche de* $i' \wedge p'_1 = p_2 \wedge k' = face$ – $\neg(p_2$ *à gauche de* $i') \wedge \neg(p_2$ *sur* $[p_1,d_1)) \wedge p'_1 = intersection(\Delta,i') \wedge k' = arete$ – $\neg(p_2$ *à gauche de* $i') \wedge p_2$ *sur* $[p_1,d_1) \wedge p'_1 = d_1 \wedge k' = sommet$

La précondition du test des faces spécifiant que $p_1 \neq p_2$ précise uniquement que le déplacement doit être non nul. Cette condition est mise en place pour éviter de chercher une prédiction de direction dans une face alors qu'aucune direction n'est prise par rapport au pas de temps précédent. Il est possible d'enlever cette précondition en vérifiant ce cas dans l'algorithme.

Déplacement depuis un sommet

Dans le cas où p_1 appartient à un sommet, tout déplacement entraîne une entrée dans une arête ou dans une face. Il s'agit donc ici d'identifier quelle nouvelle cellule est atteinte par un déplacement, et de vérifier si elle est accessible ou non.

La particule est déplacée vers une face ou le long d'une arête. Cet état est le dual de celui de la face. L'algorithme tourne autour du sommet jusqu'à ce que l'intervalle final soit atteint. Les intervalles de recherche de Δ sont définis par les angles formés par les arêtes incidentes du sommet, comme montré sur la figure 3.5. De manière similaire à l'état face, la rotation autour du sommet est effectuée dans le sens trigonométrique ou horaire selon le premier test d'orientation. Ces rotations s'effectuent selon la relation α_1 (*i.e.* $\phi_1 \circ \phi_2$) ou

son inverse (*i.e.* $\phi_2 \circ \phi_1^{-1}$).

Quelle que soit la cellule atteinte lors du déplacement le point p'_1 appartenant au sommet le plus proche de p_2 est le sommet lui-même.

L'algorithme 2 décrit l'intégralité de la phase de prédiction pour l'état sommet.

Déplacement depuis un sommet
Entrée : (p_1,i,sommet,p_2) Sortie : (p'_1,k',i',e) Préconditions : $p_1 = d_1$ Postconditions : $p_2 =$ $p_1 \vee$ p_2 *à gauche ou sur* $[p_1, d_2) \wedge p_2$ *à droite de* $[p_1, plgt(\phi_1^{-1}(i'), 0)) \wedge$ $p'_1 = d_1$ *et l'une des possibilités suivantes :* – ("libre" $\vee$ "contact") $\wedge p_2 = d_1 \wedge k' =$ *sommet* – "collision" $\wedge k' =$ *sommet* – ("libre" $\vee$ "contact") $\wedge \overrightarrow{p_1 p_2}$ colinéaire à $\overrightarrow{d_1 d_2} \wedge k' =$ *arete* – "libre" $\wedge p_2$ *à gauche de* $i' \wedge k' = face$

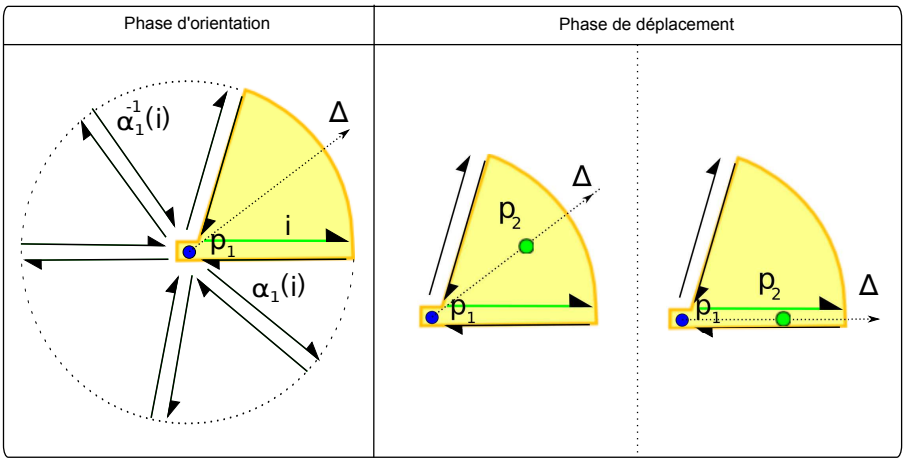

Figure 3.5 : A gauche : Orientation sur un sommet. L'orbite du sommet définit une partition en secteurs angulaires. A droite : Phase de déplacement de l'état sommet.

Algorithme 2 Déplacement depuis un sommet

– *sommetOrient* (p_1,i,p_2) =
  si p_2 sur le sommet d_1 **alors**
    $(d_1, sommet, i, libre/contact(i))$
  sinon
    si p_2 à gauche ou sur $[p_1, plgt(\phi_1^{-1}(i), 0))$ **alors**
      sommetOrientG $(p_1, \alpha_1^{-1}(i), p_2)$
    sinon
      sommetOrientD (p_1, i, p_2)
    fin si
  fin si

–*sommetDepl* (p_1, i, p_2) =
  si p_2 sur i **alors**
    $(d_1, arete, i, libre/contact(i))$
  sinon
    $//p_2$ est à gauche de i
    si i est marqué **alors**
      $(p_1, sommet, i,$"collision"$)$
    sinon
      $(d_1, face, \phi_1(i),$"libre"$)$
    fin si
  fin si

–*sommetOrientG* (p_1, i, p_2) =
  si p_2 à gauche ou sur $[p_1, plgt(\phi_1^{-1}(i), 0))$ **alors**
    sommetOrientG $(p_1, \alpha_1^{-1}(i), p_2)$
  sinon
    sommetDepl (p_1, i, p_2)
  fin si

–*sommetOrientD* (p_1, i, p_2) =
  si p_2 à droite de $[p_1, d_2)$ **alors**
    sommetOrientD $(p_1, \alpha_1(i), p_2)$
  sinon
    sommetDepl (p_1, i, p_2)
  fin si

Déplacement depuis une arête

Dans le cas où p_1 appartient à une arête, un déplacement vers un point p_2 peut engendrer 3 cas. Ce déplacement peut être un déplacement vers une face, un glissement le long de cette arête, ou encore un déplacement vers un des sommets de l'arête.

La phase d'orientation permet d'indiquer si p_2 quitte l'arête en direction d'une face ou s'il appartient à la droite passant par l'arête i'. Si p_2 quitte l'arête, on vérifie si la face vers laquelle il se dirige est libre ou non. Qu'elle le soit ou non, le point p'_1 est égal au point p_1, car le point le plus proche de p_2 appartenant à l'arête est le point p_1.

La phase de déplacement concerne les déplacements le long de l'arête. Nous testons si le déplacement franchit un des sommets d_1 ou d_2. Si p_2 appartient à l'intervalle $]d_1, d_2[$, le déplacement reste sur l'arête et p'_1 vaut p_2. Sinon p'_1 vaut d_1 ou d_2 selon le sommet franchi par le déplacement, l'état k' est alors un sommet et le brin i' est assigné à un brin du sommet correspondant ($\phi_1(i')$ ou $\phi_1(\phi_2(i'))$ selon le sommet atteint).

L'algorithme 3 décrit l'ensemble de ces tests.

Déplacement depuis une arête
Entrée : (p_1,i,arête,p_2) Sortie : (p'_1,k',i',e) Préconditions : $p_1 \in]d_1, d_2[$ Postconditions : *l'une des possibilités suivante :* – $libre/contact(i) \wedge p'_1 = p_2 \wedge k' = arete$ – "collision" $\wedge p'_1 = p_1 \wedge k' = arete$ – "libre" $\wedge p_2$ *à gauche de* $i' \wedge k' = face$ – $libre/contact(i) \wedge p'_1 = d_1 \wedge d_1 \in \Delta \wedge k' = sommet$

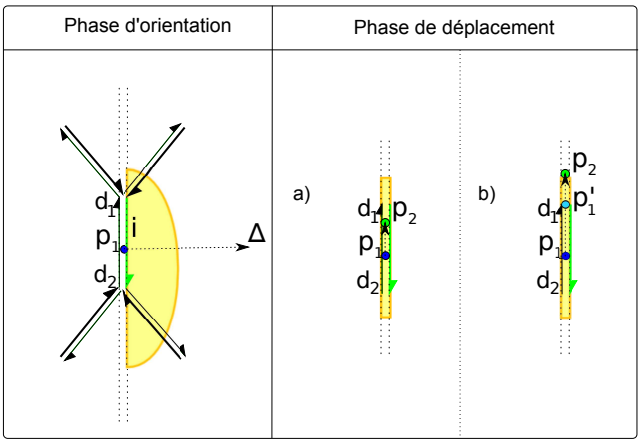

Figure 3.6 : A gauche : Orientation sur une arête. Un test d'orientation indique si le déplacement va vers une face. A droite : Phase de déplacement de l'état arête. On teste si le déplacement atteint un sommet.

Algorithme 3 Déplacement depuis une arête

– *areteOrient* (p_1,i,p_2) =
  si p_2 est à gauche de i **alors**
    testAcces (p_1,i,p_2)
  sinon si p_2 sur i **alors**
    areteDepl (p_1,i,p_2)
  sinon si p_2 est à droite de i **alors**
    testAcces $(p_1, \phi_2(i),p_2)$
  fin si

– *areteDepl* (p_1, i,p_2) =
  si p_2 est avant d_1 **alors**
    si p_2 est avant d_2 **alors**
      $(p_2, arete, i,libre/contact(i))$
    sinon
      soit $i' = \phi_2(i)$ dans
      $(d_1, sommet, i',libre/contact(i'))$
    fin si
  sinon
    $(d_1, sommet, i,libre/contact(i))$
  fin si

– *testAcces* (p_1, i,p_2) =
  si i est marqué **alors**
    $(p_1, arete, i,$"collision"$)$
  sinon
    $(p_1, face, \phi_1(i),$"libre"$)$
  fin si

3.2.2 Déplacement complet

Nous reprenons ici le cas général d'un déplacement d'une position p_t à une position p_{t+1}.

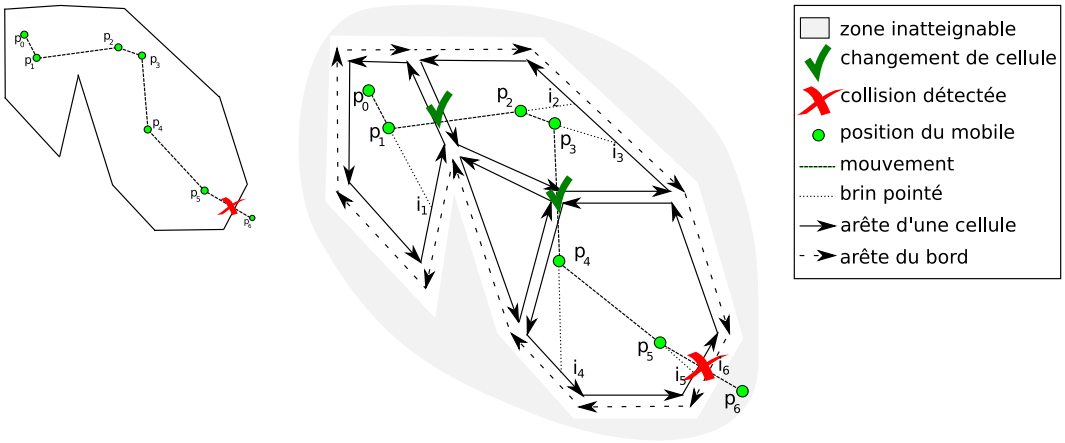

Figure 3.7 : A gauche : une particule se déplace dans un environnement 2D. Au milieu : la décomposition de l'environnement en cellules nous permet de tester les zones potentielles de collision pendant le déplacement de manière locale.

La décomposition en cellules convexes permet de limiter les tests de collisions aux arêtes du bord de la cellule courante, pour peu que l'on sache à tout moment dans quelle cellule on se trouve. Ainsi si p_{t+1} est contenu dans la même cellule que p_t une collision ne peut arriver car il n'y a pas de changement de cellule. Sinon, nous testons la cellule adjacente. Si cette cellule est infranchissable il y a collision. Sinon nous réitérons avec cette cellule jusqu'à trouver la cellule contenant p_{t+1} ou jusqu'à identifier une collision (schéma 3.7).

L'intérêt de cette approche est le suivant : si l'on sait vers quelle arête se déplace le mobile, alors un seul test d'intersection est nécessaire pour savoir si le mobile reste dans la cellule courante ou franchit un de ses bords. Ainsi lorsqu'une particule suit une trajectoire régulière, c'est-à-dire rectiligne ou faiblement courbe, le fait de mémoriser quelle arête est coupée par la droite Δ permet de réduire le nombre de tests à effectuer lors de la phase d'orientation. Il s'agit ici en quelque sorte de prédire la prochaine direction du déplacement à partir du déplacement précédent.

En suivant ce principe nous allons, pour chaque pas de temps, mémoriser l'état de la particule, afin de permettre une recherche dans le voisinage de cette dernière au pas de temps suivant.

Nous avons présenté l'ensemble des algorithmes élémentaires permettant

de tester l'intersection entre un déplacement et une cellule de l'environnement. Il s'agit maintenant, pour un déplacement donné, de décomposer la trajectoire en un enchaînement de ces briques élémentaires.

A un instant t, l'état d'une particule est donné par sa position p_t, le type k_t de la cellule le contenant (face, arête, ou sommet) et l'arête visée de la cellule courante que nous nommons i_t. Au temps $t+1$, le point p_t est déplacé selon une trajectoire, que l'on suppose rectiligne, jusqu'à un point p_{t+1}. Par la suite nous nommons S_t le triplet (p_t, k_t, i_t) représentant l'état courant d'une particule.

Nous notons $S_t^0 = (p_t^0, i_t^0, k_t^0)$ le triplet au début d'un déplacement que l'on initialise par les valeurs de S_t. L'algorithme a pour but de trouver, de manière incrémentale, le triplet final S_t^n correspondant à la prédiction S_{t+1}.

Pour identifier l'état S_{t+1}, plusieurs cellules peuvent être traversées (y compris des arêtes et des sommets). Lorsque cela se produit, des états intermédiaires S_t^j sont calculés jusqu'à obtenir le triplet final S_t^n. En d'autres mots, le mouvement est décomposé selon la subdivision de l'environnement. La question du calcul de l'état (collision, libre ou contact) est déduite automatiquement à partir du brin de prédiction et de la dimension de la cellule renvoyée.

Dans un état $S_t^j = (p_t^j, i_t^j, k_t^j)$ soit S_{t+1} a été trouvé, soit une collision est survenue, ou bien l'état doit être modifié. Le premier cas survient lorsque p_{t+1} se trouve dans l'intervalle défini par k_t^j et i_t^j. Le second cas survient lorsqu'une zone non franchissable est atteinte. Le troisième cas survient lorsque k_t^j ou i_t^j ne correspondent pas à la nouvelle position. Dans ce cas l'algorithme continue avec l'état intermédiaire S_t^{j+1} suivant.

En général, les déplacements sont suffisamment petits pour assurer que seul un faible nombre de changements d'arêtes est nécessaire pendant un pas de temps. Néanmoins, notre méthode traite le cas général où un nombre indéfini de cellules peuvent être franchies lors d'un pas de temps.

Le brin i_t^j nous permet de savoir s'il y a eu collision, changement d'état, ou encore si nous avons trouvé l'état S_{t+1}. Si le point p_{t+1} reste dans l'intervalle défini par le type k_t^j de la cellule et i_t^j alors on a trouvé le bon état. Sinon, soit le franchissement est interdit et nous reportons alors une collision, soit nous devons passer au type k_t^{j+1} de cellule et au brin i_t^{j+1} correspondant au franchissement de cet intervalle.

Le dernier S_t^n obtenu permet donc de dire si un déplacement est autorisé ou s'il entraîne un contact, voire une collision franche, avec une des arêtes infranchissables de l'environnement.

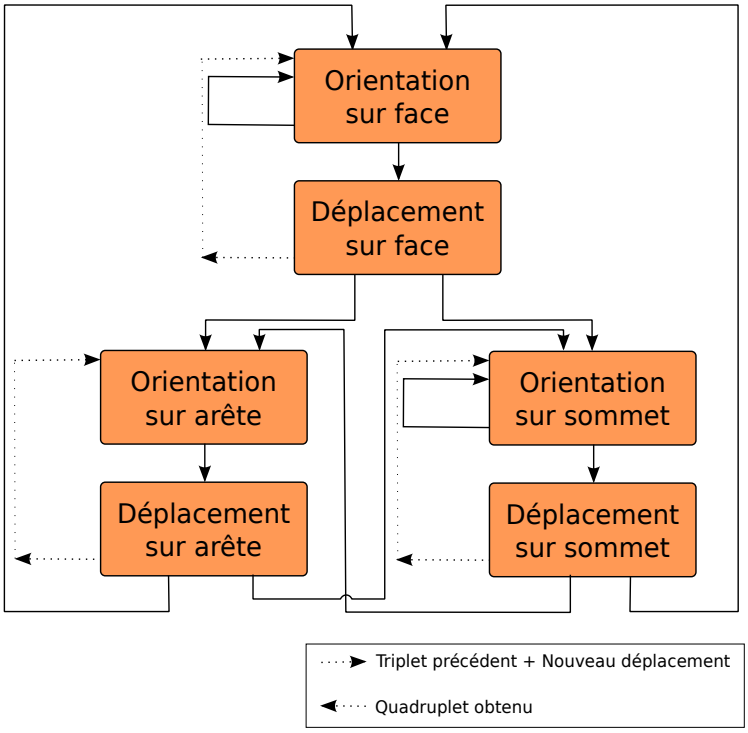

Figure 3.8 : Recherche du quadruplet correspondant au triplet de départ et à une position à atteindre par l'enchaînement des différents algorithmes.

Lorsque le suivi nécessite le passage par plusieurs cellules, les différentes prédictions s'enchaînent tel qu'illustré par le schéma 3.8.

Une optimisation est à noter lorsque l'on passe d'une face à une autre en passant par une arête. L'information concernant la position du point de destination par rapport à l'arête testée est donnée lorsque l'on quitte la première face, l'algorithme concernant le déplacement depuis une arête n'effectue alors qu'une vérification du marqueur de la face vers laquelle la particule se dirige.

Nous utilisons le schéma 3.9 pour illustrer la décomposition d'un déplacement selon les cellules de l'environnement. Le cas de l'initialisation n'est pas considéré : on suppose connue la cellule contenant le point d'origine, à cette initialisation aucune direction de déplacement n'est donnée.

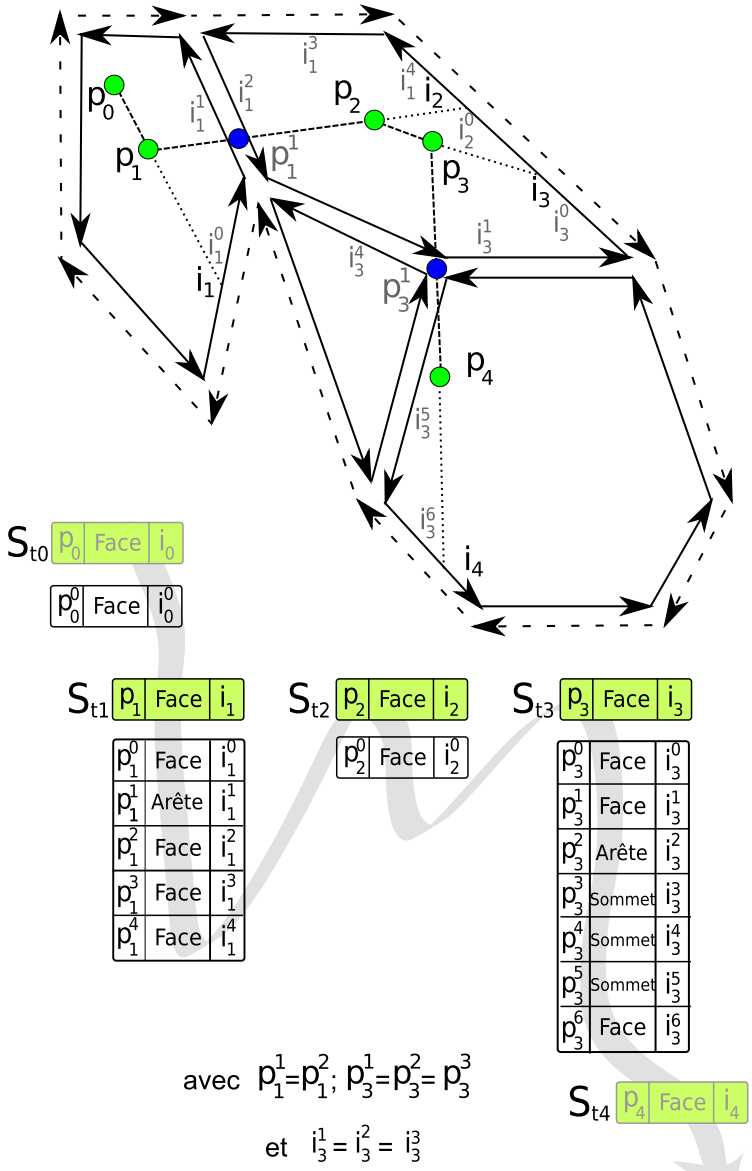

Figure 3.9 : Détails de la décomposition du mouvement selon les cellules traversées.

Étant donné un point au temps 1, il possède un état S_{t1} défini par sa position actuelle, la dimension de la cellule auquel il appartient : une face, et un brin de cette face i_1. Lorsqu'au temps 2, ce point est déplacé vers le point p_2, un test d'intersection face/déplacement est amorcé. La phase d'orientation identifie un changement de trajectoire et indique le brin i_1^1 comme brin de prédiction. La phase de déplacement indique qu'un changement de cellule a lieu et calcule le point d'intersection avec l'arête au point p_1^1. On effectue

alors un test d'intersection arête/déplacement en considérant un déplacement depuis p_1^1, vers le point p_2. Le test au niveau de l'arête indique que celle-ci est franchissable et amorce alors un test face/déplacement. La phase d'orientation de ce test indique le brin i_1^4, la phase de déplacement confirme qu'il n'y a plus de changement de cellule. Le nouvel état S_{t2} est donc trouvé.

Quand la trajectoire est régulière, peu de tests sont à effectuer. Ainsi, pour passer de l'état S_{t2} à S_{t3} seules une phase d'orientation sans changement et une phase de déplacement d'un test face/déplacement sont nécessaires. Quand aucun changement de cellule et d'orientation n'est nécessaire, 3 tests d'orientation point/droite sont effectués.

Récapitulatif

Nous avons présenté l'ensemble des algorithmes permettant de tester de manière optimum des déplacements dans les cellules élémentaires d'un environnement du plan. La composition de ces algorithmes élémentaires permet, lorsque la cellule d'origine du déplacement est connue, de ne tester que le sous-ensemble de cellules contenant un déplacement. Ainsi le maintien des informations concernant les états des particules au cours du temps permet de réduire le nombre de tests d'orientation et d'intersections nécessaires.

3.3 Déplacement de particules dans l'espace

Les déplacements en dimension 3 suivent le même principe que ceux en dimension 2. Le changement principal provient du nombre de cas possibles. L'intersection entre la droite Δ peut survenir avec quatre types de cellules : des volumes, des faces, des arêtes et des sommets. A chaque déplacement, on cherche toujours, pour une cellule donnée, le brin par lequel la droite Δ change potentiellement de cellule. En dimension 2, les intervalles de recouvrement étaient définis par des triangles. En dimension 3, ces intervalles sont remplacés par des tétraèdres.

De la même façon qu'en dimension 2, nous définissons un ensemble de prédicats (tableau 3.2), illustré par la figure 3.10, permettant de préciser des opérations géométriques. Ces opérations géométriques sont utilisées ultérieurement dans les algorithmes de recherche du nouvel état et nous permettent d'assurer la validité de l'enchaînement des états.

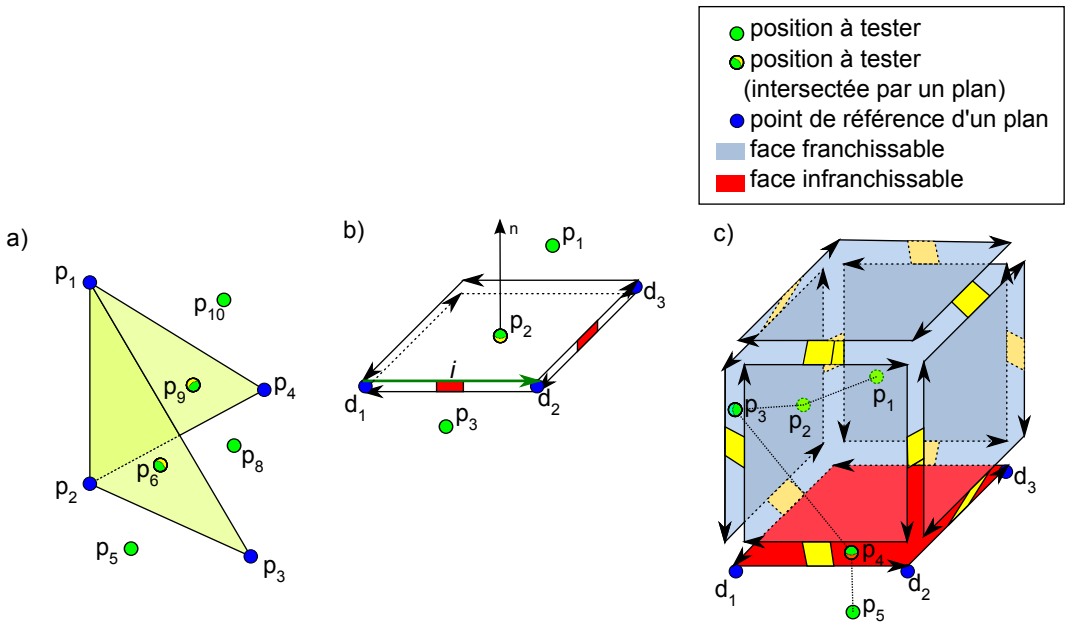

Figure 3.10 : Représentation graphique des prédicats 3D. a) prédicats à gauche, à droite est sur ; p_8, p_9 et p_{10} sont à gauche de (p_1, p_2, p_3) et p_{10} est à gauche de (p_1, p_2, p_4), p_5 est à droite de (p_1, p_2, p_3) et p_5, p_6 et p_8 sont à droite de (p_1, p_2, p_4), p_6 est sur (p_1, p_2, p_3) et p_9 est sur (p_1, p_2, p_4). b) prédicats dessus, sur et dessous le plan de i (i est plongé par 4 sommets, d_1, d_2 et d_3 suffisent pour définir le plan) ; p_1 est au dessus de i, p_2 est sur i, p_3 est en dessous de i. c) prédicats d'appartenances et de collisions ; p_1 et $p_2 \in volume(i)$, p_3 et $p_4 \in bord(volume(i))$ et appartiennent aux faces du volume ; les déplacements définis par $[p_1, p_2)$ et $[p_2, p_3)$ sont "libre", le déplacement défini par $[p_3, p_4)$ est en "contact", le déplacement défini par $[p_4, p_5)$ est en "collision".

Notations :

p_1 : position de départ d'un déplacement
p_2 : position à atteindre
$p, p_i, \ldots$: points dans l'espace
O : origine
(p_i, p_j, p_k) : trois points formant un plan
. : produit scalaire i : brin de la carte
$plgt(i, 0)$: plongement de sommet du brin i
$obstacle(i)$: booléen indiquant si un brin i est franchissable
"libre","contact","collision" : état d'un déplacement
$< k > (i)$: orbite d'une k-cellule

Prédicats :

Prédicats mathématiques :
$distanceSign(p, (p_i, p_j, p_k)) := p.normale(p_i, p_j, p_k) - distance(O, (p_i, p_j, p_k))$

Prédicats d'orientation sur des coordonnées :
p *à gauche de* $(p_i, p_j, p_k) := distanceSign(p, (p_i, p_j, p_k)) > 0$
p *à droite de* $(p_i, p_j, p_k) := distanceSign(p, (p_i, p_j, p_k)) < 0$
p *sur* $(p_i, p_j, p_k) := distanceSign(p, (p_i, p_j, p_k)) = 0$

Prédicats sur des brins :
p *au dessus de* $i := p$ *à droite de* $(plgt(i, 0), plgt(\phi_1(i), 0), plgt(\phi_1(\phi_1(i)), 0)$
p *en dessous de* $i := p$ *à gauche de* $(plgt(i, 0), plgt(\phi_1(i), 0), plgt(\phi_1(\phi_1(i)), 0))$
p *sur* $i := p$ *sur* $(plgt(i, 0), plgt(\phi_1(i), 0), plgt(\phi_1(\phi_1(i)), 0)$

Prédicats d'appartenance :
$p \in volume(i) := \forall d \in < \phi_1, \phi_2 > (i), p$ *au dessus de* i
$p \in face(i) := p$ *sur* $i \wedge \neg p$ *sous* $< \phi_1, \phi_2 > (i)$ß
$p \in arete(i) := p \in [plgt(i, 0), plgt(\phi_2(i), 0)]$
$p \in sommet(i) := p = plgt(i, 0)$
$p \in bord(volume(i)) := \neg(p \in volume(i)) \wedge \forall d \in < \phi_1, \phi_2 > (i), (p$ *au dessus de* $i \vee p$ *sur* $i)$
$p \in bord(face(i)) := p \in face \wedge \exists d \in < \phi_1 > (i), p \in arete(d)$

Prédicats d'états :
$collision(i) :=$ p_2 *en dessous de* $i \wedge obstacle(i)$
$contact(i) :=$ p_2 *sur* $i \wedge (obstacle(i) \vee obstacle(\phi_2(i)))$
$libre(i) := \forall d \in < k > (i), \neg(collision(d)) \wedge \neg(contact(d))$
$libre/collision(i) :=$ *si* $collision(i)$ *alors* "collision" *sinon* "libre"
$libre/contact(i) :=$ *si* $contact(i)$ *alors* "contact" *sinon* "libre"

Tableau 3.2 : Ensemble des prédicats utilisés en dimension 3

3.3.1 Tests d'intersections : cellules/déplacement

Dans l'espace, les différentes cellules composant la partition de l'environnement sont les volumes, les faces, les arêtes et les sommets. Comme précédemment, chacun des algorithmes démarre avec un triplet (p_1,k,i) et une destination p_2 et renvoie un quadruplet (p'_1,k',i',e).

Déplacement depuis un volume

Lorsque p_1 se trouve dans un volume, le brin de prédiction à trouver définit un tétraèdre. Ce tétraèdre est formé par le point p_1, le brin courant i et un point sur la face que nous appelons s_p (s_p se trouve sur la face que nous sommes en train de tester ; figure 3.12).

Le volume auquel appartient p_1 étant convexe, il n'existe qu'une et unique intersection potentielle entre le volume et le segment $[p_1,p_2]$. S'il n'existe pas d'intersection, le déplacement est inclus dans le volume, p'_1 est égal à p_2. S'il existe une intersection, le point p'_1 correspond au point d'intersection et i' correspond à la cellule où il se situe. Le problème consiste ici à identifier si le bord du volume est franchi ou atteint lors du déplacement de la particule.

La phase d'orientation consiste à tourner le tétraèdre basé sur le brin de prédiction autour de p_1 afin d'identifier quelle face du volume la demi-droite Δ intersecte.

Dans un premier temps, nous tournons sur la face de i afin de trouver le dièdre formé par les plans (p_1,s_p,d_2) et (p_1,s_p,d_1) contenant p_2. Tant que Δ n'est pas contenu dans cet angle, nous tournons sur la face avec les liens ϕ_1 ou ϕ_1^{-1}, comme effectué précédemment sur les faces pour les déplacements dans le plan. Le point s_p correspond en quelque sorte à l'équivalent du point p_1 autour duquel la recherche d'angle était effectuée pour les déplacements depuis une face dans le plan.

En dimension 3, le point p_2 doit également se trouver en dessous du plan (p_1,d_1,d_2) afin de localiser la face coupée par la demi-droite Δ. Sans ce test, rien n'assure qu'une intersection n'a pas lieu avec une des autres faces comprises dans l'intervalle diédral trouvé. Si ce n'est pas le cas, i est remplacé par $\phi_2(i)$, c'est-à-dire, la face du volume courant adjacente à celle testée actuellement selon l'arête i, avant de continuer la phase d'orientation. Lors de ce changement de face, un nouveau s_p est calculé afin de respecter la contrainte d'inclusion à la face qui a été prédéfinie pour le cas 2D.

De la même façon que pour le cas des faces dans le plan, nous définissons le sens de recherche de la phase d'orientation selon le résultat obtenu par le premier test d'orientation.

La phase de déplacement teste si le segment $[p_1,p_2]$ coupe la face se trouvant à la base du tétraèdre. En cas d'intersection, le déplacement passe soit par la face, l'arête ou le sommet : le point p'_1 est placé au point d'intersection et l'état de k' est modifié en conséquence. Sinon le déplacement s'effectue uniquement dans le volume : p'_1 est égal à p_2.

L'algorithme 4 décrit l'ensemble de ces étapes. La figure 3.11 schématise l'organisation des appels de fonction de ce dernier.

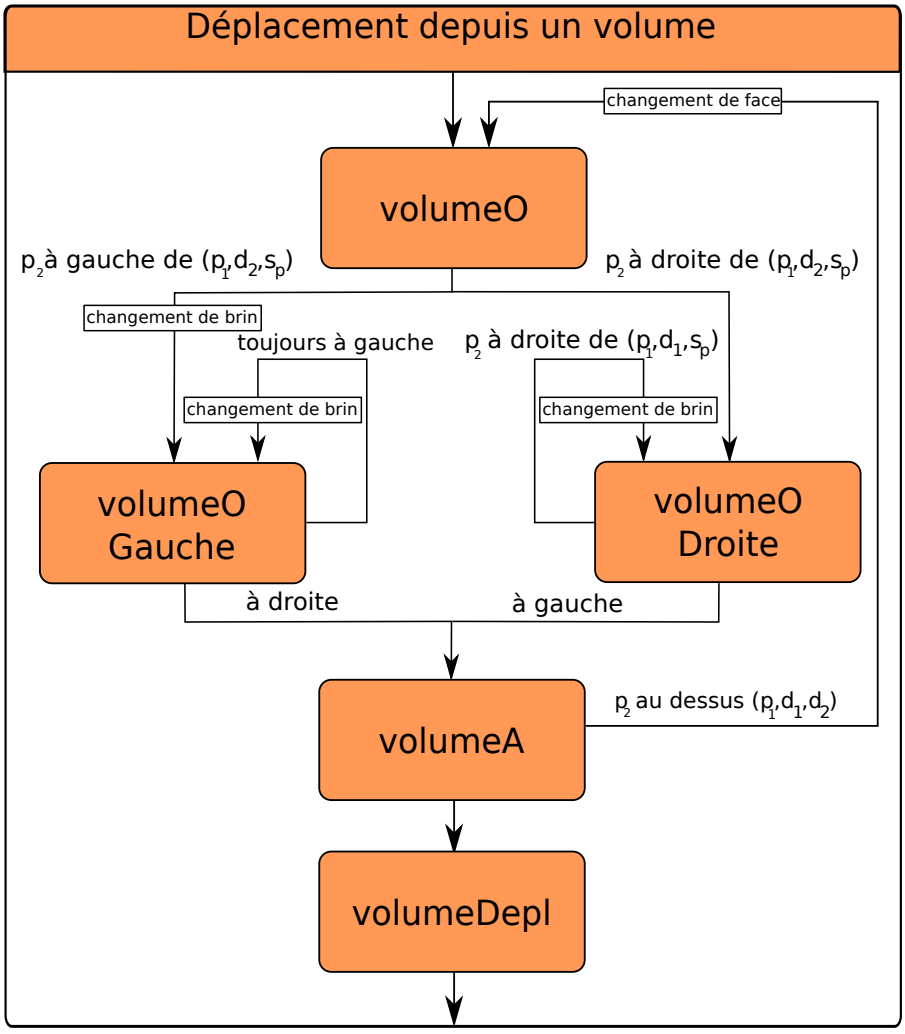

Figure 3.11 : Schéma représentant l'enchainement d'appels de fonctions de l'algorithme 4.

Déplacement depuis un volume

Entrée :
$(p_1, i, \text{volume}, p_2)$

Sortie :
(p'_1, i', k', e)

Préconditions :
$p_1 \neq p_2 \wedge$
$p_1 \in volume(i) \vee$
$(d \in < \phi_2 \circ \phi_1 > (i), \exists p \in face(d) \wedge i \notin < \phi_1 > (d) \wedge p_2 \ avant \ d) \vee$
$(d \in < \phi_2 \circ \phi_1 > (i), \exists p \in arete(d) \wedge i \notin < \phi_1 > (< \phi_2 > (d))$
$\wedge p_2 \ avant \ < \phi_2 > (d)) \vee$
$(d \in < \phi_2 \circ \phi_1 > (i), \exists p \in sommet(d) \wedge i \notin < \phi_1 > (< \alpha_1 > (d))$
$\wedge p_2 \ avant \ < \alpha_1 > (d))$

Postconditions :
p_2 *à gauche de* $(p_1, s_p, d_1) \wedge p_2$ *à droite de* $(p_1, s_p, d_2) \wedge$
$\neg(p_2$ *au dessus de* $(p_1, d_1, d_2) \wedge \overrightarrow{p_1 p_2}$ *non colinaire a* $\overrightarrow{p_1 s_p}$
et l'une des possibilités suivantes :

- "libre" $\wedge$ p_2 *au dessus de* $i' \wedge p'_1 = p_2 \wedge k' = volume$
- p_2 *en dessous ou sur* $i' \wedge p'_1 = intersection(\Delta, i') \wedge$
  $\neg(p_2 \ sur \ (p_1, d_1, d_2)) \wedge k' = face$
- p_2 *en dessous ou sur* $i' \wedge p'_1 = intersection(\Delta, i') \wedge p_2 \ sur \ (p_1, d_1, d_2) \wedge$
  $\neg(p_2 \ sur \ (p_1, s_p, d_1)) \wedge k' = arete$
- p_2 *en dessous ou sur* $i' \wedge p'_1 = intersection(\Delta, i') \wedge p_2 \ sur \ (p_1, d_1, d_2) \wedge$
  $p_2 \ sur \ (p_1, s_p, d_1) \wedge k' = sommet$

Les préconditions stipulant que p_1 n'est pas situé sur une des faces testées sont nécessaires afin d'éviter de tester des tétraèdres dégénérés pour lesquels les tests d'orientation ne donnent pas d'informations valides. Dans le cas où p_1, p_2 et s_p sont alignés, la recherche du bon intervalle sur une face n'est pas concluant, il est alors nécessaire de déplacer le point s_p afin de régler ce problème. Ce déplacement peut être effectué aléatoirement sur la face.

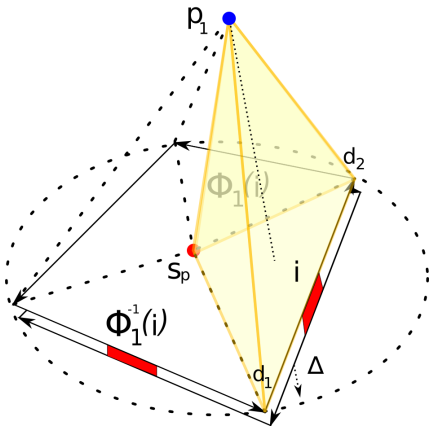

Figure 3.12 : Phase d'orientation de l'état volume ; seule une face du volume est montrée pour une question de lisibilité.

Algorithme 4 Déplacement depuis un volume

– *volumeO* $(p_1,i,s_p,p_2) =$
si p_2 à gauche ou sur (p_1,s_p,d_2) **alors**
volumeOGauche $(p_1,\phi_1(i),s_p,p_2)$
sinon
volumeODroite (p_1,i,s_p,p_2)
fin si

–*volumeOGauche* $(p_1,i,s_p,p_2) =$
si p_2 à gauche ou sur (p_1,s_p,d_2) **alors**
volumeOGauche $(p_1,\phi_1(i),s_p,p_2)$
sinon
volumeA (p_1,i,s_p,p_2)
fin si

–*volumeDepl* $(p_1,i,s_p,p_2) =$
si p_2 au dessus de i **alors**
$(p_2, volume, i,$"libre")
sinon si p_2 est sur (p_1,d_1,d_2) **alors**
si p_{t+1} est sur (p_1,s_p,d_1) **alors**
$(d_1, sommet, i,$
$libre/contact(i))$
sinon
$(intersection(\Delta,i), arete, i,$
$libre/contact(i))$
fin si
sinon si p_2 au dessous de i **alors**
$(intersection(\Delta,i), face, i,$
$libre/collision(i))$
sinon
$(intersection(\Delta,i), face, i,$
$libre/contact(i))$
fin si

–*volumeODroite* $(p_1,i,s_p,p_2) =$
si p_2 à droite de (p_1,s_p,d_1) **alors**
volumeODroite $(p_1,\phi_1^{-1}(i),s_p,p_2)$
sinon
volumeA (p_1,i,s_p,p_2)
fin si

– *volumeA* $(p_1,i,s_p,p_2) =$
si p_2 au dessus de (p_1,d_1,d_2) **alors**
soit $i' = \phi_1(\phi_2(i))$ dans
volumeO $(p_1,i', pointDe(i'),p_2)$
sinon
volumeDepl (p_1,i,s_p,p_2)
fin si

Déplacement depuis un sommet

La prédiction de l'état sommet est la version duale de celle du volume. Les faces adjacentes à un sommet appartenant à un même volume forment une ombrelle ou un cône comme illustré sur le schéma 3.13. Comme chaque volume est convexe, les ombrelles au niveau d'un sommet sont convexes également. Le point s_p est choisi sur la médiatrice de ce cône.

Nous cherchons le tétraèdre défini par le point p_1, l'arête i (incidente au sommet) et le point s_p, qui contient Δ. L'algorithme consiste, dans un premier temps, à faire tourner i autour de l'axe défini par p_1 et s_p suivant la relation $\phi_1 \circ \phi_2$ (ou $\phi_2 \circ \phi_1^{-1}$ son inverse). Une fois ce i trouvé, si p_2 se trouve en dehors du tétraèdre selon le plan défini par la face de i, i est alors associé au volume adjacent : $\phi_3(i)$. A la fin de ce test d'orientation, le bon tétraèdre est trouvé. Il ne reste plus, comme dans le cas du volume, qu'à identifier si le déplacement intersecte une arête, une face ou un volume afin de quitter le sommet.

L'algorithme 5 récapitule ces étapes. Ponctuellement nous utilisons la notation d_1 et d_2 pour noter les plongements de sommet des brins $\phi_2(i)$ et $\phi_1^{-1}(i)$.

Déplacement depuis un sommet
Entrée : (p_1,i,sommet,p_2) Sortie : (p'_1,i',k',e) Préconditions : $p_1 \neq p_2 \wedge\ p_1 \in sommet(i) \wedge\ s_p = plgt(i,0) + \sum_{d \in <\alpha_1>(i)} normal(d)$ Postconditions : $p'_1 = plgt(i',0) \wedge$ *p_2 au dessus ou sur $i' \wedge p_2$ à gauche ou sur* $(plgt(i',0), plgt(\phi_1(i),0), s_p)$ $\wedge p_2$ *à droite de* $(plgt(\alpha_1(i'),0), plgt(\phi_1(\alpha_1(i)),0), s_p)$ *et l'une des possibilités suivantes :* – "libre" $\wedge$ p_2 *au dessus de* $i' \wedge k' = volume$ – ("libre" $\vee$ "contact") $\wedge p_2$ *sur* $i' \wedge p_2 \notin arete(i') \wedge k' = face$ – ("libre" $\vee$ "contact") $\wedge p_2$ *sur* $i' \wedge p_2 \in]d_1, d_2) \wedge k' = arete$ – "collision" $\wedge k' = sommet$

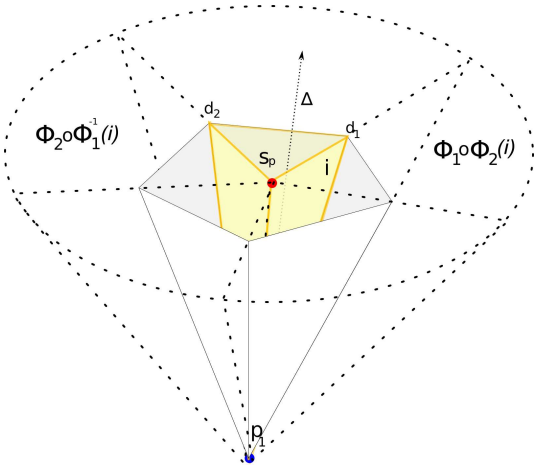

Figure 3.13 : Phase d'orientation de l'état sommet. L'ombrelle est décomposée en secteurs angulaires. Le secteur angulaire comprenant Δ est cherché avant de vérifier l'appartenance de Δ au volume courant. Cette vérification peut entraîner un changement d'ombrelle et la recherche d'un nouveau secteur angulaire. Un seul volume du sommet est représenté pour une question de visibilité.

Algorithme 5 Déplacement depuis un sommet

– *sommetO* (p_1,i,s_p,p_2) =
si p_2 à gauche de (p_1,d_2,s_p) **alors**
sommetOG $(p_1,\phi_2(\phi_1^{-1}(i)),s_p,p_2)$
sinon
sommetOD (p_1,i,s_p,p_2)
fin si

–*sommetOG* (p_1,i,s_p,p_2) =
si p_2 à gauche de (p_1,d_2,s_p) **alors**
sommetOG $(p_1,\phi_2(\phi_1^{-1}(i)),s_p,p_2)$
sinon
sommetA (p_1,i,s_p,p_2)
fin si

–*sommetD* (p_1,i,s_p,p_2) =
si i est marqué **alors**
$(p_1,sommet,i,$"collision")
sinon si p_2 est sur (p_1,d_1,d_2) **alors**
si p_2 est sur (p_1,s_p,d_1) **alors**
$(p_1,arete,i,libre/contact(i))$
sinon
$(p_1,face,i,libre/contact(i))$
fin si
sinon
$(p_1,volume,i,$"libre")
fin si

–*sommetOD* (p_1,i,s_p,p_2) =
si p_2 à droite de (p_1,d_1,s_p) **alors**
sommetOD $(p_1,\phi_1(\phi_2(i)),s_p,p_2)$
sinon
sommetA (p_1,i,s_p,p_2)
fin si

– *sommetA* (p_1,i,s_p,p_2) =
si p_2 au dessus de (p_1,d_1,d_2) **alors**
soit $i' = \phi_3(\phi_2(i))$ dans
sommetO $(p_1,i',pointDans(i'),p_2)$
sinon
sommetD (p_1,i,s_p,p_2)
fin si

Déplacement depuis une face

Lorsque p_1 est dans une face, nous réutilisons le principe présenté dans l'algorithme en dimension 2. Une contrainte apportée par le passage à la dimension 3 est la nécessité d'utiliser des tests d'orientation point/plan, afin d'éviter de calculer des projections supplémentaires. Les tests d'orientation sont basés sur les plans formés par : le point p_1, les sommets d_1 et d_2 et le point p_1 translaté le long de la normale à la face pour trouver le bon triangle.

La phase d'orientation dans l'état face est similaire à celle du cas 2D excepté pour une particularité. Le point p_2 peut quitter la face en passant au dessus ou en dessous du plan formé par celle-ci (figure 3.14). Ceci est vérifié par un premier test d'orientation. Si le point quitte la face nous vérifions alors que le volume dans lequel p_2 se dirige est libre ou non. Lorsque le point p_2 se situe dans le plan de la face et que l'un de ces deux volumes se trouve être marqué comme un obstacle, l'état de la particule est un contact.

Déplacement depuis une face

Entrée :
(p_1,i,face,p_2)

Sortie :
(p'_1,i',k',e)

Préconditions :
$p_1 \neq p_2 \wedge$
$p_1 \in face(i) \vee (p_1 \in arete(d), d \in < \phi_1(i) > \wedge i \neq d)$

Postconditions :
p_2 *à gauche de* $(p_1, d_1, (n(i)+p_1)) \wedge p_2$ *à droite de* $(p_1, d_2, (n(i)+p_1))$
et l'une des possibilités suivantes :

- "libre" $\wedge$ p_2 *au dessus de* $i' \wedge p'_1 = p_1 \wedge k' =$ *volume*
- ("libre" $\vee$ "contact") $\wedge$ p_2 *sur* $i' \wedge p'_1 = p_2 \wedge k' = face$
- "collision" $\wedge$ p_2 *en dessous de* $i' \wedge p'_1 = p_1 \wedge k' = face$
- p_2 *sur* $i' \wedge p'_1 = intersection(\Delta, i') \wedge \neg(p_2$ *sur* $(p_1, d_1, (n(i)+p_1))) \wedge k' =$ *arete*
- p_2 *sur* $i' \wedge p'_1 = d_1 \wedge p_2$ *sur* $(p_1, d_1, (n(i)+p_1)) \wedge k' =$ *sommet*

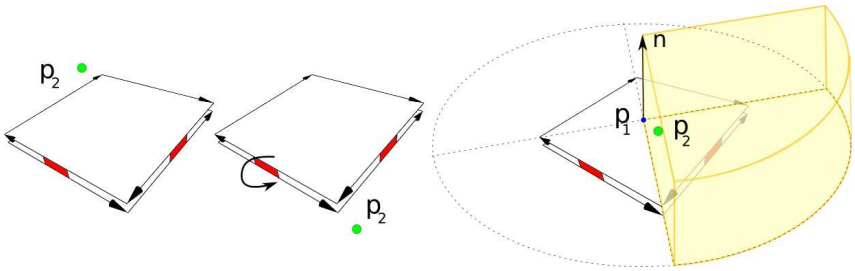

Figure 3.14 : Phase d'orientation de l'état face. On teste d'abord si p_2 est au-dessus ou sous la face. Si ce n'est pas le cas, on tourne autour de la face pour trouver le bon secteur angulaire.

Algorithme 6 Déplacement depuis une face

– *faceOV* $(p_1,i,s_p,p_2) =$
si p_2 en dessous de i **alors**

si $\phi_3(i)$ franchissable **alors**
soit
$i' = faceNonCoplanaire(\phi_3(i))$
dans
$(p_1, volume, i'$,"libre")
sinon
$(p_1, face, i$,"collision")
fin si
sinon si p_2 au dessus de i **alors**
soit $i' = faceNonCoplanaire(i)$
dans
$(p_1, volume, i'$,"libre")
sinon
faceO (p_1,i,s_p,p_2)
fin si

– *faceO* $(p_1, i,s_p,p_2) =$
si p_2 à gauche de $(p_1, d_2,p_1+n(i))$
alors
faceOG $(p_1, \phi_1(i),s_p,p_2)$
sinon
faceOD $(p_1, \phi_1^{-1}(i),s_p,p_2)$
fin si

–*faceD* $(p_1, i,s_p,p_2) =$
si p_2 avant $(d_1, d_2, d_1 + n(i))$ **alors**
$(p_2, face, i$,*libre/contact*$(i))$
sinon
si p_2 sur $(p_1, d_1, p_1 + n(i))$ **alors**
$((intersection(\Delta, i), sommet, i,$
libre/contact$(i))$
sinon
$((intersection(\Delta, i), arete, i,$
libre/contact$(i))$
fin si
fin si

–*faceOG*$(p_1, i,s_p,p_2) =$
si p_2 à gauche de $(p_1, d_2,p_1+n(i))$
alors
faceOG $(p_1, \phi_1(i),s_p,p_2)$
sinon
faceD (p_1, i,s_p,p_2)
fin si

–*faceOD* $(p_1, i,s_p,p_2) =$
si p_2 à droite de $(p_1, d_1,p_1+n(i))$ **alors**

faceOD $(p_1, \phi_1^{-1}(i),s_p,p_2)$
sinon
faceD (p_1, i,s_p,p_2)
fin si

Déplacement depuis une arête

Lorsque le mobile est dans l'état arête, nous utilisons le même principe que celui de l'état sommet en dimension 2. L'intervalle contenant Δ est défini ici par deux faces formant l'arête au niveau d'un volume (figure 3.15). Une fois cet intervalle trouvé, nous vérifions si p_2 se trouve sur la droite définie par l'arête, passe sur une face ou va dans le volume adjacent. Si le point reste sur la droite, nous vérifions alors si p_2 reste dans l'intervalle $]d_1, d_2[$ ou s'il atteint un sommet. Après ces tests, il est alors possible de définir l'état final (en testant le marqueur d'atteignabilité).

Déplacement depuis une arête
Entrée : (p_1,i,arête,p_2) Sortie : (p'_1,i',k',e) Préconditions : $p_1 \in arete(i)$ Postconditions : p_2 *au dessus ou sur* $i' \wedge p_2$ *à droite de* $\phi_3(\phi_2(i')) \vee$ $p_2 \in arete(i')$ *et l'une des possibilités suivantes :* – "libre" $\wedge$ p_2 *au dessus de* $i' \wedge p'_1 = p_1 \wedge k' = volume$ – "collision" $\wedge$ p_2 *au dessus de* $i' \wedge p'_1 = p_1 \wedge k' = arete$ – "collision" $\wedge$ p_2 *sur* $i' \wedge p_2 \notin arete(i') \wedge p'_1 = p_1 \wedge k' = arete$ – ("libre" $\vee$ "contact") $\wedge$ p_2 *sur* $i' \wedge p_2 \notin arete(i') \wedge p'_1 = p_1 \wedge k' = face$ – ("libre" $\vee$ "contact") $\wedge$ p_2 *sur* $i' \wedge p_2 \in]d_1, d_2[\wedge p'_1 = p_2 \wedge k' = arete$ – ("libre" $\vee$ "contact") $\wedge$ p_2 *sur* $i' \wedge p_2 \notin]d_1, d_2[\wedge p_2 \in]d_2, d_1) \wedge p'_1 = d_1 \wedge k' =$ *sommet*

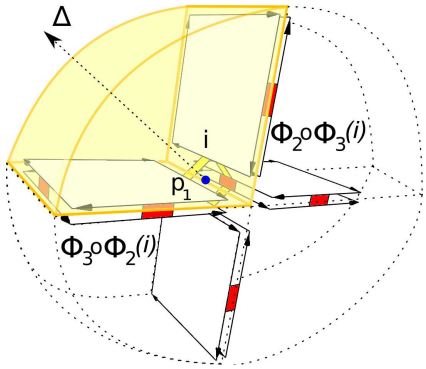

Figure 3.15 : Phase d'orientation de l'état arête. On tourne autour de l'arête en cherchant le secteur angulaire, défini par deux plans et contenant Δ.

Algorithme 7 Déplacement depuis une arête

– *areteO* $(p_1,i,s_p,p_2) =$
si p_2 a gauche de $\phi_3(\phi_2(i))$ **alors**
areteOG $(p_1,\phi_3(\phi_2(i)),s_p,p_2)$
sinon
areteOD (p_1,i,s_p,p_2)
fin si

– *areteOG* $(p_1,i,s_p,p_2) =$
si p_2 a gauche de $\phi_3(\phi_2(i))$ **alors**
areteOG $(p_1,\phi_3(\phi_2(i)),s_p,p_2)$
sinon
areteD (p_1,i,s_p,p_2)
fin si

–*areteOD* $(p_1,i,s_p,p_2) =$
si p_2 à droite de i **alors**
areteOD $(p_1,\phi_2(\phi_3(i)),s_p,p_2)$
sinon
areteD (p_1,i,s_p,p_2)
fin si

–*areteD* $(p_1,i,s_p,p_2) =$
si p_2 sur (d_1,d_2) **alors**
si p_2 est avant d_1 **alors**
si p_2 est avant d_2 **alors**
$(p_2,arete,i,libre/contact(i))$
sinon
soit $i' = \phi_2(i)$ dans
$(d_1,sommet,i',libre/contact(i'))$
fin si
sinon
$(d_1,sommet,i,libre/contact(i))$
fin si
sinon si p_2 au dessus de i **alors**
si i franchissable **alors**
soit
$i' = faceNonPlanaire(i,\phi_2(i))$
dans
$(p_1,volume,i'$,"libre")
sinon
$(p_1,arete,i$,"collision")
fin si
sinon
si i franchissable **alors**
$(p_1,face,i$,"libre")
sinon
$(p_1,arete,i$,"collision")
fin si
fin si

3.3.2 Déplacement complet

En dimension 3, l'enchaînement d'états se fait de manière similaire à la dimension 2 en intégrant les cellules de dimension 3 comme montré sur le schéma 3.16.

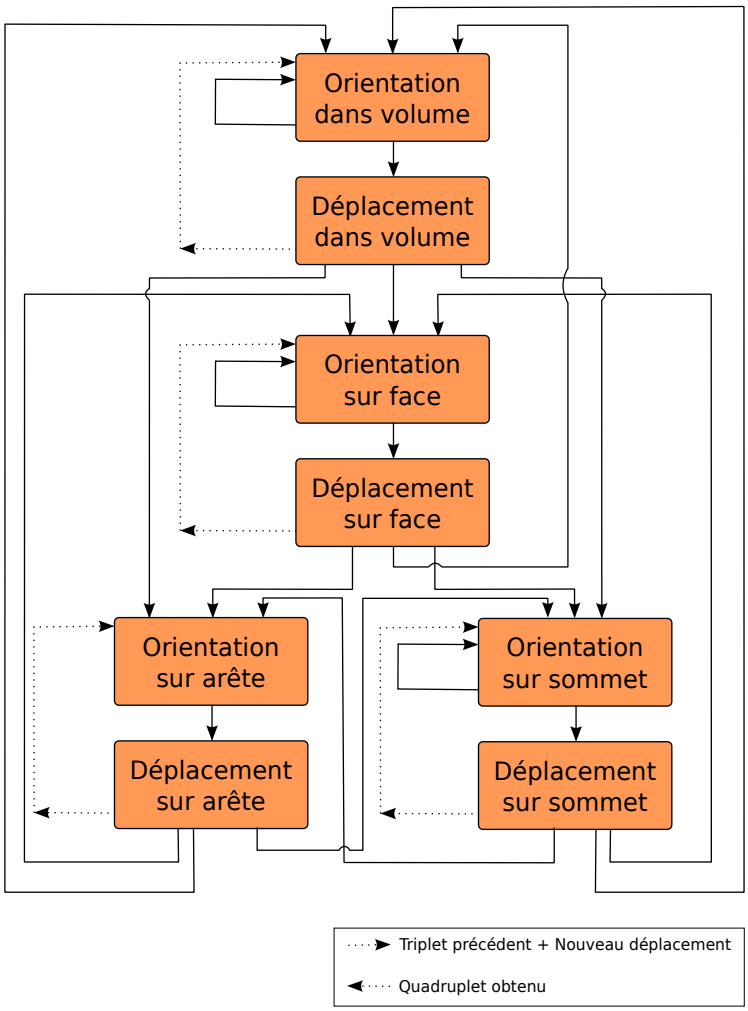

Figure 3.16 : Recherche du quadruplet correspondant au triplet de départ et à une position à atteindre par l'enchaînement des différents algorithmes.

En dehors de la modification des divers algorithmes pour correspondre aux déplacements dans l'espace, la seule modification au niveau de l'enchaînement des états correspond à l'inclusion de l'état volume.

3.4 Gestion des déformations de l'environnement

Les méthodes présentées dans les sections précédentes permettent de tester la collision de particules en déplacement dans un environnement statique. Les applications physiques ou médicales nécessitent de simuler des situations réelles de manière précise et requièrent donc la gestion de déformations. Ces déformations peuvent être de deux types.

Dans un premier temps, nous considérons l'ensemble des déformations topologiques. Les déformations topologiques correspondent à la modification de la subdivision en termes de cellules (*e.g.* suppression ou ajout d'une face, création d'un lien entre deux volumes, etc.). Ce type de déformations peut intervenir lors de simulations de sutures, de brûlures ou encore de fractures d'éléments de l'environnement.

Dans un deuxième temps, nous considérons l'ensemble des déformations géométriques. Les déformations géométriques, dans notre cadre, consistent en des déplacements de plongements associés à notre structure de représentation (*e.g.* déplacement d'un sommet d'un maillage). Ce type de déformations intervient lorsque les outils d'interactions, ou des forces extérieures, modifient l'environnement. Par exemple, si un poids est appliqué sur un organe, ce dernier se déforme.

Lorsque l'environnement subit des déformations, les méthodes précédentes doivent être adaptées. En effet, l'état réel et l'état maintenu S_t de la particule peuvent différer, ce qui peut invalider les préconditions nécessaires à une exécution robuste des algorithmes présentés (*e.g.* si après une déformation de l'environnement, une particule ayant pour état $(p_1, face, i)$ ne satisfait plus la précondition $p_1 \in face(i)$, alors l'algorithme peut engendrer un état incorrect).

Notre algorithme de prédiction, en plus d'être indépendant de la taille de la scène, se montre également plus efficace que les structures hiérarchiques lors de déformations. En effet, seules les particules se trouvant dans une zone déformée doivent être mises à jour. De plus lorsqu'une cellule partitionnant l'environnement est modifiée, seule cette dernière, et ses voisines proches affectées par cette déformation doivent être mises à jour : aucune mise à jour globale n'est à effectuer.

Dans cette partie, nous présentons comment adapter les algorithmes précédents afin de permettre la gestion des déformations de l'environnement. Nous commençons par présenter le traitement des déformations topologiques. Ensuite, nous traitons le cas des déformations géométriques ainsi que la question

des remaillages locaux nécessaires à la conservation de la propriété de convexité des cellules.

3.4.1 Déformations topologiques

Grâce à la structure topologique sous-jacente, les changements de topologie de la subdivision sont totalement supportés. Nous considérons trois types d'opérations topologiques : la couture, la simplification et le découpage de cellules. Des mises à jour des prédictions sont à prendre en compte uniquement pour les particules contenues dans les cellules subissant une modification.

La figure 3.17 illustre ces différentes opérations dans le plan, l'opération de couture consiste à rajouter un lien ϕ_2 avec une face, l'opération de simplification retire une arête entre deux faces cousues par un lien ϕ_2, l'opération de découpage crée deux faces à partir d'une seule suivant une droite de découpe.

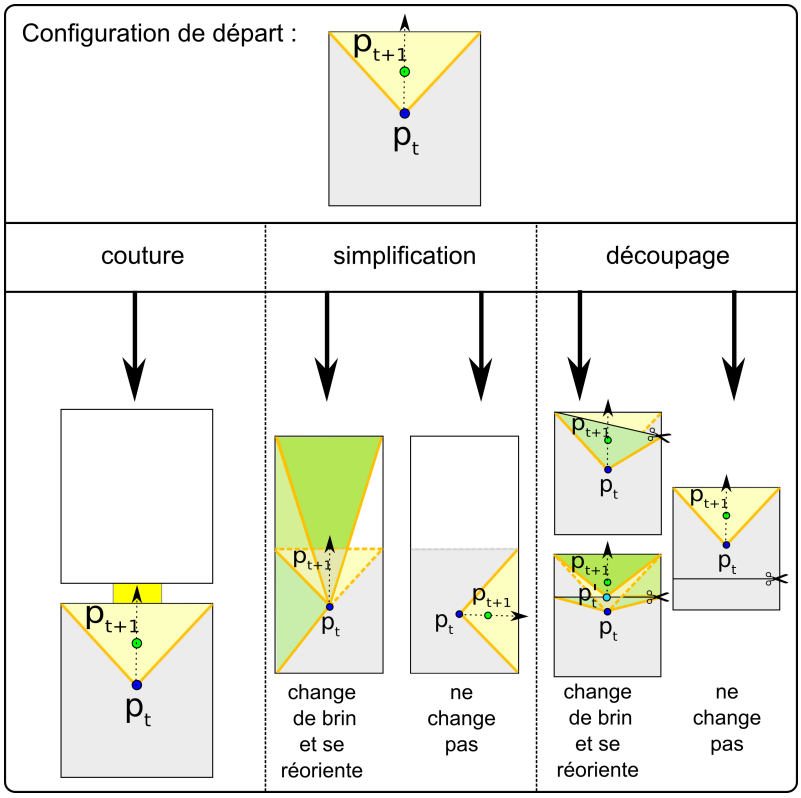

Figure 3.17 : Gestion des modifications topologiques.

Couture : lorsque la cellule contenant p_t est cousue avec une autre, la prédiction ne change pas. La cellule adjacente est automatiquement utilisée lors du prochain pas de l'algorithme.

Simplification : cette opération consiste à supprimer une cellule de dimension $(n-1)$ entre deux cellules de dimension n (*i.e.* supprimer une arête entre deux faces ou une face entre deux volumes). Lorsque p_t appartient à la cellule sur le point de subir une simplification, deux cas surviennent :

- les brins de la $(n-1)$-cellule sur le point d'être supprimés ne prennent part à aucune prédiction. Dans ce cas, aucune modification n'est nécessaire (*e.g.* p_t vise une arête qui n'est pas supprimée) ;
- une particule vise un des brins à supprimer. Dans ce cas, nous remplaçons ce brin par un brin adjacent de la cellule courante. Un nouveau pas d'orientation corrige la prédiction.

Découpage : cette opération consiste à ajouter une cellule de dimension $(n-1)$ dans une cellule de dimension n (*i.e.* ajouter une arête coupant une face ou une face coupant un volume). Ceci engendre trois cas :

- p_t et p_{t+1} sont de chaque coté de la cellule ajoutée, nous prenons alors un brin de la $(n-1)$-cellule qui vient d'être ajoutée, une phase d'orientation corrige alors automatiquement la prédiction.
- p_t et p_{t+1} sont du même coté de la cellule ajoutée et la prédiction est toujours correcte. Dans ce cas aucune mise à jour n'est requise.
- p_t et p_{t+1} sont du même coté de la cellule ajoutée et la prédiction est à l'opposée de celle-ci (i.e. appartient à la cellule adjacente), nous prenons un brin de la nouvelle $(n-1)$-cellule afin d'obtenir une configuration valide qui nous permet de nous réorienter.

En résumé, les modifications topologiques nécessitent dans certains cas le remplacement des brins de prédiction et un nouveau pas d'orientation. Les opérations topologiques présentées permettent la gestion des coupures et sutures. Les opérations topologiques non évoquées ici peuvent être traitées de manière similaire avec des mises à jour locales.

3.4.2 Déformations géometriques

Lorsqu'une particule se déplace dans un maillage déformable, nous considérons un pas de temps de l'animation comme composé de deux déplacements : le déplacement de particules de la position p_t à p_{t+1} dans l'environnement statique, puis, le mouvement du maillage en conservant les particules fixes. Une fois ce deuxième déplacement effectué il est possible qu'une correction soit à apporter sur la prédiction courante afin de la conserver valide.

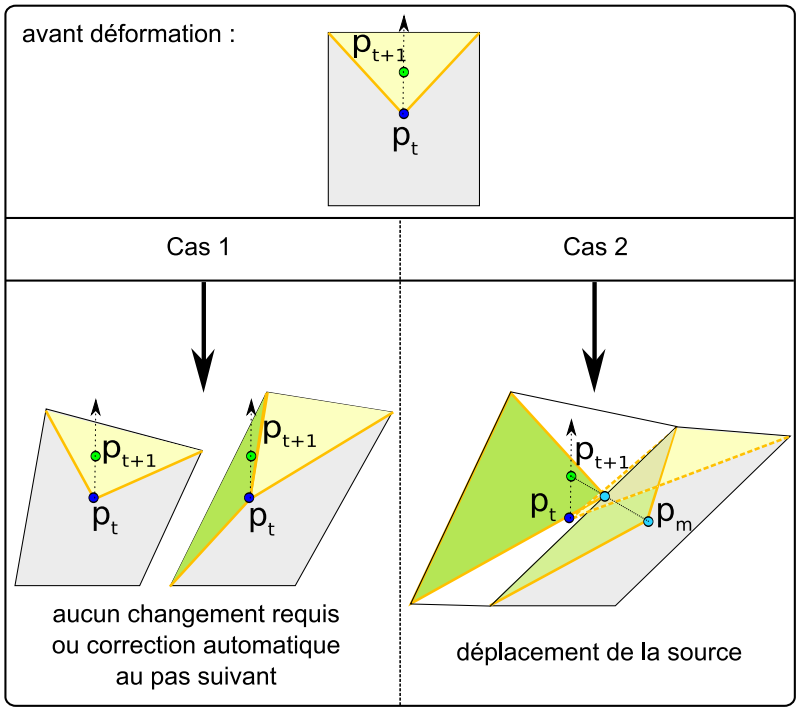

Figure 3.18 : Mise à jour locale pour gérer les déformations.

Deux cas, illustrés par la figure 3.18, se produisent :

- Cas 1 : p_t reste dans la cellule courante, la prédiction est toujours valide ou est corrigée à la prochaine prédiction.
- Cas 2 : p_t n'est plus dans la cellule précédemment visée. Ce cas survient lorsque le bord de la cellule se déplace au-dessus de p_t. Dans ce cas, afin de trouver la nouvelle cellule contenant p_t, nous simulons un déplacement depuis un point p_m se situant dans la cellule précédemment visée. Ce déplacement réinitialise la prédiction avec une configuration valide. Le point p_m peut être le barycentre de la cellule ou tout point contenu dans la cellule (*e.g.* le milieu d'une diagonale).

Dans le cadre applicatif, le premier cas nécessite de tester si p_t se trouve

toujours dans la bonne cellule. Si cette information est déductible à partir du déplacement de l'objet (c'est à dire lorsque la déformation est causée par l'outil d'interaction) ce cas peut être considéré. Lorsque la déformation n'a pas de lien direct avec l'outil, ce test nécessite n tests d'orientation (un pour chaque arête, ou face, de la cellule de la prédiction). Il est alors préférable d'appliquer directement le second cas. Une étude statistique ultérieure démontre qu'un ou deux tests d'orientation supplémentaires sont nécessaires en moyenne pour corriger l'état d'une particule.

3.4.3 Contrainte de convexité

Si une déformation entraîne la perte de convexité de certaines cellules, un remaillage de ces cellules ainsi que des cellules voisines doit être effectué. Les principes concernant les changements topologiques explicités précédemment sont alors utilisés pour assurer la validité de la prédiction après ces modifications.

Le remaillage consiste habituellement à fusionner des cellules non convexes avec les cellules voisines et potentiellement à décomposer la cellule générée.

Les auto-collisions de l'environnement sont détectées durant la vérification de la propriété de convexité. En effet, une auto-collision de l'environnement ne peut survenir que lorsque le déplacement d'un sommet aboutit en une cellule dégénérée (*i.e.* un polyèdre plat). Notre structure permet de localiser de manière efficace ces configurations.

Pour une cellule déformée, il est aisé d'identifier si elle est convexe ou concave. Un remaillage efficace et robuste n'est cependant pas trivial et est actuellement un problème ouvert dans notre système.

3.5 Conclusion

Nous avons présenté dans ce chapitre l'ensemble des algorithmes de déplacements de particules dans des environnements partitionnés. Le découpage de l'ensemble des algorithmes en tant que blocs atomiques dépendants de la dimension de la cellule traversée permet la définition précise des conditions d'entrée et de sortie de chaque fonction. Chacun des blocs est défini afin d'optimiser le nombre de tests d'orientation et d'intersections nécessaires pour suivre le déplacement d'une particule dans un ensemble de cellules. Le fait de mé-

moriser à chaque pas de temps l'état d'une particule permet alors le suivi et le maintien d'informations concernant la localisation et les propriétés de contact des déplacements de celle-ci.

La gestion de la déformation des environnements, que cela soit au niveau topologique ou au niveau géométrique, consiste alors à s'assurer du maintien de la validité des états calculés. La propriété de localité des états que nous avons définis, réduit le coût nécessaire à ce maintien.

Chapitre 4

Déplacement d'arêtes

Sommaire

Nous commençons par décrire ce qui motive la recherche des collisions entre les arêtes d'un objet mobile et son environnement. Nous présentons ensuite deux algorithmes répondant à ce problème.

4.1 Introduction

L'utilisation de particules est une approche simple permettant de gérer efficacement la détection de collision pour des objets de grandes tailles dans des simulations. Cependant, ce type de détection de collision est limité par le fait que des collisions sont manquées là où il n'y a pas de particules au niveau de la surface, particulièrement le long d'éléments tranchants, tel que montré sur la figure 4.1. Des méthodes proposent un échantillonnage dense et massif afin de recouvrir intégralement les objets avec des particules sphériques [BYM05]. Cette méthode limite les interpénétrations au niveau des arêtes, mais ne résout pas de manière satisfaisante le problème, car le nombre de particules nécessaires peut devenir extrêmement élevé.

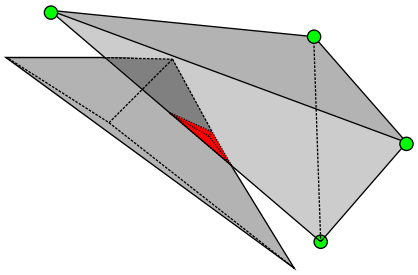

Figure 4.1 : Collision manquée le long d'une arête franche.

Nous présentons dans ce chapitre une extension de l'algorithme de déplacement des particules, permettant la détection de collision entre des arêtes de l'objet et l'environnement à l'intérieur duquel il navigue. De même que pour les particules, la méthode suppose que les cellules de l'environnement sont convexes.

Nous proposons deux méthodes. La première méthode consiste à effectuer des tests de collision lors du déplacement d'une arête de manière quasi-continue ; nous définissons cette notion dans la partie correspondante. Cette méthode, dérivée des algorithmes du chapitre précédent, est proposée et présentée dans le plan, puis dans l'espace. La deuxième méthode consiste à élaborer un algorithme de détection de collision d'arêtes de manière continue, afin de répondre à la même problématique de manière plus précise. Cette extension nécessite de définir un algorithme de recherche dans un graphe de cellules.

4.2 Déplacement d'arêtes de manière quasi-continue

L'extension que nous proposons consiste principalement à lancer des particules le long des arêtes en utilisant les algorithmes de déplacement présentés précédemment. Cette extension peut utiliser tout autre algorithme de détection de collision basé sur des particules. Les informations qui doivent être fournies par l'algorithme sont : la présence ou non de collision lors d'un déplacement et lorsqu'une collision survient le point d'impact de celle-ci, la normale à la surface de l'objet de ce point et la cellule sur laquelle elle a lieu. Nous commençons par expliquer la méthode dans le plan avant de présenter le cas général de l'espace.

4.2.1 Déplacement dans le plan

Nous présentons ici notre méthode de détection de collision limitée au plan. Nous introduisons le déplacement d'une arête, présentons les diverses réponses possibles pour la simulation, puis nous proposons certaines améliorations de cette méthode afin de limiter les calculs à effectuer.

Déplacement d'une arête

Considérons une arête entre deux particules p_t et q_t. Lorsque p_t est déplacée vers sa position suivante p_{t+1}, alors que q_t est immobile, l'arête balaye un triangle correspondant à un déplacement élémentaire. Une collision a lieu lors de ce déplacement si ce triangle, que nous appelons τ, rentre en intersection avec une cellule marquée comme un obstacle (un sommet ou une arête dans le plan). Comme l'environnement est modélisé par une partition continue, une intersection ne peut survenir que si l'une des trois arêtes du triangle τ intersecte un obstacle ou si un obstacle est complètement inclus dans τ.

Nous imposons par la suite qu'aucun obstacle de la partition ne soit plus petit que le plus grand déplacement triangulaire élémentaire pouvant être parcouru lors d'une simulation, c'est à dire qu'aucun obstacle ne peut être totalement inclus dans τ. Cette contrainte est liée à la surface couverte par un déplacement. Cette surface est proportionnelle à la longueur du déplacement (c'est-à-dire la distance de p_t à p_{t+1}) et à l'intervalle d'échantillonnage des particules sur le maillage (en d'autres termes à la taille de l'arête). Dans les simulations physiques classiques, cette contrainte est raisonnable et ne devrait pas limiter le cadre d'application de la méthode. Elle justifie l'appellation quasi-continue pour caractériser cette méthode de détection de collision.

Le déplacement complet d'une arête pour un pas de temps déplace alternativement les deux particules placées à ses extrémités. Le déplacement d'une particule de l'arête forme un triangle τ défini par les arêtes $[p_t, q_t]$, $[p_t, p_{t+1}]$ et $[p_{t+1}, q_t]$. Au départ, avant le déplacement, l'arête $[p_t, q_t]$ n'est pas en collision, sinon le contact aurait été géré précédemment. Ensuite, la particule p_t est déplacée vers la position p_{t+1} en utilisant l'algorithme de déplacement de particules, si ce déplacement est possible, il n'y a pas de collision le long de l'arête $[p_t, p_{t+1}]$. Enfin, la dernière arête $[p_{t+1}, q_t]$ du triangle τ est testée en lançant une particule temporaire de la position p_{t+1} à la position q_t, en utilisant un deuxième test de déplacement de particule. Dans un second temps, q_t est déplacé vers q_{t+1} de manière similaire et ce ainsi de suite comme indiqué sur la figure 4.2.

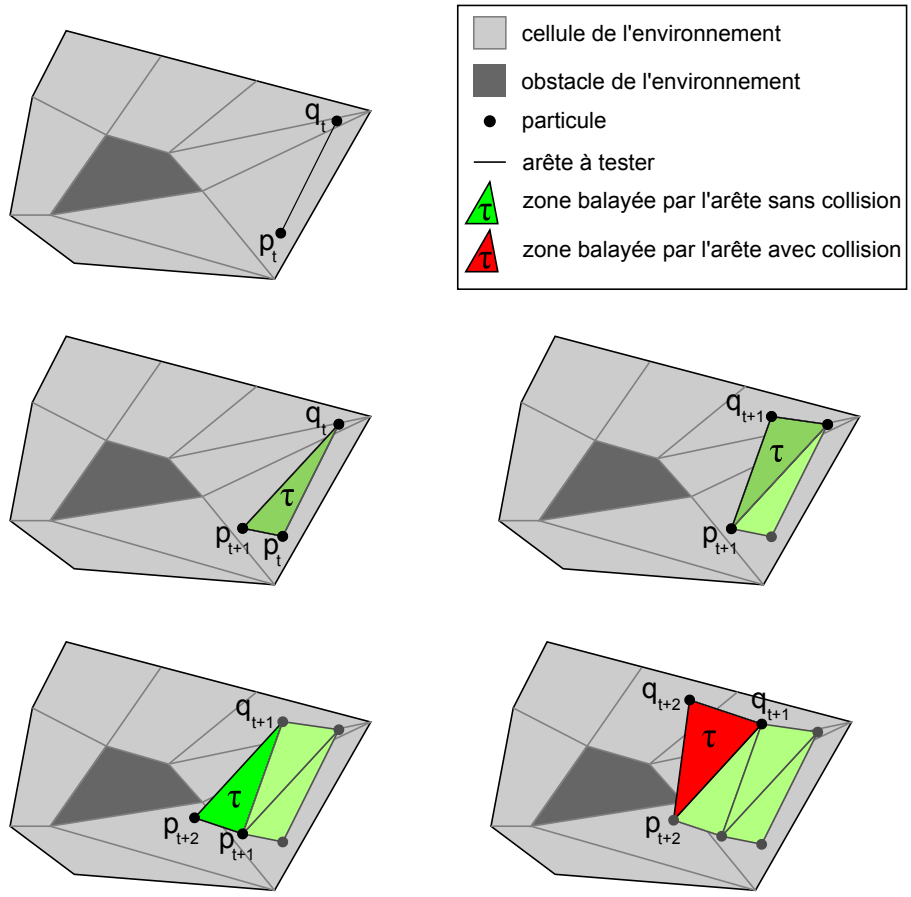

Figure 4.2 : Déplacements successifs des sommets d'une arête et zones élémentaires couvertes lors du déplacement utilisé pour la détection de collision.

L'algorithme ci-après résume la routine du déplacement d'une particule rattachée à i arêtes. Une collision au niveau d'une arête donne lieu à une réponse d'état *collisionArete* afin de permettre la différenciation avec la collision d'une particule.

Notations
p_1 : position de départ d'un déplacement
p_2 : position à atteindre
q_i : position d'une particule voisine par une arête
S_1 : état d'une particule précisant sa cellule d'appartenance et un brin visé
S_2 : état d'une particule précisant sa position atteinte, sa cellule d'appartenance, un brin visé et éventuellement des informations de collision
S^i_{arete} : état d'une particule précisant une collision d'arête et indiquant la localisation géométrique de l'impact, le type de cellule où elle a lieu et un brin de cette cellule

Algorithme 8 Déplacement d'une particule avec test d'arêtes

– *deplArete* (p_1,S_1,p_2) =
  S_2 = déplacement particule p_1 à l'état S_1 vers p_2
  pour tout q_i particules adjacentes **faire**
    création d'une particule temporaire p_{tmp} à l'état S_2
    S^i_{arete} = déplacement de la particule p_{tmp} à l'état S_2 vers q_i
    si S^i_{arete} est en collision **alors**
      renvoie S^i_{arete}
    fin si
  fin pour
  renvoie S_2

Lorsqu'une collision survient, des calculs supplémentaires sont nécessaires afin de signaler précisément les informations concernant celle-ci. Ces informations permettent alors au système physique de calculer la réponse correspondante. Nous décrivons deux méthodes pour la recherche exacte de la localisation de la première collision, puis nous proposons une réponse statique ou adaptative une fois que cette localisation est obtenue.

Recherche du premier point de collision

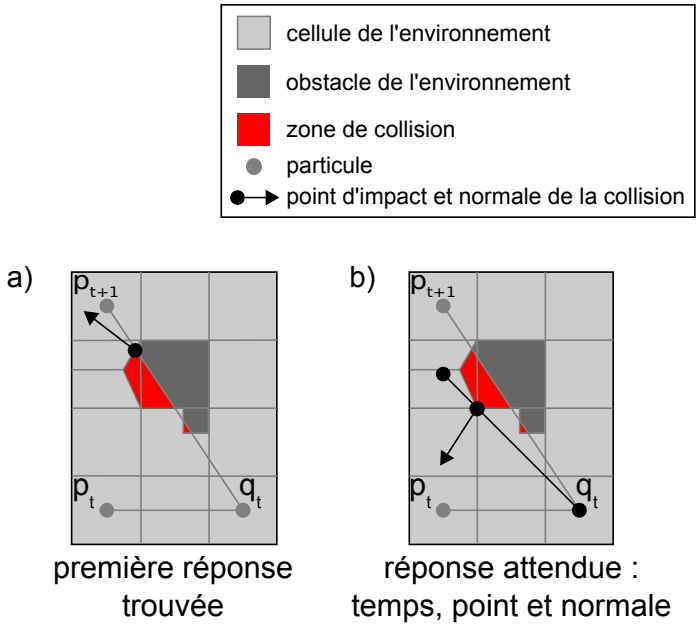

Figure 4.3 : a) Information inexacte concernant le point de collision, la normale et le temps auquel survient la collision en lançant une particule le long de l'arête. b) Information attendue.

Lorsqu'une collision est repérée au niveau d'une arête, l'algorithme de déplacement de particules renvoie la première cellule infranchissable. Cette information indique la présence d'un obstacle, mais ne fournit pas le premier point de contact, ni la configuration précise de celui-ci comme l'illustre la figure 4.3.a. Si des informations précises concernant le contact sont requises, deux méthodes peuvent être utilisées.

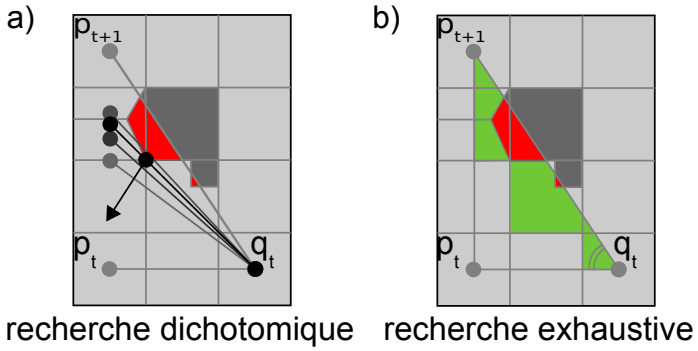

Figure 4.4 : Deux exemples de méthodes de recherche du point de collision, de la normale et du temps auquel survient la collision . a) Méthode dichotomique. b) Méthode exhaustive

La première consiste à utiliser une recherche dichotomique le long du segment $[p_t, p_{t+1}]$, jusqu'à ce que le segment le plus proche de la position souhaitée soit trouvé. Avec un nombre d'itérations borné, cette méthode est de coût faible. Le résultat est une approximation du contact réel, mais fournit une information assez précise pour être utilisé. La normale de la collision renvoyée est alors la normale au sommet du dernier brin intersecté, comme montré sur la figure 4.4.a.

Lorsqu'un calcul précis de la collision est nécessaire, une recherche exhaustive est plus appropriée. Elle consiste à chercher l'ensemble des cellules, marquées comme des obstacles, traversées lors du déplacement de la particule temporaire parcourant l'arête. Il faut ici utiliser une itération particulière de l'algorithme de déplacement de particules, qui poursuit l'ensemble des tests jusqu'à atteindre la position à atteindre tout en mémorisant l'ensemble des cellules traversées. Le sommet, où a lieu la collision, est alors le sommet d'une cellule obstacle rencontrée possédant le plus petit angle par rapport à la position d'origine de l'arête.

Cette recherche consiste à trouver l'ensemble des sommets se situant en dessous du segment $[p_{t+1}, q_t]$ et appartenant à une cellule obstacle franchie lors du déplacement ; en vert sur la figure 4.4.b. Une fois l'ensemble de ces sommets trouvés, une simple mesure d'angle permet d'identifier le premier

point de collision.

Réponse à une collision

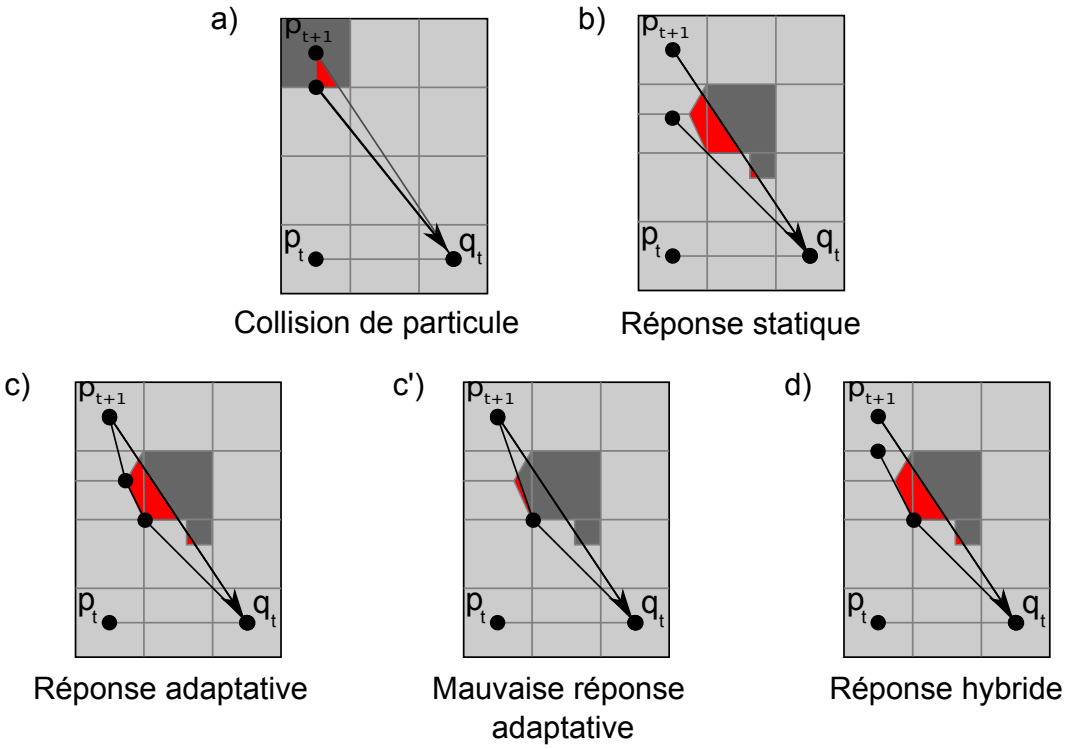

Figure 4.5 : Différents types de réponses possibles à des collisions.

Différentes réponses à des collisions sont envisageables. Le choix de la réponse appropriée dépend des propriétés physiques du modèle employé et de la capacité de ce dernier à supporter des maillages adaptatifs. Nous détaillons ici quelques réponses possibles illustrées par la figure 4.5. Il s'agit ici de réponses pour des modèles d'objets supportant des déformations.

Une collision peut intervenir au niveau d'une particule comme sur la figure 4.5.a. Dans ce cas, la particule est arrêtée au point de contact.

Lorsqu'une collision d'arête intervient nous proposons deux types de réponses : une réponse statique et une réponse adaptative. La réponse statique consiste à déplacer la particule jusqu'à la position correspondant au temps de la collision comme montré sur la figure 4.5.b.

La réponse adaptative consiste à insérer une ou des nouvelles particules au(x) point(s) de contact. Ce type de réponse est adapté aux objets dé-

formables avec un modèle physique adaptatif. Lorsqu'une recherche dichotomique est utilisée, une unique particule est ajoutée. Suite à quoi, les deux nouvelles arêtes doivent alors être testées pour identifier des éventuelles collisions. Il est possible que l'ajout d'une seule particule ne suffise pas à éviter l'ensemble des collisions. En effet, comme le montre le schéma 4.5.c', l'ajout d'une seule particule sur l'arête peut donner lieu à une collision manquée. Pour la deuxième vérification, une réponse statique – menant ainsi à une réponse hybride (figure 4.5.d) – ou une réponse adaptative peut être choisie selon l'heuristique définissant la subdivision d'une arête.

L'utilisation d'une recherche exhaustive, quant à elle, permet d'identifier directement l'ensemble des sommets posant des contraintes et permet de gérer l'ensemble des collisions simultanément comme sur la figure 4.5.c.

L'algorithme de déplacement de particules rend possible l'ajout de particules à la volée : seules les arêtes concernées par une subdivision subissent des mises à jour. Une heuristique, dépendante de la simulation, est cependant nécessaire afin de s'assurer qu'il n'y a pas trop de particules ajoutées.

Par exemple, on peut décider de subdiviser une arête si le contact est proche du milieu de celle-ci et si elle est assez longue. Si ces conditions ne sont pas satisfaites, on utilise une réponse statique.

Exploitation de la convexité et de la cinétique

Nous proposons deux optimisations permettant d'éliminer des tests d'arêtes à effectuer dans ce cadre. La première méthode consiste à éliminer les tests d'arêtes lorsque les cellules formant l'environnement sont suffisamment importantes pour les inclure totalement. La deuxième méthode consiste à éliminer les tests d'arêtes des particules de polygones fermés selon le déplacement de ces derniers.

Élimination des arêtes selon l'environnement

Lorsque la surface balayée par les arêtes est totalement incluse dans une cellule de l'environnement, il n'est pas possible de rencontrer une collision d'arête. Cette propriété découle directement de la propriété de convexité des cellules de l'environnement. Soit une arête $[p_t, q_t]$ sans collision appartenant à une cellule, si l'on déplace p_t vers une position p_{t+1} appartenant à cette même cellule alors le triangle p_t, q_t, p_{t+1} est totalement inclus dans la cellule selon la

définition des ensembles convexes. Cette notion est reprise par le schéma 4.6.

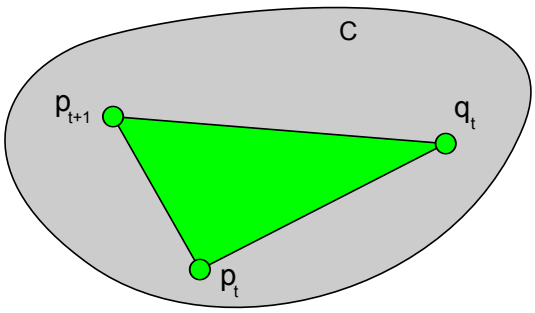

Figure 4.6 : Pour un ensemble convexe C, si p_t, p_{t+1} et q_t appartiennent à C alors la surface couverte par le triangle p_t, q_t, p_{t+1} appartient à C.

En identifiant cette situation il est possible d'assurer, sans effectuer de tests, qu'une collision ne peut survenir. En pratique, l'élimination de tests selon ce principe fonctionne comme suit. Si l'arête avant déplacement n'intersecte pas le bord d'une cellule et si le déplacement d'une particule n'entraîne pas de changement de cellule, alors le lancer de particule au niveau de l'arête peut être économisé. Lorsqu'un lancer de particule le long de l'arête doit être effectué, on mémorise si ce dernier entraîne un changement de cellule, cette information est alors utilisée pour le déplacement suivant.

Élimination des arêtes arrières

Dans le cas du déplacement d'un objet en dimension 2, chaque particule est déplacée à tour de rôle. Lorsque le déplacement concerne un polygone fermé, il est possible de ne tester que les arêtes dont le déplacement est dirigé vers l'extérieur du polygone.

Nous supposons, comme auparavant, que le maillage à déplacer est initialisé sans collision et qu'il ne franchit pas de petits obstacles au cours d'un balayage élémentaire. Partant de cette hypothèse, on peut affirmer que les arêtes ayant un déplacement vers l'intérieur du polygone ont déjà été testées et sont libres de toute collision. Cette idée, évoquée dans l'état de l'art [Van94], s'inspire des méthodes d'élimination de faces arrières provenant des méthodes de rendu. Le cas des polygones ouverts nécessite cependant de tester l'intégralité des arêtes, car il n'existe pas de notion d'intérieur.

Sur la figure 4.7, on voit que deux lancers de particules temporaires peuvent être économisés. Le déplacement de la particule au sommet doit cependant être effectué pour s'assurer du maintien de l'état de cette dernière.

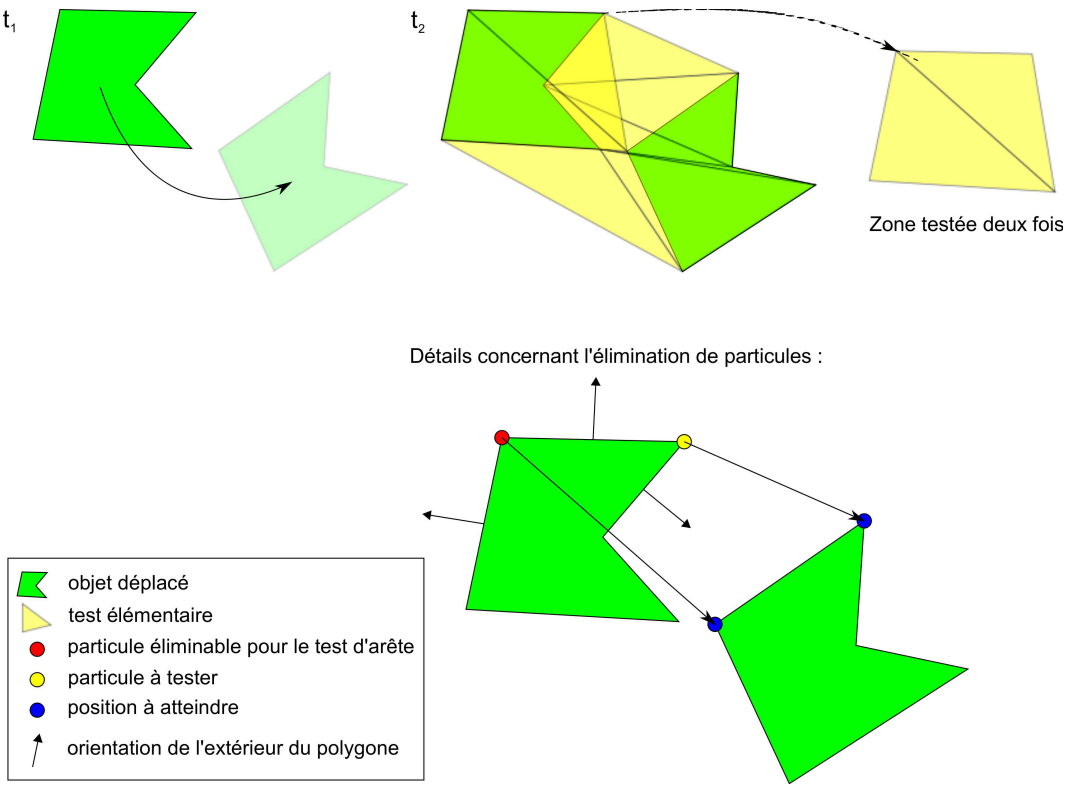

Figure 4.7 : Déplacement d'un polygone fermé et optimisation. Au temps t_1 l'objet n'est pas en collision et possède une position à atteindre, au temps t_2 l'algorithme de déplacement d'arête permet de tester l'espace balayé pour des collisions mais certaines zones peuvent être testées plusieurs fois. En bas : détails concernant le critère d'élimination de zones à tester, lorsqu'une particule est déplacée vers l'intérieur de l'objet l'espace balayé n'est pas testé.

Afin de savoir si des tests d'arêtes peuvent être économisés il est nécessaire d'effectuer 2 tests d'orientation supplémentaires sur l'ensemble des particules. Au bas de la figure 4.7 on voit que la particule rouge a pu être éliminée, car sa position à atteindre est orientée vers l'intérieur du polygone en considération de ces deux arêtes adjacentes. La particule jaune doit être testée car elle est dirigée vers l'extérieur du polygone pour une de ces arêtes adjacentes.

4.2.2 Déplacement dans l'espace

Le principe présenté dans la section précédente s'applique directement en dimension 3. Les sommets du maillage sont déplacés de manière successive. Pour chaque particule, des particules temporaires sont lancées le long des arêtes comme le montre la figure 4.8. Sur cette figure, chaque zone colorée en jaune

est alors considérée comme libre de toute collision.

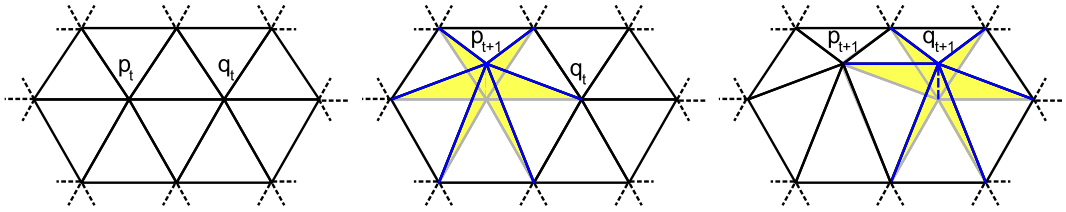

Figure 4.8 : Déplacement successif de particules pour le déplacement d'un maillage en dimension 3, des particules sont également lancées sur les arêtes adjacentes pour tester la présence ou l'absence de collision.

Comme dans le plan, lancer une particule le long d'une arête permet de détecter une collision mais ne fournit pas tous les détails de celle-ci. Telle que représenté sur la figure 4.9, la première collision trouvée peut donner une mauvaise information quant au point d'impact et à la normale de la collision. Les deux méthodes, dichotomique et exhaustive, présentées auparavant peuvent être utilisées ici pour affiner les informations de contact.

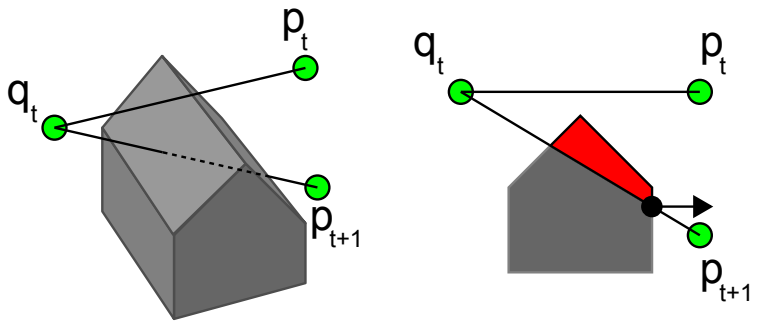

Figure 4.9 : Comme en dimension 2, les informations concernant une collision peuvent être inexactes, la recherche exhaustive ou dichotomique peut être appliquée.

La différence principale due au changement de dimension provient de l'incapacité à signaler toutes les collisions. Dans le plan, tant qu'aucun obstacle de taille inférieure au déplacement n'est rencontré : toutes les collisions sont détectées. En dimension 3, seules les collisions sommet/environnement et arête/environnement sont signalées. La figure 4.10 résume l'ensemble des configurations de collision que l'on peut rencontrer en dimension 3.

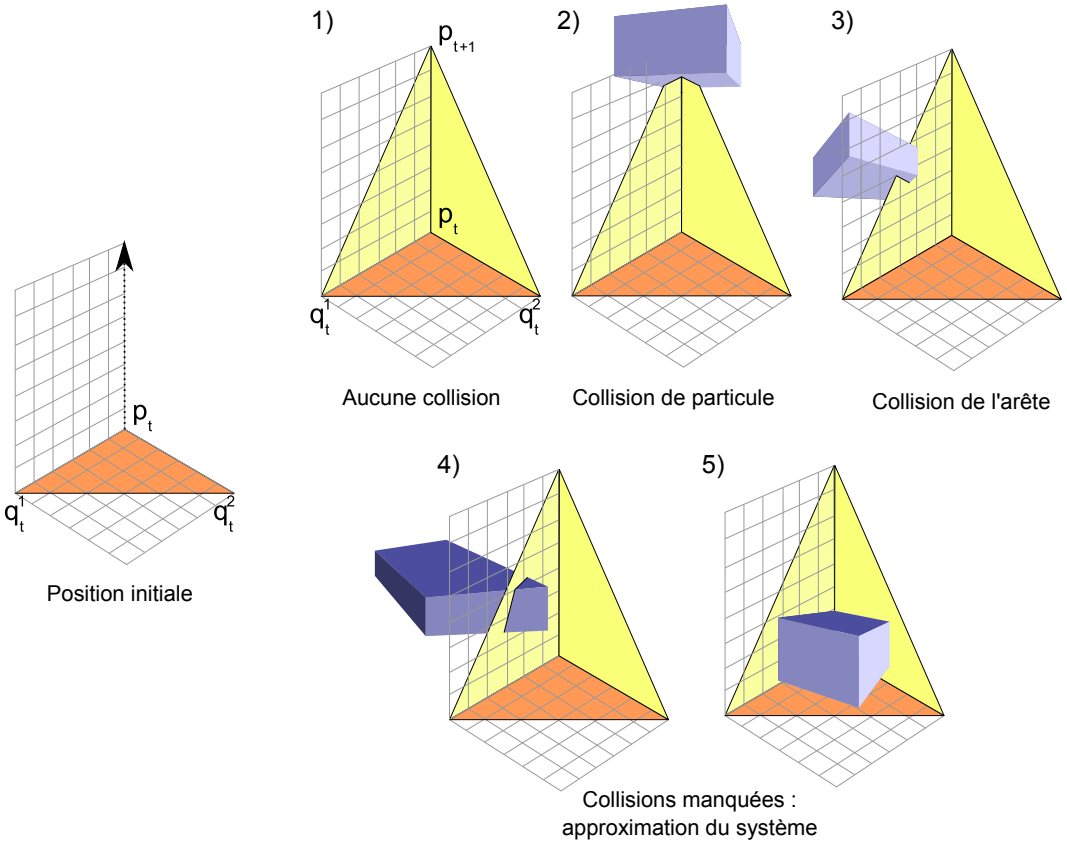

Figure 4.10 : Ensemble des types de collisions détectées et non-détectées avec le déplacement quasi-continu d'arête.

Sur le schéma 4.10, les cas 1 à 4 sont exactement ceux que l'on trouve dans le plan. Le cas 4 étant le cas où un obstacle est trop petit par rapport à la surface balayée par l'arête.

Comme on le voit sur le schéma pour le cas 5, les collisions entre les sommets de l'environnement et du maillage en déplacement ne sont pas testées. Ainsi, un sommet de l'environnement peut intersecter une face de l'objet en déplacement sans être détecté.

La plus grande intersection possible, dans ce cas, correspond à une pyramide inversée dont les arêtes, incidentes au sommet de l'environnement, reposent sur les arêtes de l'objet en déplacement. Le volume de cette pyramide est borné par la taille des faces de l'objet en déplacement et par les déplacements des particules. L'angle que forme le sommet de l'environnement localement influence également le volume d'interpénétration. Plus l'angle est aigu, plus ce volume peut être important.

Dans des scènes composées de matériaux souples, avec peu de pics, et un échantillonnage suffisant de l'objet en déplacement, le volume d'interpénétration est raisonnablement borné. De plus, dans la plupart des applications de simulation, l'objet en déplacement est fortement échantillonné, tandis que l'environnement est représenté de manière plus grossière. Cela rend les cas d'interpénétrations moins fréquents et moins importants.

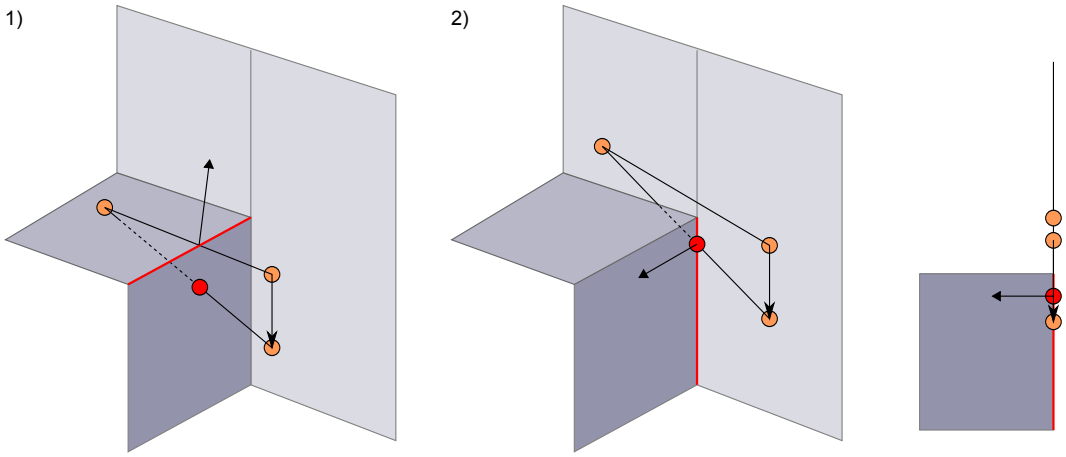

Figure 4.11 : Cas particulier de recherche de normale avec la méthode dichotomique.

La figure 4.11 illustre un autre cas particulier auquel il est nécessaire de prêter attention lors du déplacement d'une arête en dimension 3 lors de l'utilisation de la recherche par dichotomie. Le cas de gauche représente le cas fréquent où la normale de la collision peut être trouvée sans problème. Dans le cas de droite on voit cependant que la collision d'arête indique un brin de collision appartenant à un coin. La normale du plan formé par ce brin et la particule adjacente est donc fausse. Ainsi, lorsque la collision d'une arête est identifiée et renvoie une collision au niveau d'une arête de l'environnement, il est nécessaire de procéder à une vérification déterminant si l'arête renvoyée est bien celle qui a provoqué la collision. Le cas de la recherche exhaustive ne crée pas de problème.

Le fait d'avancer les particules à tour de rôle entraîne des approximations. L'ordre choisit parmi ces particules influence donc évidemment les imprécisions produites vis à vis des déplacements imposés par le modèle physique. Cependant, les pas de temps étant faible, ces approximations restent limitées, bien que difficiles à mesurer sans comparaison à des simulations réelles.

Pour éviter celles-ci, il faudrait faire progresser l'ensemble des particules de manière synchronisée. Pour ce faire, lorsqu'une collision survient au niveau d'un déplacement de particule ou d'arête, l'information devrait être propagée à l'ensemble des particules. Le déplacement de toutes les particules, stoppé au

premier point de collision, permettrait alors au système physique de réévaluer les forces à prendre en compte. Cette méthode rend la complexité du déplacement dépendante du nombre total de particules simulées, ce qui peut nuire à l'interactivité de simulation. Ce lien à la totalité de l'objet déplacé est d'autant plus pénalisant lorsque de multiples collisions sont rencontrées lors d'un seul pas de temps.

Une solution intermédiaire consisterait à déplacer les particules par couples au niveau des arêtes. Cette solution, quelque peu moins coûteuse, nécessiterait cependant de mémoriser, pour l'ensemble des arêtes attachées à une particule, l'ancienne et la nouvelle configuration obtenu pour chaque arête. La position atteinte la plus contrainte devrait alors être conservée pour la suite de la simulation.

Déplacement d'un polyèdre

Comme pour le déplacement d'un polygone fermé dans le plan, le déplacement d'un polyèdre est effectué par déplacement successif de chacune des particules.

Lors du déplacement d'un polyèdre fermé, l'optimisation proposée pour le déplacement d'un polygone fermé peut également être appliquée. Cependant, les angles au niveau d'un sommet dans l'espace ne sont pas formés uniquement par deux arêtes. Le changement de dimension complexifie donc les tests à effectuer. De ce fait, savoir si un déplacement est dirigé vers l'intérieur de l'objet, ou vers l'extérieur, peut s'avérer trop coûteux en termes de calculs à effectuer. Afin de minimiser les calculs, nous éliminons les tests de collision d'arêtes d'un sommet uniquement lorsque l'ensemble des faces autour de ce sommet nous permet de définir que le déplacement se fait vers l'intérieur du polyèdre.

Ainsi dès que le déplacement peut se diriger vers l'extérieur au niveau d'une des faces autour du sommet (c'est-à-dire que le déplacement va dans le sens de la normale de la face considérée) alors le test est effectué.

Nous effectuons une étude de ce procédé dans le chapitre d'analyse des algorithmes.

4.3 Détection de collision d'arêtes de manière continue

Il est parfois nécessaire de s'assurer qu'aucune collision ne soit manquée lors d'un déplacement d'arête. La méthode quasi-continue précédente ne permet pas de le garantir lorsque les déplacements effectués sont trop grand et qu'il existe de petits obstacles. Nous proposons ici une méthode basée sur une recherche dans un graphe afin d'assurer que cela ne se produise pas.

Cette méthode recoupe l'idée de la recherche exhaustive présentée précédemment dans ce chapitre. Nous décrivons ici uniquement l'algorithme en dimension 2, l'extension en dimension 3, suit le même principe.

L'extension en dimension 3 de cette méthode permet également de considérer le déplacement de face de manière quasi-continue. Le déplacement des faces s'effectue selon le même principe que précédemment : il s'agit d'approximer le déplacement d'une face par des déplacements d'arêtes.

4.3.1 Déplacement dans le plan

La détection de collision continue pour le déplacement d'arête, que nous proposons, se base sur une recherche de l'ensemble des cellules balayées lors d'un déplacement. Plus exactement, il s'agit, pour la dimension 2, d'identifier l'ensemble des brins de l'environnement dont l'intersection avec le triangle τ, décrit dans la section précédente, est non vide. L'identification de ces brins permet de détecter si un obstacle est franchi lors d'un déplacement, ou si ce dernier est valide. Pour effectuer cette identification, nous mémorisons les brins des cellules intersectées par les arêtes $[p_t, q_t]$ et $[p_t, p_{t+1}]$. A partir de ces brins, nous cherchons tous les brins contenus dans τ, ces derniers nous indiquent si une cellule infranchissable est balayée par le déplacement. Si le déplacement ne contient pas d'obstacle, nous mémorisons, parmi les brins contenus dans τ, les brins intersectant l'arête $[p_{t+1}, q_t]$. Ces brins sont utilisés pour le prochain déplacement. Si plusieurs brins infranchissables sont rencontrés, un tri de ces derniers selon l'angle à la position de base de l'arête permet de définir le premier point de collision.

De manière plus détaillée, la méthode de déplacement d'une arête est la suivante : à chaque pas de temps, nous mémorisons l'ensemble des brins qu'elle intersecte. La particule d'un de ses sommets est déplacée vers sa position à atteindre tout en mémorisant l'ensemble des brins traversés.

Le déplacement et la mémorisation du déplacement d'une particule sont nécessaires. En effet, il est possible que l'arête ne coupe aucun brin avant un déplacement, ou encore que le déplacement soit aligné avec l'arête. De la même façon, il est nécessaire de mémoriser l'ensemble des brins précédents car le déplacement d'une particule peut être effectué sans intersecter de brins alors que le balayage de l'arête en rencontre.

Enfin, ces mémorisations permettent, lorsqu'aucun brin n'était intersecté par l'arête au pas de temps précédent et que la particule au sommet ne franchit pas de cellules, de n'effectuer aucun test grâce à la propriété de convexité des cellules de l'environnement. Ce dernier point est lié à la technique d'élimination de tests selon l'environnement présenté précédemment.

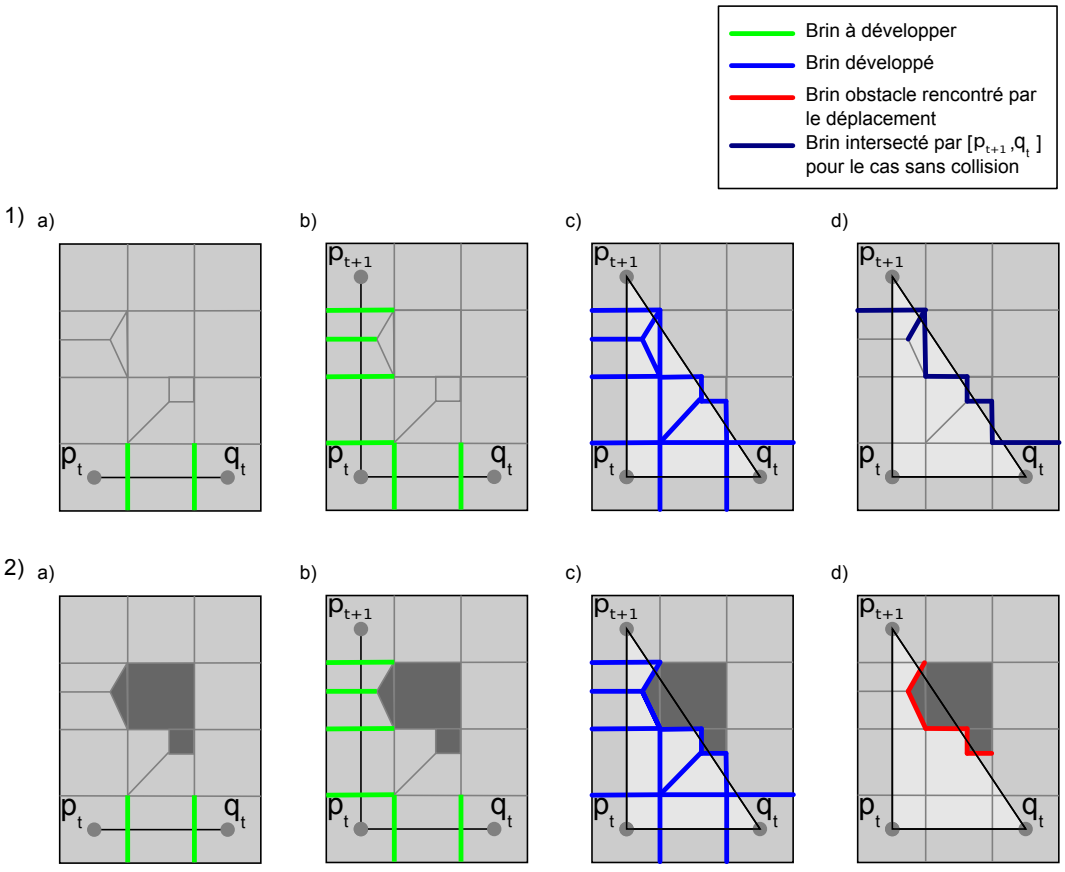

Figure 4.12 : Détection de collision d'arête de manière continue. 1) Cas sans collision. 2) Cas avec collision. a) L'arête est à sa position initiale, b) une particule est déplacée, c) on identifie les brins balayés par ce déplacement, d) on identifie l'ensemble des brins intersectant la nouvelle position de l'arête ou le front de la collision.

L'algorithme commence donc avec une liste de brins intersectés par l'arête ou par le déplacement de la particule. A partir de cette liste non-ordonnée, nous cherchons l'ensemble des brins compris dans le triangle balayé par le déplace-

ment. A partir d'un brin de la liste, nous effectuons une phase de développement. Le développement consiste à tester l'inclusion des brins adjacents (selon les orbites de sommets) dans le triangle τ. Si un brin adjacent est inclus dans le triangle de balayage, alors il est rajouté à la liste et est traité par la suite. Le traitement de la liste est effectué jusqu'à ce que l'ensemble des brins qu'elle contient aient été développés.

Dès qu'un brin a été développé, il n'est plus testé par la suite, évitant ainsi d'effectuer le test d'inclusion de manière répétée pour un même brin. Lorsqu'un brin marqué comme un obstacle est repéré pendant le balayage, il n'est pas considéré comme étant à développer. En effet, l'ensemble des brins marqués comme des obstacles qui ne sont pas accessibles à partir de brins non marqués ne peuvent faire partie du front de la collision.

La figure 4.12 illustre cet algorithme. Le cas 1) décrit le cas sans collision avec l'état final mémorisé, le cas 2) le cas avec collision. Dans le 2ème cas, l'ensemble des brins mémorisés pour l'itération suivante est dépendant de la réponse physique souhaitée.

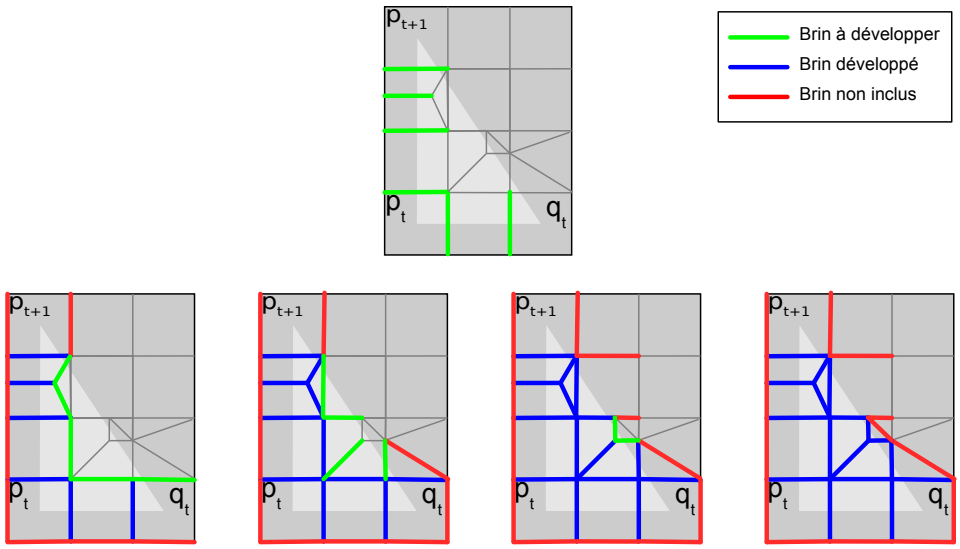

Figure 4.13 : Détection de collision d'arête de manière continue : décomposition de la phase de recherche de la zone balayée.

Nous expliquons maintenant la phase de recherche dans la partition de manière à limiter le nombre de tests d'inclusion dans la zone balayée. A l'initialisation, l'ensemble des brins à développer consiste en l'ensemble des brins coupés par l'arête avant déplacement et l'ensemble des brins coupés par le déplacement de la particule. Un brin est à développer si le test l'inclusion de celui-ci dans la zone balayée est positif. A partir d'un brin à développer, nous testons alors si le brin suivant dans l'orbite du sommet et le brin précédent de

cette même orbite sont à développer et ceci pour les deux sommets du brin. Le fait d'effectuer ce balayage dans les deux sens de l'orbite permet d'éviter de tests inutiles. En effet, si au niveau de l'orbite d'un sommet, en partant d'un brin à développer, nous trouvons un brin à gauche et à droite non inclus, alors l'ensemble des brins inclus entre ces deux brins n'est pas inclus dans le déplacement. La figure 4.13 montre un exemple de recherche des brins intersectés.

4.3.2 Détection de collision de faces quasi-continue

Il est facile d'étendre la détection de collision d'arête de manière continue au déplacement de face de manière quasi-continue. Cette extension suit le même principe que l'utilisation de l'algorithme de déplacement de particules de manière continue pour la détection de collision d'arêtes de manière quasi-continue. Il suffit de décomposer le mouvement d'une face en un ensemble de balayages continus d'arêtes. Cette extension est limitée aux faces triangulaires afin d'assurer une couverture complète de la face.

Déplacement d'une face

Pour déplacer une face de manière quasi-continue, nous considérons la surface du volume balayé par le déplacement de celle-ci. Cette surface est alors testée grâce à la méthode du déplacement d'arête de manière continue.

La figure 4.14 illustre le déplacement d'une face. Une face, à un temps t, est déplacée vers une nouvelle position en $t+1$. Chacun des sommets est déplacé à tour de rôle. Lorsqu'un sommet est déplacé, trois triangles sont balayés grâce à l'algorithme de détection de collision d'arête de manière continue. Deux de ces triangles sont formés par la position précédente du sommet, sa nouvelle position et par les sommets adjacents. Le troisième triangle testant la face est formé par la nouvelle position du sommet et les deux sommets adjacents.

Ce déplacement est qualifié de quasi-continu car il est possible qu'un obstacle plus petit que le volume couvert par le déplacement de la face soit manqué lors de ces tests.

Les algorithmes de recherche basés sur la dichotomie ou sur une recherche exhaustive, décrits dans le cas du déplacement d'arête quasi-continu, sont également valides ici. Seule la dimension des tests est ajustée.

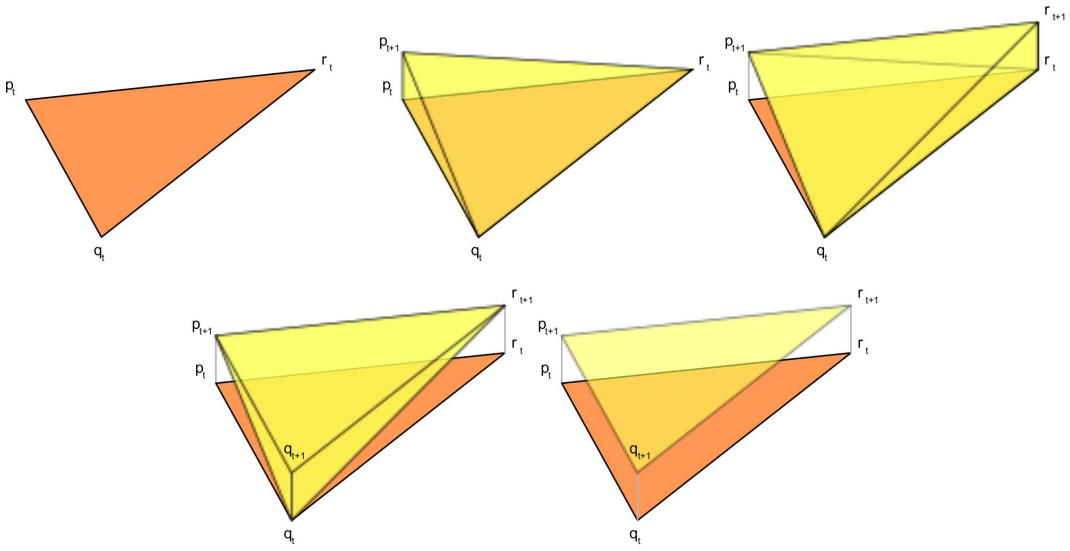

Figure 4.14 : Utilisation du déplacement d'arête continu pour le déplacement de face de manière quasi-continue.

Déplacement d'un polyèdre

Lors du déplacement d'un polyèdre fermé, la méthode d'élimination des tests effectués pour les arêtes proposée pour le cas des déplacements d'arêtes de manière quasi-continue peut être repris de manière similaire. Cette méthode permet à nouveau d'économiser des tests de faces superflus. En effet, si l'on considère comme auparavant qu'un maillage ne contient aucun obstacle manqué lors des tests, alors seules les faces se dirigeant vers l'extérieur doivent être testées.

Le gain, en termes de nombre de tests à effectuer, de ces tests d'élimination est accru ici. En effet, un déplacement de face nécessite plus de tests qu'un déplacement d'arêtes.

4.4 Conclusion

Nous avons présenté dans ce chapitre des algorithmes de déplacement d'arêtes dans le plan et dans l'espace. Selon les propriétés de la partition, il est possible de proposer diverses implantations apportant des précisions variables concernant les tests de détection de collision.

La première méthode proposée, qui consiste à effectuer des déplacements de manière quasi-continue, est une extension directe de l'algorithme du dé-

placement de particules du chapitre précédent. Cette méthode ne permet pas de traiter complètement le cas des faces. Dans le cas où l'on ne possède pas d'échantillonnage fin de l'objet en déplacement, ce dernier doit alors être raffiné si l'on désire s'assurer de la validité des réponses des tests de collisions.

Lorsque l'on souhaite assurer que l'ensemble des obstacles est identifié sans approximation dans toute situation, la méthode de déplacement d'arête de manière continue est à utiliser. Le fait de rechercher l'ensemble des obstacles balayés durant le déplacement d'une arête permet en effet d'identifier toutes les collisions sans approximations.

Le choix entre ces deux méthodes est fortement lié à l'échantillonnage en particules, au niveau de la surface de l'objet en déplacement, et par l'environnement utilisé au cours de la simulation. La méthode d'échantillonnage temporel utilisé au cours de la simulation est en général prédéfinie par le modèle physique et les outils d'interactions utilisés. Il est ainsi possible, en prenant en compte l'ensemble de ces paramètres, de déterminer la méthode à privilégier.

Chapitre 5

Modèles physiques et implantation

Sommaire

Dans ce chapitre nous présentons l'ensemble des modèles physiques mis en œuvre afin de valider nos algorithmes de déplacements. Nous donnons ensuite un schéma d'intégration de ces modèles physiques par rapport au système de détection de collision. Pour conclure ce chapitre, nous donnons également des pistes concernant les accélérations matérielles utilisables au niveau d'une application de simulation intégrant nos algorithmes.

5.1 Introduction

Nous effectuons ici une brève introduction aux modèles physiques, avant de présenter ceux mis en œuvre dans nos applications.

Lors des simulations que nous considérons, la gravité, les forces de contact et les forces d'interactions sont appliquées. Si seules ces forces sont appliquées

alors les objets déplacés se retrouvent aplatis, allongés ou dans des configurations non réalistes. Il est donc nécessaire de créer un lien entre les différentes entités des objets en déplacement, afin que ceux-ci gardent une forme cohérente au cours du temps.

Pour ce genre de simulations, les objets peuvent avoir diverses propriétés physiques. Nous ne considérons dans ce chapitre que le cas des maillages déformables. La gestion des objets rigides est également possible en utilisant les algorithmes que nous avons présentés. Une adaptation est cependant nécessaire au niveau du système de détection de collision lorsqu'un obstacle est rencontré. Il est en effet nécessaire de mémoriser pour chaque pas de temps, le premier temps de collision rencontré sur l'ensemble des éléments déplacés. Une fois ce temps trouvé, il faut alors reculer l'ensemble des éléments à ce temps là, afin de recalculer selon une physique rigide l'ensemble des nouvelles directions prises.

Nous ne détaillons pas les problèmes de stabilité numérique qui sont communément rencontrés dans ce genre de traitements. Afin de limiter les problèmes d'instabilité d'évaluation de vitesse, nous utilisons la méthode de Runge-Kutta à l'ordre 4. Cette méthode permet une meilleure approximation des solutions d'équations différentielles représentant la vitesse et les accélérations lors d'une simulation à pas de temps discrets.

5.2 Simulation physique pour objets déformables

Dans les simulations chirurgicales, la méthode des éléments finis (*Finite element method* - FEM) est fréquemment utilisée. Il s'agit d'une méthode numérique permettant la simulation de modèles physiques continus par la résolution d'équations aux dérivées partielles. Cette méthode, bien qu'appropriée au cadre que nous visons, peut parfois être coûteuse en terme de temps de calcul. Certains travaux se concentrent sur l'accélération des temps de calcul de FEM, que nous ne traitons pas ici. Nous renvoyons le lecteur vers des articles traitant ce problème [MG04, NPF05, ISF07, MKB+08].

Nous présentons ici deux modèles physiques pour la simulation d'objets déformables rapidement implantables et interactifs. Ces méthodes sont la méthode des masses-ressorts, et la méthode du *shape-matching*.

Un système masse-ressort se base sur des contraintes définies aux niveaux des arêtes du modèle et nécessite certaines informations topologiques de celui-ci. La méthode du shape-matching, quant à elle, se base uniquement sur l'estimation du positionnement d'un nuage de points afin de correspondre à une

configuration initiale donnée.

5.2.1 Masse-ressort

Un système masse-ressort est un système mécanique à un degré de liberté. Il s'agit ici de placer des ressorts virtuels à des endroits clés des objets à simuler. Ces ressorts appliquent des contraintes selon leur rigidité dans une seule et unique direction. Un système masse-ressort fait rentrer en compte trois paramètres : une force de raideur, une force d'amortissement et une force extérieure.

L'application de la force du ressort r s'exprime comme suit :

$$r = k * (d_{repos} - ||p_i - p_j||).\frac{p_i - p_j}{||p_i - p_j||} \tag{5.1}$$

Dans cette équation k correspond à la raideur du ressort et d_{repos} à la longueur au repos de l'arête entre les particules p_i et p_j. La force r est appliquée selon le vecteur défini par l'arête entre les particules proportionnellement à la taille du segment $[p_i, p_j]$. Cette force attire les particules l'une vers l'autre lorsque l'arête est allongée, les éloigne lorsque l'arête à une taille réduite et est nulle lorsque l'arête à une longueur égale à sa longueur au repos.

La force d'amortissement réduit la force du ressort par rapport à la vitesse relative des particules. Cette force permet de diminuer les effets d'oscillation que l'on peut rencontrer dans un tel système.

La force extérieure correspond à la somme des forces de contact, la gravité ainsi que des forces d'interaction.

Il est parfois souhaitable d'avoir des ressorts angulaires afin de contraindre les liaisons de pivot entre les différentes arêtes du modèle. Lorsque les contraintes appliquées ne considèrent que des critères de distance pour chaque arête, le système est sous-contraint et peut se replier sur lui-même.

Par la suite, nous mettons en application notre algorithme pour la simulation d'un cathéter, représenté par une ligne brisée d'arêtes. En plus des ressorts sur les arêtes, des ressorts sont ajoutés entre deux particules espacées d'une particule, tel qu'illustré par la figure 5.1, afin de rigidifier la ligne brisée.

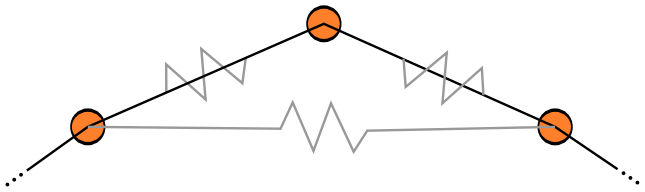

Figure 5.1 : Exemple de positionnement de ressort pour la simulation d'un cathéter.

Bien que nous ne revenions pas par la suite sur la notion de raffinement des systèmes masse-ressort, il est également possible de considérer des systèmes adaptatifs, afin d'échantillonner plus finement les zones à fortes courbures ou en collision [HPH96, HH98].

5.2.2 Shape-Matching

Modèle standard

Pour l'implantation de notre physique d'objets déformables nous avons effectué une implantation de la méthode présentée dans [MHTG05]. Cette méthode permet de simuler la physique d'objets déformables représentés par un ensemble de particules sans information de connectivité. Nous donnons ici le détail de cette méthode.

L'idée de base est la suivante. Un ensemble de i particules de masses m_i est donné dans une configuration de repos. Une configuration au repos définit l'ensemble des positions des particules tel qu'aucune force interne à l'objet ne soit à appliquer. On note p_0^i la position initiale de chaque particule. Chaque déplacement de particule est considéré comme indépendant des autres et est régi uniquement par : l'ensemble des forces externes (tel que la gravité) et les réponses aux collisions dues à l'environnement dans lequel elle se déplace. A un pas de temps t, chaque particule se trouve à une position p_t^i.

Le *shape-matching* consiste alors à trouver la transformation optimale permettant d'obtenir la configuration au repos. Dans le cadre du *shape-matching* linéaire, cela revient à calculer la matrice de rotation R et les vecteurs de translation T et T_0 qui minimisent l'équation 5.2.

$$\Sigma_i m_i (R(p_0^i - T_0) + T - p_t^i)^2 \tag{5.2}$$

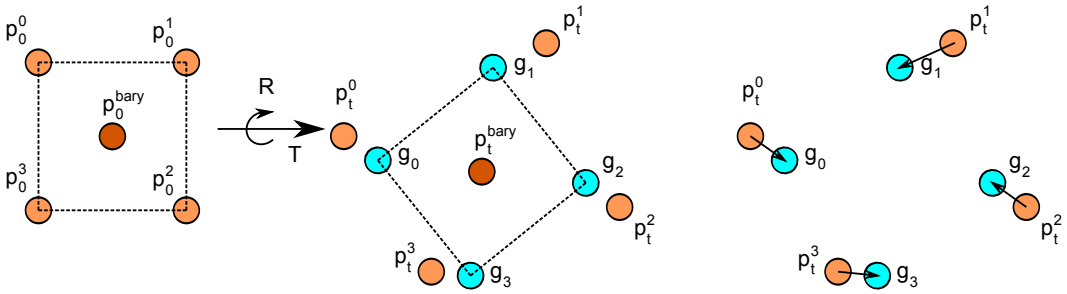

Figure 5.2 : A gauche la configuration au repos ; à un temps t on calcule la transformation minimale correspondante à la position au repos ; la simulation ramène les points vers la position calculée.

Une fois cette rotation et cette translation identifiées, il s'agit alors de ramener les sommets de l'objet vers la configuration au repos comme l'illustre la figure 5.2. Le problème est décomposé en deux étapes. La première étape consiste à trouver les translations T et T_0 concernant les deux configurations. A partir de ces translations, on exprime les coordonnées des particules dans un repère local permettant d'identifier la rotation R.

Les vecteurs de translation optimaux se basent sur les barycentres de la configuration au repos et de la configuration modifiée, notés respectivement p_0^{bary} et p_t^{bary} , soit :

$$T_0 = p_0^{bary} = \frac{\sum_i m_i p_0^i}{\sum_i m_i}, T = p_t^{bary} = \frac{\sum_i m_i p_t^i}{\sum_i m_i} \tag{5.3}$$

La rotation optimale est trouvée en fonction des positions relatives des points à leur barycentre soit $q_i = p_0^i - p_0^{bary}$ et $p_i = p_t^i - p_t^{bary}$. Afin d'extraire la rotation R de la transformation linéaire optimale A, il s'agit alors de minimiser le terme $\sum_i m_i(Aq_i - p_i)^2$. La transformation optimale s'exprime de la manière suivante :

$$A = A_{pq}A_{qq} = (\sum_i m_i p_i q_i^T)(\sum_i m_i q_i q_i^T)^{-1} \tag{5.4}$$

Le terme A_{qq} est une matrice symétrique ne contenant qu'un coefficient de mise à l'échelle et non de rotation. La rotation optimale R se trouve donc être la rotation incluse dans la matrice A_{pq} qui peut être trouvée par une décomposition polaire $A_{pq} = RS$, où la partie symétrique S vaut $\sqrt{A_{pq}^T A_{pq}}$ et

la rotation R est $A_{pq}S^{-1}$. Une fois cette rotation extraite, il est possible de calculer les positions optimales g_t^i pour les particules p_t^i comme décrit dans l'équation 5.5.

Le calcul de S^{-1} est effectué en diagonalisant la matrice $A_{pq}^T A_{pq}$ par des rotations jacobiennes.

$$g_t^i = R(p_0^i - p_0^{bary}) + p_t^{bary} \tag{5.5}$$

A partir des positions à atteindre, il est possible de définir la modification de la vitesse des sommets afin que ceux-ci se déplacent vers la configuration au repos. L'utilisation d'un paramètre de pondération sur le vecteur vitesse $(g_t^i - p_t^i)/(\Delta t)$ permet de faire varier l'élasticité du modèle lors de collisions.

Modèle adaptatif

Comme vu dans le chapitre 4, il est possible d'utiliser une réponse adaptative, au niveau des arêtes, qui ajoute des particules lors de collision. Lorsque ce type de réponse est utilisé, la méthode du *shape-matching* présentée ci-dessus est alors légèrement modifiée. Nous considérons, dans ce cas, que la position de repos d'une particule ajoutée est l'interpolation des positions de repos des particules. Cette interpolation est faite en fonction du rapport de longueur de part et d'autre de la particule ajoutée dans la configuration actuelle.

$$p_0^a = r * p_0^i + q_0^i(1 - r) \tag{5.6}$$

Dans l'équation 5.6, p_0^a est le vecteur de repos d'une particule insérée dans le système, p_0^i et q_0^i sont les vecteurs de repos des particules formant l'arête en collision. r est le ratio représentant la position de la collision par rapport à la longueur de l'arête.

Le paramètre m_i, exprimé dans le terme à minimiser exposé précédemment, est utilisé, dans ce cas, pour pondérer la considération des nouvelles particules. Si trop de poids est donné à des particules insérées, le calcul de la transformation minimum tend à faire basculer le mobile dans la direction des insertions.

Lors de la suppression d'une particule ajoutée, il n'y a pas de modifications particulières à faire au niveau du modèle physique.

5.3 Intégration collision/physique

L'intégration du système de détection de collision et du système physique est décrit dans cette section. Les données concernant la cinétique comprennent : la position, la vitesse et les informations de contact. Ces données sont rattachées aux sommets de l'objet en déplacement par l'association de particules à chaque orbite de sommet. Les données concernant la mécanique sont, quant à elles, rattachées à des éléments de l'objet ou à l'objet dans sa globalité. Par exemple, la simulation masse-ressort associe une raideur à chaque arête, la simulation de type *shape-matching* associe des positions de repos à chaque sommet.

Les informations de collisions, ou de contacts, sont définies par la position et la configuration des particules (*i.e.* collision au niveau d'un plan, d'une arête ou d'un sommet, rattachée à une normale). Le fait de mémoriser un historique des contacts permet également de définir s'il s'agit d'un contact de type glissement, rebond, ou encore s'il s'agit d'un contact bloquant.

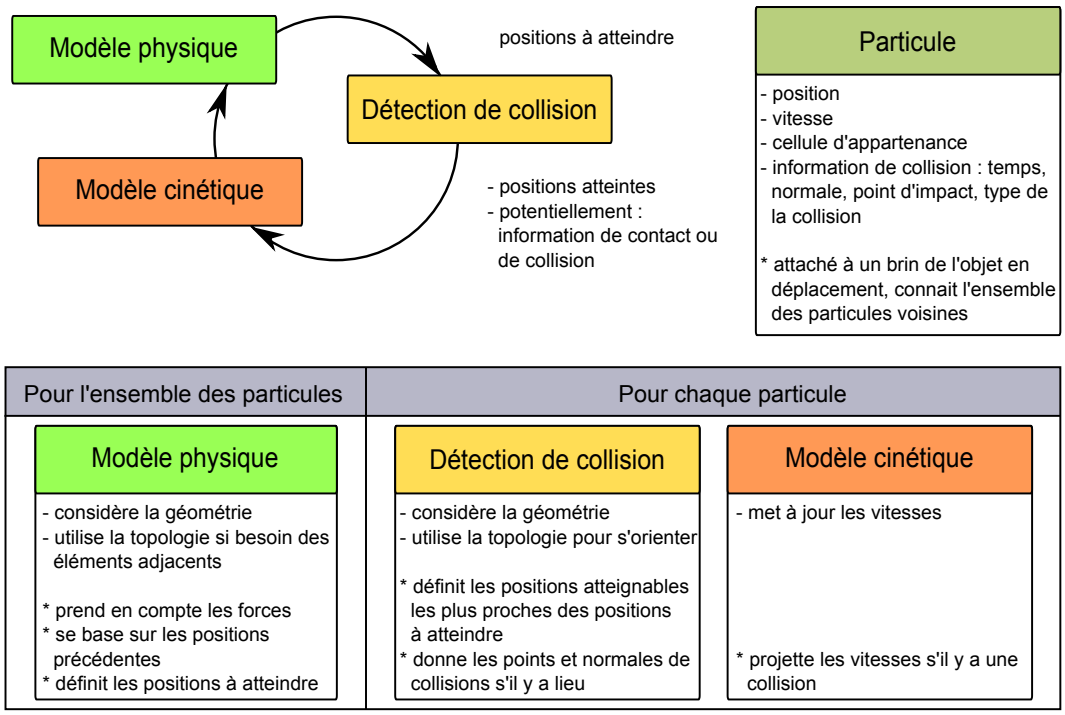

Figure 5.3 : Vue générale de l'intégration collision/physique.

La boucle de simulation est illustrée par la figure 5.3. Au cours de celle-ci, le système physique calcule l'ensemble des forces à appliquer aux sommets selon leurs configurations actuelles. Le système de détection de collision essaye

ensuite de déplacer chacun des sommets à tour de rôle et indique les positions réellement atteintes, ainsi que les informations de contact lorsqu'une collision survient. Les positions et vitesses des sommets sont modifiées en conséquence et renvoyées au système physique pour la prochaine itération.

Nous décrivons ici un modèle d'intégration permettant de simuler des objets déformables.

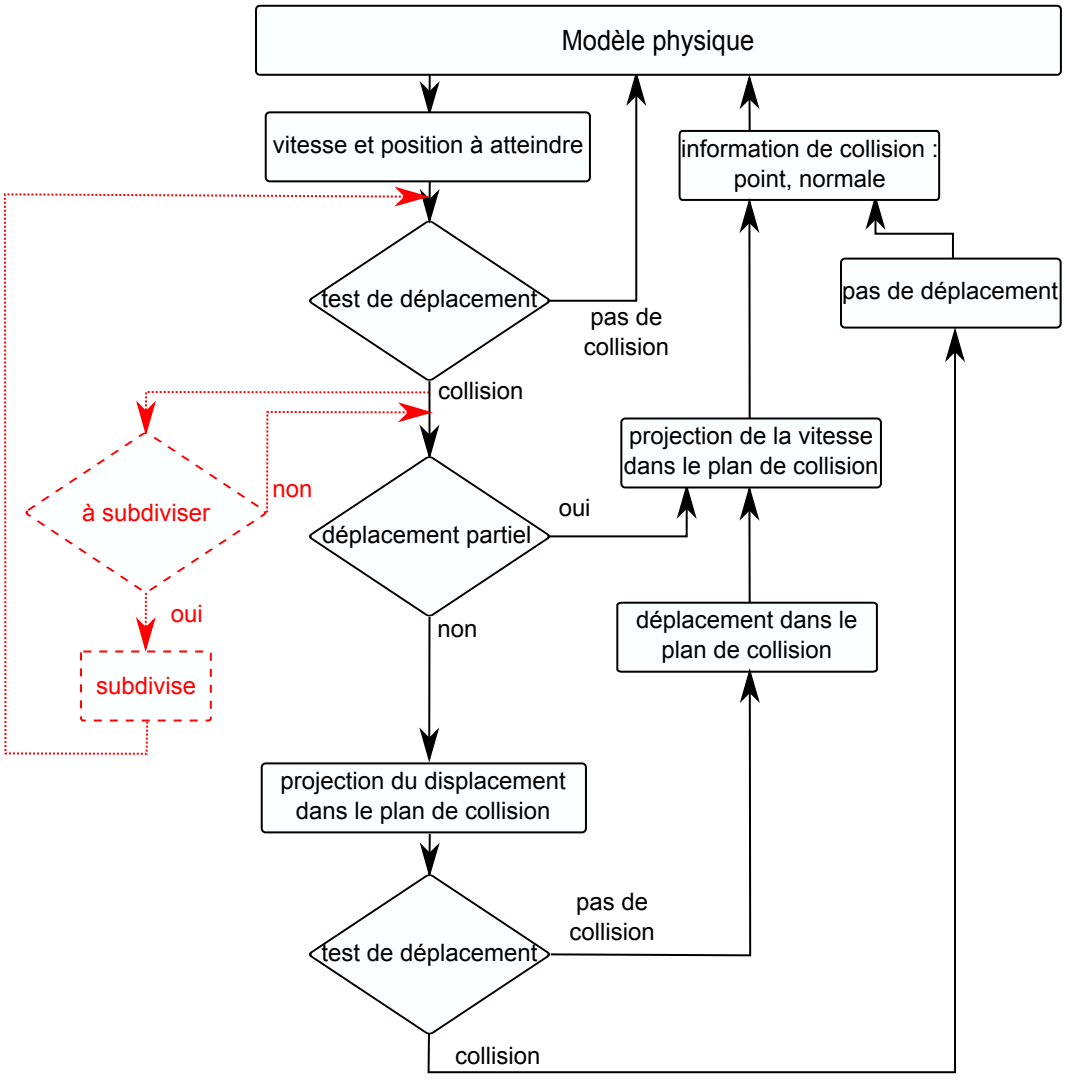

Figure 5.4 : Détection de collision et modèle cinétique.

Le schéma de la figure 5.4 détaille le système de détection de collision et les réponses. Lorsqu'un sommet est déplacé d'une position p_t à une position p_{t+1}, nous déterminons si une collision est survenue ou non. Si aucune collision ne survient, le sommet est déplacé, sa position et sa vitesse sont mis à jour.

Nous considérons, dans la suite de ces explications, que la détection de collision se fait en utilisant le déplacement d'arête. Pour utiliser la détection de collision sur les particules uniquement, il suffit de considérer, dans la suite de la description, qu'aucune collision d'arête n'est détectée.

Lorsque le déplacement d'un sommet entraîne une collision au niveau de ce dernier ou au niveau d'une arête adjacente à ce sommet, différentes réponses sont proposées.

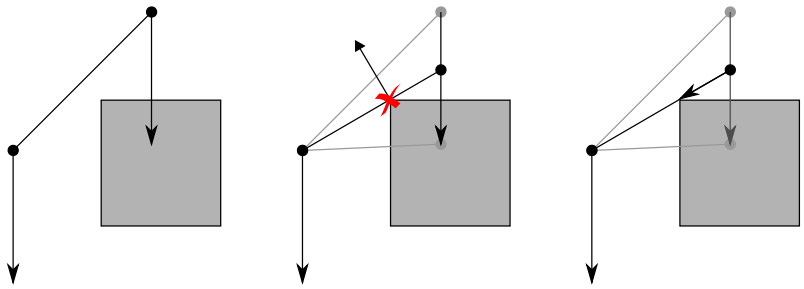

Figure 5.5 : Premier cas : déplacement partiel possible.

Premier cas : un déplacement partiel est possible. Le sommet est déplacé jusqu'à la position atteignable et sa vitesse est projetée dans le plan de la collision. Un nouveau déplacement de ce sommet est possible s'il rebondit dans la direction de la normale à la collision, ou s'il glisse le long du plan de celle-ci.

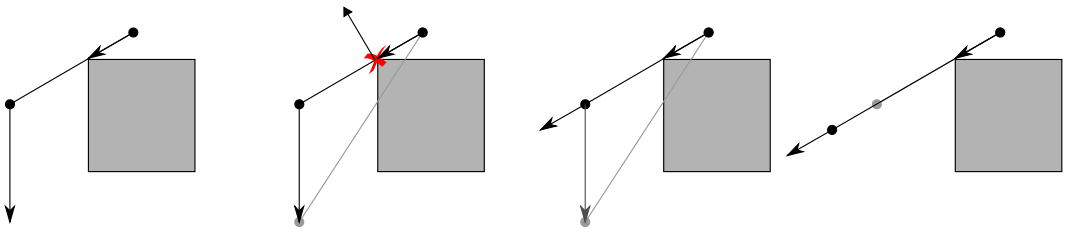

Figure 5.6 : Deuxième cas : aucun déplacement possible.

Deuxième cas : aucun déplacement n'est possible. Cela signifie qu'une collision est déjà survenue au niveau d'une arête adjacente et que le déplacement n'est plus possible, ou que le modèle physique pousse le sommet vers la zone de collision. Ce cas peut arriver également lorsque, pour une question numérique par exemple, un contact n'a pas encore été identifié, les forces calculées par le modèle physique ignorent alors la contrainte imposée. Pour éviter de recalculer la nouvelle position à atteindre au niveau du modèle physique, nous choisissons de projeter le déplacement dans le plan de la collision et de retenter un déplacement dans celui-ci.

Lorsque le blocage du déplacement est dû à une collision d'arête, le premier sommet de l'arête est déplacé partiellement et le deuxième se retrouve bloqué. Cette réponse permet au deuxième sommet de glisser le long du plan de collision identifié par le premier sommet.

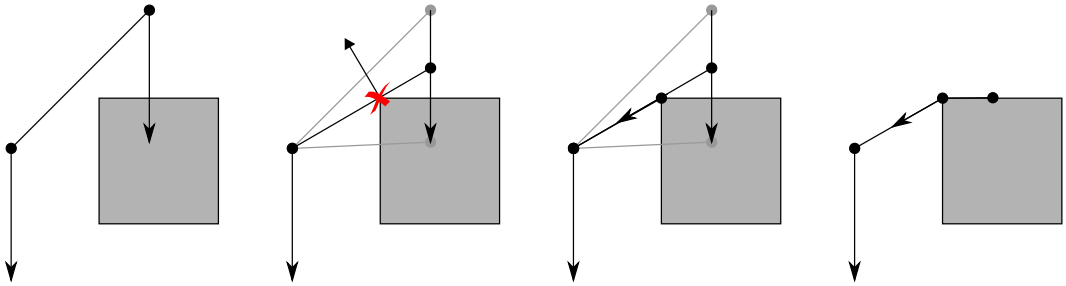

Figure 5.7 : Troisième cas : subdivision d'une arête.

Troisième cas : une collision intervient le long d'une arête et le modèle physique gère le raffinement adaptatif du maillage. Si les critères de subdivision sont remplis (*e.g.* s'il y a une longueur d'arête considérée comme suffisante de part et d'autre de la collision), l'arête est subdivisée, autrement la réponse standard est utilisée. Le sommet inséré est alors déplacé de manière similaire aux particules d'origine. Lorsque le contact disparaît et que la position du sommet ajouté est une interpolation linéaire des positions des deux sommets adjacents avec une force relative faible, la particule est supprimée, restaurant ainsi l'arête dans sa configuration d'origine.

Le modèle physique peut se baser sur une évaluation globale ou sur une évaluation locale, afin de calculer les déplacements à effectuer. Alors que le système de détection de collision est effectué sur chaque cellule atomique (*i.e.* sommets ou arêtes) indépendamment. Les réponses proposées ici démontrent la capacité de notre système de détection de collision à gérer différents types de modèles physiques. D'autres réponses peuvent être mises en œuvre, afin de correspondre à d'autres propriétés physiques. L'intérêt des réponses que nous avons proposées se trouve dans le fait qu'elles permettent d'empêcher l'interpénétration des objets en déplacement avec l'environnement. Les sommets et les arêtes peuvent être bloqués sur les surfaces de contact et leurs vitesses projetées sur les plans tangents. Les forces internes élastiques générées par les modèles physiques entraînent alors des rebonds ou des glissements des objets en déplacement.

5.4 Accélération matérielle

L'implantation actuelle n'est pas adaptée à l'exécution sur des cartes graphiques. En effet, l'accès à la topologie et à la géométrie des cartes entraînerait un coût de transfert d'informations du processeur vers la carte graphique trop important pour obtenir un réel apport au niveau du temps d'exécution. Bien que les langages actuels de programmation pour les cartes graphiques permettent la mise en œuvre d'algorithmes de plus en plus complexes, la question de la gestion d'environnements de grandes tailles n'a actuellement pas de solution. La quantité de mémoire disponible sur les cartes graphiques est encore à l'heure actuelle une limite pour ce type d'applications. Les méthodes basées images se permettent de ne faire qu'un rendu partiel de la scène afin d'effectuer de la détection de collision. Nos algorithmes généralisés à des déplacements de tailles inconnues ne sont actuellement pas transférables sur carte graphique.

En revanche, la parallélisation des algorithmes sur des processeurs à cœurs multiples est possible. Nous donnons ici les indications permettant cette parallélisation pour les déplacements de particules et pour le cas des déplacements d'arêtes.

5.4.1 Particules

Chaque déplacement de particule est indépendant, il est donc possible de faire progresser simultanément l'ensemble des particules. Seule la phase de calcul global au niveau du système physique nécessite une synchronisation pour le cas de modèles tel que le *shape-matching*. La considération d'un système masse-ressort permet une limitation de cette synchronisation par sous-ensemble de particules : deux particules pour les ressorts de distance, trois particules pour les ressorts angulaires.

5.4.2 Arêtes

Comme chaque déplacement d'arête est considéré sommet par sommet, il est possible de paralléliser l'exécution de cet algorithme. A la différence du déplacement des particules, le déplacement de deux particules voisines par une arête peut causer des problèmes d'accès mémoire concurrent, c'est pourquoi il est nécessaire de définir les ensembles de particules autorisant une exécution simultanée des algorithmes de déplacement. Comme le montre la figure 5.8, la

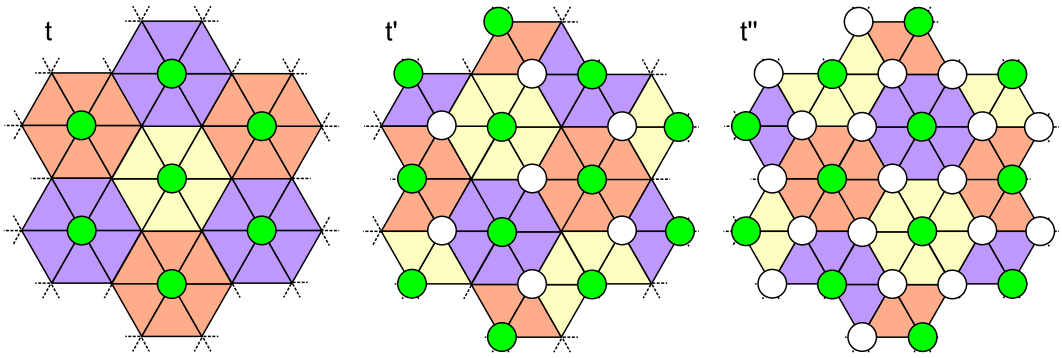

Figure 5.8 : Regroupement de particules pour la détection de collision sur les arêtes. A gauche : les 7 particules en vert peuvent être traitées simultanément. Au milieu, les 12 particules vertes peuvent être traitées. A droite : les 12 particules vertes peuvent être traitées.

parallélisation consiste à trouver des ensembles de sommets ayant des arêtes indépendantes et à réitérer jusqu'à ce que l'ensemble des particules soit déplacé. Bien que le problème du regroupement soit loin d'être trivial, cette méthode peut permettre l'accélération de l'exécution en répartissant les tâches sur divers processeurs.

5.5 Conclusion

Comme nous l'avons vu dans ce chapitre, l'intégration d'un système physique dans le système de détection de collision est assez directe. Les systèmes physiques présentés sont adaptés à des modèles d'objets déformables. Le fait d'utiliser les algorithmes de déplacements de particules ou d'arêtes est indépendant de l'intégration : si aucune information n'est donnée quant à la collision d'arête, le système physique réagit uniquement sur les informations données par les particules. On peut ainsi, décider d'ignorer des détections de collisions au niveau d'arêtes et permettre des approximations au niveau des particules lorsque le temps de calcul donné par le cadre de la simulation est borné.

Dans le cadre de simulation chirurgicale, le cas des déformations que nous avons présenté peut être considéré dans le sens opposé. Si l'objet en déplacement est rigide, il s'agit alors de déplacer l'ensemble des obstacles rencontrés selon la position à atteindre définie par l'objet rigide. Cela revient à identifier les points d'impacts de l'objet rigide avec l'environnement déformable et de pousser la zone de l'environnement correspondante au point d'impact jusqu'à la position à atteindre de l'objet rigide.

L'architecture actuelle des ordinateurs tend vers la parallélisation de tâches de part l'utilisation de cœurs multiples ou de l'utilisation du GPU. Nos algorithmes permettent la parallélisation sur de multiples cœurs. La question de l'utilisation des cartes graphiques pour l'accélération de l'exécution peut se résoudre de deux façons. Il est possible que la quantité de mémoire des cartes graphiques augmente de manière suffisante pour pouvoir charger directement l'ensemble des informations concernant l'environnement et les objets naviguant directement. Il est également envisageable de ne charger à la volée que l'ensemble des cellules, et potentiellement les cellules voisines, contenant des particules suivant les changements de cellules. Une étude concernant les temps de transfert est à envisager afin de valider cette approche.

Chapitre 6

Analyse théorique et pratique

Sommaire

Ce chapitre présente l'analyse des différents algorithmes proposés. Dans un premier temps, nous effectuons une analyse théorique de ces algorithmes. Une fois cette analyse fournie, nous la complétons par un ensemble de cas pratiques pour lesquels diverses mesures sont effectuées.

6.1 Introduction

Afin de valider les algorithmes que nous avons présentés, nous effectuons ici une analyse théorique de leur complexité. Cette analyse exhibe l'ensemble des critères qui influencent leurs temps d'exécution. Cette étude montre le faible coût du maintien des informations et du processus de détection de collision.

Afin de conforter le bien fondé de la complexité théorique, nous effectuons également un ensemble de tests sur des cas pratiques donnant ainsi lieu à une étude statistique et à une analyse des performances.

6.2 Complexité théorique

Dans cette section, une analyse de la complexité théorique est fournie. L'analyse est effectuée afin d'indiquer les critères à considérer pour estimer le coût calculatoire d'un déplacement unitaire. La définition de ces nombreux critères est nécessaire afin de fournir une quantification de la complexité de nos méthodes de détection de collision. Il est en effet inévitable de définir un cadre d'utilisation de nos algorithmes pour pouvoir les évaluer.

La complexité d'un déplacement est évaluée selon le nombre de tests d'orientation point/droite ou point/plan nécessaires afin de vérifier s'il y a, ou non, collision.

Le nombre de tests est fortement dépendant de la taille et de la direction du déplacement. Il est également lié au *stabbing number* [HKM95] de l'environnement –le nombre maximum de cellules traversées avant de rencontrer un obstacle– et à la taille de ses cellules.

Sans prendre en considération l'animation, la complexité d'un déplacement est en $O(n)$ où n est le nombre de faces formant la scène. En effet, un déplacement peut, dans des cas dégénérés, traverser l'ensemble des faces de la scène durant un pas de temps. Les schémas d'intégration standards, en simulation physique, fixent implicitement une limite maximum à la taille d'un déplacement afin d'obtenir de bonnes évaluations numériques. C'est pourquoi, le nombre de faces traversées peut être considéré comme borné. En restant dans le cas général, nous pouvons donc supposer qu'un déplacement ne couvre pas une distance plus grande que la dimension moyenne d'une cellule du maillage. En pratique, ces déplacements sont beaucoup plus faibles (de plusieurs ordres de grandeur). Cette précondition suppose également que la décomposition convexe de la scène est régulière, c'est-à-dire qu'il y a une faible différence entre la plus petite et la plus grande des cellules.

Nous étudions la complexité des algorithmes dans le cadre d'un environnement statique, puis d'un environnement déformable. Cette étude nous permet une première comparaison théorique avec un modèle hiérarchique classique.

6.2.1 Complexité en environnement statique

Nous abordons la complexité pour le déplacement d'une particule, avant de traiter le cas du déplacement d'arête. Cette étude est effectuée uniquement pour des environnements statiques.

Déplacement de particules

En nous basant sur les hypothèses exprimées ci-dessus, une particule, lors d'un pas de temps, ne peut traverser que : les faces, arêtes et sommets bornant un volume ; les volumes, arêtes et sommets bornant une face ; les volumes, faces et sommets incidents à une arête ; ou encore les volumes, faces et arêtes incidents à un sommet.

Si nous notons $deg(i_t)$ le degré de la cellule du brin i_t visé par une particule, alors le nombre de tests d'orientations nécessaires pour trouver la prédiction suivante est en $O(deg(i_t))$. En d'autres mots, le nombre de brins visités, c'est à dire le nombre de triangles ou de tétraèdres de prédiction testés, est proportionnel au nombre de cellules adjacentes à la cellule courante.

En considérant un maillage M en son entier, ce nombre est borné par le degré maximum de ces cellules, que nous notons $maxDegree(M)$.

Par exemple, dans le plan, pour une scène composée uniquement de cellules quadrilatérales, chaque cellule comporte 4 brins. Le degré des faces est donc de 4. Avec un maximum de 4 quadrilatères incident à un sommet, le degré des sommets est également de 4. De la même manière, avec un maximum de 2 quadrilatères au niveau d'une arête, le degré des arêtes est de 2. Dans ce cas $maxDegree(M)$ vaut donc 4.

Pour une scène dans l'espace, composée uniquement de cellules hexaédriques, chaque cellule comporte 6 faces composées de 4 brins. Le degré des volumes est donc de 24. Avec un maximum de 8 hexaèdres incidents à un sommet (avec 3 faces incidentes au sommet pour chaque hexaèdre), le degré des sommets est également de 24. Avec un maximum de 4 hexaèdres au niveau d'une arête (avec 2 faces incidentes pour chaque hexaèdre), le degré des arêtes est de 8. Dans ce cas $maxDegree(M)$ vaut donc 24.

La phase d'orientation des algorithmes composant les tests de collisions se base sur des rotations strictes (uniquement en sens horaire ou trigonométrique). De part ce fait, on peut aisément réduire cette borne à $maxDegree(M)/2$ en nous basant sur l'hypothèse de la régularité des cellules.

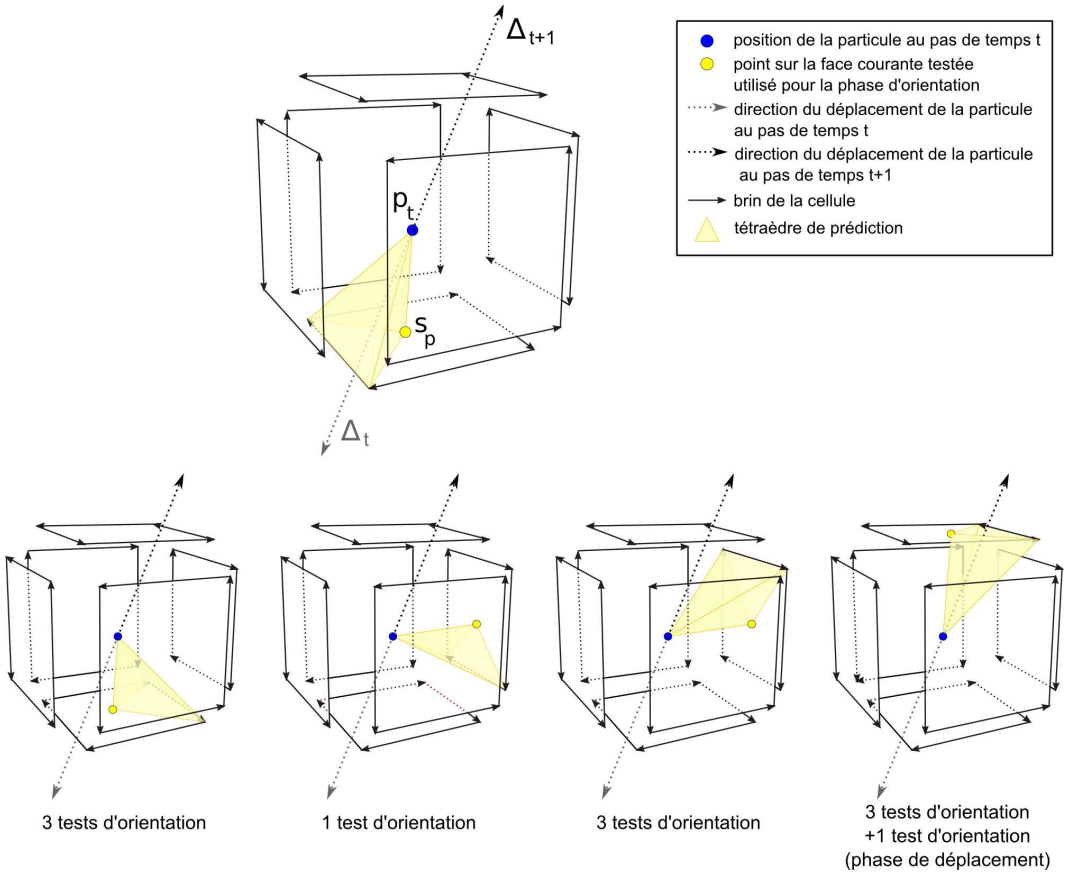

Figure 6.1 : Complexité maximale pour le cas d'un déplacement dans un volume. La particule p_t se dirigeait selon la demi-droite Δ_t, au pas de temps suivant elle prend la direction opposée Δ_{t+1}. Il s'agit du cas où la prédiction au pas de temps précédent est la plus éloignée, en terme de nombre de tests d'orientation, de la nouvelle prédiction à obtenir.

La figure 6.1 illustre cette réduction de borne au niveau d'un volume. La complexité maximale d'un déplacement survient lorsque la nouvelle direction du déplacement est dans le sens opposé de celle prise précédemment. Sur le cas d'un hexaèdre, cela revient à effectuer 11 tests d'orientation.

Si la taille maximale d'un déplacement est connue à l'avance, l'ensemble des cellules atteignables est borné : par un disque en dimension 2 ou par une sphère en dimension 3. L'environnement étant fixe, le nombre maximum *maxCell* de cellules, pouvant être contenues dans ce disque, est connu. Cela signifie qu'une particule ne peut traverser que *maxCell* cellules lors d'un déplacement. Si nous multiplions ce nombre par le nombre de tests d'orientation par cellule, on obtient une estimation de la complexité totale de nos algorithmes de déplacement.

La complexité totale de la détection de collision de p particules se déplaçant dans un maillage M de taille n et régulier est donc en $O(maxCell \times (maxDegree(M) \times p)$ et est indépendante de n. La complexité de l'algorithme peut donc être considérée en temps constant.

Déplacement d'arête quasi-continu

La complexité du déplacement d'arête de manière quasi-continue dépend uniquement de la complexité de l'algorithme de déplacement de particules. Nous considérons que les sommets ont des valences v et que la longueur des arêtes n'est pas supérieure à la taille moyenne d'une cellule de l'environnement, alors la complexité maximale à considérer est en $O((v+1) \times deg(i_t))$. En effet, un sommet ayant une valence de v nécessite $v+1$ lancers de particules : un lancer de particule au sommet, puis v lancers de particules le long des arêtes.

Technique d'élimination des arêtes arrières

Lorsque l'on considère l'optimisation proposée au niveau des polygones, dans le plan, et au niveau des polyèdres, dans l'espace, cette complexité doit être pondérée. Comme vu dans le chapitre 4, les particules associées à l'objet se déplaçant vers l'extérieur de l'objet sont les seules nécessitant d'effectuer des tests de détection de collision d'arêtes. Pour analyser l'apport de cette technique d'élimination, nous posons x le pourcentage d'arêtes pour lesquels les tests doivent être effectués.

En considérant cette optimisation, la complexité du déplacement d'arête de manière quasi-continue est ramenée à :

$$O(v \times n + n \times (v+1) \times deg(i_t) \times x + n \times deg(i_t) \times (1-x)) \qquad (6.1)$$

La valeur x correspond au nombre de particules dont on doit tester les collisions au niveau des arêtes. n est le nombre total de particules formant l'objet en déplacement. Ainsi il y a $n \times (v+1) \times deg(i_t) \times x$ tests à effectuer pour ces particules et uniquement $n \times deg(i_t) \times (1-x)$ tests à effectuer pour celle que l'on a pu éliminer.

Cette complexité comprend $v \times n$ tests servant à éliminer les cas où le déplacement se trouve vers l'objet à déplacer. Comme expliqué précédemment,

dans l'espace, le critère de concavité ou de convexité d'un angle au niveau d'un polyèdre peut être coûteux en termes de tests. Nous définissons donc de manière stricte le test consistant à vérifier que la particule va vers l'objet. Une particule est considérée comme se déplaçant vers l'intérieur d'un objet uniquement si l'ensemble des tests d'orientation avec les faces environnantes l'indique. Ainsi il y a au plus $v \times n$ tests afin de permettre de définir qu'une particule se dirige vers l'intérieur ou non.

La complexité sans élimination des particules arrière est en :

$$O(n \times (v+1) \times deg(i_t)) \tag{6.2}$$

Nous prenons un exemple dans l'espace afin de définir à partir de quelle valeur de x l'élimination des tests de collision d'arêtes arrières devient avantageux.

On considère, pour chaque particule, que le nombre de tests nécessaires pour détecter les collisions est minimal. C'est-à-dire, chaque déplacement testé requiert le nombre minimal de tests d'orientation. En dimension 3, cela revient à effectuer uniquement 4 tests d'orientation afin de vérifier l'inclusion au tétraèdre de prédiction. On suppose également que le nombre moyen d'arêtes autour d'un sommet est de 6.

L'élimination des particules est donc avantageuse lorsque :

$$\begin{aligned} v + (v+1) \times deg(i_t) \times x + deg(i_t) \times (1-x) &< (v+1) \times deg(i_t) \\ 6 + 28x + 4 - 4x &< 28 \\ x &< 0.75 \end{aligned}$$

Selon ces valeurs, si l'élimination permet de ne pas effectuer de tests de collisions d'arêtes sur 25% des particules, ce test est avantageux. Nous avons pris ici $deg(i_t) = 4$ ce qui est le cas le plus avantageux pour le déplacement de particules. Pour la collision d'arêtes, il est cependant rare que la prédiction de la particule corresponde directement avec la direction de chacune des arêtes environnantes.

Déplacement d'arête continue

En se basant sur les mêmes hypothèses qu'auparavant, la complexité théorique du déplacement d'arête de manière continue est également dépendante du degré de la cellule contenant les arêtes. Pour déplacer une arête, deux déplacements de particules sont nécessaires, lorsque ces déplacements sont totalement inclus dans une cellule de l'environnement, le coût de la détection de collision de manière continue est nul. Lorsque le bord d'une cellule est franchi, il est alors nécessaire de développer l'ensemble des brins intersectés. Ce développement a un coût limité et est proportionnel au degré des cellules qu'il franchit.

6.2.2 Complexité théorique en environnement déformable

Notre méthode, lors de déformations ou de changements de topologie, ne nécessite que des mises à jour locales au niveau des particules. Nous rappelons et exprimons le surcoût maximum de la complexité, pour les différents cas de déformations pris en compte dans le chapitre de déplacement de particules. Ce surcoût est, lui aussi, exprimé en fonction du nombre de tests d'orientation requis.

Modification topologique :
- Couture : coût nul ;
- Simplification :
  - les brins de la $(n-1)$-cellule sur le point d'être supprimés ne prennent part à aucune prédiction : coût nul ;
  - une particule vise un des brins à supprimer : coût maximum = modification du brin i_t par le brin $\phi_2(i_t) + deg(i_t)/2$.
- Découpage :
  - p et p_{t+1} sont de chaque coté de la cellule ajoutée : coût maximum = modification du brin i_t par un brin de la $(n-1)$-cellule $+deg(i_t)/2$;
  - p_t et p_{t+1} sont du même coté de la cellule ajoutée et la prédiction est toujours correcte : coût nul.
  - p_t et p_{t+1} sont du même coté de la cellule ajoutée et la prédiction est à l'opposé de celle-ci (i.e. appartient à la cellule adjacente) : coût maximum = modification du brin i_t par un brin de la $(n-1)$-cellule $+deg((n-1)-cellule)/2$.

Modification géométrique :
- Cas 1 : p_t reste dans la cellule courante : coût maximum = $deg(i_t)/2$;
- Cas 2 : p_t n'est plus dans la cellule précédemment visée : coût maximum = coût remplacement p_t par un point inclus dans la cellule $+deg(i_t)/2$.

6.2.3 Comparaison avec les méthodes hiérarchiques

Dans les approches classiques, chaque élément e (des particules, des arêtes ou des faces) est testé avec une hiérarchie de boîtes englobantes. Cette approche donne une complexité en $O(6.e.log(n))$. En effet, tester l'intersection avec une boîte englobante demande (habituellement) 6 tests d'orientation. De plus, un arbre contenant les volumes englobants d'un environnement de taille n nécessite en moyenne $log(n)$ requêtes pour tester l'appartenance à un volume. Cette dépendance à la taille complète de l'environnement devient pénalisante lorsque les environnements sont de tailles importantes.

Le fait de se baser sur la cohérence temporelle permet de diminuer cette complexité. La complexité passe alors d'une complexité logarithmique à une complexité amortie constante. Cet amortissement peut être obtenu en mémorisant l'ensemble des volumes englobants testés, ainsi seule une partie de la hiérarchie doit être testée.

Lors de déformations ou de modifications de l'environnement, les méthodes hiérarchiques nécessitent $m.log(n)$ opérations où m est le nombre de cellules déformées (c'est-à-dire le nombre de feuilles de l'arbre qui doivent être mises à jour dans un arbre de taille n). Des hypothèses fortes concernant les modifications doivent être mises en place afin de réduire ce surcoût.

C'est la complexité de cette mise à jour, dépendante de la taille de la scène, qui pénalise le plus les méthodes de type BVH, et c'est donc dans ce cas que notre approche s'avère plus efficace.

6.3 Analyse statistiques et performances

Nous effectuons, dans cette section, une analyse statistique et des comparaisons de performances sur différents cas d'école. Ces analyses nous servent à valider la complexité théorique exprimée précédemment. Nous traitons le cas du déplacement de particules avant d'aborder le déplacement d'arêtes. Pour le cas des particules, nous effectuons également une analyse statistique concernant les environnements déformables.

La complexité est plus élevée dans le cas de déplacements dans l'espace et permet la comparaison à des méthodes classiques de hiérarchie de volumes englobants. De plus, toutes les applications de simulations chirurgicales sont, à notre connaissance effectuées en dimension 3. Seule cette dimension est con-

sidérée pour l'ensemble des analyses pratiques.

Les performances des algorithmes proposés ont été mesurées sur un PC utilisant un processeur 2.4GHz Intel Core2 et 2GB de mémoire. Seul un cœur du processeur est utilisé pour l'ensemble des simulations.

6.3.1 Déplacement de particules

Nous effectuons dans un premier temps une analyse statistique concernant le nombre de mises à jour nécessaires dans l'algorithme de prédiction du déplacement de particules, c'est à dire le nombre de brins visités. Une fois cette analyse effectuée, nous comparons les performances de notre méthode avec une méthode hiérarchique.

Analyse statistique des prédictions de collisions

Méthode : nous simulons un flot de particules en déplacement soumis à une force de gravitation dans un environnement subdivisé. L'environnement pivote autour d'un axe afin de conserver les particules en mouvement. Pour cette simulation nous considérons un ensemble de 5x5x28 hexaèdres. La figure 6.2 donne une illustration du maillage utilisé. La simulation consiste à laisser rebondir librement 1000 particules durant 1000 pas de temps. Cela nous donne 1000000 de tests de collision dans des configurations correspondant à des données réelles.

Pour cette étude statistique, nous comptons le nombre de changements nécessaires à chaque particule au niveau du système de prédiction, c'est-à-dire le nombre de tests d'orientation nécessaires afin de mettre à jour les informations. Deux cas sont considérés : un premier cas avec un environnement statique et un deuxième subissant de fortes déformations au cours de la simulation. Le schéma 6.3 représente la cinématique de cette étude pour le cas déformable.

Nous effectuons également des mesures sur un maillage statique de réseau vasculaire composé d'un ensemble de tétraèdres. Pour cette scène nous laissons également tomber 1000 particules durant 1000 pas de temps et mesurons le nombre de changements nécessaires à chaque particule pour maintenir les états du système de prédiction.

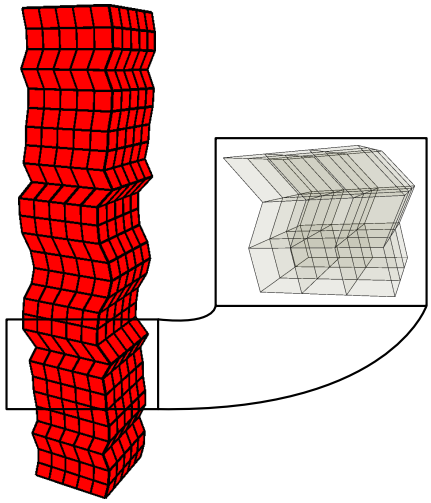

Figure 6.2 : Structure du maillage déformable utilisé pour l'analyse.

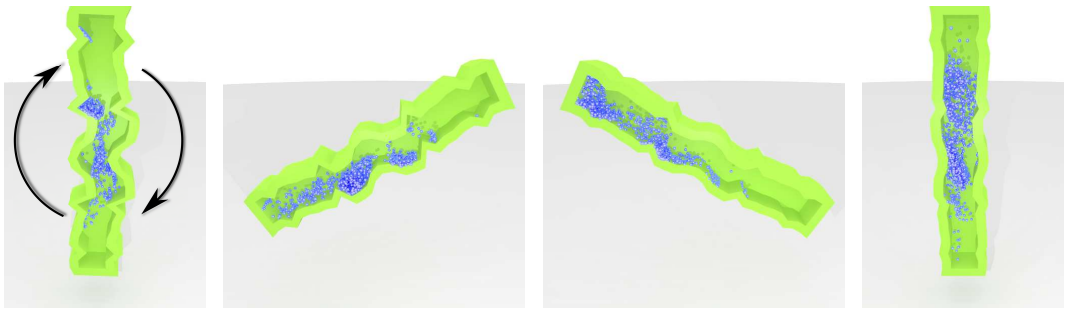

Figure 6.3 : 1000 particules dans un environnement déformable composé de 5x5x28 hexaèdres ; l'environnement est pivoté autour d'un point central selon un axe afin de garder les particules en déplacement quasiment constant.

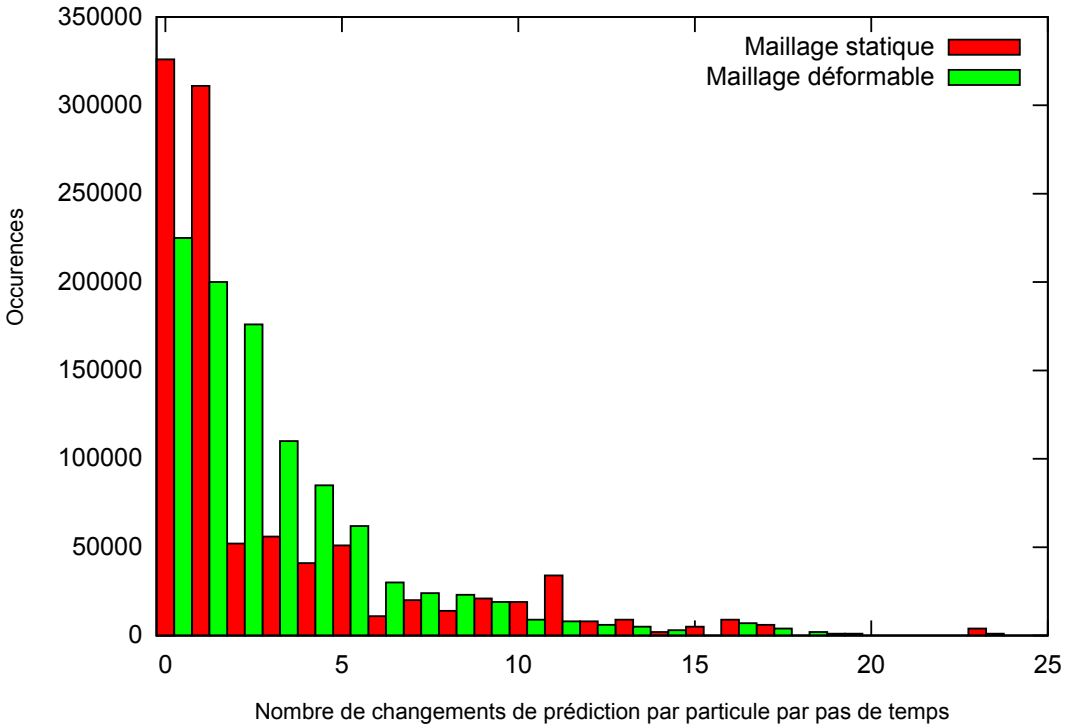

Figure 6.4 : Nombre de rotations sur une face et nombre de changements de faces lors de la phase de prédiction pour 1000 particules soumises à la gravité pendant 1000 pas de temps.

Analyse : comme le montre le graphique 6.4, il n'est pas nécessaire dans la plupart des cas de changer de prédiction. Cela montre également que la complexité maximum ne survient qu'occasionnellement. En pratique, le nombre de tests d'orientation nécessaires est proche de la complexité minimale.

Sur 1000000 d'itérations, les résultats expérimentaux donnent, pour le cas statique, une moyenne de 6,75 tests d'orientation par particule. Dans 33% des cas, la prédiction reste valide et ne nécessite donc que 4 tests afin d'être validée (un test pour chaque face formant le tétraèdre de prédiction). Dans 31% des cas, la mise à jour de la prédiction n'implique qu'un seul décalage à une arête adjacente et donc uniquement un test supplémentaire. Les autres cas, nécessitant deux ou plus de modifications, surviennent rarement.

Le graphique 6.6 évalue le même critère dans le maillage montré en figure 6.5. Le nombre moyen de tests d'orientation est de 4,27, ce qui est significativement inférieur à celui mesuré ci-dessus. Cela montre que la complexité globale de la scène, qui est cette fois-ci composée de plusieurs milliers de tétraèdres, n'implique pas de perte d'efficacité de l'algorithme. Au contraire, comparée

aux tests précédents, la complexité moyenne est plus faible. La complexité en $deg(i_t)$ ne survient que lorsqu'un changement de cellule ou un changement de direction dû à une collision ou à l'interaction intervient. Cependant plus les volumes sont grands, moins le premier cas survient, le deuxième cas n'est par contre pas influencé par la taille des volumes et est strictement dépendant du modèle physique et/ou du déplacement indiqué par l'utilisateur. Il est également notable que le degré des volumes considérés agit sur l'espace mémoire nécessaire : plus les volumes ont un degré important, plus ils ont un coût mémoire faible.

Nous revenons sur le graphique 6.4 concernant la gestion des déformations de l'environnement. Le fait de déformer les cellules du maillage n'augmente pas de manière conséquente le nombre de tests d'orientation nécessaire, qui est d'approximativement 6,85. Cela montre que le nombre de tests supplémentaires nécessaires à la mise à jour de la prédiction lors d'une déformation reste faible. Dans cet exemple, le surcoût est de 0,1 test par particule. Cela signifie que seule 1 particule sur 10 nécessite une mise à jour supplémentaire de sa prédiction locale.

Figure 6.5 : 1000 particules dans un réseau vasculaire représenté par une subdivision volumique de 12k tétraèdres.

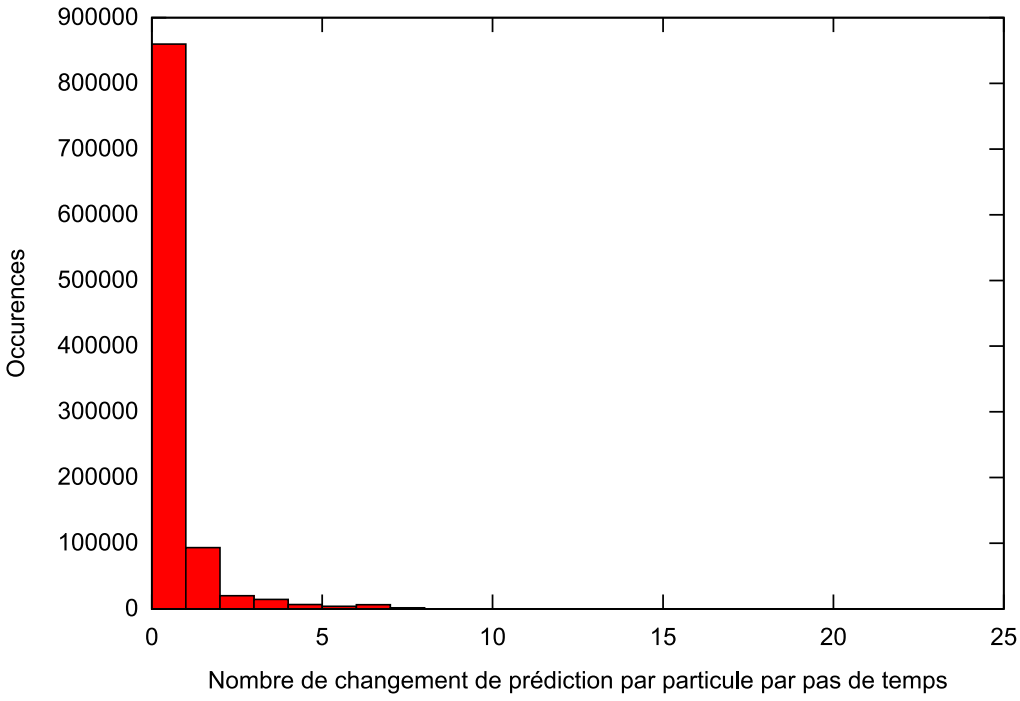

Figure 6.6 : Nombre de rotations sur une face et nombre de changements de faces lors de la phase de prédiction pour 1000 particules soumises à la gravité pendant 1000 pas de temps.

Test de performance

Nous effectuons ici des tests visant à mesurer l'efficacité de notre méthode comparée à une méthode classique de type BVH. Une étude complète et générale des temps d'exécutions est difficile, nous focalisons ici notre étude sur le coût impliqué par la gestion de scènes complexes.

Méthode : le test de performance compare notre algorithme avec une hiérarchie de AABB. L'implantation de la hiérarchie utilisée provient de la librairie de simulation physique Bullet [Bul]. Nous simulons le déplacement de 1000 particules pendant 1000 pas de temps dans une scène dont nous faisons croître la complexité. Toutes les particules sont initialisées à une position aléatoire et chutent sous l'effet de la gravité. L'environnement est constitué de n hexaèdres volumiques, nous modifions le nombre de ces derniers afin d'augmenter la complexité comme illustré par la figure 6.7.

La hiérarchie de AABB est utilisée pour limiter le nombre de tests d'intersections segments/triangle sur cette même scène. Les extrémités de ces segments correspondent au point de départ et d'arrivée d'une particule en déplacement. Seuls les temps concernant la détection de collision sont pris en compte dans les deux cas.

Analyse : la figure 6.8 donne les résultats numériques de cette expérimentation. Notre méthode est moins coûteuse en termes de temps pour les tests de détection de collision. La différence se trouve dans le fait que les AABB utilisés n'exploitent pas la cohérence temporelle. Un autre point significatif à observer concerne le profil des deux courbes. Le coût de notre méthode est maintenu constant malgré l'augmentation de la complexité de la scène, alors que dans les mêmes conditions l'utilisation d'une hiérarchie de AABB subit une augmentation en $log(n)$. Cette augmentation est directement liée à la structure utilisée. Nous n'effectuons pas de comparaison avec le cas déformable, le surcoût du maintien du BVH serait encore plus important.

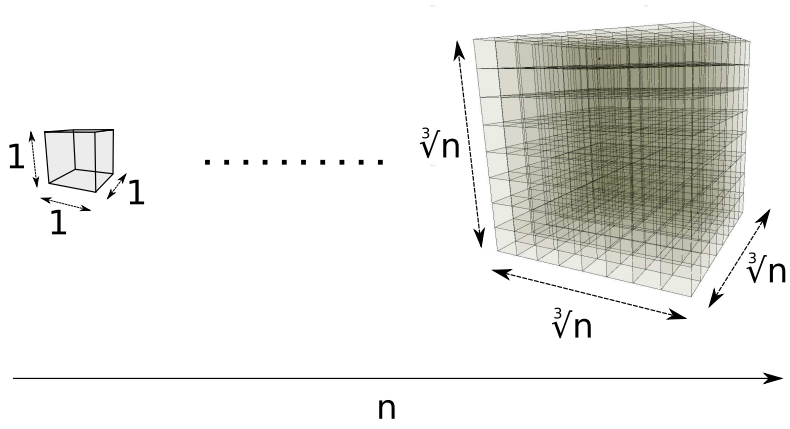

Figure 6.7 : Construction d'un maillage de taille n pour le test de performance.

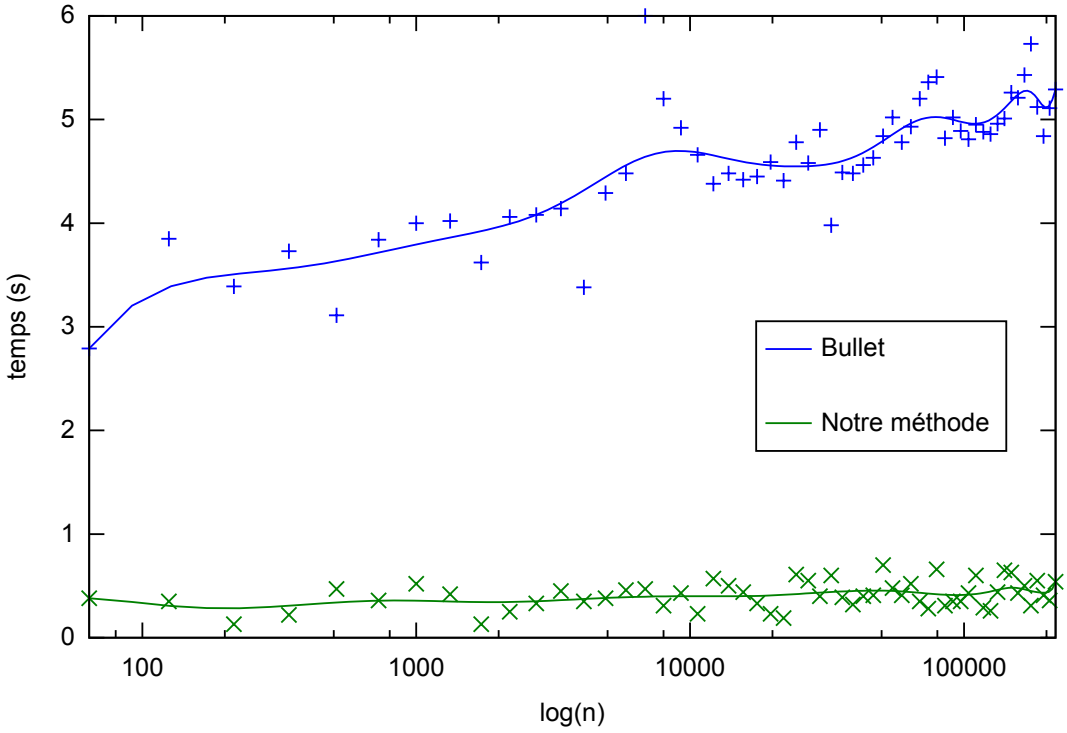

Figure 6.8 : Temps de calcul de la détection de collision en fonction de la complexité de la scène pour 1000 particules pendant 1000 pas de temps selon une échelle logarithmique.

6.3.2 Déplacement d'arête de manière quasi-continue

Nous effectuons une analyse du nombre de volumes testés par l'algorithme de déplacement quasi-continu d'arête. Une fois cette analyse effectuée, nous comparons les performances de notre méthode avec une méthode hiérarchique.

Analyse statistique du nombre de volume testés

Méthode : nous évaluons ici de manière statistique le nombre de volumes testés lors de l'utilisation de l'algorithme de déplacement d'arête effectué de manière quasi-continue. Nous utilisons un système masse-ressort pour simuler le déplacement d'un cathéter dans un réseau vasculaire (figure 6.9). Des ressorts de distances et d'angles sont utilisés afin d'assurer une certaine rigidité au cathéter. L'utilisation du système de détection de collision sur les arêtes rend possible l'intégration du calcul de moments lorsqu'une collision intervient au niveau de l'une d'entre elle. La physique du cathéter que nous avons implanté n'est pas utilisable pour une simulation chirurgicale et est mise en place ici uniquement dans le cadre de mesures d'essais. Afin d'obtenir une simulation plus réaliste, on peut se tourner vers des méthodes qui consistent à résoudre un système de contraintes [LCDN06] utilisant les informations de collisions retournées par nos algorithmes.

Au cours de cette simulation, nous déplaçons un cathéter à l'intérieur d'un réseau vasculaire tétraédrisé. La longueur du cathéter, et donc des arêtes le formant, augmente lors de la simulation. Ainsi le nombre de tests géométriques nécessaires augmente avec la taille des arêtes, comme le précise notre analyse de complexité théorique précédente.

Analyse : le graphique 6.10 indique le nombre de tétraèdres testés pour la détection de collision en fonction de la longueur des arêtes. Comme la méthode consiste à lancer des particules le long des arêtes pour détecter les collisions, le nombre de tétraèdres testés augmente avec la longueur des arêtes. On voit apparaître des pics au niveau du graphique dus à des relaxations de contraintes du système (le cathéter glisse soudainement à un endroit où il avait une collision bloquante précédemment), ce qui implique de plus grands déplacements. Comme on peut le voir sur le graphique, le nombre de tétraèdres testés en comparaison du nombre de tétraèdres formant la scène reste faible : moins de 3% de l'ensemble des tétraèdres sont testés.

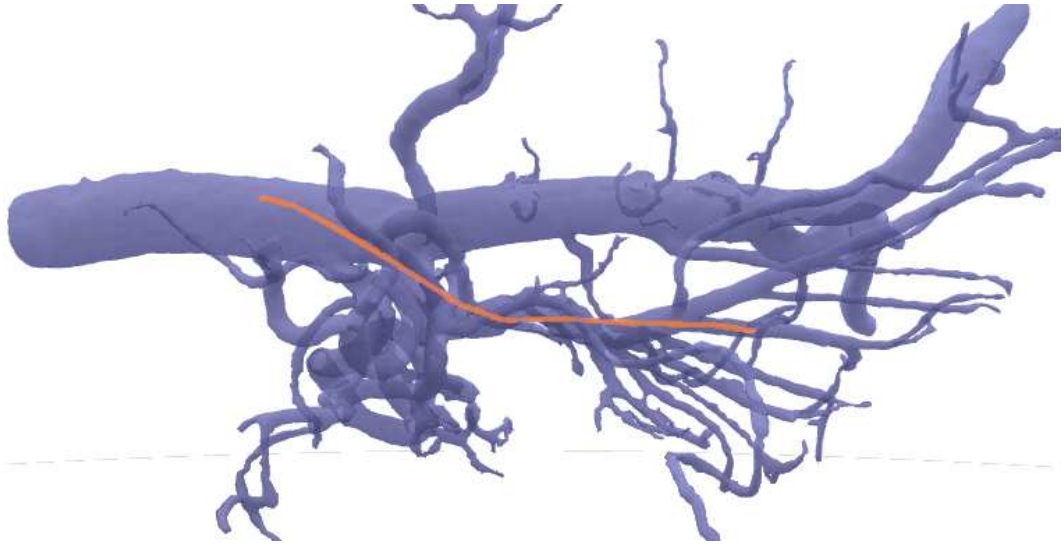

Figure 6.9 : Un cathéter en déplacement dans un réseau vasculaire. Le cathéter est échantillonné par un ensemble de 20 particules à l'intérieur d'un maillage volumétrique composé de 200 000 tétraèdres.

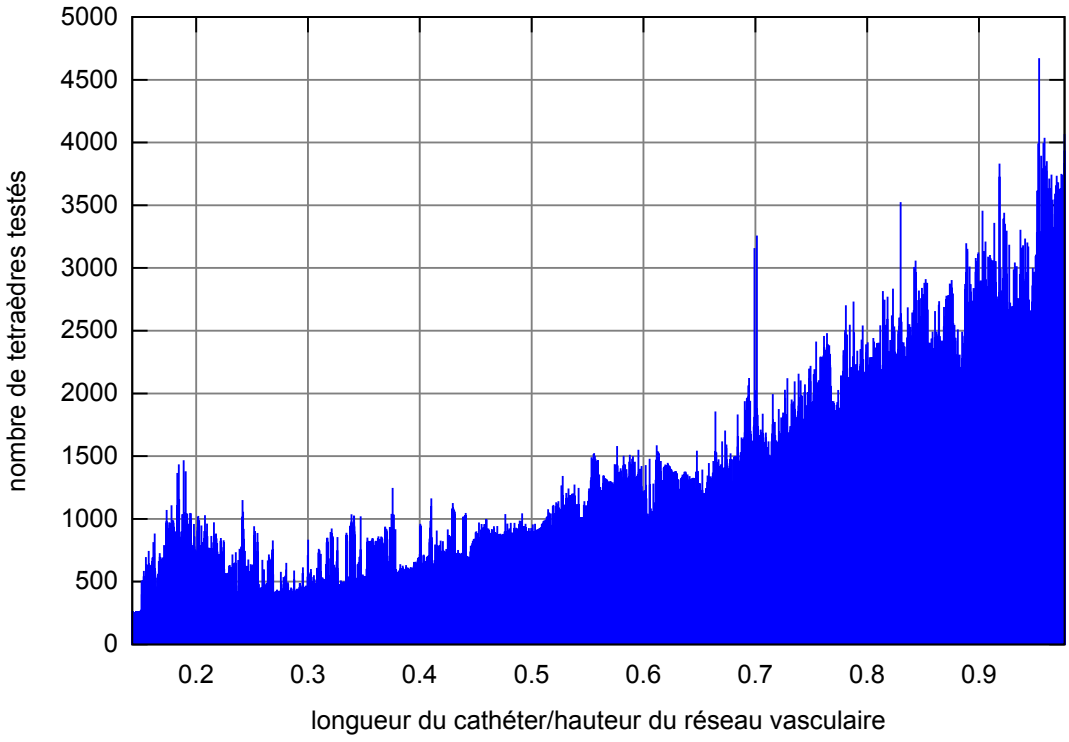

Figure 6.10 : Nombre de tétraèdres testés en fonction de la taille du cathéter.

Test de performance

Méthode : comme précédemment, le test de performance consiste à comparer notre algorithme avec une hiérarchie de AABB de la librairie de simulation physique Bullet. Nous simulons le déplacement de 225 cubes déformables échantillonnés par 8 particules pendant 1000 pas de temps dans une scène dont nous faisons croître la complexité. Chaque cube déformable est simulé selon le modèle du *shape-matching* avec maillage statique. L'ensemble des cubes est initialisé avec une impulsion aléatoire. L'environnement est composé de n hexaèdres volumiques dont certains sont des obstacles, afin que des collisions d'arêtes aient lieu, nous modifions le nombre d'hexaèdres composant la scène afin d'augmenter la complexité comme le montre la figure 6.11. Seule la détection de collision est prise en compte pour la mesure des temps de calcul.

Deux tests ont été effectués avec des AABB :
- un test considère des intersections segments/triangles (comme celui effectué pour les particules) dans le même esprit que nos tests de collision effectués sur les arêtes ;
- un test considère des intersections triangles/triangles : un triangle correspond au balayage de l'arête entre deux pas de temps. Ces tests exploitent une optimisation de Bullet qui utilise dans ce cas la librairie GIMPACT pour les tests d'intersections triangle/triangle.

Nous avons également effectué des tests concernant le changement de topologie en ajoutant des obstacles dans la scène de manière aléatoire. Ces changements de topologie n'influencent cependant pas fortement les temps de calcul et nécessiterait la mise à jour du BVH, nous n'analysons donc pas plus les performances de l'algorithme dans ce cadre.

Analyse : le graphique 6.12 donne les résultats numériques de l'expérience. Notre méthode surpasse les autres par un facteur de 3. Les tests segments/triangle subissent une augmentation logarithmique en fonction de la complexité de l'environnement (dû à la structure hiérarchique de volume englobant de type AABB). La droite que l'on voit se démarquer, en considérant uniquement les points de temps plus faibles sur l'échelle semi-logarithmique au niveau des mesures, correspond à l'augmentation de la profondeur de l'arbre hiérarchique. Les tests triangle/triangle et notre méthode conservent une complexité constante indépendante de la complexité de l'environnement. Nous supposons que les tests triangles/triangles permettent de diminuer le coût des tests à effectuer en regroupant des ensembles de tests d'orientation. Notre méthode reste constante malgré l'augmentation de la complexité de part l'utilisation du mécanisme de prédiction et de la cohérence temporelle.

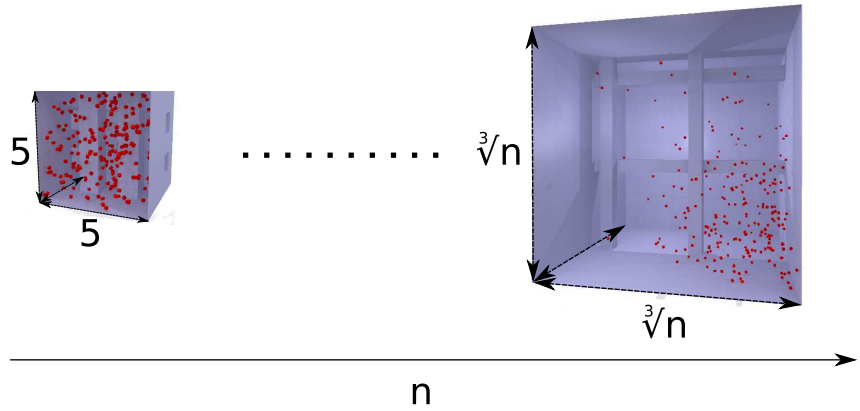

Figure 6.11 : Construction d'un environnement de taille n contenant 225 hexaèdres déformables pour le test de performance.

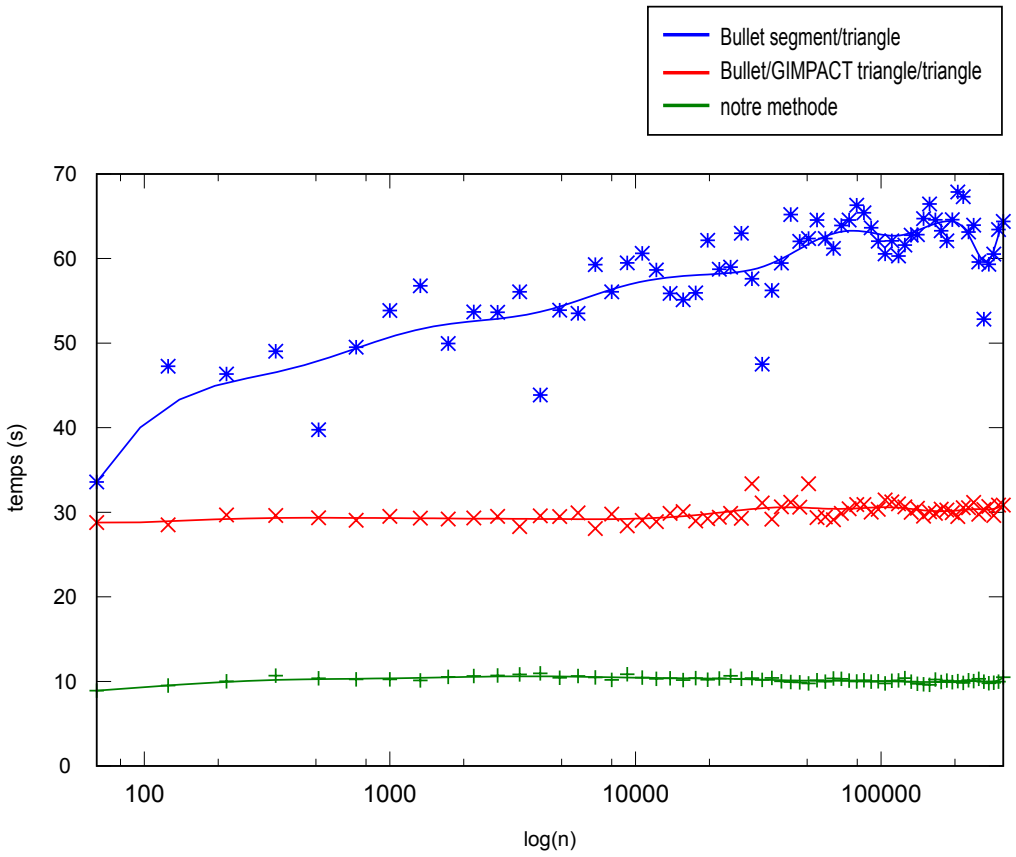

Figure 6.12 : Temps de calcul de la détection de collision en fonction de la complexité n de l'environnement pour 225 hexaèdres déformables pendant 1000 pas de temps ; notre méthode requiert environ 10ms par pas de temps ce qui représente approximativement 5μs par particule.

6.3.3 Élimination de tests d'arêtes

Nous effectuons des analyses concernant l'élimination des tests d'arêtes pour les arêtes totalement incluses et pour l'élimination des arêtes arrières. Ces études visent à observer sur un exemple le nombre de tests éliminés.

Élimination des particules selon l'environnement

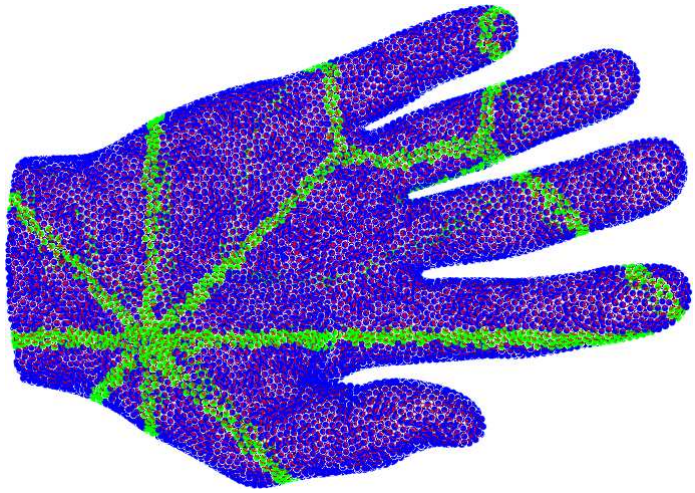

Figure 6.13 : Élimination des tests d'arêtes pour les particules selon l'environnement : seules les arêtes dont le déplacement d'une particule entraîne un franchissement de cellule ou dont la position précédente coupait une cellule sont testées. Les particules bleues sont les particules dont les arêtes sont totalement incluses dans les cellules, les particules vertes sont les particules à tester pour une éventuelle collision d'arête.

Méthode : pour l'ensemble de ces tests nous utilisons un maillage de main échantillonné avec environ 12000 particules. Lorsqu'une particule ne franchit pas de cellules et que l'arête au pas de temps précédent était totalement incluse dans une même cellule, le test d'arête est ignoré. La figure 6.13 illustre ce maillage lorsque celui-ci est inclus dans un environnement composé de pyramides. Les particules bleues n'effectuent pas de tests de collision d'arêtes car leurs déplacements sont totalement inclus dans une même cellule. Les particules en vert sont celles dont l'arête à la position précédente ou le déplacement de la particule implique un franchissement de cellules.

Ce maillage est lâché dans un environnement simple composé uniquement de pyramides. La figure 6.14 représente une partie de la cinématique de la simulation.

Nous mesurons le pourcentage de tests d'arêtes économisé ainsi que le gain de temps apporté par cette élimination. Deux mesures de temps sont effectuées : une mesure sans élimination et une mesure avec élimination des particules selon l'environnement. Nous regroupons les temps de calcul par tranche de 50 pas de temps afin d'amortir les effets d'arrondis. L'ensemble des mesures effectuées ne comprend que la phase de détection de collision.

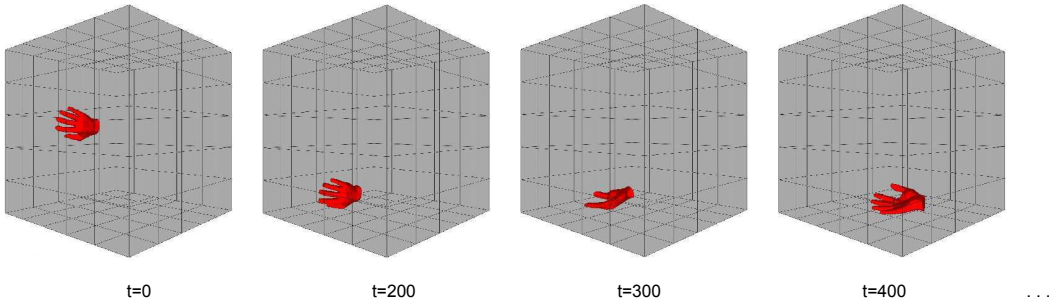

Figure 6.14 : Animation utilisée pour les différentes mesures.

Analyse : le graphique 6.15 représente l'ensemble des particules qui ont pu être éliminées par le test proposé. On constate que le pourcentage d'élimination de tests d'arêtes avoisine les 75%. A l'initialisation, un nombre important de particules a pu être éliminé des tests. Cela s'explique par le fait que les particules ont des vitesses faibles et franchissent donc peu de cellules. A partir du pas de temps 1200, le pourcentage d'élimination se stabilise. L'objet repose sur le plan inférieur de l'environnement et peu de changements de configurations ont lieu.

Le graphique 6.16 illustre la mesure de temps. On note sur les temps mesurés une amélioration d'environ 55% des temps de calcul avec l'élimination des particules selon l'environnement.

Cette optimisation est dépendante de la taille des cellules formant l'environnement. Le cas où aucun test ne peut être supprimé n'est pas coûteux en termes de temps de calcul mais uniquement d'un point de vue de la mémoire. Il est en effet nécessaire pour cette méthode de mémoriser les arêtes intersectées par une cellule de l'environnement. Le pourcentage d'arêtes éliminables est également dépendant de la vitesse des particules : plus les particules se déplacent rapidement, proportionnellement à la taille des cellules, plus il est possible qu'elles franchissent le bord d'une cellule lors d'un déplacement.

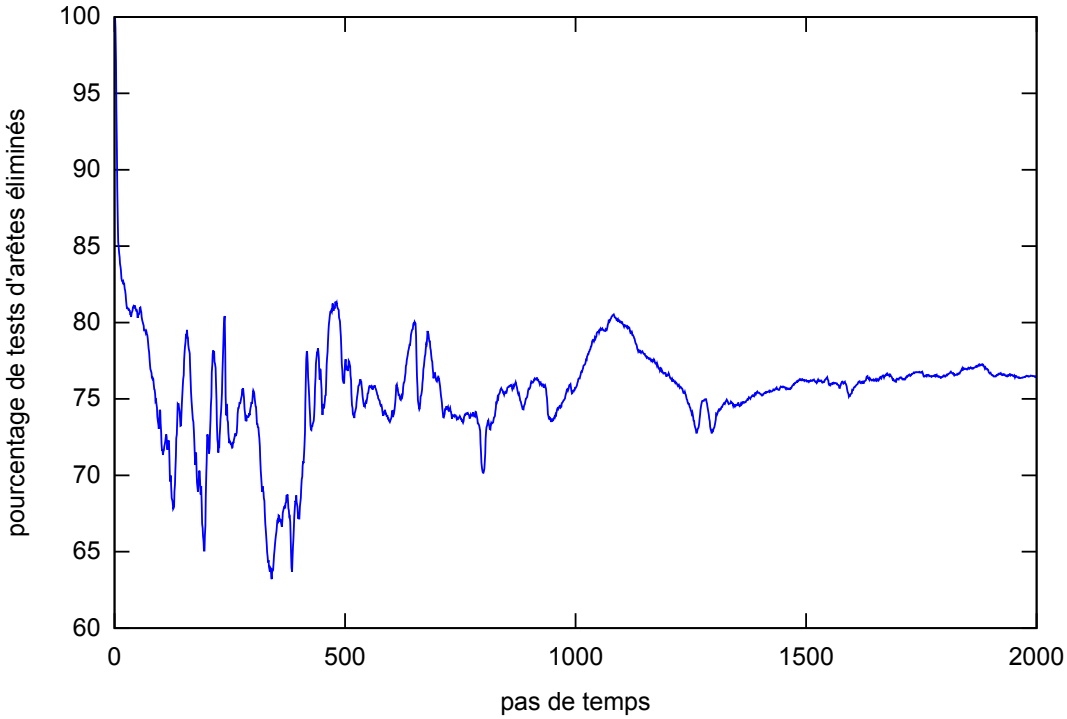

Figure 6.15 : Pourcentage de particules n'effectuant pas de tests de collisions d'arêtes en fonction du temps pour une main échantillonnée avec 11898 particules dans un environnement composé de pyramides.

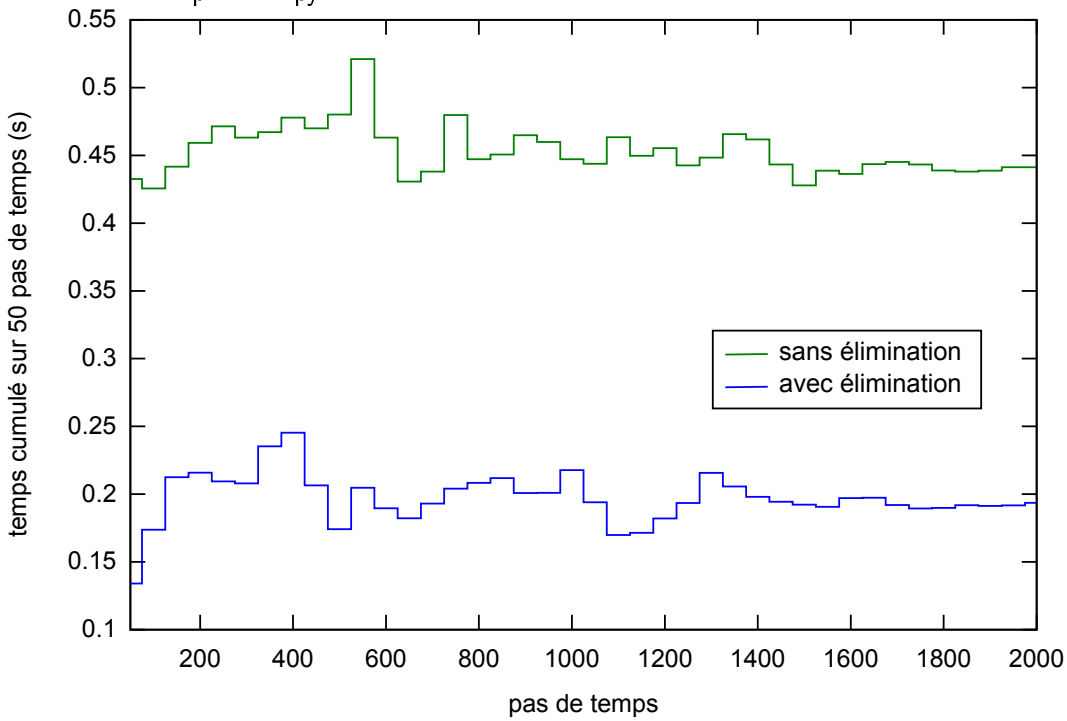

Figure 6.16 : Temps cumulé sur 50 pas de temps de l'ensemble des tests de détection de collision sans éliminations de tests d'arêtes et avec élimination des tests d'arêtes selon l'environnement.

Élimination des arêtes arrières

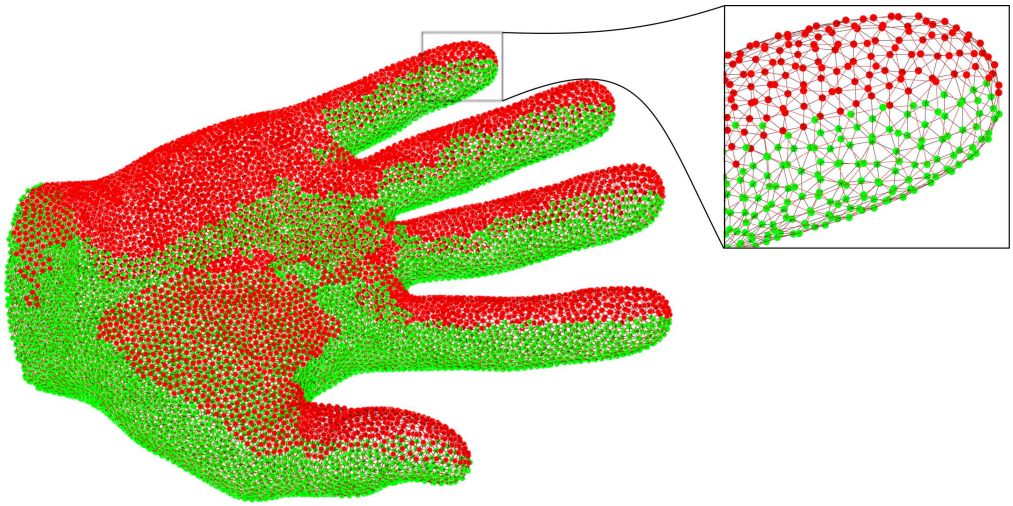

Figure 6.17 : Élimination des tests d'arêtes pour les particules se dirigeant vers l'objet : les particules en vert effectuent les tests d'arêtes, les particules rouge n'effectuent que le suivi des particules.

Méthode : pour l'ensemble de ces tests nous utilisons le même maillage que précédemment échantillonné avec environ 12000 particules. Lorsque le déplacement indiqué par le modèle physique et l'ensemble des forces se dirige vers l'intérieur de l'objet, les collisions d'arêtes ne sont pas effectuées. La figure 6.17 illustre ce maillage lorsque celui-ci tombe à la verticale. L'ensemble des particules dans la direction de la gravité sont testées. L'ensemble des particules rouges a pu être éliminé des tests de collisions d'arêtes.

La cinématique de la simulation utilisée pour l'analyse quantitative est similaire à celle utilisée pour l'élimination des arêtes arrières.

Analyse : le graphique 6.18 représente l'ensemble des particules qui ont pu être éliminées par le test proposé. Pendant la première phase de chute de l'objet, on voit que ce pourcentage reste stable. L'objet est alors uniquement en translation et ne subit pas de variation. Lors de la suite de la simulation, l'objet se heurte sur une face ce qui entraîne un enchaînement de rotations. Ces rotations font alors varier le pourcentage d'arêtes éliminables. Malgré les fortes variations observables, on constate que le pourcentage de particules reste entre 40% et 50%. Selon la complexité théorique ce pourcentage est suffisant pour une diminution du nombre de tests.

Afin de valider ce point, nous effectuons des mesures de temps sur cette même animation avec et sans élimination de particules de la même façon qu'auparavant.

Le graphique 6.19 illustre cette mesure. On note une amélioration d'environ 30% des temps de calcul.

Il est à noter que ce type d'optimisation, et donc les mesures effectuées ici, est intégralement dépendant du maillage et du déplacement de l'objet. Le pire des cas que l'on peut rencontrer est une rotation d'un objet tel que chacune des particules suit un mouvement tangent aux faces de son orbite. Dans un tel cas, non seulement toutes les arêtes doivent être testées, mais en plus chaque particule effectue des tests supplémentaires.

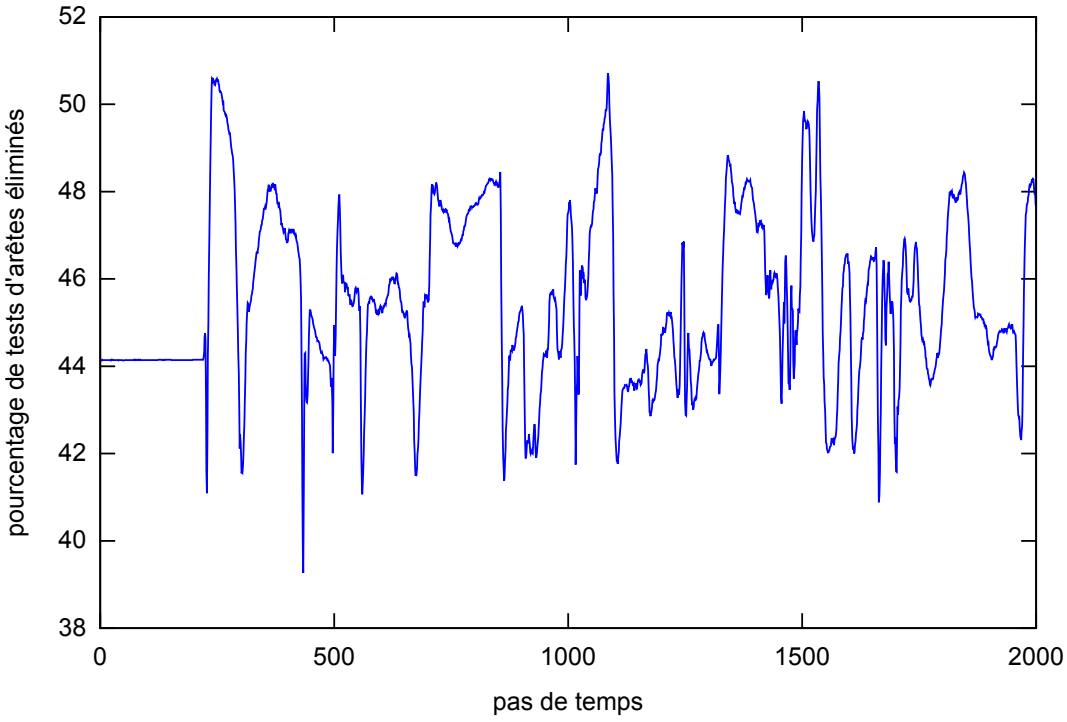

Figure 6.18 : Pourcentage de particules n'effectuant pas de tests de collisions d'arêtes en fonction du temps pour une main échantillonnée avec 11898 particules tombant dans un environnement composé de pyramides.

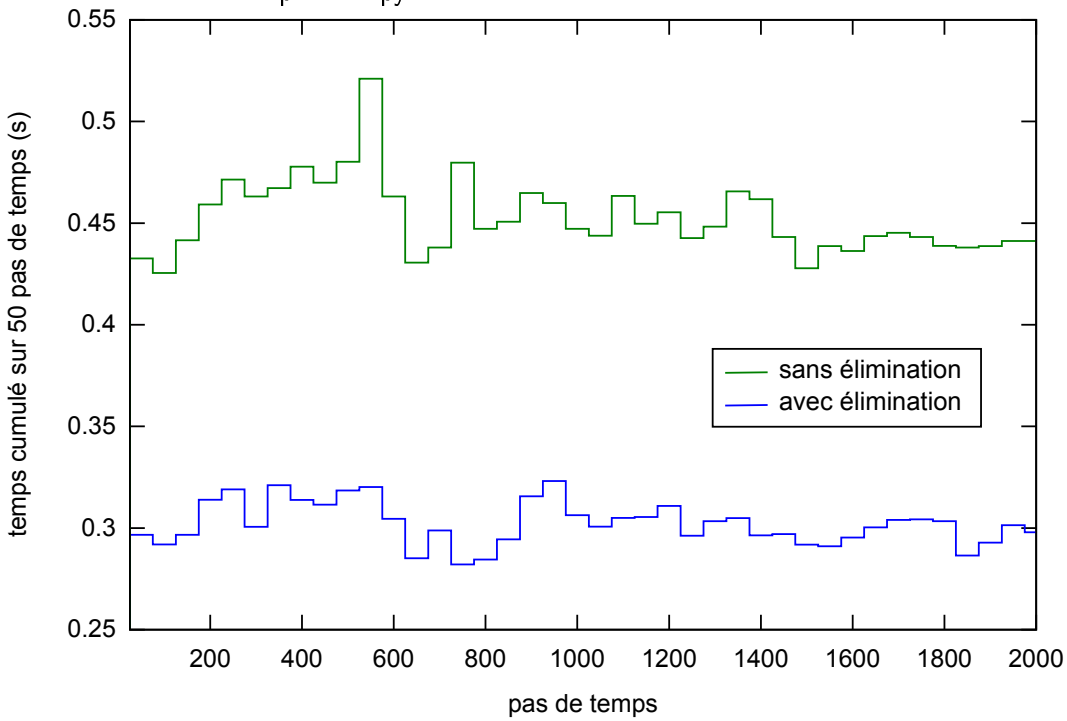

Figure 6.19 : Temps cumulé sur 50 pas de temps de l'ensemble des tests de détection de collision avec et sans élimination des arêtes arrières.

Méthode hybride

Méthode : dans une dernière analyse nous proposons de combiner les deux méthodes d'élimination sur la même scène. Parmi l'ensemble des particules nous identifions trois types de particules :

- les particules dont les arêtes sont totalement incluses dans une cellule de l'environnement. Ces arêtes ne nécessitent pas d'être testées pour la détection de collision ;
- les particules dont les arêtes se trouvent dans plusieurs cellules et qui se déplacent vers l'intérieur de l'objet. Ces arêtes ne doivent pas être testées pour la détection de collision, elles le seront lorsque le déplacement se fera vers l'extérieur de l'objet ;
- les particules dont les arêtes se trouvent dans plusieurs cellules et qui se déplacent vers l'extérieur de l'objet. Ces arêtes doivent être testées pour la détection de collision.

Analyse : le graphique 6.20 représente l'ensemble des particules qui ont pu être éliminées par le test proposé. On constate que le pourcentage d'élimination de tests d'arêtes avoisine les 86%.

Comme auparavant, nous validons le gain de temps par cette élimination en effectuant des mesures de temps sur cette même animation selon le même procédé. Le graphique 6.21 illustre cette mesure. Nous faisons également apparaître le cas de l'élimination selon l'environnement sur ce graphique. On note sur les temps mesurés une amélioration d'environ 20% des temps de calcul de la méthode hybride par rapport à la méthode d'élimination uniquement selon l'environnement.

Nous avons également fait apparaître le temps de la détection de collision uniquement pour les particules sur cette même scène. La technique simple de détection de collision sur les arêtes multiplie par 9 le temps total des tests. La technique hybride d'élimination de tests d'arêtes permet de limiter cette augmentation à un coefficient de 3 pour des tests tout aussi précis.

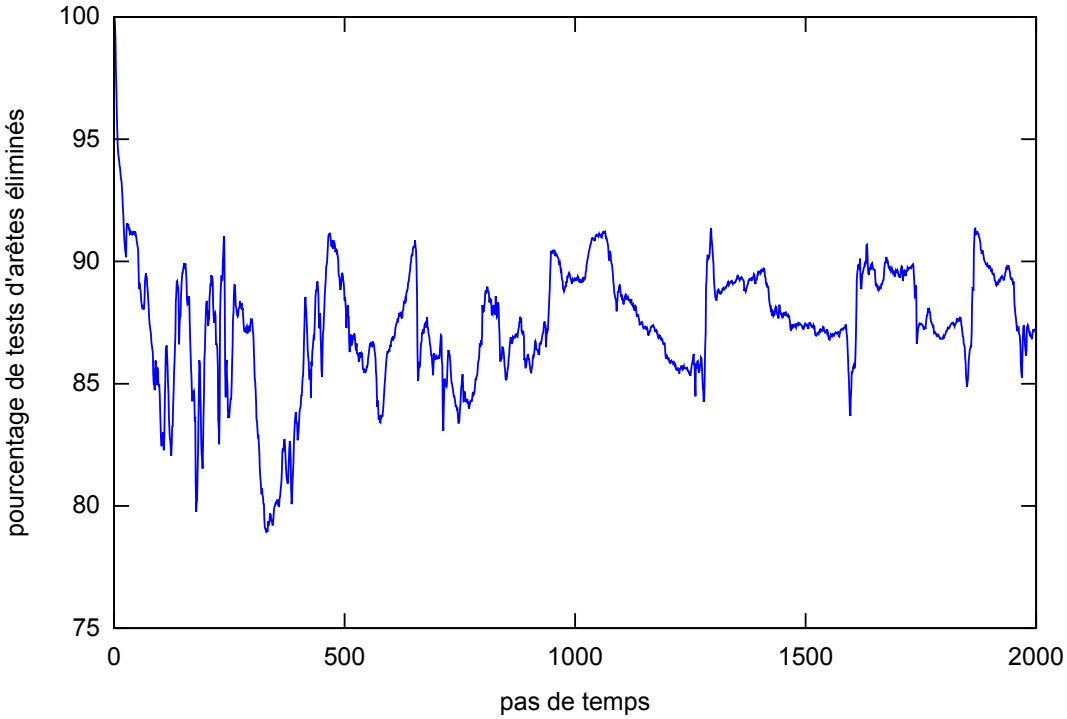

Figure 6.20 : Pourcentage de particules n'effectuant pas de tests de collisions d'arêtes en fonction du temps pour une main échantillonnée avec 11898 particules tombant dans un environnement composé de pyramides.

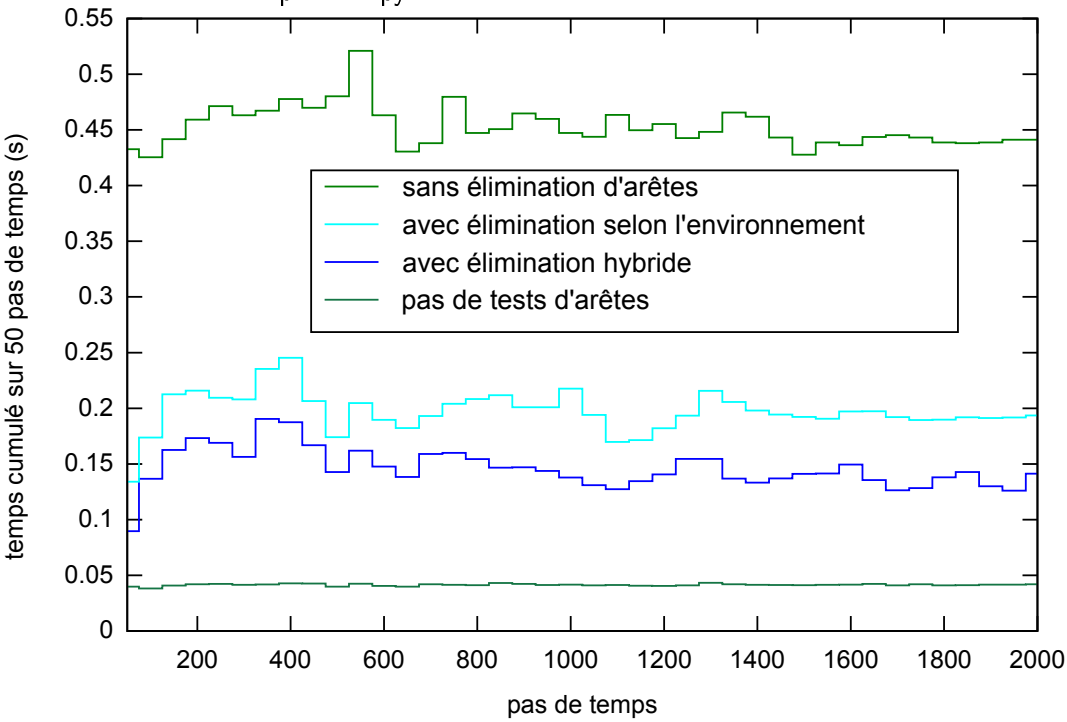

Figure 6.21 : Temps cumulé sur 50 pas de temps de l'ensemble des tests de détection de collision d'arêtes sans élimination, avec élimination selon l'environnement, avec la méthode d'élimination hybride et sans tests d'arêtes.

6.4 Conclusion

L'ensemble des résultats de ce chapitre démontre l'efficacité des algorithmes proposés. Toutes les scènes mises en place pour mesurer les performances réelles lors de l'intégration avec des modèles physiques valident les propriétés théoriques de l'algorithme. Les scènes proposées suivent les contraintes classiques que l'on retrouve en simulation physique et chirurgicale. Le cas de simulations plus chaotiques pour lesquelles les objets en déplacement effectuent de grands déplacements soudains n'est pas traité car il sort du cadre de ce mémoire.

On voit bien apparaître au cours des différentes courbes que la complexité est constante selon le nombre d'éléments atomiques déplacés. Les hiérarchies de volumes englobants peuvent atteindre une complexité amortie constante, en mémorisant à chaque pas de temps l'ensemble des nœuds de la hiérarchie à prendre en compte pour la détection de collision. Cependant, même avec cette mémorisation, lorsque l'environnement devient déformable, des mises à jour sont à effectuer sur l'ensemble de la hiérarchie à chaque déformation. L'avantage de notre méthode se situe dans le fait que seules les cellules de notre partition sont mises à jour lors d'une déformation. Le surcoût au niveau des algorithmes de déplacement est négligeable.

Les analyses comparatives doivent être pondérées. De part notre méthode, les scènes consistent en des subdivisions volumiques. La taille des cellules de ces subdivisions influence les résultats de nos différentes expériences. De même, les pas de temps et vitesses des objets en déplacement doivent être pris en compte vis à vis du volume de ces cellules. Une étude permettant de s'abstraire de l'ensemble de ces paramètres est néanmoins difficile à fournir. Nous avons néanmoins essayé d'apporté diverses mesures afin de mettre en évidence la complexité constante et le principe de localité de nos algorithmes.

Les différentes optimisations consistant en l'élimination d'arêtes sont également testées dans le cadre de simulation d'objets polyédriques fermés. Ces optimisations sont dépendantes du cadre applicatif, leurs gains sont variables selon l'environnement et les déplacements des objets.

Conclusion

Au cours de ce mémoire, notre objectif était de concevoir des algorithmes de détection de collision permettant de répondre aux exigences d'applications contraignantes vis à vis de ce processus. Parmi ces applications, le cas de la simulation chirurgicale est un cas particulièrement difficile. Les exigences de ce type de simulations portent aussi bien sur la notion d'interactivité et de précision que sur la capacité à gérer des situations difficiles en modélisation géométrique, en l'occurrence, la gestion des déformations et des changements de topologie pour des environnements complexes. De plus, la simulation physique se doit d'être précise et réduit d'autant plus le temps disponible pour le processus de détection de collision.

Nous avons proposé diverses méthodes s'appliquant à ce type de contraintes. Nous rappelons ici les différentes contributions de ce mémoire avant de présenter différentes perspectives aux travaux présentés.

1 Rappel des contributions

Nous avons présenté divers algorithmes de détection de collision dans des environnements modélisés par des subdivisions de l'espace (surfaciques ou volumiques). Pour supporter nos algorithmes, ces subdivisions doivent être com-

posées uniquement de cellules convexes. Par contre, nous n'ajoutons pas de contraintes concernant la nature de ces cellules qui peuvent être des tétraèdres, des hexaèdres ou tous polyèdres convexes. Nos algorithmes exploitent les informations de voisinage présentent dans le modèle des cartes combinatoires. Cette structure a été utilisée pour sa capacité à gérer des déformations topologiques et géométriques pour des coûts faibles, tout en fournissant des requêtes d'adjacence et d'incidence optimales.

Dans un premier temps, nous avons proposé un système permettant la détection de collision de particules de manière continue basé sur un algorithme de prédiction de trajectoire. Cet algorithme permet, lorsque les trajectoires des particules sont régulières, d'optimiser le nombre de tests d'orientation à effectuer afin de déterminer si un déplacement entraîne ou non une collision.

A partir de ce système de gestion de particules, nous avons présenté deux extensions concernant le déplacement d'arêtes de manière quasi-continue et de manière continue. Ces extensions sont particulièrement adaptées à notre gestion de particules. Néanmoins, elles sont suffisamment génériques pour être intégrées à d'autres algorithmes de déplacement de particules capables de fournir des informations similaires.

Les algorithmes développés ont été intégrés à des modèles physiques existant afin de permettre diverses mesures de performances.

Un système de détection de collision générique

Le système de détection de collision que nous avons présenté est générique selon plusieurs critères. Nous revenons sur chacun de ces critères point par point.

Nos algorithmes de détection de collision sont indépendants du modèle physique utilisé. Comme nous l'avons vu, il suffit que le modèle physique soit capable d'indiquer aux sommets des objets des positions à atteindre, pour qu'il puisse être intégré. De même, la mise à jour des données cinétiques n'influe en rien sur le déroulement des algorithmes. L'ensemble des tests de collisions a été développé selon cette notion d'indépendance. Le cas où la cohérence temporelle est élevée est évidemment avantageux vis à vis des techniques mises en place, mais le cas général d'un déplacement quelconque est également géré.

Le système que nous proposons est également adapté à tout types de simulations. Les cas simples de déplacement d'objets, tout comme les cas plus évolués

considérant des objets déformables topologiquement et géométriquement, peuvent être traités. Nous avons principalement mis l'accent sur la gestion des objets totalement inclus dans un environnement, si l'on souhaite gérer le cas d'un environnement ouvert, voire infini, il est possible d'ajouter des cellules libres à la volée dans la partition afin de ne pas être limité à un environnement borné.

De plus, le fait de ne pas limiter les cellules de notre environnement à des éléments simpliciaux (*i.e.* des triangles ou des tétraèdres), nous permet une représentation plus compacte en mémoire de nos scènes.

Un système efficace

Le système de détection de collision que nous avons développé, en plus d'être générique, s'avère efficace pour l'ensemble des situations que nous avons testées.

Les temps de calcul obtenus permettent son utilisation dans le cadre d'applications de simulation temps-réel. Les comparaisons, dans divers scénarii, avec le système de détection de collision de la librairie Bullet démontrent sa compétitivité.

Au cours de ces scénarii, nous avons également pu mettre en valeur l'avantage apporté par le principe de localité au niveau des temps de calculs. Lorsque la complexité de la scène est augmentée, les performances de nos algorithmes restent constantes.

La gestion des déformations géométriques et topologiques est également effectuée de manière efficace. Malgré l'imbrication de nos objets, les coûts nécessaires à la mise à jour des différentes informations que nous exploitons sont réduits et impliquent un surcoût mineur. De plus, ces mises à jour sont maitrisées et n'impliquent pas la mise en place d'un système global. Lorsqu'une partie de l'environnement est modifiée, seules les cellules de cette partie et les particules qu'elles contiennent doivent être mises à jour. Notre système est ainsi capable de traiter les scènes pour lesquelles les objets et l'environnement sont déformés et supporte des changements de topologie. De ce fait, il nous est possible d'effectuer des simulations de sutures ou encore de découpes pour des simulations chirurgicales.

De plus, nos algorithmes s'avèrent précis dans le sens où ils permettent de détecter l'ensemble des collisions qui surviennent lors d'un déplacement.

Ils empêchent les interpénétrations des sommets et des arêtes des mobiles en déplacement avec l'environnement. Une implantation efficace de la détection de collision des faces n'est actuellement pas disponible. Cela implique l'apparition éventuelle d'interpénétrations entres les faces des mobiles et l'environnement. Notre précision reste néanmoins suffisante pour l'application à la simulation d'opérations dans le corps humain. Le cas des interpénétrations des faces survient lorsque des parties infranchissables de l'environnement présentent des angles aigus. Or l'ensemble des organes modélisés dans le cadre d'applications chirurgicales est constitué de surfaces faiblement courbes et ne génère donc pas d'imprécisions.

Notre algorithme est également capable d'intégrer des optimisations proposées dans la littérature. Nous avons mis en place un système d'élimination de tests grâce à la méthode du *back-face culling* afin de supprimer des tests de collisions d'arêtes lorsque cela ne s'avérait pas nécessaire.

L'utilisation d'un partitionnement de l'environnement en cellules convexes a également permis de mettre en place une optimisation basée sur les propriétés géométriques des cellules. Cette optimisation s'avère très performante lorsque la taille des arêtes des objets déplacés a un faible ratio comparé à la taille des cellules de l'environnement. De plus, cette optimisation s'avère compatible avec l'élimination des arêtes arrières évoquée ci-dessus.

2 Perspectives

Nos perspectives dans le cadre de ces travaux s'articulent autour de plusieurs points : l'extension des algorithmes proposés, la gestion de l'environnement et l'intégration de systèmes physiques et d'interfaces.

Une implantation concrète de l'algorithme de détection de collision d'arêtes de manière continue en dimension 3 reste à effectuer. Cette extension doit alors être étudiée en terme de performances, tant dans le cas du déplacement d'arêtes que dans le cas du déplacement de faces de manière quasi-continue. Suite à cette étude, il est envisageable de traiter le cas de la détection de collision de face de manière continue en suivant le même principe. Nous envisageons une approche similaire de développement du graphe des cellules rencontrées au cours d'un déplacement de face.

Au cours de l'élimination des arête arrières, les particules continuent à être suivies. L'élaboration d'un algorithme permettant de n'assurer que le suivi des particules se déplaçant vers l'extérieur de l'objet et faisant passer les particules

se déplaçant vers l'intérieur en particules *dormantes* pourrait être intéressant. Lorsqu'une particule doit être activée pour le suivi, il serait alors possible d'identifier la cellule la contenant à partir d'une particule active proche. Cette extension nécessite l'élaboration d'une heuristique permettant de décider quelle particule active doit être utilisée pour réactiver le suivi.

Nous n'avons pas traité le cas de l'auto-collision et de la détection de collision de multiples objets dans un même environnement. Une implantation de ces tests peut également être proposée en exploitant les propriétés de la subdivision de l'environnement. Pour cela, chaque cellule de l'environnement doit mémoriser l'ensemble des primitives qu'elle contient. Ensuite, les primitives contenues dans une même cellule sont testées. Dans ce cas, le partitionnement de l'environnement, en plus de rendre local les tests de collisions objet/environnement, fournit un système d'identification des zones d'intérêt à la manière d'une feuille d'un BVH.

Des solutions sont encore à proposer au niveau de la gestion de l'environnement. Divers travaux portant sur la subdivision en éléments convexes peuvent être intégrés pour le partitionnement initial de l'environnement. La question du remaillage local et efficace lors de déformation topologique et géométrique reste cependant ouverte. Des algorithmes locaux sont à fournir pour permettre d'assurer une complexité linéaire en fonction du nombre de cellules à modifier.

L'intégration de représentations multirésolutions est également une perspective envisageable, divers travaux concernant les cartes combinatoires ont déjà été effectués à ce propos. Il s'agirait ici de proposer des méthodes permettant de raffiner les zones à considérer pour les collisions lorsque celle-ci sont proches d'un obstacle et d'utiliser une représentation plus grossière dans les autres cas.

Bibliographie

[BFA05] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 2, New York, NY, USA, 2005. ACM.

[BGTG03] Daniel Bielser, Pascal Glardon, Matthias Teschner, and Markus Gross. A state machine for real-time cutting of tetrahedral meshes. *11th Pacific Conference on Computer Graphics and Applications*, pages 377–386, 2003.

[Bri93] E. Brisson. Representing geometry structures in d dimensions : Topology and order. *Discrete & Computational Geometry*, pages 387–426, 1993.

[BT95] S. Bandi and D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. In *Proc. Computer Graphics Forum*, volume 14, pages 259–70, 1995.

[Bul] Bullet physics library. http://bulletphysics.org/.

[BYM05] Nathan Bell, Yizhou Yu, and Peter J. Mucha. Particle-based simulation of granular materials. In *SCA : Proc. of the ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages

77–86, New York, NY, USA, 2005.

[Cam97] Stephen Cameron. Enhancing gjk: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[CD99] D. Cazier and J.F. Dufourd. A formal specification of geometric refinements. *Visual Computer*, 15:279–301, 1999.

[CLMP95] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: an interactive and exact collision detection system for large-scale environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–ff., New York, NY, USA, 1995. ACM Press.

[CTM08] Sean Curtis, Rasmus Tamstorf, and Dinesh Manocha. Fast collision detection for deformable models using representative-triangles. In *Proc. of the symposium on Interactive 3D graphics and games*, pages 61–69. ACM, 2008.

[DK90] David P. Dobkin and David G. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. In *ICALP '90: Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, pages 400–413, London, UK, 1990. Springer-Verlag.

[DO00] John Dingliana and Carol O'Sullivan. Graceful degradation of collision handling physically based animation. *Comput. Graph. Forum*, 19(3), 2000.

[Ebe04] David Eberle. Primitive tests for collision detection. *Collision Detection and Proximity Queries, SIGGRAPH 2004 Course*, 2004.

[EG86] H. Edelsbrunner and L. Guibas. Topologically sweeping an arrangement. In *Proc. of the 18th ACM Symposium on the Theory of Computing (Berkeley)*, pages 389–403, 1986.

[EL00] Stephen Ehmann and Ming Lin. Swift: Accelerated proximity queries using multi-level voronoi marching. Technical report, Chapel Hill, NC, USA, 2000.

[EL01] Stephen A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *in Computer Graphics Forum*, pages 500–510, 2001.

[EL07] Mathias Eitz and Gu Lixu. Hierarchical spatial hashing for real-time collision detection. *IEEE International Conference on Shape Modeling and Applications, SMI '07.*, pages 61–70, 2007.

[FBF77] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

[For95] Steven Fortune. Polyhedral modelling with exact arithmetic. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, pages 225–234, 1995.

[GD04] Stéphane Guy and Gilles Debunne. Monte-carlo collision detection. Technical Report RR-5136, INRIA, March 2004.

[Gei00] Bernhard Geiger. Real-time collision detection and response for complex environments. In *CGI '00: Proceedings of the International Conference on Computer Graphics*, page 105, 2000.

[GGK06] Alexander Greß, Michael Guthe, and Reinhard Klein. Gpu-based collision detection for deformable parameterized surfaces. *Computer Graphics Forum*, 25(3):497–506, September 2006.

[GHPT89] Michel Gangnet, Jean-Claude Hervé, Thierry Pudet, and Jean-Manuel Van Thong. Incremental computation of planar maps. In *Proc. of ACM-SIGGRAPH Conf. on Computer Graphics*, volume 23, pages 345–354, 1989.

[GHZ99] Leonidas J. Guibas, David Hsu, and Li Zhang. H-walk: hierarchical distance computation for moving convex bodies. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 265–273. ACM, 1999.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of [see also IEEE Transactions on Robotics and Automation]*, 4(2):193–203, Apr 1988.

[GKJ+05] Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, and Dinesh Manocha. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph.*, 24(3):991–999, 2005.

[GKLM07] Naga K. Govindaraju, Ilknur Kabul, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection among deformable models using graphics processors. *Comput. Graph.*, 31(1):5–14, 2007.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, New York, USA, 1996. ACM Press.

[GLM04] Naga K. Govindaraju, Ming C. Lin, and Dinesh Manocha. Fast and reliable collision culling using graphics hardware. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 2–9, New York, NY, USA, 2004. ACM.

[GNRZ02] Leonidas Guibas, An Nguyen, Daniel Russel, and Li Zhang. Collision detection for deforming necklaces. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 33–42, New York, NY, USA, 2002. ACM.

[GO05] T. Giang and C. O'Sullivan. Closest feature maps for time-critical collision handling. In *Workshop on Virtual Reality Interaction and Physical Simulation*, 2005.

[GRLM03] Naga K. Govindaraju, Stephane Redon, Ming C. Lin, and Dinesh Manocha. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 25–32, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[GSS89] L. Guibas, D. Salesin, and J Stolfi. Epsilon geometry: building robust algorithms from imprecise computations. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 208–217, 1989.

[GST09] Marc Gissler, Ruediger Schmedding, and Matthias Teschner. Time-critical collision handling for deformable modeling. *Comput. Animat. Virtual Worlds*, 20(2–3):355–364, 2009.

[HFSQ01] W. Huagen, F. Zhaowei Fan, G. Shuming, and P. Qunsheng. A parallel collision detection algorithm base on hybrid bounding volume hierarchy. In *Proc. of CAD and Graphics*, pages 521–528, 2001.

[HH98] P. Howlett and W.T. Hewitt. Mass-spring simulation using adaptive non-active points. *Computer Graphics Forum*, 17(3):345–353, 1998.

[HIZ+02] Kenneth E. Hoff, III, Andrew Zaferakis, Ming Lin, and Dinesh Manocha. Fast 3d geometric proximity queries between rigid and deformable models using graphics hardware acceleration. In *Technical Report TR02-004, Department of Computer Science, UNC Chapel Hill*, 02.

[HK09] Shu-Wei Hsu and John Keyser. Statistical simulation of rigid bodies. In *SCA '09: Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 139–148, New York, NY, USA, 2009. ACM.

[HKM95] Martin Held, James T. Klosowski, and Joseph S. B. Mitchell. Evaluation of collision detection methods for virtual reality fly-throughs. In *Canadian Conference on Computational Geometry*, pages 205–210, 1995.

[HPH96] Dave Hutchinson, Martin Preston, and Terry Hewitt. Adaptive refinement for mass/spring simulations. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 31–45, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[Hub93] P.M. Hubbard. Interactive collision detection. In *IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–31, 1993.

[Hub95] Philip M. Hubbard. Collision detection for interactive graphics applications. *IEEE Trans. on Visualization and Comput. Graph.*, 1(3):218–230, 1995.

[HZLM01] Kenneth E. Hoff, III, Andrew Zaferakis, Ming Lin, and Dinesh Manocha. Fast and simple 2d geometric proximity queries using graphics hardware. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 145–148, New York, NY, USA, 2001. ACM.

[ISF07] Geoffrey Irving, Craig Schroeder, and Ronald Fedkiw. Volume conserving finite element simulations of deformable models. *ACM Trans. Graph.*, 26(3):13, 2007.

[JCA06] Lionnel Joussemet, André Crosnier, and Claude Andriot. Espions: A novel algorithm based on evolution strategy for fast collision detection. In *EuroHaptics*, pages 159–167, 2006.

[JFSO06] J. J. Jiménez, F. R. Feito, R. J. Segura, and C. J. Ogáyar. Particle oriented collision detection using simplicial coverings and tetra-trees. *Computer Graphics Forum*, 25(1):53–68, 2006.

[JH08] Hanyoung Jang and JungHyun Han. Fast collision detection using the a-buffer. *Vis. Comput.*, 24(7):659–667, 2008.

[JP04] Doug L. James and Dinesh K. Pai. BD-Tree: Output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (SIGGRAPH 2004)*, 23(3), 2004.

[JS08] Juan J. Jiménez and Rafael J. Segura. Collision detection between complex polyhedra. *Comput. Graph.*, 32(4):402–411, 2008.

[JT80] C.L. Jackins and S.L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, volume 14:pages 249–270, 1980.

[KCB09] Pierre Kraemer, David Cazier, and Dominique Bechmann. Extension of half-edges for the representation of multiresolution subdivision surfaces. *The Visual Computer*, 25(2):149–163, 2009.

[KHH+09] Duksu Kim, Jae-Pil Heo, Jaehyuk Huh, John Kim, and Sung-Eui Yoon. HPCCD: Hybrid parallel continuous collision detection. *Computer Graphics Forum (Pacific Graphics)*, 28(7), 2009.

[KHM+98] James T. Klosowski, Martin Held, Joseph S.B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, volume 4(1):21–36, 1998.

[KNF04] S. Kimmerle, Matthieu Nesme, and François Faure. Hierarchy accelerated stochastic collision detection. In *Vision, Modeling, and Visualization*, pages 307–314, Stanford, California, 2004.

[KPLM98] Shankar Krishnan, Amol Pattekar, Ming C. Lin, and Dinesh Manocha. Spherical shell: a higher order bounding volume for fast proximity queries. In *WAFR '98: Proceedings of the third workshop on the algorithmic foundations of robotics on Robotics : the algorithmic perspective*, pages 177–190, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[KZ03] Jan Klein and Gabriel Zachmann. Adb-trees: Controlling the error of time-critical collision detection. In *In 8th International Fall Workshop Vision, Modeling and Visualisation (VMV)*, pages 37–46. Springer, 2003.

[LAM01] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics, Short presentations*, 2001.

[LAM03] Thomas Larsson and Tomas Akenine-Möller. Efficient collision detection for models deformed by morphing. *The Visual Computer*, 19:164–174, 2003.

[LAM06] Thomas Larsson and Tomas Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers and Graphics*, 30(3):450–459, 2006.

[LC91] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1008–1014 vol.2, Apr 1991.

[LC98] Tsai-Yen Li and Jin-Shin Chen. Incremental 3d collision detection with hierarchical data structures. In *VRST '98: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139–144, New York, NY, USA, 1998. ACM.

[LCDN06] Julien Lenoir, Stephane Cotin, Christian Duriez, and Paul F. Neumann. Interactive physically-based simulation of catheter and guidewire. *Computers & Graphics*, 30(3):416–422, 2006.

[LCF05] Rodrigo G. Luque, ao L. D. Comba, Jo and Carla M. D. S. Freitas. Broad-phase collision detection using semi-adjusting bsp-trees. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 179–186, New York, NY, USA, 2005. ACM.

[Lev66] Cyrus Leventhal. Molecular model-building by computer. *Scientific American*, volume 214(6), 1966.

[LGLM99] Eric Larsen, Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. 1999.

[Lie91] Pascal Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Comput. Aided Des.*, 23(1):59–82, 1991.

[Lie94] Pascal Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *Int. J. Comput. Geometry Appl.*, 4(3):275–324, 1994.

[LM04] Ming C. Lin and Dinesh Manocha. *Handbook of Discrete and Computational Geometry*, chapter 35 - Collision and proximity queries. CRC Press, 2004.

[LqWhX07] Li Liu, Zhao qi Wang, and Shi hong Xia. A volumetric bounding volume hierarchy for collision detection. In *Computer-Aided Design and Computer Graphics, 2007 10th IEEE International Conference on*, pages 485–488, 2007.

[LSGR05] Peiran Liu, Xiaojun Shen, Nicolas Georganas, and Gerhard Roth. Multi-resolution modeling and locally refined collision detection for haptic interaction. In *3DIM '05: Proceedings of the Fifth International Conference on 3-D Digital Imaging and Modeling*, pages 581–588, Washington, DC, USA, 2005. IEEE Computer Society.

[LSM08] Pierre-François Léon, Xavier Skapin, and Philippe Meseure. A topology-based animation model for the description of 2D models with a dynamic structure. In *Proceedings of Virtual Reality Interactions and Physical Simulations 2008*, 2008.

[LT07] Alex Lindblad and George Turkiyyah. A physically-based framework for real-time haptic cutting and interaction with 3d continuum models. In *Proc. of the ACM symposium on Solid and physical modeling*, pages 421–429, 2007.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *GI '04: Proceedings of Graphics Interface 2004*, pages 239–246, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM Trans. Graph.*, 24(3):471–478, 2005.

[Mil95] Victor Milenkovic. Practical methods for set operations on polygons using exact arithmetic. In *In Proc. 7th Canad. Conf. Comput. Geom*, pages 55–60, 1995.

[Mir98] Brian Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Trans. Graph.*, 17(3):177–208, 1998.

[MKB+08] Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. Polyhedral finite elements using harmonic basis functions. *Computer Graphics Forum*, 27(5):1521–1529, 2008.

[MKE03] J. Mezger, S. Kimmerle, and O. Etzmuss. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG*, 11:322–329, 2003.

[MO06] C. Mendoza and C. O'Sullivan. Interruptible collision detection for deformable objects. *Comput. Graph.*, 30(3):432–438, 2006.

[Mou04] David M. Mount. *Handbook of Discrete and Computational Geometry*, chapter 38 - Geometric Intersection. CRC Press, 2004.

[MPT99] William A. McNeely, Kevin D. Puterbaugh, and James J. Troy. Six degree-of-freedom haptic rendering using voxel sampling. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 401–408, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[NAT90] Bruce Naylor, John Amanatides, and William Thibault. Merging bsp trees yields polyhedral set operations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 115–124, New York, USA, 1990. ACM Press.

[NPF05] Matthieu Nesme, Yohan Payan, and François Faure. Efficient, physically plausible finite elements. In J. Dingliana and F. Ganovelli, editors, *Eurographics (short papers)*, august 2005.

[OCSG07] Miguel A. Otaduy, Olivier Chassot, Denis Steinemann, and Markus Gross. Balanced hierarchies for collision detection between fracturing objects. *Virtual Reality Conference, IEEE*, 0:83–90, 2007.

[OD01] Carol O'Sullivan and John Dingliana. Collisions and perception. volume 20, pages 151–168, New York, NY, USA, 2001. ACM.

[OL03] Miguel A. Otaduy and Ming C. Lin. Clods: dual hierarchies for multiresolution collision detection. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 94–101, Aire-la-Ville, Switzerland, 2003. Eurographics Association.

[PF05] Matt Pharr and Randima Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Addison-Wesley Professional, 2005.

[Pro97] Xavier Provot. Collision and self-collision handling in cloth model dedicated to design garment. *Graphics Interface*, pages 177–189, 1997.

[RKC02a] Stephane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Fast continuous collision detection between rigid bodies. In *Proceedings of Eurographics*, 2002.

[RKC02b] Stephane Redon, Abderrahmane Kheddar, and Sabine Coquillart. Hierarchical back-face culling for collision detection. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[SBGS] R.A Schumaker, B. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation.

[SGG+06] Avneesh Sud, Naga Govindaraju, Russell Gayle, Ilknur Kabul, and Dinesh Manocha. Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Trans. Graph.*, 25(3):1144–1153, 2006.

[She97] Jonathan Richard Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, 18(3):305–363, 1997.

[SJC05] Min Hong Sunwha Jung and Min-Hyung Choi. An adaptive collision detection and resolution for deformable objects using spherical implicit surface. *LNCS*, 3514:735–742, 2005.

[SK06] N. Saenghaengtham and P. Kanongchaiyos. A collision detection algorithm using particle sensor. pages 1 –6, 2006.

[SKSH03] Mikhail Senin, Nikita Kojekine, Vladimir Savchenko, and Ichiro Hagiwara. Particle-based collision detection. pages 1–8, 2003.

[TCYM09] M. Tang, S. Curtis, S. Yoon, and D. Manocha. Iccd: Interactive continuous collision detection between deformable models using connectivity-based culling. pages 544–557, 2009.

[THM+03] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling, Visualization VMV'03*, pages 47–54, 2003.

[TKZ+04] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure,

N. Magnetat-Thalmann, and W. Strasser. Collision detection for deformable objects. In *Eurographics State-of-the-Art Report (EG-STAR)*, pages 119–139. Eurographics Association, 2004.

[TMT10] Min Tang, Dinesh Manocha, and Ruofeng Tong. Mccd: Multi-core collision detection between deformable models. *Graphical Models*, 72(2):7–23, 2010.

[TTSD06] Oren Tropp, Ayellet Tal, Ilan Shimshoni, and David P. Dobkin. Temporal coherence in bounding volume hierarchies for collision detection. *Int. J. Shape Model.*, 12(2):pages 159–178, 2006.

[Tur90] Greg Turk. Interactive collision detection for molecular graphics. Technical report, Chapel Hill, NC, USA, 1990.

[TYM08] Min Tang, Sung-Eui Yoon, and Dinesh Manocha. Adjacency-based culling for continuous collision detection. *Vis. Comput.*, 24(7):545–553, 2008.

[Van94] G. Vanecek. Back-face culling applied to collision detection of polyhedra. *Journal of Visualization and Computer Animation*, volume 5(1):55–63, 1994.

[vdB97] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, 1997.

[VT01] Kevin Vlack and Susumu Tachi. Fast and accurate spacio-remporal intersection detection with the gjk algorithm. *ICAT : International Conference on Artificial Reality and Telexistence*, pages 79–85, 2001.

[WB05] Wingo Sai-Keung Wong and George Baciu. Gpu-based intrinsic collision detection for deformable surfaces: Collision detection and deformable objects. *Comput. Animat. Virtual Worlds*, 16(3-4):153–161, 2005.

[WB06] Wingo Sai-Keung Wong and George Baciu. A randomized marking scheme for continuous collision detection in simulation of deformable surfaces. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 181–188, New York, NY, USA, 2006. ACM.

[WHG84] Hank Weghorst, Gary Hooper, and Donald P. Greenberg. Improved computational methods for ray tracing. *ACM Trans. Graph.*, 3(1):52–69, 1984.

[WLW+06] Y. Wang, W. Li, T. Wang, W. Guo, and Z. Zhang. An efficient collision detection of complex deformable objects based on particle swarm optimization algorithm. In *International Conference on Machine Learning and Cybernetics*, pages 3964 – 3969, 2006.

[WZ06] René Weller and Gabriel Zachmann. Kinetic separation lists for continuous collision detection of deformable objects. In *Third Workshop in Virtual Reality Interactions and Physical Simulation (Vriphys)*, Madrid, Spain, 6–7 November 2006.

[WZ09a] Rene Weller and Gabriel Zachmann. Inner sphere trees for proximity and penetration queries. In *2009 Robotics: Science and Systems Conference (RSS)*, Seattle, WA, USA, 2009.

[WZ09b] Rene Weller and Gabriel Zachmann. A unified approach for physically-based simulations and haptic rendering. In *Sandbox '09: Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*, pages 151–159, New York, NY, USA, 2009. ACM.

[YCM07] Sung-Eui Yoon, Sean Curtis, and Dinesh Manocha. Ray tracing dynamic scenes using selective restructuring. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 55, New York, NY, USA, 2007. ACM.

[ZK07] Xinyu Zhang and Young J. Kim. Interactive collision detection for deformable models using streaming aabbs. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):318–329, 2007.

[ZW06] Gabriel Zachmann and Rene Weller. Kinetic bounding volume hierarchies for deformable objects. In *VRCIA '06: Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, pages 189–196, New York, NY, USA, 2006. ACM.