Titre: Title:	Détection de collisions d'objets déformables en temps réel pour la simulation chirurgicale
Auteur: Author:	François Léonard
Date:	2021
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Léonard, F. (2021). Détection de collisions d'objets déformables en temps réel pour la simulation chirurgicale [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/9123/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/9123/
Directeurs de recherche: Advisors:	Benoît Ozell
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Détection de collisions d'objets déformables en temps réel pour la simulation chirurgicale

FRANÇOIS LÉONARD
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2021

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Détection de collisions d'objets déformables en temps réel pour la simulation chirurgicale

présenté par **François LÉONARD**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Martine BELLAÏCHE, présidente
Benoît OZELL, membre et directeur de recherche
Thomas HURTUT, membre

REMERCIEMENTS

Je tiens à remercier Benoît Ozell pour m'avoir offert l'opportunité de travailler sur ce projet. Je tiens également à remercier Vincent Magnoux pour son aide et tous les membres du laboratoire LIRV pour leur soutien moral.

RÉSUMÉ

La simulation chirurgicale permet aux chirurgiens de pratiquer certaines opérations dans un environnement virtuel. Elle offre un moyen de développer les connaissances et les aptitudes motrices et techniques requises sans risque pour des patients et à un coût moindre que la plupart des méthodes traditionnelles. Pour y parvenir, les simulateurs doivent reproduire fidèlement des situations médicales complexes de façon interactive et avec un comportement physique réaliste des organes et tissus simulés.

Un aspect important de la simulation chirurgicale est la détection de collisions. Cette étape consiste à identifier en temps réel les contacts entre les surfaces d'objets déformables généralement représentées par des maillages de triangles. De plus, elle doit être adaptée aux diverses interactions complexes qu'exige la simulation de certaines procédures, comme la découpe d'objets avec un scalpel ou d'autres instruments chirurgicaux. De telles découpes entraînent des changements dans la topologie des objets qui peuvent impacter l'algorithme de détection.

La recherche présentée ici porte sur la détection de collisions dans un environnement de simulation d'objets déformables permettant la découpe des objets à l'aide d'un outil virtuel. L'objectif principal est de développer un algorithme capable de détecter en temps réel les collisions d'objets pouvant subir des changements de topologie. L'algorithme doit pouvoir détecter rapidement les collisions aussi bien entre objets que les auto-collisions, c'est-à-dire les collisions d'une surface avec elle-même, et ce pour des environnements contenant plusieurs dizaines de milliers de triangles.

La méthode implémentée est basée sur le partitionnement de l'espace comme méthode pour élaguer rapidement les triangles trop éloignés pour être en collision. Pour permettre la simulation d'environnements complexes en temps réel, l'algorithme a été adapté au calcul en parallèle sur processeur multicœur.

Afin d'évaluer la performance de l'algorithme et l'atteinte des objectifs, une série d'expériences a été effectuée. D'abord, différentes étapes de l'algorithme ont été testées pour mesurer leur efficacité à élaguer les triangles des surfaces et accélérer la détection. Les résultats montrent que le choix des volumes englobants employés a un impact important sur la performance. Ils démontrent aussi l'efficacité de la méthode des cônes normaux, une technique d'accélération de la détection des auto-collisions.

La performance de l'algorithme a été comparée avec celle d'une autre méthode de détection de collisions implémentée dans la plateforme logicielle SOFA et notre méthode s'avère plus

rapide, particulièrement pour la détection des auto-collisions. Cependant, les résultats ne permettent pas de garantir la simulation en temps réel sur CPU. Néanmoins, les expériences effectuées sur un processeur multicœur démontrent que la méthode est bien adaptée à la parallélisation et qu'un gain important peut être attendu en portant l'algorithme sur GPU. La discussion présente des améliorations possibles pour augmenter davantage l'accélération sur processeur multicœur de même que des changements qui seraient bénéfiques pour adapter l'algorithme au calcul sur GPU.

Le fonctionnement de la méthode avec les changements de topologie n'a pas pu être démontré. Le problème est dû à l'incompatibilité de la méthode de réponse aux collisions utilisée avec l'algorithme de découpe et la méthode utilisée pour le calcul des déformations. Ce problème n'invalide pas la méthode de détection choisie et certains changements sont discutés pour le résoudre.

ABSTRACT

Surgical simulation allows surgeons to practice certain operations in a virtual environment. It offers a way to develop the required knowledge as well as the technical and motor skills needed without any risk for patients and at a lower cost than most traditional methods. For this, simulators must be able to reproduce complex medical situations realistically with the proper physical behaviour of the simulated organs and tissues.

An important aspect of surgical simulations is collision detection. It consists in identifying in real-time the contacts between the surface of deformable objects generally represented with a triangles mesh. Moreover, the collision detection must be adapted to the complex interactions required to simulate certain procedures, like cutting an object with a scalpel or other surgical instruments. Such cuts cause topological changes of the object which can impact the detection algorithm.

This project focuses on collision detection in a simulator of deformable objects which allows interactive cuts with a virtual tool. The main objective is to develop an algorithm to detect collisions in real-time between objects which can undergo topological changes. The algorithm must be able to detect rapidly the collisions between different objects as well as self-collisions, that is, collisions of a surface with itself, for scenes containing tens of thousands of triangles.

The implemented method is based on spatial partitioning as a way to rapidly cull triangles too far from one another to be in collision. To allow the real-time simulation of complex environments, the algorithm was adapted to parallel computing on multi-core processors.

In order to evaluate the performance of the algorithm, a series of experiments was carried out. Firstly, different steps of the algorithm were tested to measure their efficiency at culling the surface triangles and accelerating the detection. The results showed that the choice of bounding volumes has an important impact on computation time. They also demonstrated the efficiency of the normal cone method, a technique used to accelerate the detection of self-collisions.

The performance of the algorithm was compared with another collision detection method implemented in the SOFA framework. Our method was found to be faster, especially with the detection of self-collisions. However, the results do not guarantee real-time performance on CPU. Nevertheless, the experiment performed on a multi-core processor showed that the method is well adapted to parallel computing and that an important gain should be obtained by porting it on GPU. Some possible improvements are discussed to further increase the

acceleration on multi-core CPU and to adapt the algorithm to GPU computing.

The effectiveness of the method with topological changes could not be demonstrated. The problem is due to the incompatibility of the collision response method used with the cutting algorithm and the method used to compute the objects deformations. This problem does not invalidate the spatial partitioning method developed for this project and some possible changes are discussed to resolve it.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AABB	*Axis Aligned Bounding Boxe* (Boîte englobante alignée avec les axes)
AVL	Adelson-Velsky et Landis
BSP	*Binary Space Partitioning* (Partitionnement binaire de l'espace)
BVH	*Bounding Volume Hierarchy* (Hiérarchie de volumes englobants)
BVTT	*Bounding Volume Test Tree* (Arbre de test de volumes englobants)
CPU	*Central Processing Unit* (Processeur)
DOP	*Discrete Oriented Polytope* (Polytope orienté discret)
GPU	*Graphics Processing Unit* (Processeur graphique)
LBVH	*Linear Bounding Volume Hierarchy* (Hiérarchie de volumes englobants linéaire)
LDI	*Layered Depth Images* (Images de profondeurs par couches)
OBB	*Oriented Bounding Boxe* (Boîte englobante orientée)

CHAPITRE 1 INTRODUCTION

1.1 Simulation chirurgicale

La pratique de la chirurgie est une activité complexe qui exige une grande expertise. Les erreurs médicales au cours d'opérations peuvent être lourdes de conséquences et le chirurgien doit pouvoir faire face à de multiples complications. Pour cette raison, les futurs chirurgiens doivent parfaire leur éducation durant de nombreuses années afin d'acquérir les connaissances et aptitudes requises avant de pouvoir exercer de façon autonome dans une salle d'opération.

Une part importante de cette éducation consiste en l'entraînement par la pratique. L'approche pédagogique à l'entraînement chirurgical a changé considérablement au cours des dernières décennies. Avant de pouvoir acquérir les aptitudes nécessaires par la pratique supervisée en salle d'opération, les futurs chirurgiens passent de plus en plus de temps à s'entraîner par des moyens plus sécuritaires, comme en utilisant des animaux, des cadavres ou des mannequins médicaux. Cependant, ces méthodes souffrent de certaines limitations. La similarité entre l'anatomie humaine et animale est limitée et les normes éthiques placent des contraintes à l'usage d'animaux. Les cadavres sont dispendieux et leur disponibilité est limitée. Les mannequins médicaux peuvent être utiles pour acquérir certaines aptitudes de bases, mais leur réalisme est insuffisant pour représenter la complexité de l'anatomie et la physiologie humaine. De nouvelles méthodes d'entraînement faisant usage des avancées technologiques sont de plus en plus répandues. Notamment, la simulation chirurgicale est un domaine prometteur et en pleine croissance. Plusieurs entreprises offrent déjà des solutions pour certains types d'opérations.

La simulation chirurgicale vise à reproduire l'aspect visuel et haptique ressenti par un chirurgien durant une procédure. Elle permet à l'apprenti de répéter la pratique des techniques et de gérer les complications jusqu'à atteindre l'expertise nécessaire. Cet entraînement se fait sans les limitations des autres méthodes mentionnées plus haut, sans mettre en danger des patients et potentiellement de façon plus économique. La recherche a prouvé les bénéfices d'inclure la simulation dans l'éducation des chirurgiens : les aptitudes acquises par la simulation se traduisent en de meilleures performances en salle d'opération [1, 2]. En plus de l'éducation de nouveaux chirurgiens, les simulateurs peuvent aussi servir à la planification d'opérations pour des chirurgiens expérimentés. Cette approche permet d'améliorer les aptitudes des chirurgiens, de réduire les coûts hospitaliers et d'améliorer les résultats chez les patients [2].

La simulation chirurgicale comporte plusieurs défis techniques. Le principal est la modélisation des organes et tissus humains. Ils consistent en des objets déformables qui doivent être simulés de façon interactive et en temps réel. Un simulateur doit pouvoir représenter la mécanique de ces objets pour calculer leurs déplacements dans l'espace et leurs déformations. La méthode des éléments finis et divers modèles sans maillage sont souvent utilisés à cette fin. Un autre défi important est la détection des collisions. Celui-ci consiste à détecter les points de contact entre les surfaces d'objets et à corriger leur dynamique pour éviter qu'ils pénètrent l'un dans l'autre. Enfin, certaines procédures requièrent de simuler des interactions complexes avec les tissus et organes comme des coupures ou incisions à l'aide de scalpels.

1.2 Détection de collisions dans la simulation chirurgicale

La détection de collisions est le problème auquel ce projet de recherche s'intéresse. La détection de collisions pour la simulation chirurgicale comporte certaines problématiques particulières.

La première vient du fait que les objets simulés sont déformables. La surface de ces objets est généralement composée d'un maillage de triangles. Trouver les points de contact entre deux objets consiste donc, dans un premier temps, à trouver leurs triangles qui sont en contact. La position relative de ces triangles les uns par rapport aux autres change lorsque les objets subissent des déformations et ils agissent donc en quelque sorte comme un ensemble d'objets différents.

Une autre difficulté avec les objets déformables est la détection des auto-collisions, c'est-à-dire la détection des collisions d'une surface avec elle-même. Certains tissus et organes peuvent être recourbés sur eux-mêmes au cours d'une opération. La détection des auto-collisions est nécessaire pour empêcher qu'une section d'une surface pénètre dans une autre.

Pour un rendu visuel suffisant, des milliers de triangles par objet peuvent être nécessaires et la détection de collisions devient alors très coûteuse. Puisque la simulation chirurgicale doit être interactive, elle doit pouvoir être exécutée en temps réel. Ce critère impose des contraintes de performance importantes sur tous les aspects de la simulation, y compris la détection de collisions, et de nombreuses méthodes d'accélération ont été proposées dans la littérature.

Enfin, les interactions avec les objets, comme la découpe, peuvent affecter l'efficacité de ces méthodes d'accélération. Une attention particulière doit donc être portée aux implications de ces interactions lorsqu'un algorithme de détections de collisions est développé.

1.3 Contexte du projet

Ce travail vise à développer un algorithme de détection de collisions dans un environnement de simulation nommé Scyther [3–5]. L'objectif de ce projet était de développer une méthode de simulation d'objets déformables et de découpe à l'aide d'un outil virtuel. Ce projet, de même que l'algorithme de détection de collisions présenté dans ce mémoire, sont implémentés dans une librairie nommée Scyther.

Ce projet utilise SOFA [6], une plateforme logicielle libre pour la simulation physique en temps réel avec un accent sur la simulation médicale. SOFA décompose en composantes indépendantes les différents aspects de la simulation comme la discrétisation des objets, les forces et contraintes, les solutionneurs numériques ou les algorithmes de détections de collisions. Cette architecture permet de facilement remplacer ces composantes avec celles développées et ainsi de se servir de SOFA comme banc d'essai.

1.4 Plan du mémoire

Le chapitre suivant présente une revue de littérature sur la détection de collisions avec une attention particulière sur les aspects les plus pertinents pour la simulation chirurgicale. Les notions générales y sont d'abord présentées, suivies des différentes méthodes de détection existantes. La fin du chapitre présente les conclusions retenues sur l'état de l'art ainsi que les objectifs et les hypothèses de ce projet de recherche. Le chapitre 3 décrit en détail l'algorithme de détection de collisions développé ainsi que les expériences réalisées. Le chapitre 4 présente les résultats obtenus dans ces expériences. Ceux-ci sont discutés au chapitre 5, ainsi que les limitations observées et les améliorations possibles. Enfin, le chapitre 6 présente une synthèse des travaux effectués et quelques avenues possibles d'améliorations futures.

CHAPITRE 2 REVUE DE LITTÉRATURE

La détection de collisions est un problème important dans plusieurs applications comme la robotique, les jeux vidéos, la simulation chirurgicale et l'animation 3D [7]. La question de détecter les contacts entre les objets simulés a été largement étudiée. Plusieurs approches ont été proposées et leur efficacité dépend grandement de l'application où elles sont utilisées.

Ce chapitre vise à présenter un aperçu des approches proposées dans la littérature pour la détection de collisions dans le contexte de la simulation d'objets déformables et plus spécifiquement de la simulation chirurgicale. D'abord, la section 2.1 discute des considérations générales des algorithmes de détection de collisions afin de définir quelques termes et concepts importants. Ensuite, les sections 2.2 à 2.6 présentent les principales méthodes existantes pour l'accélération de la détection de collisions. La section 2.7 résume les méthodes de détection et de réponses aux collisions implémentées dans SOFA. Enfin, la section 2.8 présente une synthèse de l'état de l'art et les objectifs et hypothèses de ce projet de recherche.

2.1 Fonctionnement général de la détection de collisions

2.1.1 Pipeline de collision

La détection et la réponse aux collisions sont généralement gérées séparément. La détection consiste à trouver les points de contact entre les objets et à calculer les informations nécessaires à la réponse aux collisions comme la distance de séparation ou de pénétration ou le volume d'intersection des objets. Ensuite, la réponse aux collisions utilise ces informations pour calculer les forces ou les contraintes à appliquer sur les objets pour modifier leur mouvement et éviter qu'ils pénètrent l'un dans l'autre.

On appelle primitives de collision les éléments dynamiques de la simulation entre lesquels les collisions sont détectées. Avec la simulation d'objets déformables, les primitives sont souvent les sommets, arêtes et faces des triangles du maillage de surface, les éléments finis ou d'autres structures superposées aux objets comme des sphères qui approximent leur volume.

Une approche par force brute consisterait à comparer la position de chaque paire de primitives dans l'environnement simulé. Avec n primitives, cela consisterait à effectuer $n(n-1)/2$ tests de collisions, correspondant à une complexité $O(n^2)$, ce qui peut rapidement devenir trop coûteux pour la simulation en temps réel. Pour cette raison, la détection de collisions est souvent divisée en au moins deux étapes : la phase large et la phase étroite.

La phase large consiste à accélérer la détection de collisions en trouvant les paires de primitives qui sont potentiellement en collision à l'aide de tests rapides mais peu précis. Généralement, il s'agit d'identifier et d'éliminer les objets et les primitives trop éloignés pour être en contact. La phase étroite consiste généralement à effectuer des tests entre les primitives identifiées lors de la phase large pour détecter avec exactitude les collisions entre les objets. Cette organisation de la détection de collisions en étapes successives s'appelle le pipeline de collision et a été proposée initialement par Hubbard [8].

2.1.2 Détection de collisions discrète et continue

Les collisions peuvent être détectées de façon discrète ou continue. Avec la détection de collisions discrète, les contacts sont détectés seulement à des instants discrets. À chaque pas de simulation, l'algorithme détermine si les objets s'intersectent ou s'ils sont suffisamment proches pour être traités comme étant en collision. Seulement la position à cet instant est considérée et les objets sont traités comme s'ils étaient stationnaires à chaque pas. La principale information calculée est la distance minimale entre les primitives ou leur profondeur de pénétration dans l'objet. Cette information est ensuite utilisée par la méthode de réponse aux collisions, généralement afin de calculer des forces qui seront ajoutées au système dynamique pour repousser les primitives en contact.

La détection de collisions continue prend également en compte le déplacement des objets. Elle interpole linéairement la position et le temps exacts du contact à partir des informations des objets entre deux pas de simulation consécutifs. Cette méthode est plus longue à calculer puisque trouver le premier point de contact entre deux primitives correspond principalement à calculer les racines d'une équation cubique qui décrit la distance entre elles. L'information sur les points de contact est ensuite utilisée par la réponse aux collisions, généralement pour calculer des contraintes de position ou de vélocité. La méthode de résolution du système dynamique doit être adaptée pour prendre en compte ces contraintes lors du calcul des calculs de déformation des objets.

La détection de collisions discrète est plus simple et rapide à calculer, mais elle n'empêche pas les objets d'entrer l'un dans l'autre si la force de répulsion n'est pas suffisante ou si les objets s'approchent trop rapidement. De plus, un objet peut potentiellement en traverser un autre sans qu'une collision soit détectée si le déplacement de l'objet entre deux pas est trop grand relativement à la taille des objets. La détection de collisions continue règle ce problème et permet une simulation physiquement plus réaliste et potentiellement plus stable, mais augmente la complexité du calcul [9]. Non seulement la détection des collisions continue est plus coûteuse, mais la résolution des systèmes dynamiques avec contraintes est aussi plus

lente que de simplement ajouter des forces comme il est possible avec la détection discrète.

2.1.3 Simulation en temps réel

La simulation en temps réel, ou interactive, est un aspect essentiel à la simulation chirurgicale. Pour que la simulation soit interactive, l'efficacité de l'algorithme de détection de collisions est particulièrement importante puisque celle-ci contribue souvent à une grande partie du temps de calcul requis. Pour que la simulation paraisse fluide, un rendu visuel entre 20 et 60 Hz est acceptable. Cependant, le rendu haptique exige une fréquence de simulation plus élevée, soit entre 300 et 1000 Hz [10].

2.1.4 Problématiques de la simulation chirurgicale

La détection de collisions pour la simulation chirurgicale comporte plusieurs problématiques particulières. La première vient du fait que les tissus simulés sont des objets déformables. Les premières méthodes de détection de collisions s'appliquaient principalement aux objets rigides [7]. Plusieurs aspects viennent compliquer la question avec les objets déformables.

Premièrement, plusieurs des méthodes proposées dans la littérature nécessitent une phase de prétraitement. Cette phase consiste souvent à construire des structures de données d'accélération, comme des hiérarchies de volumes englobants, des structures de partitionnement de l'espace ou des champs de distances. Ces méthodes sont expliquées dans les sections 2.3 à 2.5. Avec des objets déformables, ces structures d'accélération doivent être mises à jour ou recalculées pour tenir compte des changements de forme des objets. Cela rend ces méthodes moins efficaces que pour les objets rigides.

Deuxièmement, la déformation d'objets fait en sorte que ceux-ci peuvent entrer en contact avec eux-mêmes. Ainsi, contrairement aux cas des objets rigides, la simulation d'objets déformables requiert la détection des auto-collisions en plus de la détection des collisions entre différents objets. Avec certaines méthodes, la détection des auto-collisions requiert un traitement spécial tel que précisé dans les sections suivantes.

Troisièmement, une autre problématique de la simulation chirurgicale concerne la découpe des tissus pour simuler l'usage d'un scalpel. La découpe d'un objet entraîne un changement de sa topologie et peut même le séparer en deux objets distincts. Avec les méthodes de détection de collisions qui emploient une structure de données d'accélération, la découpe doit aussi modifier cette structure pour l'adapter à la nouvelle topologie, ce qui peut réduire leur efficacité.

2.2 Sweep and prune

La première méthode d'accélération de la détection de collisions présentée est celle de *sweep and prune*. Il s'agit d'un algorithme de phase large qui permet de détecter rapidement des volumes englobants en intersection et ainsi trouver les paires d'objets pouvant être en collision. Il fut d'abord présenté par Baraff [11] sous le nom de *sort and sweep*. Les publications subséquentes, comme Cohen *et al.* [12], utilisent plutôt le nom *sweep and prune*.

L'algorithme fonctionne en construisant des boîtes englobantes autour des objets. Ensuite, les coordonnées de leurs bornes inférieure et supérieure sont triées le long d'un certain nombre d'axes (voir figure 2.1). Cela permet de trouver les volumes englobants dont la projection sur l'axe se chevauche. Si leurs projections se chevauchent sur chacun des axes, les volumes englobants sont alors nécessairement en intersection. On peut alors faire des tests plus précis, mais plus coûteux, pour déterminer si les objets englobés sont en contact.

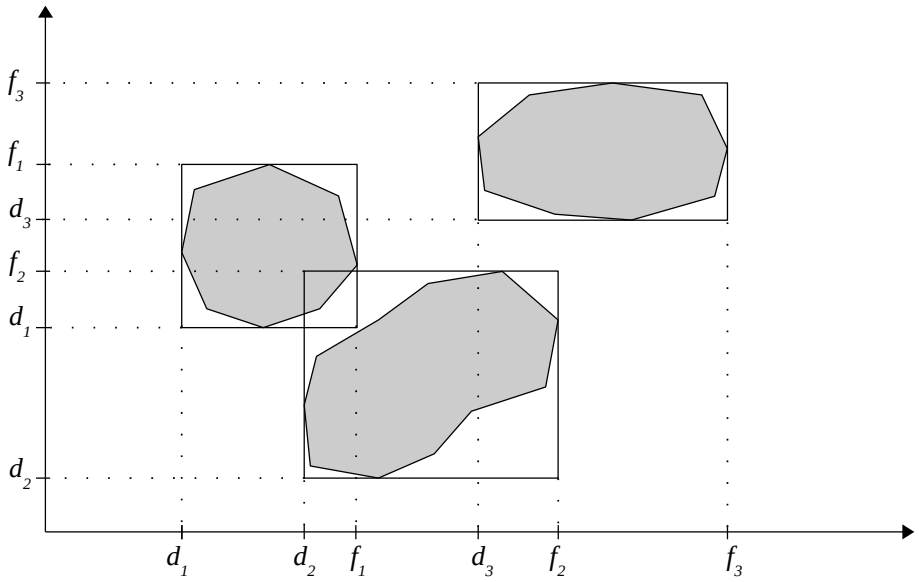


Figure 2.1 Exemple 2D de l'algorithme *sweep and prune* pour trois objets avec la projection des débuts (d_i) et des fins (f_i) de leur volume englobant.

Le *sweep and prune* peut exploiter la cohérence temporelle puisque les objets ne se déplacent généralement que sur de courtes distances entre deux pas de simulation consécutifs. Pour cette raison, la projection des bornes sur les axes change peu à chaque pas et peut donc être triée rapidement. Les algorithmes de tri efficaces pour trier des listes presque déjà en ordre, comme le tri par insertion, sont particulièrement utiles pour cette tâche.

2.3 Hiérarchie de volumes englobants

La hiérarchie de volumes englobants est une méthode d'accélération de détection de collisions basée sur le partitionnement récursif des objets. Elle consiste en un partitionnement récursif des primitives des objets regroupées dans des volumes englobants, permettant d'éliminer rapidement un grand nombre des primitives trop éloignées pour être en collision et d'identifier celles qui ont plus de chance de l'être.

La hiérarchie prend la forme d'un arbre où chaque nœud correspond à un volume englobant (voir figure 2.2). Une hiérarchie peut être construite pour chaque objet présent dans l'environnement simulé. La racine de l'arbre correspond à un volume englobant qui entoure l'objet en entier. Chaque nœud englobe les volumes englobants de leurs nœuds enfants, jusqu'aux feuilles de l'arbre qui, quant à eux, englobent une ou quelques primitives.

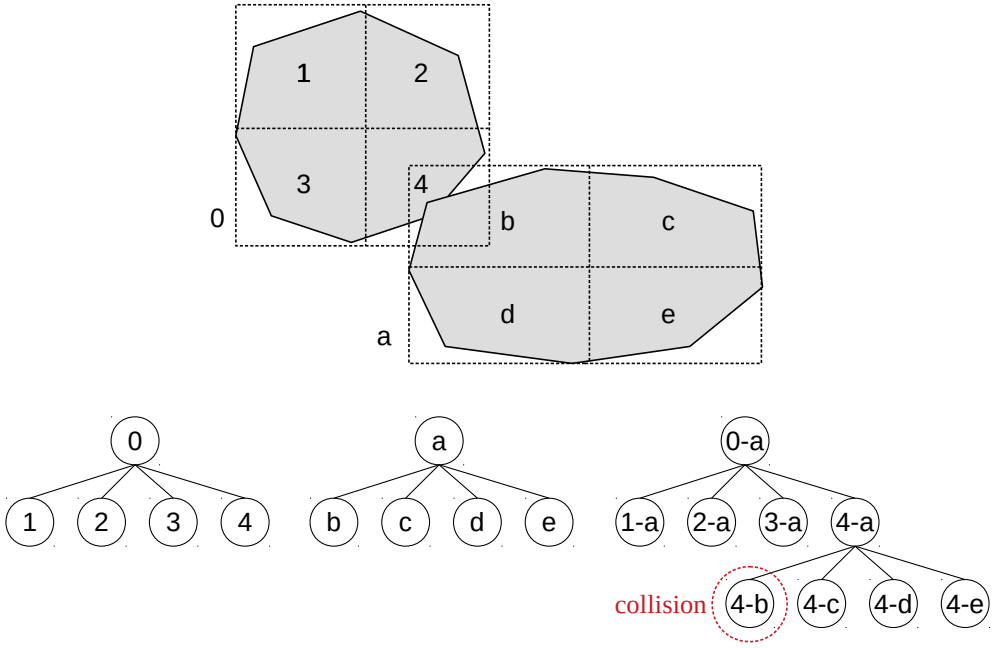


Figure 2.2 Détection de collisions entre deux objets avec des hiérarchies représentées par des arbres quaternaires (en bas à gauche et au centre) et les tests effectués entre les volumes englobants (en bas à droite).

Pour tester si deux objets sont en collision, leurs hiérarchies sont parcourues simultanément de haut en bas en testant si les volumes englobants se chevauchent. Si c'est le cas, leurs nœuds enfants sont testés à leur tour de la même manière. Sinon, on peut conclure que les primitives contenues sous ces nœuds ne sont pas en collision et ignorer le reste de la branche. Si les nœuds qui se chevauchent sont des feuilles, un test de collision est fait entre les primitives contenues dans les volumes englobants correspondants lors de la phase étroite.

Si un seul des deux nœuds est une feuille, les primitives de la feuille sont testées contre toutes les primitives contenues sous l'autre nœud.

Il existe plusieurs possibilités de volumes englobants et les plus courantes sont illustrées à la figure 2.3 : les sphères [13, 14], les boîtes alignées avec les axes (*Axis Aligned Bounding Boxes* ou AABB) [15, 16], les boîtes orientées (*Oriented Bounding Boxes* ou OBB) [17], les polytopes orientés discrets (*discrete oriented polytopes* ou k-DOPs) [18, 19] et les enveloppes convexes minimales.

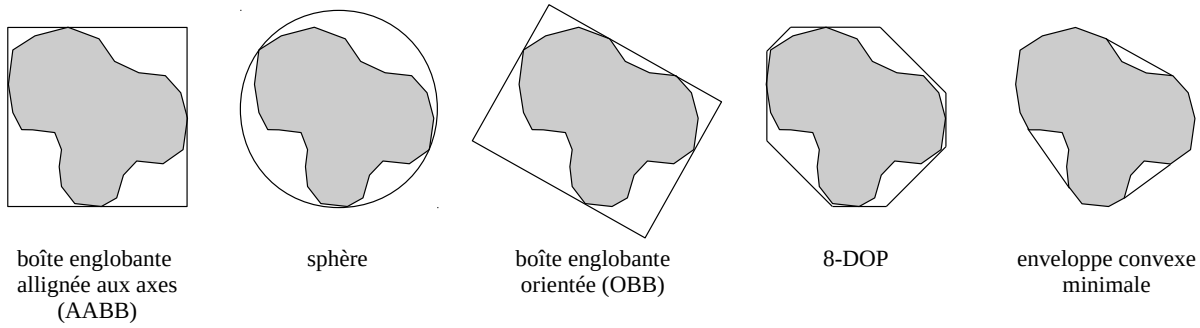


Figure 2.3 Exemples de volumes englobants

Les volumes englobants les plus simples sont plus rapides à construire et à mettre à jour à la suite de déformations. De plus, les tests d'intersections demandent moins de calcul. Cependant, ils englobent généralement les objets moins étroitement et sont donc moins efficaces à réduire le nombre de tests d'intersection de volumes englobants et à élaguer les paires de primitives. Au contraire, les volumes englobants les plus complexes sont souvent meilleurs à réduire le nombre de tests d'intersections, mais sont plus coûteux à construire, à mettre à jour et à tester s'ils sont en intersection.

2.3.1 Détection des auto-collisions

Les hiérarchies de volumes englobants peuvent aussi être utilisées pour les auto-collisions d'objets déformables. Plutôt que de tester les volumes englobants des hiérarchies de deux objets comme décrit plus haut, la hiérarchie est testée contre elle-même. Cependant, les volumes englobants adjacents ont de fortes chances d'être en intersection, diminuant l'efficacité de cette méthode. Pour régler ce problème, plusieurs heuristiques ont été développées [7].

L'une des plus utilisées est la méthode des cônes normaux proposée par Provot [20] où l'absence d'auto-collisions dans la région d'une surface est détectée à partir de sa courbure. L'idée est que si la région n'est pas suffisamment recourbée sur elle-même, il ne peut y avoir

d'auto-collisions. La courbure est évaluée à partir des normales des triangles de la région. Un cône est calculé de façon à inclure la direction de ces normales et son angle d'ouverture est suffisant pour déterminer l'absence d'auto-collisions (voir figure 2.4). Si l'angle d'ouverture α du cône est supérieur à $\pi/2$, il y a possibilité d'intersection. Autrement, aucune auto-collision n'est possible et cette région peut être ignorée.

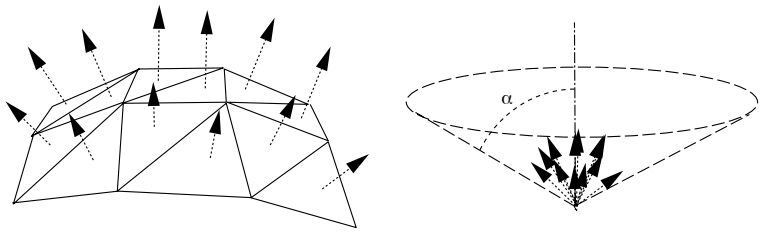


Figure 2.4 Cône normal d'une région de surface [20]. Si $\alpha < \pi/2$, aucune auto-collision n'est possible dans cette région.

Cette méthode est utilisée avec les hiérarchies de volumes englobants pour identifier des branches de l'arbre ne pouvant contenir d'auto-collisions. Les cônes sont construits en parcourant la hiérarchie de bas en haut. D'abord, les feuilles contiennent un seul triangle et donc une seule normale. Un cône est alors construit avec cette normale comme axe central et un angle d'ouverture $\alpha = 0$. Puis, pour chaque nœud, un cône est construit de façon à contenir les cônes des nœuds enfants.

Cette méthode a été étendue à la détection de collisions continues par Tang *et al.* [21]. Dans les feuilles de la hiérarchie, plutôt que commencer avec la normale du triangle et un angle $\alpha = 0$, un cône est construit de façon à contenir la direction de la normale durant tout l'intervalle de temps du pas de simulation.

2.3.2 Mise à jour des volumes englobants après déformations

La détection de collisions avec les hiérarchies de volumes englobants a d'abord été développée pour les collisions entre objets rigides. Avec des objets déformables, une difficulté s'ajoute puisque les volumes englobants doivent être mis à jour régulièrement.

Larsson et Akenine-Möller [22] ont comparé plusieurs heuristiques pour mettre à jour une hiérarchie de volumes englobants avec des AABBs. En particulier, ils ont comparé la mise à jour des nœuds de l'arbre de bas en haut et de haut en bas. Ils ont trouvé que lorsque de nombreux nœuds situés en profondeur dans l'arbre sont atteints au moment de la détection de collisions (parce que leurs volumes englobants se chevauchent), la mise à jour de bas en haut de façon incrémentale est plus efficace. Dans ce cas, les volumes englobants des nœuds feuilles

sont ajustés en fonction de la position des primitives qu'ils englobent, puis les nœuds parents sont ensuite mis à jour un niveau après l'autre. Cependant, lorsque seulement quelques nœuds situés en profondeur sont atteints, la mise à jour de haut en bas, effectuée en même temps que la détection de collisions, est plus efficace. Cette stratégie nécessite de tenir compte de l'ensemble des primitives associées à la branche du nœud mis à jour, mais elle permet de mettre à jour les nœuds qu'au besoin lorsque ceux-ci sont atteints durant la détection de collisions.

Les auteurs ont donc développé une méthode hybride généralement plus rapide que la mise à jour de haut en bas et de bas en haut. Avec cette méthode, la moitié supérieure de l'arbre est mis à jour de bas en haut, et le reste de haut en bas seulement lorsque ces nœuds sont atteints durant la détection de collisions. Les mêmes auteurs [23] ont modifié la méthode pour prendre en compte la cohérence temporelle. Lors de la détection de collisions, les nœuds atteints sont marqués comme «actifs». Au pas de simulation suivant, les nœuds actifs sont mis à jour de bas en haut, et les autres nœuds sont seulement mis à jour de haut en bas lorsqu'ils sont parcourus.

Une autre stratégie utilisée consiste à omettre ou à simplifier la mise à jour des volumes englobants durant plusieurs pas de simulation. C'est le cas dans Mezger *et al.* [24], où les volumes englobants de type k-DOP sont élargis d'une certaine distance et mis à jour que lorsque les sommets de la surface se déplacent davantage que cette distance. Ainsi, la mise à jour est accélérée lorsque les objets se déplacent et se déforment lentement de même que pour les petits pas de simulation.

2.3.3 Mise à jour après changements de topologie

Les hiérarchies de volumes englobants doivent également être mises à jour ou reconstruites à la suite de changements de topologie. C'est le cas lors de la simulation de découpes ou de fractures des objets. En plus de modifier la connectivité de leurs nœuds, les objets peuvent aussi être scindés en deux objets distincts. Pour continuer à élaguer efficacement les paires de primitives de collision, les volumes englobants et leur structure hiérarchique doivent être restructurés. De plus, les coupures et fractures créent de nouvelles surfaces exposées, entraînant la création de nouvelles primitives qui doivent être ajoutées à la hiérarchie de volumes englobants. Plusieurs méthodes ont été proposées pour traiter efficacement cette difficulté.

Otaduy *et al.* [25] présentent une méthode pour restructurer dynamiquement les hiérarchies à la suite de fractures plutôt que de les reconstruire entièrement. Les hiérarchies sont implémentées comme des arbres AVL, des arbres binaires automatiquement équilibrés, avec chaque nœud représentant un volume englobant aligné avec les axes et les feuilles contenant

chacune un triangle de la surface. Lors de fractures, l'arbre est restructuré par l'ajout et la suppression de triangles et des opérations élémentaires maintiennent l'équilibre de l'arbre. Les auteurs montrent que pour les fractures progressives (lorsque moins de 10% des triangles sont affectés), leur méthode est jusqu'à 20 fois plus rapide que la reconstruction complète de l'arbre.

Jung et Lee [26] ont présenté une méthode de découpe de modèle sans maillage où les hiérarchies de volumes englobants sont reconstruites lorsque les objets sont scindés en deux. Pour ce faire, la connectivité des nœuds est enregistrée avec un graphe non dirigé où la connectivité est définie par un critère de visibilité. Lors d'une coupure, le graphe est modifié en éliminant les liens intersectant la surface de découpe. La reconstruction de la hiérarchie est enclenchée lorsqu'un parcours en largeur du graphe, à partir d'un nœud arbitraire, n'atteint pas au moins un nœud.

2.3.4 Parallélisation

Les méthodes de détection de collisions utilisant des structures hiérarchiques consistent essentiellement à parcourir en profondeur les arbres de volumes englobants. Plusieurs algorithmes parallèles pour accélérer le parcours de ces arbres ont été publiés.

Kumar *et al.* [27] ont analysé le parcours d'arbres en parallèle et démontré que l'efficacité du parcours est fortement influencée par le choix de régime de répartition de charge sur les nœuds de calcul et sur l'architecture parallèle employée. Kumar *et al.* [28] ont également analysé l'évolutivité de plusieurs algorithmes de répartition de charge avec des tâches de taille très variable pour différentes architectures parallèles. Deux stratégies de répartition de charge pour le parcours en profondeur d'arbres en parallèle sont présentées par Reinefeld et Schnecke [29]. Ils y ont proposé une méthode où des paquets de travail de taille fixe sont utilisés pour l'équilibrage dynamique de charge.

L'une des difficultés avec la détection de collisions basée sur les hiérarchies de volumes englobants est que les tests d'intersection entre les niveaux supérieurs des hiérarchies offrent peu de parallélisme. Pour augmenter le parallélisme, il est possible d'exploiter la cohérence temporelle avec les fronts d'arbres de tests de volumes englobants (BVTT pour *bounding volume test tree*) initialement proposés par Klosowski *et al.* [18] et utilisés dans plusieurs travaux subséquents [30–33]. Les BVTT sont des arbres représentant toutes les paires de volumes englobants pouvant être testés entre eux. Le front du BVTT consiste en l'ensemble des nœuds où une intersection a été détectée le pas précédent de même que les nœuds adjacents. Plutôt que de recommencer le parcours d'arbre depuis la racine, on commence par les nœuds du front. Ainsi, moins de tests d'intersections de volumes englobants sont effectués et ces

tests peuvent être davantage effectués en parallèle.

Plusieurs auteurs ont présenté des méthodes basées sur la parallélisation hybride sur CPU et GPU. Kim *et al.* [34] ont présenté une telle méthode pour la détection de collisions continue. Le processeur multicœur y est utilisé pour effectuer le parcours des hiérarchies afin d'élaguer les primitives trop éloignées pour être en collision alors que les GPUs sont utilisés pour les tests d'intersection entre les paires de primitives. Les calculs sur CPU et GPU peuvent se chevaucher dans une certaine mesure, ce qui accélère davantage la détection. Leurs résultats ont été calculés avec un CPU Intel i7 quad-core de 3.2 GHz et deux cartes graphiques Geforce GTX285 et présentent des temps de calcul moyen de la détection de collisions de 6.8 à 53.8 ms pour des scènes de 32 000 à 252 000 triangles. La méthode offre une accélération d'un ordre de grandeur par rapport au calcul en série.

Pour accélérer la construction des hiérarchies avec le parallélisme, Lauterbach *et al.* [35] ont présenté la méthode des hiérarchies de volumes englobants linéaires (LBVH, pour *linear bounding volume hierarchy*). Les LBVH sont basés sur la courbe de Lebesgue qui permet de produire un mappage des coordonnées des primitives vers une liste d'index en une dimension. Une fois la liste triée en parallèle, des grappes de primitives rapprochées dans l'espace se trouvent dans des segments contigus de la liste, ce qui sert à les répartir dans les nœuds de l'arbre. Initialement présentée pour le lancer de rayons, la méthode a été adaptée à la détection de collisions [36]. Une fois les primitives réparties dans les différents nœuds de la hiérarchie par la méthode des LBVH, les volumes englobants de chaque nœud sont construits en parallèle de bas en haut.

2.3.5 Synthèse

Les hiérarchies de volumes englobants présentent une des méthodes de détection de collisions les plus populaires et les plus rapides. Cependant, la détection des auto-collisions, la mise à jour après déformations ou changements de topologie, et la parallélisation présentent chacune un niveau de complexité supplémentaire. Plusieurs travaux abordent une ou deux de ces difficultés, mais aucun article trouvé ne présente un algorithme capable de traiter tous ces cas à la fois.

2.4 Partitionnement de l'espace

D'autres méthodes pour accélérer la détection des collisions s'appuient sur le partitionnement de l'espace. Plutôt que de partitionner les primitives des objets comme avec la méthode des hiérarchies de volumes englobants, c'est l'espace de l'environnement simulé qui est subdivisé,

soit selon une grille uniforme ou de manière hiérarchique.

Avec cette approche, l'espace est subdivisé en régions distinctes, ou cellules, et les primitives de collisions sont regroupées selon ces régions pour réduire le nombre de tests à effectuer. Si deux primitives se trouvent dans la même région, ou dans des régions adjacentes, elles sont testées entre elles. Sinon, on conclut qu'elles sont trop éloignées pour être en collision. Parfois, des volumes englobants sont utilisés autour des objets et c'est dans l'intersection de ces volumes que l'espace est partitionné.

2.4.1 Représentation du partitionnement de l'espace

Avec cette approche, il est important d'avoir une structure de données efficace pour représenter le partitionnement de l'espace. Ce choix peut avoir une influence déterminante sur l'efficacité à réduire le nombre de paires de primitives à comparer et la mémoire requise. Plusieurs structures de données ont été proposées à cette fin (voir figure 2.5), notamment les grilles uniformes [37–41], les octrees [42, 43], les arbres k-d [44, 45] et les arbres de partition binaire de l'espace (*Binary Space Partitioning*, ou BSP). Glass [46] offre une analyse détaillée des différentes méthodes de partitionnement de l'espace.

Grilles uniformes : Avec les grilles uniformes, l'espace est subdivisé en cellules de taille égale et les positions des primitives sont discrétisées par rapport à la taille de ces cellules. Pour cela, leurs coordonnées sont divisées par la taille des cellules et arrondies à l'entier. Le résultat est associé à un index 1D par une fonction de hachage qui sert à enregistrer les informations sur les primitives dans une table de hachage. Cette table permet de réduire la mémoire requise. Une fois toutes les primitives enregistrées, celles qui se trouvent dans la même cellule ou des cellules adjacentes sont testées entre elles.

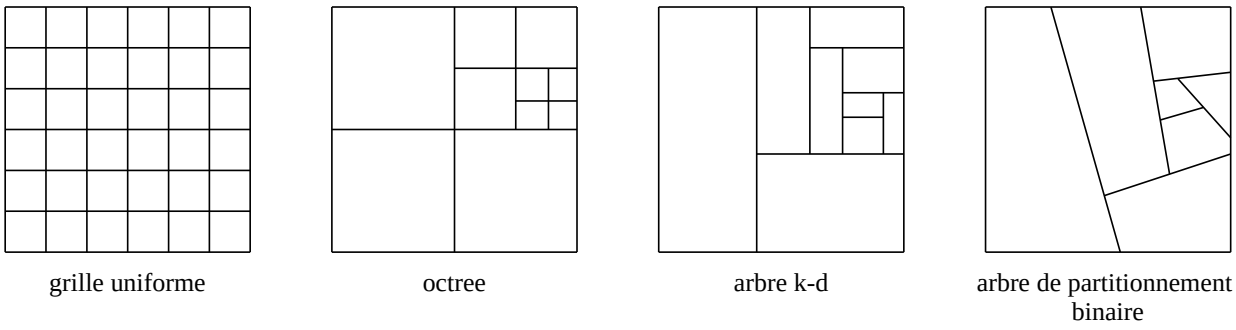


Figure 2.5 Différents types de partitionnement de l'espace

L'efficacité de cette méthode à réduire le nombre de tests de primitives dépend de la taille des cellules, de la fonction de hachage et de la taille de la table de hachage. Par exemple, si un grand nombre de primitives se trouvent très près l'une de l'autre, certaines cellules pourraient contenir un grand nombre de primitives à tester et l'avantage du partitionnement de l'espace serait limité. Une limitation des grilles régulières est donc qu'elles ne s'adaptent pas à la position des primitives dans l'environnent. Les techniques suivantes tentent de régler ce problème en partitionnant davantage les régions de l'espace les plus peuplées.

Octrees : Les octrees peuvent être utilisés pour faire une subdivision multirésolution de l'espace. Avec cette méthode, la racine de l'arbre représente l'espace à subdiviser. Cet espace est subdivisé en huit régions et si plusieurs primitives se trouvent dans une de ces régions, celle-ci est à son tour subdivisée en huit. Ce partitionnement continue récursivement jusqu'à ce qu'une profondeur prédéfinie soit atteinte, ou jusqu'à ce que les régions ne possèdent qu'une seule primitive [46].

Arbres k-d : Les arbres k-d sont des arbres binaires permettant aussi un partitionnement récursif de l'espace. Cette fois-ci, pour chaque niveau de l'arbre, l'espace est subdivisé de part et d'autre d'un plan perpendiculaire à l'un des trois axes. Comme avec les octrees, la racine représente l'espace à subdiviser. Pour construire les niveaux suivants de l'arbre, on doit choisir l'orientation et la position du plan de partitionnement. Il existe plusieurs façons de faire ce choix, comme prendre la médiane le long de l'axe ou la position qui produit l'arbre le plus équilibré [46]. Les primitives sont attribuées à l'une des deux branches selon de quel côté de la partition elles se trouvent. Puis, les partitions non vides sont à leur tour partitionnées en deux jusqu'à ce qu'une condition soit remplie, comme un nombre minimum de primitives dans une partition. Une fois toutes les primitives placées, celles qui se trouvent dans la même partition sont testées entre elles.

Arbre de partition binaire (*BSP-tree*) : La partition binaire de l'espace est une généralisation des arbres k-d où le plan de partitionnement peut être orienté arbitrairement. Le choix des plans de partitionnement est plus compliqué et peut être fait selon divers critères. Luque *et al.* [45] ont présenté un algorithme où les plans sont choisis selon un critère basé sur le nombre d'objets contre lesquels le plan doit être testé, le ratio d'objets de part et d'autre du plan et le nombre d'objets contenu des deux côtés à la fois. Ce calcul ajoute une complexité à la détection de collisions, mais ce problème est mitigé en prenant en compte la cohérence temporelle et en modifiant les partitions seulement lorsque les objets se sont déplacés considérablement.

2.4.2 Parallélisation

Plusieurs implémentations sur GPU de la détection de collisions basée sur le partitionnement de l'espace ont été proposées. Le Grand [47] a développé une méthode avec CUDA qui utilise une grille uniforme avec une table de hachage pour élaguer rapidement les paires d'objets n'étant pas en collision dans un modèle de particules. La méthode est un ordre de grandeur plus rapide que son implémentation sur CPU.

Liu *et al.* [48] ont présenté un algorithme implémenté entièrement sur GPU pour élaguer les collisions entre un très grand nombre d'objets. La méthode combine l'algorithme de *sweep and prune* avec un partitionnement de l'espace à deux niveaux pour mitiger la grande densité d'objets pouvant être projetés sur un même intervalle d'un axe. Pour ce faire, l'algorithme de *sweep and prune* est appliqué simultanément à de nombreux objets en détectant en parallèle les collisions dans une même cellule ou dans des cellules adjacentes. De plus, une analyse en composantes principales est utilisée pour déterminer la meilleure direction de balayage de l'algorithme de *sweep and prune* afin d'accélérer encore davantage la détection de collisions.

Pabst *et al.* [41] ont proposé un algorithme hybride CPU et GPU basé sur le partitionnement de l'espace pour la détection de collisions entre objets rigides et déformables. La méthode utilise toujours une grille uniforme avec une table de hachage, mais elle est étendue aux maillages de triangles plutôt qu'aux particules. Avant le partitionnement de l'espace, les objets actifs sont identifiés par un test de chevauchement de volumes englobants de type k-DOP. Ensuite, des sphères englobantes sont calculées pour chaque triangle. La taille des cellules de la grille de partitionnement de l'espace est calculée à partir du rayon de la sphère la plus grande, plus un epsilon déterminé par l'utilisateur. La phase étroite est basée sur les tests élémentaires sommet-triangle et arête-arête. L'algorithme gère aussi bien la détection de collisions discrète que continue de même que les auto-collisions.

Sa performance a été mesurée avec un processeur quad-core Intel Core i7 à 3,2GHz et un à quatre GPUs NVidia GT X295. Avec un seul GPU, les collisions dans des scènes de 7600 à 146 000 triangles sont calculées entre 6.7 et 128.1 ms pour la détection discrète et entre 5.9 et 184.8 ms pour la détection continue. Avec 4 GPUs, le temps de calcul est diminué à entre 5.4 et 39.9 ms pour la détection discrète et entre 4.9 et 77.7 ms pour la détection continue. L'algorithme est bien adapté aux architectures multi-CPU et multi-GPU. Cependant, puisqu'elle est basée sur un partitionnement de l'espace avec une grille uniforme, elle est moins efficace à paralléliser des modèles avec des tailles de triangles très variables.

Pour répondre à cette limitation, Wong *et al.* [49] ont proposé une méthode basée sur le concept de « triangles virtuels ». L'idée est de subdiviser les triangles les plus grands en

plusieurs triangles de plus petite taille. Cette subdivision est dite « virtuelle » parce que ces nouveaux triangles sont utilisés uniquement pour la détection de collisions et sont recalculés à chaque pas de simulation. Cette méthode permet de réduire considérablement le nombre de tests de collisions et la mémoire requise sur le GPU, et donc d'augmenter la performance de l'algorithme.

Finalement, les mêmes auteurs [43] ont présenté un autre algorithme entièrement sur GPU pour pallier les problèmes de taille et de distribution variable des triangles inhérents aux méthodes basées sur les grilles uniformes. Cette fois-ci, l'algorithme est basé sur les octrees pour adapter le partitionnement de l'espace de façon hiérarchique. La construction, la mise à jour et le parcours d'arbres en parallèle requièrent des opérations complexes offrant une répartition du travail en parallèle relativement faible. Pour cette raison, l'octree est représenté sur GPU par une « grille d'octree », c'est-à-dire un tableau de taille égale au nombre de cellules de l'octree en entier.

Les auteurs présentent une méthode pour construire l'octree en choisissant le niveau de partitionnement de chaque branche de façon à minimiser le nombre de tests élémentaires. Plutôt que de supprimer ou d'ajouter des nœuds, l'arbre est construit en marquant les nœuds des différents niveaux comme « feuille », « interne » ou « inactif ». Une fois construite, cette grille d'octree permet le mappage des triangles à la cellule du niveau où les tests seront effectués. Ensuite, le procédé de distribution des tâches présenté par Fan *et al.* [50] est utilisé pour répartir les tests sur les paires de triangles entre les fils d'exécution et éviter les tests redondants.

La performance de l'algorithme a été mesurée sur une carte NVIDIA GTX 780 et les collisions dans des scènes de 20 000 à 146 000 triangles sont calculées entre 3,2 à 20,8 ms par pas. Maintenir le tableau de la grille d'octree requiert beaucoup de mémoire, mais puisque leur algorithme est entièrement sur GPU, les données sont transférées du CPU au GPU qu'une seule fois au début de la simulation.

2.4.3 Synthèse

Les méthodes de détection de collisions basées sur le partitionnement de l'espace ont comme avantage que les structures de données générées sont indépendantes de la topologie des objets, ce qui peut être avantageux pour la simulation avec changement de topologie. De plus, les techniques de partitionnement de l'espace s'adaptent bien à la parallélisation.

2.5 Champ de distances

Une autre approche pour l'accélération de la détection de collisions est d'utiliser des champs de distances où la distance à la surface des objets est précalculée de façon à y avoir accès rapidement durant la simulation.

Un champ de distances est une fonction scalaire représentant la distance minimale d'une surface fermée à chaque point de l'espace. Cette structure de données est précalculée lors de l'initialisation pour une certaine région autour de la surface. De plus, l'intérieur de l'objet peut être représenté par des valeurs négatives pour indiquer la distance de pénétration. Ainsi, durant la simulation, les informations nécessaires à la réponse aux collisions, comme la distance à l'objet, la normale et la distance de pénétration, sont obtenues très rapidement.

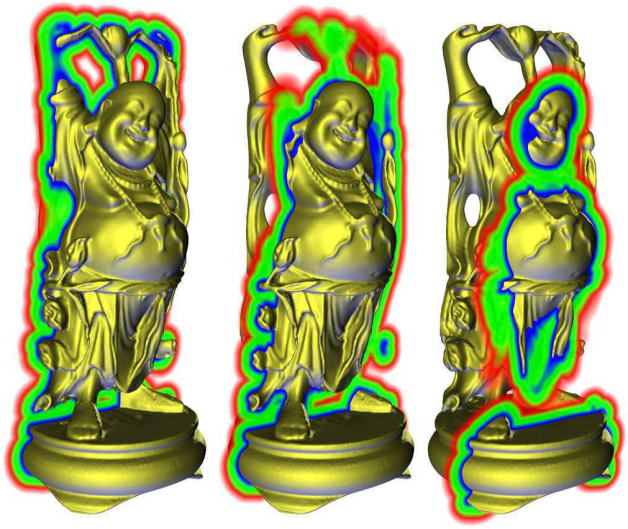


Figure 2.6 Exemples de champ de distances autour du «Happy Buddha» [7]

2.5.1 Représentation du champ de distances

L'un des choix importants pour la détection de collisions avec les champs de distances est la façon de représenter ces champs. Les structures de données utilisées sont généralement les mêmes que celles présentées pour le partitionnement de l'espace à la section 2.4.1, soit les grilles uniformes, les octrees et les arbres de partition binaire de l'espace [7]. Ces structures permettent de représenter la distance à la surface aux divers points du champ. La distance entre ces points peut ensuite être obtenue par interpolation.

Les grilles uniformes permettent de représenter la distance à la surface aux divers points du champ puis d'évaluer la distance à un point quelconque par interpolation trilinéaire. Cette

méthode est facile à implémenter, rapide et fonctionne bien avec des surfaces relativement lisses. Cependant, avec des surfaces plus irrégulières, par exemple avec des régions pointues ou dentelées, la précision des distances peut être insuffisante à moins d'augmenter la résolution de la grille. Cela peut être coûteux en mémoire et en temps de construction de la grille.

Pour pallier ce problème, des techniques ont été proposées afin d'adapter la résolution d'échantillonnage du champ de distances [51]. Avec les octrees, les cellules sont subdivisées davantage près des parties de la surface possédant plus de détails. Elles sont subdivisées tant que l'interpolation trilinéaire n'approxime pas suffisamment bien la distance à la surface. Cette structure peut demander beaucoup moins de mémoire qu'une grille uniforme et permet de représenter avec précision la distance à des surfaces très irrégulières.

Quant à eux, les arbres de partition binaire de l'espace permettent de représenter le champ de distances par un ensemble d'approximations linéaires. Les plans de partitionnement séparent des régions à l'intérieur desquelles la distance à la surface est approximée par une fonction linéaire en trois dimensions. Plusieurs algorithmes pour choisir les plans de partitionnement sont présentés par Wu et Kobbelt [52]. Cette méthode permet de compresser le champ de distances encore davantage que les octrees [52], mais la construction des arbres est coûteuse en temps de calcul.

2.5.2 Construction du champ de distances

Plusieurs méthodes ont été proposées pour calculer la distance minimale à la surface durant la construction du champ de distances [53]. Par exemple, certaines méthodes utilisent une organisation hiérarchique de la surface pour accélérer le calcul de la distance à chaque point du champ. D'autres méthodes consistent à propager la distance entre la surface et des voxels depuis les voxels voisins.

2.5.3 Synthèse

Les champs de distances permettent de gérer la détection des collisions très efficacement, mais demandent un pré-calcul important pour la construction du champ. Mettre à jour le champ de distances est également très coûteux. Pour cette raison, cette méthode est peu utilisée pour la détection de collisions entre objets déformables en temps réel [7].

Cependant, cette méthode peut être appropriée pour détecter les collisions d'un objet rigide, comme un outil chirurgical, avec des objets déformables. Cela est dû au fait qu'un champ de distances est seulement nécessaire pour l'un des deux objets. Ainsi, un champ peut être construit pour l'objet rigide et les distances entre les primitives d'une surface déformable et

l'objet rigide sont ensuite obtenues rapidement.

2.6 Méthode d'espace image

Les dernières méthodes de détection de collisions présentées dans ce chapitre sont celles basées sur le concept de l'espace image. Avec ces méthodes, le matériel graphique est utilisé pour détecter les collisions et approximer rapidement le volume d'intersection.

Ces techniques utilisent la rastérisation d'une projection en 2D de la surface des objets pour détecter les collisions. Elles ont surtout été appliquées à des maillages triangulaires de surface. Elles sont particulièrement appropriées pour être implémentées sur le matériel graphique et ne requièrent aucun pré-calcul.

Lombardo *et al.* [54] ont proposé une première application de la technique d'espace image appliquée à la simulation de chirurgie. Le matériel graphique est utilisé pour détecter en temps réel la pénétration d'un outil rigide avec un organe déformable. Leur méthode est similaire au processus de rendu graphique en infographie. Ce dernier utilise un volume de visualisation basé sur la position, l'orientation et la projection de la caméra et produit un rendu de l'intersection de l'objet avec le volume de visualisation. Pour détecter les collisions, la forme de l'outil est utilisée comme volume de visualisation (ou le volume couvert par l'outil entre deux pas consécutifs pour la détection de collisions continue). Ainsi, les faces avec lesquelles l'outil est en contact sont identifiées.

Heidelberger *et al.* [55] ont proposé une autre technique d'espace image pour la détection de collisions entre objets déformables. Cette technique emploie la méthode des *Layered Depth Images* (LDI) [56]. Cette méthode consiste à représenter la surface d'un objet par des images rastérisées, ou couches, où la valeur des pixels représente la profondeur de la surface dans une direction donnée. Ces valeurs de profondeur peuvent être interprétées comme l'intersection de rayons parallèles entrant et sortant de l'objet.

La méthode présentée par Heidelberger *et al.* consiste en trois étapes. Premièrement, chaque objet est contenu dans une boîte alignée avec les axes et les intersections de ces volumes sont détectées pour identifier les objets potentiellement en contact. Le volume d'intersection de deux volumes englobants sert de région d'intérêt dans les étapes suivantes. Deuxièmement, une représentation LDI de la surface dans la région d'intérêt est calculée pour chacun des deux objets. Troisièmement, les LDI sont utilisés pour faire deux tests. D'abord, pour tester l'existence de collisions, les deux LDI sont combinées pour calculer une approximation du volume d'intersection. Les pixels des LDI correspondent à l'étendue du volume d'intersection et permettent d'obtenir une représentation discrétisée de ce volume. Si ce volume n'est pas

vide, il y a collision. Ensuite, un autre test permet d'identifier les sommets des objets ayant pénétré dans l'autre objet. Les coordonnées des sommets sont transformées dans le système de coordonnées des LDI pour détecter lesquels se trouvent à l'intérieur du volume d'intersection. Cette technique permet de détecter les collisions entre objets déformables efficacement, mais pas les auto-collisions.

Les mêmes auteurs [57] ont présenté une version améliorée de la méthode qui permet de détecter également les auto-collisions. Pour y arriver, les LDI sont calculées en prenant en compte l'orientation des faces des objets. Pour tester les auto-collisions pour un objet, le volume d'intérêt choisi est le volume englobant de l'objet en entier. Dans la deuxième étape, l'orientation des faces est utilisée pour analyser l'ordre des entrées et des sorties de l'objet. Si les entrées et sorties alternent correctement, il n'y a pas d'auto-collision. Si, au contraire, l'ordre des entrées et sorties est invalide, il y a auto-collision et le volume d'intersection peut être calculé comme pour les collisions entre deux objets différents.

Ces deux derniers articles présentent une méthode pour calculer les volumes d'intersection à l'aide des LDI, mais la réponse aux collisions n'y est pas traitée. Faure *et al.* [58] ont proposé une méthode pour calculer les forces de répulsion à l'aide du volume d'intersection et de son gradient (voir figure 2.7). Pour ce faire, des LDI sont construits pour trois directions orthogonales. Ensuite, le volume d'intersection et la dérivée du volume par rapport aux coordonnées des sommets sont calculés. Ces valeurs permettent de calculer les forces de contact par pixel qui sont ensuite réparties aux sommets. Cette méthode permet de calculer aussi bien les forces de répulsion que de friction. Allard *et al.* [59] ont proposé une méthode similaire, mais des contraintes de non-pénétration et de mouvement tangentiel (friction) sont calculées plutôt que des forces de pénalité pour gérer la réponse aux collisions.

Les méthodes d'espace image ne nécessitent aucun pré-calcul, les rendant particulièrement adaptées pour la simulation d'objets déformables. De plus, ces techniques n'ont pas à tenir compte de la topologie des surfaces. La simulation de coupures ne cause donc pas de difficultés additionnelles. Finalement, ces méthodes sont particulièrement adaptées pour le calcul sur GPU. Cependant, la précision et l'efficacité de ces méthodes sont limitées par la résolution de l'espace image.

2.7 Collision dans SOFA

L'algorithme de détection de collisions développé pour ce projet, décrit au chapitre suivant, est implémenté dans la librairie Scyther qui utilise la plateforme logicielle SOFA pour être testé. De plus, les performances de la méthode proposée seront comparées à l'algorithme de

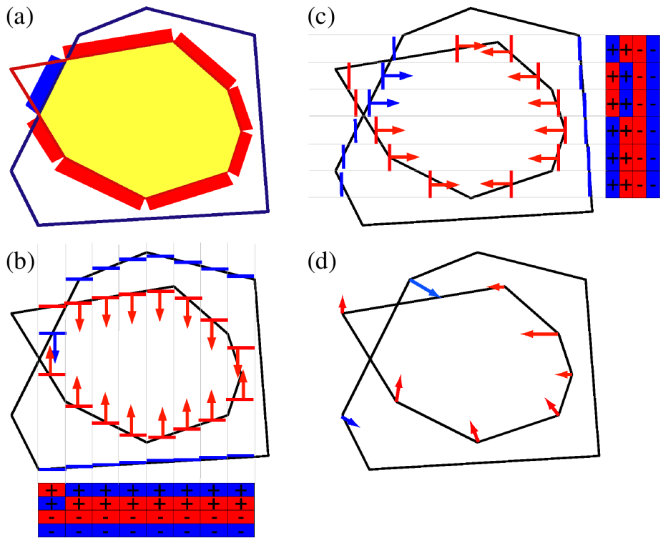


Figure 2.7 Aperçu de la détection d'intersections et de la réponse avec la méthode des LDI. (a) Deux objets en intersection, avec le volume d'intersection en jaune et les forces de pressions aux surfaces représentées par les rectangles rouges et bleus. (b) Rastérisation des surfaces dans la direction verticale. Les barres rouges et bleues dénotent les intervalles utilisés pour le calcul du volume d'intersection et les flèches montrent les forces de répulsion. En bas, les signes dénotent la direction de la normale de la surface aux entrées et sorties des objets. (c) Rastérisation des surfaces dans la direction horizontale. (d) Les forces une fois appliquées aux sommets. [58]

détection de collisions de SOFA. Il est donc utile de décrire brièvement la façon dont les collisions sont traitées dans SOFA.

Dans la boucle d'animation de SOFA, les collisions sont traitées séparément du reste de la simulation physique, généralement avant l'exécution des solutionneurs qui calculent la dynamique des objets. Le pipeline de collisions est lui-même séparé en trois parties : la réinitialisation, la détection et la réponse, chacune déclenchée par un visiteur. La première retire les réponses aux collisions détectées au pas précédent, la deuxième détecte les nouvelles collisions alors que la dernière calcule les réponses qui seront utilisées par le solutionneur.

Chaque objet pouvant entrer en collision doit être associé à une ou plusieurs structures géométriques, appelées modèle de collision, dédiées à la détection de collisions et mappées aux degrés de liberté de l'objet. Ces structures peuvent être constituées des éléments d'un maillage de surface (faces, sommets et arêtes des triangles) ou de volumes englobants. Dans le cas d'un maillage, celui-ci est différent du maillage utilisé pour le rendu visuel et peut avoir une résolution différente, permettant d'équilibrer la précision et la rapidité de la détection de collisions indépendamment du rendu visuel.

La détection de collisions prend comme entrée les modèles de collisions et retourne les paires de primitives en collision avec les points de contact associés. Cette information est passée au gestionnaire des contacts pour générer la réponse aux collisions.

Plusieurs méthodes de détections sont implémentées dans SOFA :

***Brute Force* :** Cette méthode s'appuie sur le concept des hiérarchies de volumes englobants décrit dans la section 2.3. Une hiérarchie de boîtes englobantes alignées avec les axes est créée pour chaque objet. À la base de la hiérarchie se trouvent les primitives de collision. Pour des objets déformables, ceux-ci sont le plus souvent les points, arêtes et faces des triangles du maillage de surface.

***Sweep and prune* :** Cette méthode implémente l'algorithme de *Sweep and prune* décrit à la section 2.2. Elle permet de détecter rapidement les collisions entre des boîtes englobantes alignées aux axes.

***Ray tracing* :** Cette méthode est celle du lancer de rayons décrite par Hermann *et al.* [60]. Avec cette méthode, un rayon est lancé à partir de chaque sommet de la surface dans la direction opposée à sa normale, soit vers l'intérieur de l'objet. Un contact est détecté lorsque la première intersection du rayon se fait avec la surface d'un autre objet. Pour accélérer la recherche d'éléments en intersection avec le rayon, les triangles de chaque objet sont organisés dans un octree.

La réponse aux collisions peut être faite avec une de deux méthodes : soit en introduisant des forces de pénalité ou des contraintes utilisant les multiplicateurs de Lagrange. Ceux-ci sont ensuite traités par le solutionneur avec les autres forces externes et contraintes.

2.8 Objectifs et hypothèses

2.8.1 Conclusion sur l'état de l'art

Au vu de l'état de l'art, les méthodes basées sur le partitionnement de l'espace semblent les plus appropriées pour la détection de collisions dans la librairie de simulation d'objets déformables et de découpe Scyther.

Les méthodes basées sur le partitionnement topologique des objets, comme la hiérarchie de volumes englobants (section 2.3), semblent les plus utilisées et les résultats publiés suggèrent qu'elles sont parmi les plus efficaces. Cependant, aucune étude trouvée ne prend en compte à la fois la difficulté des auto-collisions, des déformations, des changements de topologie et

de la parallélisation. Chacune de ces difficultés ajoute un coût de calcul et une complexité d'implémentation importante. Quant à elle, la méthode des champs de distances (section 2.5) est peu appropriée pour des objets déformables. Finalement, la méthode des *Layered Depth Images* (section 2.6) est une méthode approximative. De plus, l'un des principaux avantages de cette méthode est qu'elle permet d'exploiter le parallélisme sur GPU en utilisant les librairies d'affichage graphique comme OpenGl. Cependant, avec la popularité et l'accessibilité grandissantes du calcul générique sur processeur graphique au cours des dernières années, cet avantage est aujourd'hui moins pertinent.

Les méthodes basées sur le partitionnement de l'espace (section 2.4) ont plusieurs avantages pour la détection de collisions dans Scyther comparativement aux autres méthodes. D'abord, la méthode est indépendante de la topologie des objets et devrait donc être facilement adaptable à la simulation de découpes. Ensuite, elle est en principe facilement parallélisable, ce qui semble nécessaire pour exécuter la détection de collisions dans des environnements de simulation complexe en temps réel.

2.8.2 Objectif général

L'objectif général de cette étude est de concevoir et implémenter un algorithme de détection de collisions dans la librairie Scyther et son plugin d'interface à SOFA. Cet algorithme doit permettre la détection de collisions en temps réel aussi bien entre objets déformables qu'avec un outil de découpe.

2.8.3 Objectifs spécifiques

L'objectif général est décomposé en cinq objectifs spécifiques :

1. Concevoir et implémenter un algorithme de détection de collisions basée sur le partitionnement de l'espace dans Scyther.
2. Mesurer l'accélération de la détection de collisions produite par différentes étapes d'élagage de l'algorithme pour tester leur efficacité.
3. Comparer la performance de l'algorithme avec celle de la détection de collisions de SOFA.
4. Démontrer que la méthode choisie est adaptée aux changements de topologie avec l'algorithme de découpe de Scyther.
5. Paralléliser l'algorithme sur CPU multicœurs avec OpenMP et mesurer l'accélération.

2.8.4 Hypothèses

Les objectifs spécifiques ont pour but de confirmer ou d'infirmer trois hypothèses. D'abord, pour que la simulation soit interactive, la détection de collisions doit fonctionner en temps réel. Comme mentionné à la section 2.1.3, un rendu visuel entre 20 et 60 Hz est acceptable pour que la simulation paraisse fluide. Cette fréquence correspond à un temps de 17 à 50 ms par pas de simulation, comprenant toutes les étapes de la simulation dont la déformation et le déplacement des objets et la détection des collisions.

Le calcul en parallèle pourrait être nécessaire pour parvenir à une telle fréquence. La détection de collisions par partitionnement de l'espace est bien adaptée à la parallélisation et la solution implémentée pour ce travail sera testée sur un CPU à six cœurs. Ainsi, les deux premières hypothèses sont :

H1 La parallélisation sur six cœurs réduit le temps de calcul d'au moins de moitié.

H2 Un temps de calcul total d'au plus 50 ms par pas est possible pour des scènes de plusieurs dizaines de milliers de triangles.

La méthode de partitionnement de l'espace a été privilégiée parce qu'elle ne semble pas nécessiter d'adaptation pour fonctionner avec les changements de topologie lors de la découpe. Ainsi, la troisième hypothèse est :

H3 Les changements de topologie ne requièrent pas d'étapes additionnelles et ne causent pas de coût de calcul supplémentaire à la détection de collisions basée sur le partitionnement de l'espace.

CHAPITRE 3 MÉTHODOLOGIE

Ce chapitre présente tout d'abord la solution implémentée dans Scyther pour la détection de collisions. La section 3.1 donne un aperçu de l'algorithme alors que les sections 3.2 à 3.9 présentent ses différentes étapes en détail de même que la réponse aux collisions utilisée dans SOFA et la parallélisation de l'algorithme avec OpenMP. Ensuite, la section 3.10 présente les expériences réalisées pour vérifier l'atteinte des objectifs et tester les hypothèses présentées à la section 2.8.

3.1 Aperçu de l'algorithme

L'algorithme implémenté permet la détection des collisions entre objets et des auto-collisions. Comme expliqué à la section 2.8, l'approche retenue est celle du partitionnement de l'espace.

La méthode est principalement inspirée de celles de Le Grand [47] et Pabst *et al.* [41] d'où sont empruntées la technique de partitionnement par hachage spatial ainsi que l'approche pour éviter de dupliquer les tests de collisions entre deux triangles lorsque ceux-ci se rencontrent dans plusieurs cellules. Pour accélérer la détection des auto-collisions, la méthode des cônes normaux de Provot [20], mentionnée dans la section 2.3.1, a été adaptée pour élaguer les cellules de la grille de partitionnement selon un critère de courbure de la surface. De plus, une méthode a été conçue pour assigner les primitives de collisions (sommets et arêtes) à un seul triangle pour éviter de tester une même paire plusieurs fois. Finalement, la réponse aux collisions est gérée par la méthode des forces de pénalités implémentée dans SOFA.

L'algorithme 1 présente la méthode implémentée. Au début de chaque pas de simulation, les objets potentiellement en collision sont identifiés par un test d'intersection de leur volume englobant. L'intersection des deux volumes englobants constitue la région d'intérêt. Ce sont les triangles présents dans cette région qui sont considérés dans les étapes suivantes. Ensuite, la grille de partitionnement de l'espace est définie et les triangles sont associés aux cellules par une fonction de hachage de leurs coordonnées. De cette manière, une liste de triangles avec les cellules dans lesquels ils résident est construite. Cette liste est triée par ordre de valeur de hachage des cellules et les paires de triangles potentiellement en collisions dans chaque cellule sont identifiées par un test d'intersection de leur volume englobant. Finalement, les collisions entre les triangles sont détectées par des tests de proximité triangle-sommet et arête-arête.

Deux complications surviennent avec la méthode de partitionnement de l'espace. La première est qu'il faut éviter de tester une paire de triangles plusieurs fois. Cela peut survenir parce

Algorithme 1 Détection de collisions

```
mettre à jour les volumes englobants des objets
pour chaque paire d'objets faire
  // Étape 1 :
  tester l'intersection des volumes englobants des objets
  déterminer la région d'intérêt
  si une intersection est détectée alors
    // Étape 2 :
    trouver les triangles dans la région d'intérêt
    // Étape 3 :
    définir la grille
    // Étape 4 :
    construire la liste des paires cellule-triangle
    // Étape 5 :
    trier la liste des paires cellule-triangle
    // Étape 6 : Construire la liste de paires de triangles potentiellement en collision
    pour chaque cellule faire
      si auto-collisions alors
        effectuer le test des cônes normaux
      fin si
      pour chaque paire de triangles faire
        élaguer par type de cellule
        élaguer par test de volume englobant
        ajouter la paire de triangles à la liste
      fin pour
    fin pour
    // Étape 7 : Tests de paires de primitives
    pour chaque paire de triangles faire
      effectuer les tests de proximité sommet-triangle et arête-arête
      ajouter les paires de sommet-triangle et arête-arête en contact au vecteur de contacts
    fin pour
  fin si
fin pour
```

qu'un triangle peut chevaucher plusieurs cellules. Il faut donc un mécanisme pour éviter de dupliquer les tests de collision de triangles. La deuxième complication est qu'il faut éviter de tester une paire de primitives (triangle-sommet ou arête-arête) plusieurs fois. Puisque les sommets et arêtes sont partagés entre plusieurs triangles, une paire de primitives peut se présenter à travers plusieurs paires de triangles. Encore une fois, un mécanisme doit permettre de ne tester les paires de primitives qu'une seule fois. Les solutions à ces deux complications sont présentées aux sections 3.6 et 3.7 respectivement.

Les méthodes d'accélération de la détection de collisions fonctionnent essentiellement en réduisant le nombre de paires de triangles et de primitives à tester. Pour cette raison, plusieurs étapes sont ajoutées pour élaguer davantage ces paires avec des tests rapides à exécuter. Notamment, des tests d'intersection de volumes englobants permettent d'éliminer les paires de primitives trop éloignées pour être en collision avant de calculer leur proximité. De plus, la méthode des cônes normaux a été implémentée. Elle a été adaptée pour éliminer des cellules entières lors de la détection des auto-collisions d'après la normale des triangles présents dans chaque cellule.

Les sections suivantes présentent l'implémentation de l'algorithme en détail. D'abord, la section 3.2 décrit le test d'intersection des volumes englobants pour identifier les objets potentiellement en collision et définir la région d'intérêt. Ensuite, la section 3.4 décrit l'implémentation du partitionnement de l'espace dont la définition de la grille. Puis, la section 3.5 décrit la répartition des triangles dans la grille avec la fonction de hachage afin de construire la liste des paires cellule-triangle. La section 3.6 décrit la façon dont les paires de triangles à tester sont identifiées dans chaque cellule. La section 3.6.1 décrit la méthode des cônes normaux telle qu'adaptée au partitionnement de l'espace. Finalement, la section 3.7 décrit les différents tests de collision entre les primitives pour calculer la liste des contacts.

3.2 Test d'intersection des volumes englobants des objets et région d'intérêt (étape 1)

La première étape de l'algorithme est le test d'intersection des volumes englobants des objets. Chaque objet est muni d'une boîte englobante alignée avec les axes qui est mise à jour à chaque pas de la simulation afin de l'ajuster à l'objet lorsque celui-ci s'est déplacé. Ce test permet d'éliminer rapidement les paires d'objets trop éloignés pour être en collision et de sauter le reste de la détection pour cette paire d'objets.

À chaque mise à jour des boîtes, leur taille est augmentée d'une demi-fois la distance de contact de chaque côté tel qu'illustré à la figure 3.1. La distance de contact est la distance à laquelle les primitives sont considérées en contact et est un paramètre défini par l'utilisateur. Augmenter la taille de chacune des boîtes d'une demi-fois la distance de contact assure que, si la distance minimale entre deux objets est inférieure à la distance de contact, leurs boîtes seront nécessairement en intersection et les contacts entre les objets pourront être détectés.

Durant cette étape, la région d'intérêt est évaluée. Elle correspond au volume d'intersection des deux boîtes englobantes. Cette intersection est identifiée par comparaison des bornes inférieures et supérieures des deux boîtes le long de chacun des axes. Ce sont uniquement

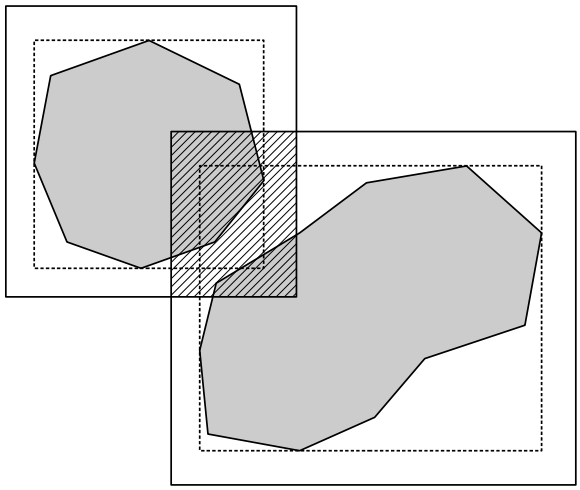


Figure 3.1 Exemple en 2D du test d'intersection des volumes englobants entre deux objets et de la région d'intérêt. Les boîtes englobantes initiales (lignes en pointillé) sont augmentées de chaque côté par la moitié de la distance de contact (ligne pleine) et leur intersection constitue la région d'intérêt (zone hachurée).

les triangles des surfaces se trouvant dans cette région qui sont considérés par le reste de l'algorithme. Pour les auto-collisions, aucun test d'intersection de volumes englobants n'est nécessaire. La région d'intérêt correspond à la boîte englobante de l'objet en entier et la détection doit être exécutée à chaque pas.

3.3 Détection des triangles dans la région d'intérêt (étape 2)

Les triangles se trouvant dans la région d'intérêt sont identifiés par un test d'intersection entre la région et les volumes englobants des triangles. Puisque de nombreux triangles peuvent être présents dans la région d'intérêt, de nombreux tests d'intersection doivent être exécutés. De plus, les volumes englobants des triangles sont réutilisés dans les étapes subséquentes où, encore une fois, de nombreux tests doivent être exécutés. Pour cette raison, les sphères englobantes et des boîtes englobantes alignées avec les axes ont été choisies. Due à leur simplicité, ce sont les volumes les plus rapides à construire et à en tester l'intersection. L'impact du choix entre ces deux volumes sur le temps de calcul de la détection de collisions sera comparé dans les expérimentations.

Comme pour les objets, la taille des volumes englobants des triangles est augmentée d'une demi-fois la distance de contact de chaque côté. Ainsi, les volumes sont nécessairement en intersection si les triangles qu'ils englobent sont à moins d'une fois la distance de contact.

Cette étape produit la liste de triangles qui seront considérés dans les étapes suivantes. Pour les auto-collisions, ces tests d'intersection ne sont pas nécessaires puisque tous les triangles de l'objet sont dans la région d'intérêt. La liste de tous les triangles de l'objet est alors utilisée.

En plus de produire la liste de triangles à tester, cette étape identifie la taille du plus grand volume englobant de triangle afin de définir la taille des cellules à l'étape suivante. Cette taille correspond au diamètre pour les sphères ou à la longueur du plus long côté pour les boîtes.

3.4 Partitionnement de l'espace (étape 3)

Le partitionnement de l'espace consiste à partitionner la région d'intérêt en une grille où, à chaque cellule de la grille, est associée la liste des triangles qui l'intersectent. La méthode de subdivision retenue est la grille uniforme où la taille des cellules est au moins égale au plus grand volume englobant de triangle. Ceci assure que chaque triangle ne peut chevaucher plus de deux cellules dans une direction donnée pour un maximum de huit cellules ($2 \times 2 \times 2$).

Comme expliqué à la section 2.4.1, les grilles uniformes peuvent être moins efficaces lorsqu'il y a une grande disparité de taille de triangles. Cependant, les maillages de surface utilisés dans ce projet sont constitués de triangles de taille relativement uniforme. De plus, l'algorithme de découpe développé dans Scyther permet de contrôler la taille maximale des triangles générés et ainsi d'assurer que les nouveaux triangles sont eux aussi de taille semblable. Les grilles uniformes ont donc été privilégiées puisque, comparativement aux grilles hiérarchiques, elles sont plus rapides à utiliser, plus simples à implémenter et plus faciles à paralléliser.

La grille de partitionnement spatial est définie à partir des coordonnées de la région d'intérêt. La taille des cellules est définie comme la taille du plus grand volume englobant de triangle plus une valeur d'epsilon de 0.001 pour éviter les problèmes liés aux erreurs d'arrondissement. Pour les scènes utilisées, cet epsilon correspond à 0.06% à 0.12% de la taille du plus grand volume englobant de triangle. La taille de la grille est augmentée d'une demi-cellule de chaque côté de la région d'intérêt pour s'assurer que le volume englobant de chaque triangle ait son centre à l'intérieur de la grille et n'en dépasse pas les bornes. Finalement, sa taille est augmentée à nouveau pour qu'elle corresponde à un nombre entier de cellules. Au final, la grille est représentée par les coordonnées de son sommet minimum, le nombre de cellules le long de chacun des axes et la taille des cellules. Elle permet de déterminer les valeurs de hachage de chaque triangle dans la prochaine étape.

3.5 Construction de la liste des paires cellule-triangle (étapes 4 et 5)

L'étape suivante consiste à trouver quelles cellules chaque triangle de la région d'intérêt chevauche pour construire la liste des paires cellule-triangle. Chaque élément de cette liste contient la valeur de hachage d'une cellule et une référence à un triangle qu'elle contient.

Pour chaque triangle, on commence par identifier les indices de la cellule contenant le centre de son volume englobant avec l'équation

$$(i, j, k) = \left\lfloor \frac{(x_c, y_c, z_c) - (x_{min}, y_{min}, z_{min})}{l} \right\rfloor \tag{3.1}$$

où (x_c, y_c, z_c) sont les coordonnées du centre, $(x_{min}, y_{min}, z_{min})$ sont les coordonnées du sommet minimum de la grille, l est la taille des cellules et $\lfloor\ \rfloor$ dénote la partie entière. Ensuite, la valeur de hachage est donnée par

$$H(i, j, k) = kn_x n_y + jn_x + i \tag{3.2}$$

où n_x, n_y et n_z sont le nombre de cellules dans la grille le long de chacun des axes. Une fois calculée, cette valeur et une référence au triangle sont ajoutées à la liste des paires cellule-triangle.

Ensuite, les indices des autres cellules avec lesquelles le volume englobant du triangle est en intersection sont identifiés. Puisque le volume englobant est nécessairement plus petit que les cellules, il ne peut chevaucher au maximum que huit cellules. Plutôt que de vérifier s'il chevauche chacune des 26 cellules adjacentes à la cellule contenant son centre, on peut identifier dans quel octant de la cellule le centre se trouve et vérifier le chevauchement avec seulement les sept cellules adjacentes à cet octant. Pour chaque cellule chevauchée, une entrée est ajoutée à la liste des paires cellule-triangle en répétant le calcul de l'équation 3.2.

Finalement, lorsque tous les triangles de la région d'intérêt ont été traités, la liste est triée par ordre de valeur de hachage des cellules avec la fonction `std::sort` de la librairie standard de C++.

3.6 Construction de la liste de paires de triangles (étape 6)

Une fois la liste des paires cellule-triangle complétée et triée, l'étape suivante consiste à construire la liste de paires de triangles potentiellement en collision. Le but du partitionnement de l'espace est d'accélérer la détection en permettant de comparer seulement les triangles se trouvant dans une même cellule plutôt que d'avoir à considérer toutes les paires

possibles.

La première étape consiste alors à identifier les intervalles de la liste des paires cellule-triangle correspondant à une même cellule. Pour ce faire, la liste est parcourue pour trouver les changements de valeur de hachage indiquant une transition d'une cellule à la suivante. Les indices du début et de la fin des éléments correspondant à une même cellule sont enregistrés et ce sont les triangles dans cet intervalle qui peuvent ensuite être testés.

Lors de la détection des auto-collisions, le test des cônes normaux est ensuite exécuté pour chaque cellule pour déterminer si les triangles qu'elle contient peuvent être entièrement ignorés. Ce test est décrit à la section 3.6.1.

Avant de procéder à tester les triangles d'une même cellule entre eux, il est nécessaire de déterminer dans quelle cellule chaque paire de triangles doit être considérée. En effet, une paire de triangles peut se retrouver dans plusieurs cellules si les deux triangles chevauchent un même ensemble de cellules. Un mécanisme est donc nécessaire pour s'assurer qu'une paire de triangles donnée ne soient considérés que dans une seule cellule. Ce mécanisme est décrit à la section 3.6.2.

Dans le cas de la détection des auto-collisions, une étape additionnelle doit être ajoutée. Puisque ce sont les triangles d'une même surface qui sont considérés, deux triangles adjacents auront nécessairement des volumes englobants qui s'intersectent et des primitives en contact. Il faut donc ignorer les paires de triangles adjacents. La technique conçue à cette fin est décrite à la section 3.6.3.

Une fois les paires de triangles d'une même cellule identifiées et les triangles adjacents élagués dans le cas des auto-collisions, un test d'intersection des volumes englobants des triangles de chaque paire est exécuté. Si les volumes sont en intersection, la paire est ajoutée à la liste de paires de triangles.

3.6.1 Test des cônes normaux

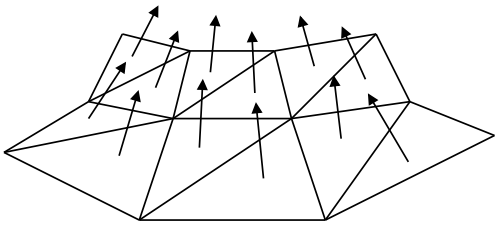
La méthode des cônes normaux est utilisée pour identifier des cellules où il ne peut y avoir d'auto-collisions et éviter des tests de collision inutiles. Elle a originellement été utilisée avec les hiérarchies de volumes englobants pour trouver des branches de la hiérarchie ne pouvant contenir d'intersection [20]. Comme expliqué à la section 2.3.1, les cônes y sont construits en parcourant la hiérarchie de bas en haut avec les cônes de chaque nœud construits à partir de ceux des nœuds enfants.

Pour ce travail, cette méthode a été adaptée au partitionnement de l'espace afin d'élaguer des cellules de la grille. Plutôt que de parcourir l'arbre d'une hiérarchie de volumes englobants, la

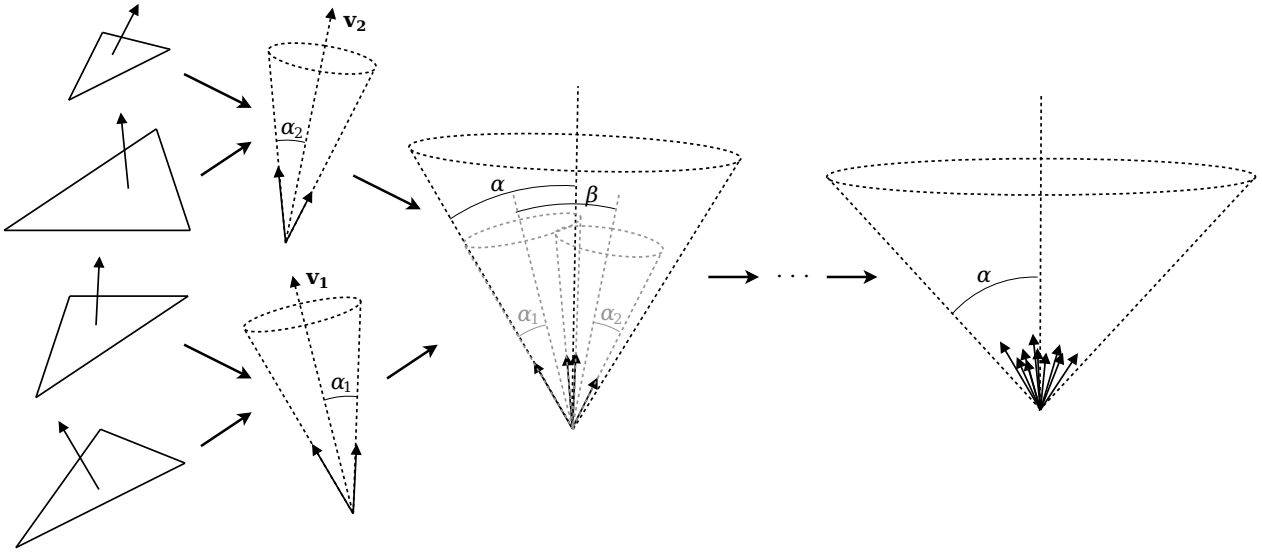
liste des triangles d'une cellule est subdivisée de façon récursive et les cônes sont construits en remontant la récursion (voir figure 3.2). Lorsque l'intervalle ne contient qu'un seul triangle, un cône est créé avec la normale du triangle comme axe central. Puisque ce cône ne contient qu'une seule normale, son angle d'ouverture est $\alpha = 0$. En remontant la récursion, de nouveaux cônes sont construits de façon à contenir les cônes des deux sous-intervalles. Si $\mathbf{v_1}$ et $\mathbf{v_2}$ sont les vecteurs de l'axe central de ces deux derniers et α_1 α_2 sont leur angle d'ouverture, l'axe du nouveau cône est calculé avec la moyenne de $\mathbf{v_1}$ et $\mathbf{v_2}$ et son angle d'ouverture est donné par

$$\alpha = \frac{\beta}{2} + \max(\alpha_1, \alpha_2) \tag{3.3}$$

où β est l'angle entre $\mathbf{v_1}$ et $\mathbf{v_2}$.



(a)



(b)

Figure 3.2 Exemple de construction de cônes normaux pour déterminer si une cellule du partitionnement spatial peut être élaguée lors de la détection des auto-collisions. (a) Section d'une surface contenue dans une cellule. (b) Début de la construction du cône normal pour quatre des triangles de la cellule et, à droite, le cône normal final pour cette cellule.

À chaque appel de la fonction récursive, si $\alpha \geq \pi/2$, le reste des calculs de cônes normaux pour cette cellule sont sautés et les paires de triangles de cette cellule devront être testées. Puisque l'angle d'ouverture ne peut qu'augmenter en combinant davantage de cônes, il est inutile de continuer les calculs de la méthode des cônes normaux pour cette cellule. Si, au contraire, la récursion est remontée au complet et α est toujours inférieur à $\pi/2$, la détection des auto-collisions dans cette cellule peut être ignorée. Cette dernière valeur de α correspond à l'angle d'ouverture du cône contenant toutes les normales de triangles dans la cellule et $\alpha < \pi/2$ implique que la surface n'est pas suffisamment courbée pour être en contact avec elle-même dans cette cellule.

3.6.2 Attribution des paires de triangles à une seule cellule

Pour éviter de faire des tests de collisions entre les mêmes triangles plusieurs fois, il faut s'assurer que les paires de triangles soient considérées que dans une seule cellule. Pour résoudre ce problème, une approche similaire à celle présentée par Le Grand [47] a été adoptée.

Lors de la construction de la grille, les cellules se font assigner une valeur numérique, ou type, différente pour chaque cellule adjacente. Pour une cellule de type T, chacune des 26 cellules adjacentes doit avoir un type $T' \neq T$. Pour une grille uniforme en 3D, huit types sont nécessaires, comme illustrés à la figure 3.3. Puisque la taille des cellules est définie par la taille du plus grand triangle, aucun triangle ne peut chevaucher deux cellules d'un même type.

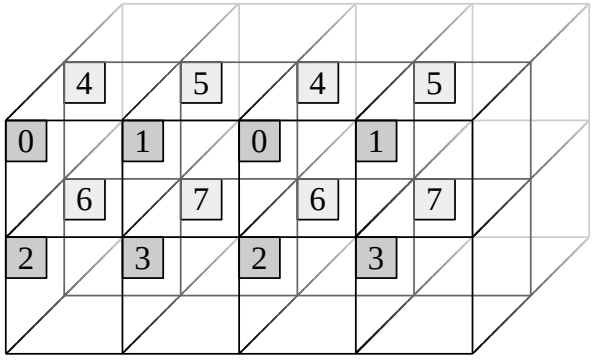


Figure 3.3 Types de cellules

Ensuite, lors de l'étape de la construction de la liste des paires cellule-triangle décrite plus haut, les types de cellules chevauchés par chaque triangle sont enregistrés. Chaque triangle possède un nombre entier où est enregistré le type de cellule contenant le centre de son volume

englobant ainsi que huit bits de contrôle pour sauvegarder le type des cellules que son volume englobant intersecte.

Puis, lors de la construction de la liste des paires de triangles à tester, ces données sont utilisées pour déterminer dans quelle cellule les paires doivent être considérées. Lorsqu'on considère une cellule de type T, on ignore les paires de triangles qu'elle contient pour lesquelles l'une des deux conditions suivantes est vraie :

1. aucun des deux triangles n'a son centre dans une cellule de type T,
2. un des deux triangles a son centre dans une cellule de type T', où $T' < T$, et T' fait partie des types de cellules chevauchés par l'autre triangle.

Ainsi, il est assuré que les paires de triangles sont traitées dans une seule cellule, soit celle du centre du triangle avec le type le plus petit.

Pour clarifier, la figure 3.4 illustre deux exemples. Dans le premier, la paire de triangles sera ignorée dans les cellules de type $T = 1$ et $T = 4$ puisqu'aucun des deux triangles n'y possède son centre. Dans la cellule $T = 3$, l'un des triangles possède son centre dans une cellule $T' = 2$, donc inférieur à 3, et qui est chevauchée par l'autre triangle. La paire y est donc également ignorée. Enfin, il ne reste que la cellule $T = 2$ où la paire n'est pas ignorée puisqu'aucune des deux conditions n'est vraie. Dans le deuxième exemple, les triangles ne partagent que la cellule $T = 2$. L'un des triangles possède son centre dans une cellule inférieure à 2, mais celle-ci n'est pas chevauchée par l'autre triangle. Ces deux triangles sont donc traités dans la cellule $T = 2$.

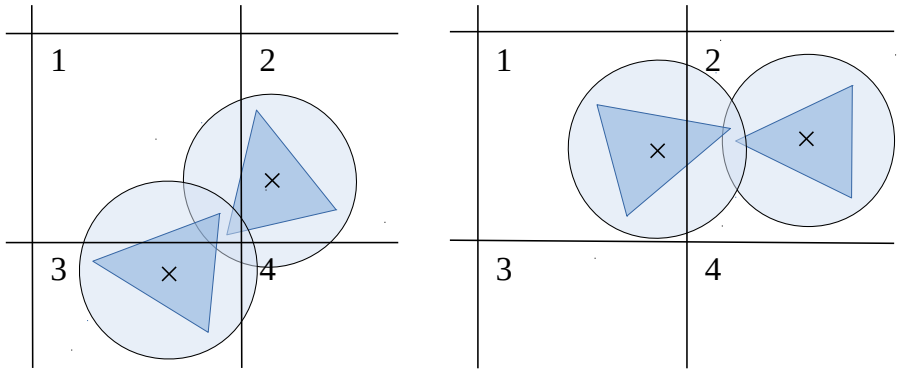


Figure 3.4 Exemples en 2D de tests basés sur les types de cellules pour déterminer dans quelle cellule une paire de triangles doit être testée.

3.6.3 Triangles adjacents

Lors de la détection des auto-collisions, il faut éviter le calcul et la création de contacts entre des triangles adjacents. Pour identifier les paires de triangles adjacents, il suffit de vérifier s'ils possèdent un sommet en commun. Chaque triangle possède la liste des index de ses trois sommets puisque cette information est nécessaire à la fin de la détection de collisions pour identifier les primitives en contact. Ces index sont également utilisés ici pour vérifier si deux triangles sont adjacents. Pour ce faire, les index des deux triangles sont comparés. Si les triangles partagent un même index, ceux-ci sont adjacents et la paire n'est pas ajoutée à la liste.

3.7 Test des paires de primitives (étape 7)

Une fois que la liste des paires de triangles potentiellement en collision est créée, il reste à déterminer lesquels sont réellement en contact. Pour être certain d'identifier tous les contacts possibles entre deux triangles, il faut tester chaque contact entre un sommet et la face d'un triangle et entre deux arêtes. Avec trois sommets et trois arêtes par triangles, jusqu'à 15 tests de primitives peuvent être nécessaires : six tests sommet-triangle et neuf tests arête-arête.

Cependant, puisque les sommets et arêtes sont partagés par plusieurs triangles, il faut s'assurer de tester une paire de primitives donnée au travers que d'une seule paire de triangles. Pour ce faire, chaque sommet et arête d'une surface est assigné à un triangle lors de l'initialisation. Les assignations sont enregistrées dans deux vecteurs où chaque élément consiste en trois valeurs booléennes indiquant si les sommets et arêtes sont assignés ou non au triangle du même index. Lorsque la surface est découpée, ces vecteurs sont mis à jour à chaque ajout ou retrait de triangles. Au moment des tests entre deux triangles, seulement les primitives assignées à ces triangles sont testées.

La détection de collisions discrète est effectuée avec des tests de proximité. Les éléments des triangles sont considérés en contact si leur distance est inférieure à la distance de contact. Le problème revient donc à calculer la distance minimale entre les sommets et la face des triangles et entre les paires d'arêtes. Pour la réponse aux collisions, il faut aussi retourner le point sur le triangle ou les arêtes où le contact est appliqué. Le calcul de ces distances et des points de contact est présenté aux sections 3.7.1 et 3.7.2.

3.7.1 Test de proximité sommet-triangle

Le test de collision entre un sommet et un triangle consiste à trouver la distance minimale entre les deux primitives. Il faut d'abord calculer le point sur le plan du triangle le plus proche du sommet. Ensuite, si ce point se trouve à l'intérieur du triangle, la distance est calculée directement à partir de ce point. Sinon, le point du triangle le plus proche du sommet se trouve sur l'une de ses arêtes ou à l'un de ses sommets et la distance est calculée à ce point.

La méthode pour calculer cette distance est empruntée à Eberly [61]. Le problème consiste à trouver la distance minimale entre un point $\mathbf{P}$ et un triangle $\mathbf{T}(u, v)$ défini par

$$\mathbf{T}(u, v) = \mathbf{A} + u\mathbf{E}_0 + v\mathbf{E}_1, \tag{3.4}$$

où $\mathbf{A}$ est un des sommets du triangle et $\mathbf{E}_0$ et $\mathbf{E}_1$ sont deux arêtes partant de $\mathbf{A}$. La région à l'intérieur du triangle est délimitée par $u \geq 0$, $v \geq 0$ et $u + v \leq 1$. La distance au carré entre le sommet et le plan du triangle est donnée par :

$$Q(u, v) = |\mathbf{T}(u, v) - \mathbf{P}|^2 = au^2 + 2buv + cv^2 + 2du + 2ev + f \tag{3.5}$$

où $a = \mathbf{E}_0 \cdot \mathbf{E}_0$, $b = \mathbf{E}_0 \cdot \mathbf{E}_1$, $c = \mathbf{E}_1 \cdot \mathbf{E}_1$, $d = \mathbf{E}_0 \cdot (\mathbf{B} - \mathbf{P})$, $e = \mathbf{E}_1 \cdot (\mathbf{A} - \mathbf{P})$ et $f = (\mathbf{A} - \mathbf{P}) \cdot (\mathbf{A} - \mathbf{P})$. La distance minimale au plan est alors trouvée en calculant les valeurs (u', v') où $Q(u, v)$ est minimisé :

$$\nabla Q(u', v') = 2(au' + bv' + d, bu' + cv' + e) = (0, 0). \tag{3.6}$$

La solution à ce système d'équations est donnée par :

$$u' = (be - cd)/(ab - b^2) \tag{3.7}$$

$$v' = (bd - ae)/(ab - b^2). \tag{3.8}$$

Si u' et v' respectent $u' \geq 0$, $v' \geq 0$ et $u' + v' \leq 1$, le minimum se trouve à l'intérieur du triangle, soit la région 0 à la figure 3.5, et la distance du sommet au triangle peut être calculée directement avec $|\mathbf{T}(u', v') - \mathbf{P}|$. Autrement, le minimum se trouve dans l'une des régions à l'extérieur du triangle et le point du triangle le plus proche au sommet se trouve alors sur une arête ou un sommet. Ce point doit être déterminé pour pouvoir calculer la distance.

Si le minimum se trouve dans la région 1, le point le plus proche se trouve sur la droite de l'arête $u + v = 1$. La position est trouvée en minimisant la fonction $F(u'') = Q(u'', 1 - u'')$. Il faut cependant déterminer si le point est bien sur l'arête du triangle ou à l'un des deux sommets. Si $u'' \geq 1$, le point est au sommet $(1, 0)$, alors que si $u'' \leq 0$, il est au sommet $(0, 1)$.

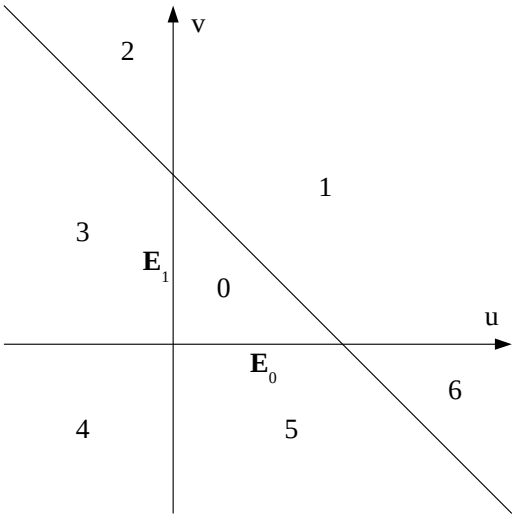


Figure 3.5 Régions du plan d'un triangle servant à déterminer si le point du triangle le plus proche d'un sommet se trouve sur sa face, sur une de ses arêtes ou à l'un de ses sommets.

Si le minimum se trouve dans la région 3 ou 5, le point le plus proche se trouve sur l'arête $u = 0$ ou $v = 0$ respectivement. Donc, (u'', v'') est trouvé en minimisant $F(v'') = Q(0, v'')$ ou $F(u'') = Q(u'', 0)$. Encore une fois, il faut distinguer entre l'arête et les sommets. Pour la région 3, le point est au sommet$(0, 0)$ si $v'' \leq 1$ et $(0, 1)$ si $v'' \geq 1$. Pour la région 5, il est au sommet $(0, 0)$ si $u'' \leq 1$ et $(1, 0)$ si $u'' \geq 1$.

Finalement, si le minimum tombe dans la région 2, 4 ou 6, le point le plus proche peut se trouver sur l'une des deux arêtes adjacentes à la région. Pour déterminer sur laquelle des deux arêtes le point se trouve, Eberly [61] propose un test basé sur le signe du gradient de Q aux deux arêtes concernées dans la direction qui s'éloigne du sommet adjacent à la région. Par exemple, pour la région 2, les deux directions sont $(0, -1)$ pour l'arête $u = 0$ et $(1, -1)/\sqrt{2}$ pour l'arête $u + v = 1$. Le test consiste alors à calculer $(0, -1) \cdot \nabla Q(0, 1)$ et $(1, -1)/\sqrt{2} \cdot \nabla Q(0, 1)$. Selon laquelle de ces deux valeurs est négative, on détermine sur quelle arête se trouve le point. Le reste du test est similaire aux autres régions : minimiser une fonction $F(u'', v'')$ avec les contraintes correspondant à l'arête en question, puis tester les inégalités pour vérifier si le point est à la position calculée ou sur l'un des deux sommets de l'arête.

3.7.2 Test de proximité arête-arête

Le test de collision entre deux arêtes consiste à trouver la distance minimale entre eux. La méthode pour calculer cette distance est empruntée à Ericson [62]. D'abord, la méthode consiste à trouver les points sur la droite de chaque arête entre lesquels la distance est minimisée puis à calculer la distance entre ces deux points.

Soit deux arêtes allant des points P_1 à Q_1 et P_2 à Q_2 respectivement comme illustré à la figure 3.6. Les droites passant le long de chaque arête sont définies par :

$$D_1(u) = P_1 + u\mathbf{d}_1, \tag{3.9}$$

$$D_2(v) = P_2 + v\mathbf{d}_2, \tag{3.10}$$

où $\mathbf{d}_1 = Q_1 - P_1$ et $\mathbf{d}_2 = Q_2 - P_2$. Les deux arêtes sont alors définies comme les points où $0 \leq u \leq 1$ et $0 \leq v \leq 1$ respectivement.

Ensuite, soit un vecteur reliant les deux droites du point $D_1(u)$ au point $D_2(v)$:

$$\mathbf{s}(u, v) = D_2(v) - D_1(u). \tag{3.11}$$

La distance minimale entre les droites est trouvée en déterminant les valeurs de u et v qui minimisent la norme de $\mathbf{s}(u, v)$. On peut constater que cette norme est minimisée lorsque $\mathbf{s}(u, v)$ est orthogonal aux deux droites, c'est-à-dire lorsque $\mathbf{d}_1 \cdot \mathbf{s}(u, v) = 0$ et $\mathbf{d}_2 \cdot \mathbf{s}(u, v) = 0$. En substituant les équations 3.9 et 3.10 dans l'équation 3.11 et en multipliant par $\mathbf{d}_1$ et $\mathbf{d}_2$, le système d'équations suivant est obtenu :

$$(\mathbf{d}_1 \cdot \mathbf{d}_1)u - (\mathbf{d}_1 \cdot \mathbf{d}_2)v = -(\mathbf{d}_1 \cdot \mathbf{r}) \tag{3.12}$$

$$(\mathbf{d}_2 \cdot \mathbf{d}_1)u - (\mathbf{d}_2 \cdot \mathbf{d}_2)v = -(\mathbf{d}_2 \cdot \mathbf{r}), \tag{3.13}$$

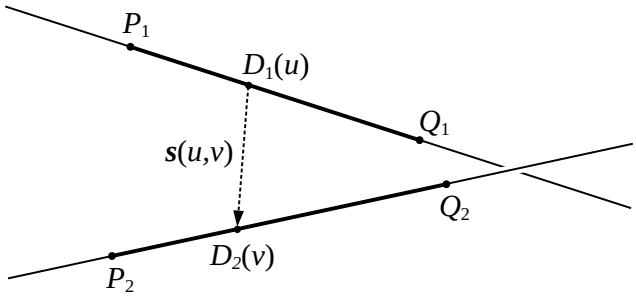


Figure 3.6 Deux arêtes inscrites dans les droites $D_1(u)$ et $D_2(v)$

où $\mathbf{r} = P_2 - P_1$. La solution est donnée par :

$$u' = \frac{(\mathbf{d}_1 \cdot \mathbf{d}_2)(\mathbf{d}_2 \cdot \mathbf{r}) - (\mathbf{d}_1 \cdot \mathbf{r})(\mathbf{d}_2 \cdot \mathbf{d}_2)}{(\mathbf{d}_1 \cdot \mathbf{d}_1)(\mathbf{d}_2 \cdot \mathbf{d}_2) - (\mathbf{d}_1 \cdot \mathbf{d}_2)^2} \tag{3.14}$$

$$v' = \frac{(\mathbf{d}_1 \cdot \mathbf{d}_1)(\mathbf{d}_2 \cdot \mathbf{r}) - (\mathbf{d}_1 \cdot \mathbf{d}_2)(\mathbf{d}_1 \cdot \mathbf{r})}{(\mathbf{d}_1 \cdot \mathbf{d}_1)(\mathbf{d}_2 \cdot \mathbf{d}_2) - (\mathbf{d}_1 \cdot \mathbf{d}_2)^2}. \tag{3.15}$$

Lorsque les valeurs trouvées respectent $0 \leq u' \leq 1$ et $0 \leq v' \leq 1$, les deux points sont à l'intérieur des arêtes et la distance entre ceux-ci est directement donnée par $|D_1(u') - D_2(v')|$. Autrement, un ou les deux points doivent être attachés au sommet de leur arête respective. Si un seul point se trouve à l'extérieur de l'arête correspondante, il est remplacé par le sommet le plus proche et le point de l'autre arête le plus près est calculé. Si les deux points sont à l'extérieur de leur arête respective, la même procédure doit être appliquée deux fois. D'abord, le point de l'arête 1 est attaché au sommet. Ensuite, le point le plus près sur la droite de l'arête 2 est calculé. Si ce point tombe à nouveau à l'extérieur de l'arête, il est attaché au sommet puis le point le plus près sur l'arête 1 est calculé.

Aux équations 3.14 et 3.15, si le dénominateur est égal à zéro, les deux droites sont parallèles et ce cas doit être géré séparément. Dans ce cas, les points de contact sont établis au centre de l'intervalle où les arêtes sont vis-à-vis l'une de l'autre. Pour ce faire, les sommets de la deuxième arête, P_2 et Q_2, sont projetés sur la droite $D_1(u)$:

$$u_{P2} = (P_2 - P_1) \cdot \mathbf{d}_1 / {\mathbf{d}_1}^2, \tag{3.16}$$

$$u_{Q2} = (Q_2 - P_1) \cdot \mathbf{d}_1 / {\mathbf{d}_1}^2. \tag{3.17}$$

Ensuite, au cas où les deux arêtes pointent dans des directions opposées, u_{P2} et u_{Q2} sont permutés si $u_{P2} > u_{Q2}$. L'intervalle où les arêtes sont vis-à-vis va de $\max(0, u_{P2})$ à $\min(1, u_{Q2})$. La position du contact sur l'arête 1 est donc :

$$u' = (\max(0, u_{P2}) + \min(1, u_{Q2}))/2. \tag{3.18}$$

Si $u' < 0$ ou $u' > 1$, les deux arêtes ne sont pas vis-à-vis et la distance minimale entre eux se trouve entre deux sommets. Autrement, la position sur l'arête 2 est obtenue en projetant $D_1(u')$ sur $D_2(v)$:

$$v' = (D_1(u') - P_2) \cdot \mathbf{d}_2 / {\mathbf{d}_2}^2, \tag{3.19}$$

La distance est alors donnée par $|D_1(u') - D_2(v')|$.

3.8 Réponse aux collisions

La réponse aux collisions est gérée par SOFA par la méthode des pénalités. Lors des tests de paires de primitives, les contacts trouvés sont enregistrés dans un vecteur et passés au gestionnaire de contacts de SOFA. Chaque élément du vecteur contient un contact décrit par :

- l'index des deux éléments des modèles de collisions concernés (triangle, arête ou sommet),
- un numéro d'identification unique à chaque contact,
- les coordonnées des deux points de contact,
- la normale du contact (pointant du premier au deuxième point de contact),
- la distance entre les deux points de contact.

Le numéro d'identification est utilisé par le gestionnaire de contacts pour éliminer les contacts redondants. Lorsque plusieurs contacts sont ajoutés à une primitive donnée, seulement celui avec la plus courte distance est préservé.

Les forces de pénalité sont ensuite créées :

$$F = k\left(\Delta x - d_{contact}\right), \tag{3.20}$$

où k, la constante de rappel, et $d_{contact}$, la distance de contact, sont des paramètres définis par l'utilisateur et Δx est la distance entre les deux points du contact. Les forces ainsi créées sont ensuite mappées aux degrés de liberté et ajoutées aux autres forces externes du modèle mécanique utilisé par le solutionneur.

3.9 Parallélisation avec OpenMP

La méthode de détection de collisions proposée peut facilement être accélérée avec la parallélisation sur processeurs multicœurs. Pour ce faire, l'algorithme a été parallélisé avec la librairie OpenMP sur le paradigme de parallélisation des données. Les quatre étapes suivantes ont été parallélisées :

- étape 2 : trouver les triangles dans la région d'intérêt,
- étape 4 : construire la liste des paires cellule-triangle (incluant la détection des cellules intersectées par chaque triangle et le calcul de leur valeur de hachage),
- étape 6 : trouver les paires de triangles à tester dans chaque cellule,
- étape 7 : tester les contacts entre les paires sommet-triangle et arête-arête.

L'algorithme 2 reprend l'algorithme 1 mais avec ces étapes indiquées par la mention «OpenMP».

Algorithme 2 Détection de collisions en parallèle

```
mettre à jour les volumes englobants des objets
pour chaque paire d'objets faire
  // Étape 1 :
  tester l'intersection des volumes englobants des objets
  déterminer la région d'intérêt
  si une intersection est détectée alors
    // Étape 2 :
    trouver les triangles dans la région d'intérêt (OpenMP)
    // Étape 3 :
    définir la grille
    // Étape 4 :
    construire la liste des paires cellule-triangle (OpenMP)
    // Étape 5 :
    trier la liste des paires cellule-triangle
    // Étape 6 : Construire la liste de paires de triangles (OpenMP)
    pour chaque cellule faire
      si auto-collisions alors
        effectuer le test des cônes normaux
      fin si
      pour chaque paire de triangles faire
        élaguer par type de cellule
        élaguer par test de volume englobant
        ajouter la paire de triangles à la liste
      fin pour
    fin pour
    // Étape 7 : Tests de paires de primitives (OpenMP)
    pour chaque paire de triangles faire
      effectuer les tests de proximité sommet-triangle et arête-arête
      ajouter les paires de sommet-triangle et arête-arête en contact au vecteur de contacts
    fin pour
  fin si
fin pour
```

Dans chacune de ces étapes, les données retournées doivent être combinées dans un même vecteur pour être utilisées dans l'étape suivante. Dans trois des quatre étapes, les données sont ajoutées au vecteur dans une section critique. Dans l'étape de la construction de la liste des paires cellule-triangle, il a été déterminé expérimentalement qu'il est plus rapide d'accumuler les données dans des vecteurs différents pour chaque fil d'exécution et de les fusionner dans un même vecteur sur un seul fil.

3.10 Expérimentations

Cette section présente les expériences effectuées pour évaluer les capacités et limitations de la méthode proposée et pour tester les hypothèses énumérées à la section 2.8. Pour l'ensemble des expériences, quelques scènes ont été préparées avec diverses tailles de maillage de surface et une avec des auto-collisions.

La première expérience vise à déterminer l'impact du choix de volumes englobants sur le temps de calcul. La deuxième a pour but d'évaluer l'efficacité de l'algorithme à détecter les auto-collisions et l'impact de la méthode des cônes normaux. Le troisième vise à comparer les performances de l'algorithme à d'autres méthodes existantes. La quatrième consiste à vérifier la capacité à traiter les collisions dans des scénarios de découpe d'objets. Finalement, la cinquième expérience a pour but d'évaluer l'accélération apportée par la parallélisation sur processeur multicœur.

3.10.1 Volumes englobants

La première expérience porte sur le type de volume englobant utilisé et le choix de les utiliser ou non dans certaines étapes de l'algorithme. Plus spécifiquement, ce sont les volumes englobants des triangles et des primitives qui sont étudiés puisque ceux-ci sont impliqués dans une grande partie des calculs. Les différentes formes de volumes englobants influencent le temps de calcul pour les mettre à jour et pour en tester l'intersection de même que l'efficacité à élaguer les triangles et les paires de primitives. Cette expérience constitue la première partie de l'objectif 2 présenté à la section 2.8.3. De plus, elle est présentée en premier parce qu'elle servira à déterminer les volumes utilisés dans les expériences suivantes.

Pour rappel, les volumes englobants des triangles sont utilisés dans plusieurs étapes de la méthode proposée. D'abord, ils servent à identifier les triangles à considérer par un test d'intersection avec la région d'intérêt. Ensuite, ils permettent d'identifier quelles cellules de la grille sont occupées par chaque triangle lors de la création de la liste de paires cellule-triangle. Finalement, ils servent à identifier les paires de triangles potentiellement en collision. Le choix de volumes englobants des triangles peut donc impacter le temps de calcul à plusieurs niveaux.

Des volumes englobants peuvent aussi être ajoutés aux primitives de collisions pour réduire le nombre de tests de proximité à exécuter. Cela implique que des volumes englobants doivent être mis à jour pour chaque arête testée. Pour les tests sommet-triangle, les volumes englobants des triangles utilisés dans les étapes précédentes sont réutilisés.

Les deux types de volumes englobants implémentés sont les sphères et les boîtes alignées

aux axes. Ainsi, quatre tests seront effectués sur l'ensemble des scènes utilisées : avec les sphères ou les boîtes englobantes et, dans les deux cas, avec ou sans tests d'intersection de volumes englobants aux primitives. On comparera le temps de calcul consacré à la détection de collisions, le nombre de paires de triangles restants et le nombre de primitives élaguées.

3.10.2 Auto-collisions

Comme expliqué au chapitre 2, la détection des auto-collisions présente une difficulté supplémentaire pour la plupart des méthodes d'accélération de la détection de collisions. Dans le cas du partitionnement de l'espace, la région d'intérêt est le volume englobant de l'objet en entier et la grille couvre donc l'objet au complet. Ainsi, tous les triangles de l'objet doivent être traités, ce qui peut être coûteux en temps de calcul.

Pour accélérer la détection des auto-collisions, la méthode des cônes normaux a été implémentée. Ce test visera donc à compléter l'objectif 2 en évaluant la performance de la détection d'auto-collisions et le gain apporté par la méthode des cônes normaux. Pour ce faire, on mesurera le temps de calcul de la détection de collisions avec et sans l'utilisation des cônes normaux avec une scène impliquant des auto-collisions. De plus, on évaluera la méthode présentée dans la section 3.6 pour identifier et élaguer les triangles adjacents.

3.10.3 Comparaison avec SOFA

L'un des principaux objectifs de ce projet est de démontrer l'efficacité de la méthode de détection de collisions proposée. À cette fin, l'objectif 3 sera réalisé en comparant la performance de la méthode implémentée avec celle de l'algorithme de détection de collisions dans SOFA.

La principale méthode disponible dans SOFA pour la détection de collisions entre objets déformables est celle basée sur les hiérarchies de volumes englobants. Les contacts entre les primitives sont détectés par des tests de proximités similaires à ceux implémentés dans Scyther. Ce test permettra donc une comparaison avec une implémentation d'une des méthodes de détection les plus répandues.

Pour effectuer ce test, des scènes identiques à celles utilisées pour les autres tests seront préparées avec SOFA, sans la librairie Scyther, en utilisant les composantes de détection de collisions de SOFA. Le temps moyen par pas de la détection de collisions sera rapportée pour chacun des deux algorithmes. Ces scènes compareront aussi bien le temps de calcul pour les collisions entre objets que les auto-collisions.

3.10.4 Découpe

L'un des avantages avancés dans cette étude pour la méthode du partitionnement de l'espace est sa simplicité par rapport à la découpe des objets. En effet, dans la boucle d'animation, la découpe est effectuée avant la détection de collisions. La détection prend en compte la position des primitives au moment présent sans égard à la topologie de la surface. Elle ne devrait donc pas être impactée par les changements de topologie de la surface, c'est-à-dire l'ajout et retrait de triangles.

Cette expérience consistera à réaliser l'objectif 4 en étudiant la détection de collisions lorsqu'un objet se fait découper. Pour ce faire, une scène sera préparée avec la découpe progressive d'un objet. Une attention particulière sera portée à la détection de collisions entre les nouvelles surfaces générées de part et d'autre du plan de découpe.

3.10.5 Parallélisation

L'un des avantages avancés pour la méthode du partitionnement de l'espace est qu'elle est bien adaptée à la parallélisation. Cette dernière expérience vise donc à réaliser l'objectif 5 en analysant l'accélération obtenue par la parallélisation avec OpenMP décrite à la section 3.9.

Pour ce faire, les temps de calcul en série et en parallèle de la détection de collisions seront comparés pour l'ensemble des scènes préparées. Les temps de calcul des différentes étapes de la détection seront également comparés.

CHAPITRE 4 RÉSULTATS

Ce chapitre présente les résultats des expérimentations introduites au chapitre précédent. La première section décrit les scènes utilisées pour produire les résultats présentés dans les sections qui suivent. Les sections suivantes présentent les résultats des expérimentations décrites à la section 3.10.

4.1 Scènes

Cette section présente les deux scènes utilisées pour obtenir l'ensemble des résultats présentés dans les sections suivantes.

La première, la scène des tores, est illustrée dans la figure 4.1. Elle consiste en deux tores entrant en collision. Celui du bas, à la verticale, est fixé à la base par des contraintes de fixation appliquées à certains degrés de liberté du volume. Le tore du haut tombe librement sur l'autre et glisse sur le côté. Les résultats présentés dans les sections suivantes correspondent aux moyennes de 300 pas de 20 ms, ce qui correspond au temps requis pour passer de la première à la dernière image de la figure 4.1. Cette scène a été reproduite avec les trois niveaux de résolution des maillages de surface illustrés à la figure 4.2. Chaque tore est formé de 1600,

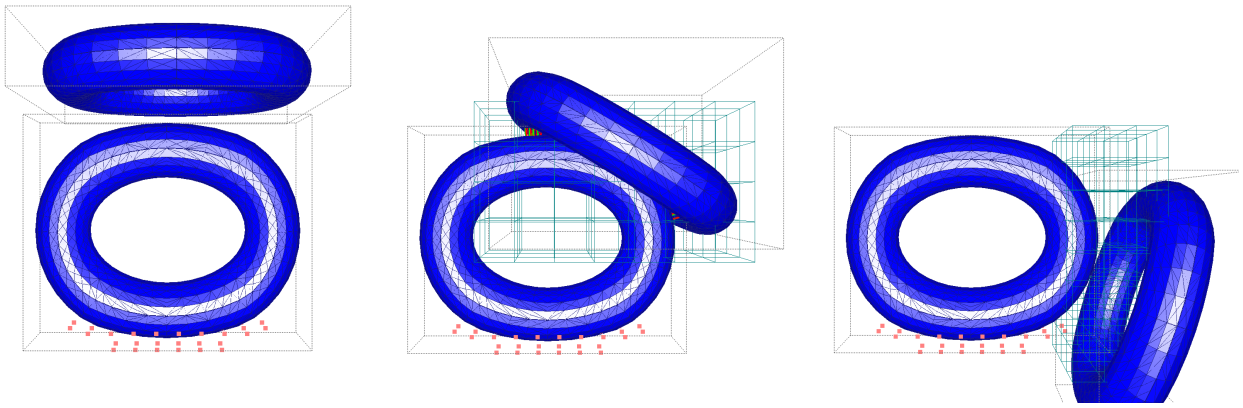


Figure 4.1 Scène des tores. Le tore du bas est fixé à sa base par des contraintes à certains degrés de liberté (carrés rouges) alors que celui du haut tombe et glisse sur l'autre. Les volumes englobants des deux objets sont visibles en gris, de même que la grille de partitionnement de l'espace, en turquoise, couvrant l'intersection des deux volumes englobants. Les courtes lignes rouges et vertes dans la figure du centre montrent les contacts détectés entre les deux objets.

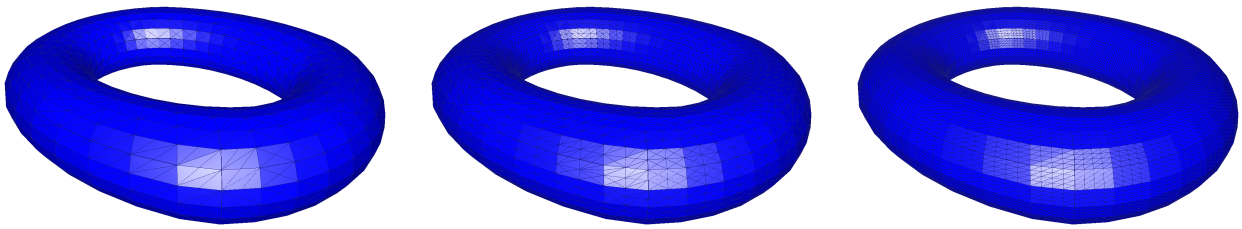


Figure 4.2 Maillages de surface de différentes résolutions utilisées pour les tests avec, de gauche à droite, 1600, 6400 et 25600 triangles.

6400 et 25600 triangles respectivement.

La seconde scène, celle du nœud, est illustrée dans la figure 4.3. Elle est employée principalement pour étudier les auto-collisions. Elle consiste en un nœud, également fixé à la base par des contraintes aux degrés de liberté, qui s'effondre sur lui-même produisant un grand nombre d'auto-collisions. La surface contient 3960 triangles et les résultats de chaque section représentent les moyennes de 370 pas de 20 ms.

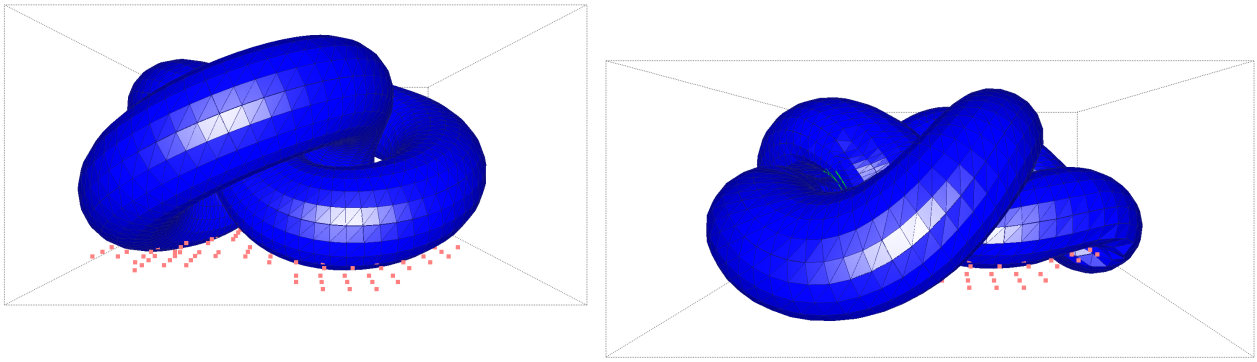


Figure 4.3 Scène du nœud. L'objet est fixé à sa base par des contraintes à certains degrés de liberté (carrés rouges). La surface retombe sur elle-même, créant des auto-collisions. La grille de partitionnement de l'espace, qui recouvre tout le volume englobant, n'est pas illustrée.

Des scènes identiques ont été produites avec SOFA, sans la librairie Scyther, pour en comparer les performances. Ces scènes emploient l'algorithme de détection de collisions de SOFA basé sur la méthode des hiérarchies de volumes englobants. Les contacts sont détectés par des tests de proximité entre les primitives similaires à ceux utilisés dans Scyther. SOFA requiert de définir comme paramètre le nombre de niveaux de la hiérarchie de volumes englobants. Ce nombre a été fixé à 12 puisque c'est le nombre qui a montré expérimentalement les meilleures performances.

Toutes les expérimentations ont été effectuées sur un ordinateur muni d'un processeur Intel i5-9400F avec six cœurs de 2.9 GHz et une fréquence *turbo* maximale de 4.1 GHz. Tous les temps présentés dans les sections suivantes correspondent à la moyenne de 10 exécutions des scènes.

4.2 Volumes englobants

Cette section présente les résultats de l'expérience décrite à la section 3.10.1 sur le choix de volumes englobants des triangles et des primitives de collision. Pour comparer l'impacte des volumes englobants sur la performance de l'algorithme, le tableau 4.1 montre le temps de calcul moyen par pas de simulation de la détection de collisions pour chacune des scènes. Pour chacune des deux formes de volumes englobants (sphères et boîtes), ces temps ont été mesurés avec et sans l'usage de test d'intersection de volumes englobants des primitives à l'étape 7. Pour la scène du nœud, la méthode des cônes normaux n'est pas utilisée. Elle sera étudiée à la section suivante.

	Sphères		Boîtes	
Scène	Avec	Sans	Avec	Sans
Tores (3200 triangles)	0.79	0.80	0.54	0.51
Tores (12800 triangles)	3.13	3.09	2.55	2.39
Tores (51200 triangles)	16.93	16.64	16.20	15.13
Nœud (3960 triangles)	19.56	20.75	11.96	12.06

Tableau 4.1 Temps moyen (en millisecondes) par pas de la détection de collisions pour les deux types de volumes englobants étudiés avec et sans tests d'intersection de volumes englobants pour les paires de primitives.

On peut voir que les boîtes englobantes sont toujours plus efficaces que les sphères englobantes. L'impact du choix de volumes englobants sur le temps de calcul dépend principalement de leur capacité à élaguer les paires de triangles lors des différentes étapes de la détection de collisions. Le tableau 4.2 montre le nombre de paires de triangles potentiellement en collision qu'y n'ont pas été élaguées et dont les primitives doivent être testées entre elles. La dernière colonne montre l'amélioration apportée par les boîtes par rapport aux sphères.

Le tableau 4.1 permet aussi d'observer l'impact de l'utilisation de volumes englobants aux primitives de collision lors des tests élémentaires. Leur impact dépend de leur capacité à élaguer les paires de primitives de même que du temps additionnel requis pour les mettre à jour à chaque pas. Pour mieux comprendre leur effet, le tableau 4.3 présente le nombre de paires de primitives identifiées dans la liste des paires de triangles et le nombre (et pourcentage) de ces paires ayant été élaguées par les tests d'intersection de volumes englobants.

Scènes	Sphères	Boîtes	Amélioration
Tores (3200 triangles)	2506	965	61%
Tores (12800 triangles)	7302	3988	45%
Tores (51200 triangles)	27759	25687	7%
Nœud (3960 triangles)	161465	83756	48%

Tableau 4.2 Nombre de paires de triangles moyen à tester par pas de simulation avec les deux types de volumes englobants étudiés et pourcentage de réduction du nombre de tests avec les boîtes par rapport aux sphères.

	Sphères		Boîtes	
Scènes	Paires de primitives	Paires élaguées	Paires de primitives	Paires élaguées
Tores (3200 triangles)	7921	1848 (23.3%)	3069	1748 (57.0%)
Tores (12800 triangles)	23990	5504 (22.9%)	13107	6084 (46.4%)
Tores (51200 triangles)	90517	18106 (20.0%)	83558	25755 (30.8%)
Nœud (3960 triangles)	507914	112862 (22.2%)	264754	131924 (49.8%)

Tableau 4.3 Nombre de paires de primitives potentiellement en collision à tester et nombre de paires élaguées par test d'intersection de volumes englobants pour les deux types de volumes étudiés.

Ces résultats seront discutés à la section 5.1. Cependant, on peut déjà observer que les boîtes réduisent davantage le temps de calcul que les sphères. De plus, il est le plus souvent légèrement plus rapide de ne pas utiliser de volumes englobants aux primitives. Ce sont ces paramètres qui ont été utilisés pour les expériences dont les résultats sont présentés aux sections suivantes.

4.3 Auto-Collisions

Dans cette section, les résultats de l'expérience sur les auto-collisions décrite dans la section 3.10.2 sont présentés. Les résultats obtenus avec la scène du nœud y sont comparés avec et sans l'usage de la méthode des cônes normaux. Le tableau 4.4 montre le temps de calcul moyen des 370 pas pour la détection de collisions et les étapes influencées par les cônes normaux. Le tableau 4.5 montre l'impact de cette méthode quant au nombre de cellules élaguées, d'éléments de la liste des paires de cellule-triangle contenues dans ces cellules et de tests élémentaires restant à exécuter avec et sans les cônes normaux.

Dans la section 3.6, on avait présenté la méthode utilisée pour identifier les triangles adjacents et empêcher la création de contacts entre eux. Lors des tests, on a observé que cette méthode est parfois insuffisante pour éviter la création de contacts superflus entre des tri-

Étapes	Sans cônes normaux	Avec cônes normaux	Amélioration
Test des cônes normaux	0.00 (0.0%)	0.80 (8.1%)	
Trouver paires de triangles à tester	3.66 (30.3%)	2.81 (28.4%)	
Tests de primitives	5.83 (48.4%)	3.69 (37.3%)	
Autres étapes	2.57 (21.3%)	2.59 (26.2%)	
Total de la détection de collisions	12.06 (100.0%)	9.90 (100.0%)	17.9%

Tableau 4.4 Temps moyen par pas (en millisecondes) et pourcentage du temps total de la détection des étapes influencées par les cônes normaux pour la scène du nœud.

Étapes	Sans cônes normaux	Avec cônes normaux	Amélioration
Cellules de la grille élaguées		1010 (87.4%)	
Triangles élagués		19466 (67.5%)	
Tests élémentaires	264754	127479	52%

Tableau 4.5 Nombre de cellules et de triangles élagués par les tests de cônes normaux et nombre de tests élémentaires restant à exécuter avec et sans les cônes normaux.

angles voisins, soit des triangles proches l'un de l'autre sur la surface. Lorsque la distance de contact est supérieure à la taille d'un triangle, un contact peut être créé entre les primitives de deux triangles de part et d'autre de ce triangle. Plusieurs de ces contacts sont montrés dans la figure 4.4. Comme mentionné précédemment, pour que le traitement des collisions soit visuellement convaincant et éviter l'interpénétration des objets, il faut choisir une distance de contact, une constante de rappel des forces de pénalité et un pas de temps adéquats. Cependant, il a été observé qu'il n'est pas toujours possible de fixer la distance de contact à une valeur suffisamment petite pour éviter la formation de contacts entre des triangles voisins.

4.4 Comparaison avec SOFA

Les résultats de la comparaison de la solution proposée avec SOFA sont présentés dans le tableau 4.6. Les valeurs correspondent au temps de calcul moyen de la détection de collisions des deux algorithmes. Pour la scène du nœud, la méthode des cônes normaux n'est pas utilisée puisqu'elle n'est pas implémentée dans SOFA.

On voit qu'avec les scènes utilisées, la méthode de subdivision spatiale implémentée est entre 19% et 66% plus rapide que la méthode basée sur les hiérarchies de volumes englobants de SOFA. La différence est particulièrement marquée avec la scène du nœud qui teste la détection des auto-collisions et diminue lorsque le nombre de triangles croît.

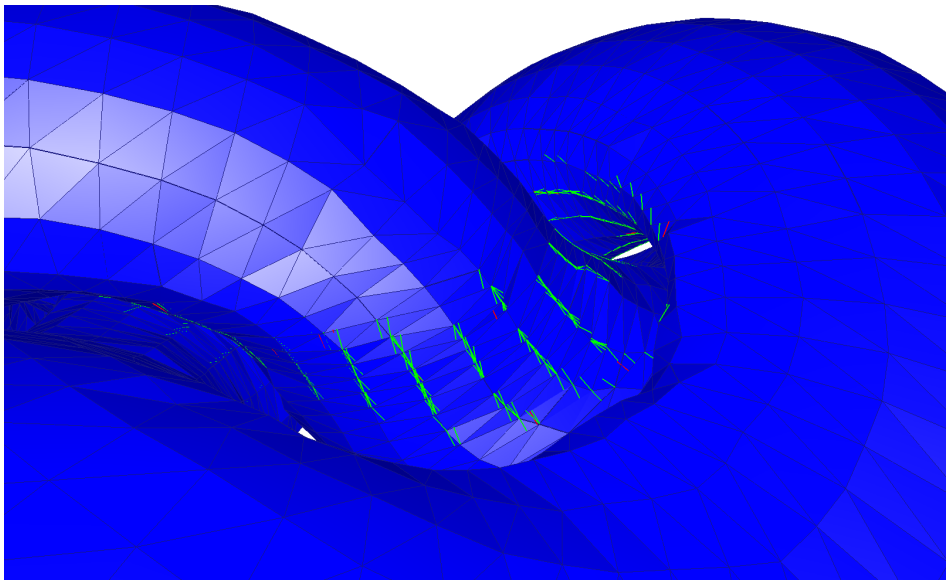


Figure 4.4 Agrandissement de la scène du nœud montrant certains des contacts entre triangles voisins. Les contacts sommet-triangle et arête-arête sont représentés en vert et en rouge respectivement.

Scène	SOFA	Scyther	Amélioration
Tores (3200 triangles)	0.69	0.53	24%
Tores (12800 triangles)	2.97	2.41	19%
Nœud (3960 triangles)	35.51	12.21	66%

Tableau 4.6 Comparaison du temps de calcul moyen par pas (en millisecondes) de la détection de collisions entre Scyther et SOFA.

4.5 Découpe

Lorsque la découpe dans Scyther est testée conjointement à la détection de collisions, on observe que l'objet découpé devient instable et se déforme de façon chaotique. La surface et les degrés de liberté sur lesquels elle est mappée semblent en quelque sorte exploser dans toutes les directions. L'algorithme de découpe de Scyther fonctionne en avançant le fil de la lame dans l'objet à chaque pas. Lorsque la lame intersecte un triangle, celui-ci est découpé, ou plutôt remplacé par de plus petits triangles de part et d'autre de la coupure. De nouveaux triangles sont aussi générés à l'intérieur de l'objet de chaque côté du plan de découpe de façon à maintenir la surface de l'objet étanche. Les deux surfaces générées de chaque côté du plan sont très rapprochées l'une de l'autre et des contacts sont détectés entre elles durant la détection de collisions. Elles sont alors repoussées par les forces de pénalités et, généralement après quelques pas, l'objet se déforme violemment et la simulation devient rapidement instable.

4.6 Parallélisation avec OpenMP

Le tableau 4.7 présente les résultats des tests de comparaison du temps de calcul entre l'exécution en série et en parallèle. Pour chaque scène, le temps moyen en millisecondes est donné pour chacune des étapes de la détection de collisions. Les colonnes correspondent au temps de l'exécution en série et en parallèle sur un processeur à six cœurs et à l'accélération obtenue pour les quatre étapes parallélisées. Pour rappel, ces étapes sont la détection des triangles dans la région d'intérêt (étape 2), la création de la liste de paires cellule-triangle (étape 4), la détection des paires de triangles potentiellement en collision (étape 6) et les tests de paires de primitives (étape 7).

Étapes de la détection de collisions	1 cœur		6 cœurs		Accélération
Tores (3200 triangles)					
2. Détecter triangles dans région d'intérêt	0.19	(36.5%)	0.10	(39.6%)	1.90
3. Définir la grille	0.00	(0.1%)	0.00	(0.6%)	
4. Créer la liste de paires cellule-triangle	0.05	(9.2%)	0.02	(7.3%)	2.61
5. Trier la liste de paires cellule-triangle	0.06	(11.4%)	0.06	(23.9%)	
6. Trouver les paires de triangles	0.11	(21.0%)	0.04	(16.1%)	2.67
7. Effectuer les tests élémentaires	0.10	(20.2%)	0.02	(9.3%)	4.33
Autres opérations	0.01	(1.5%)	0.01	(3.2%)	
Total de la détection de collisions	**0.51**	**(100.0%)**	**0.25**	**(100.0%)**	**2.05**
Total de la simulation	**15.32**		**9.79**		
Tores (12800 triangles)					
2. Détecter triangles dans région d'intérêt	0.81	(33.7%)	0.38	(38.3%)	2.13
3. Définir la grille	0.00	(0.2%)	0.01	(0.5%)	
4. Créer la liste de paires cellule-triangle	0.18	(7.4%)	0.06	(6.4%)	2.78
5. Trier la liste de paires cellule-triangle	0.27	(11.3%)	0.27	(27.3%)	
6. Trouver les paires de triangles	0.67	(27.9%)	0.16	(15.7%)	4.27
7. Effectuer les tests élémentaires	0.45	(19.0%)	0.10	(10.5%)	4.39
Autres opérations	0.01	(0.5%)	0.01	(1.4%)	
Total de la détection de collisions	**2.39**	**(100.0%)**	**0.99**	**(100.0%)**	**2.41**
Total de la simulation	**19.61**		**13.04**		
Tores (51200 triangles)					
2. Détecter triangles dans région d'intérêt	4.18	(27.6%)	1.79	(34.6%)	2.34
3. Définir la grille	0.05	(0.3%)	0.03	(0.6%)	
4. Créer la liste de paires cellule-triangle	0.75	(4.9%)	0.29	(5.6%)	2.56
5. Trier la liste de paires cellule-triangle	1.21	(8.0%)	1.25	(24.2%)	
6. Trouver les paires de triangles	6.16	(40.7%)	0.97	(18.7%)	6.36
7. Effectuer les tests élémentaires	2.73	(18.1%)	0.79	(15.2%)	3.46
Autres opérations	0.05	(0.3%)	0.05	(1.0%)	
Total de la détection de collisions	**15.13**	**(100.0%)**	**5.18**	**(100.0%)**	**2.92**
Total de la simulation	**42.08**		**27.85**		
Nœud (3960 triangles)					
2. Détecter triangles dans la région d'intérêt	0.85	(8.6%)	1.66	(34.1%)	0.51
3. Définir la grille	0.04	(0.4%)	0.02	(0.5%)	
4. Créer la liste de paires cellule-triangle	0.71	(7.2%)	0.23	(4.7%)	3.14
5. Trier la liste de paires cellule-triangle	0.95	(9.6%)	0.97	(20.0%)	
6.a Faire le test des cônes normaux	0.80	(8.1%)	0.27	(5.5%)	2.97
6.b Trouver les paires de triangles	2.81	(28.4%)	0.75	(15.5%)	3.73
7. Effectuer les tests élémentaires	3.69	(37.3%)	0.92	(19.0%)	4.00
Autres opérations	0.04	(0.4%)	0.04	(0.8%)	
Total de la détection de collisions	**9.90**	**(100.0%)**	**4.87**	**(100.0%)**	**2.03**
Total de la simulation	**56.99**		**27.84**		

Tableau 4.7 Temps de calcul (en millisecondes) des différentes étapes de la détection de collisions en série et en parallèle sur processeur multicœur et accélération obtenue.

CHAPITRE 5 DISCUSSION

Comme le démontrent les expériences réalisées, la méthode de détection de collisions basée sur le partitionnement de l'espace fonctionne bien dans les cas généraux de détection des collisions entre objets et d'auto-collisions. De plus, elle se compare favorablement à l'algorithme de hiérarchie de volumes englobants de SOFA. Selon les résultats de la section 4.4, la différence est particulièrement notable pour la détection des auto-collisions, où l'algorithme implémenté est 64% plus rapide que SOFA pour la scène test. Cependant, quelques problèmes ont été observés dans les expériences, notamment par rapport à la découpe des objets.

Ce chapitre présente une discussion des résultats. Les conclusions pouvant être tirées des expériences y sont discutées, ainsi que les problèmes et limitations rencontrés et les améliorations possibles pour y remédier. Les quatre premières sections portent sur les expériences réalisées. La dernière section porte sur la performance de l'algorithme, en particulier par rapport à l'objectif de permettre la simulation en temps réel.

5.1 Volumes englobants

5.1.1 Volumes englobants de triangles

Dans l'expérience sur les volumes englobants, les boîtes alignées avec les axes et les sphères englobantes ont été comparées pour leur efficacité à élaguer les triangles des surfaces dans les différentes étapes de l'algorithme et à accélérer la détection des collisions. À cet égard, les boîtes ont montré être considérablement plus efficaces. Selon les résultats du tableau 4.1, les boîtes offrent un gain en temps de calcul allant jusqu'à 42% comparativement aux sphères pour la scène du nœud. Dans les scènes des deux tores, ce gain diminue avec un nombre croissant de triangles.

Ce gain est attribuable principalement à l'efficacité accrue des boîtes à élaguer les paires de triangles lors des différentes étapes de la détection de collisions. Les données du tableau 4.2 montrent que de 7% à 61% moins de paires de triangles doivent être testées avec les boîtes qu'avec les sphères.

L'efficacité des volumes englobants à élaguer les paires de triangles dépend principalement de leur capacité à englober les triangles étroitement, diminuant ainsi leur volume et par conséquent leur probabilité d'intersecter d'autres volumes. C'est ce qui explique l'avantage observé avec les boîtes. Notamment, lorsque l'orientation d'un triangle s'approche de l'un des plans du système de coordonnées, l'épaisseur de sa boîte englobante approche de la distance

de contact, réduisant ainsi son volume et, par conséquent, sa probabilité d'intersecter une autre boîte. Au contraire, les sphères englobantes ont toujours une épaisseur considérable dans l'axe de la normale du triangle et ce peu importe leur orientation.

5.1.2 Volumes englobants des primitives de collisions

En ce qui concerne l'utilisation de volumes englobants aux primitives de collision, le tableau 4.1 montre qu'ils ralentissent généralement la détection des collisions plutôt que de l'accélérer. Pour les deux types de volumes englobants, le temps de calcul y est légèrement plus élevé avec que sans les volumes englobants. C'est le cas pour toutes les scènes à l'exception de celle du nœud et celle des tores avec la plus base résolution et utilisant les sphères.

Pourtant, comme le montre le tableau 4.3, les tests d'intersection de volumes englobants permettent d'élaguer un grand nombre de paires de primitives. Avec les boîtes englobantes, entre 30.8% et 57.0% des paires sont élaguées en moyenne. Cependant, le coût additionnel nécessaire à leur mise à jour et aux tests d'intersection est plus grand que le coût des tests de proximité sommet-triangle et arête-arête qu'ils permettent d'éviter. C'est pourquoi ils n'ont pas été employés dans les autres expériences.

5.2 Auto-collisions

5.2.1 Test des cônes normaux

Dans l'expérience portant sur les auto-collisions, les résultats démontrent que la méthode des cônes normaux améliore la performance de l'algorithme. Dans la scène utilisée, la méthode réduit le temps de calcul de 17.9%. Ce gain est obtenu en élaguant, en moyenne, 87.4% des cellules de la grille occupées. Celles-ci contiennent 67.5% des triangles et leur élagage entraîne une réduction de 52% du nombre de tests élémentaires à exécuter. Les deux étapes influencées par les cônes normaux, soit l'identification des paires de triangles à tester dans chaque cellule et les tests élémentaires, occupent plus de la 78.8% du temps de calcul de l'algorithme. C'est pourquoi la méthode permet de réduire le temps de l'algorithme de façon importante.

5.2.2 Triangles voisins

La méthode choisie pour éviter la détection de contacts entre les triangles adjacents s'est avérée inadéquate avec la réponse aux collisions par forces de pénalité. Lorsque la distance de contact est supérieure à la taille des triangles, des contacts sont détectés entre des triangles proches, mais non adjacents. Ce problème est plus important avec des maillages de surface

de plus haute résolution et ajoute un temps de calcul superflu en calculant des contacts où il ne devrait pas y en avoir.

De plus, une fraction des gains mesurés avec les tests de cônes normaux provient de l'élagage d'une partie de ces contacts superflus. Ces tests prennent en compte l'orientation des triangles dans chaque cellule pour déterminer s'ils peuvent être ignorés pour le reste des calculs d'auto-collisions. Si la courbure de la surface dans une cellule donnée est suffisamment faible, ses triangles sont élagués et le calcul des contacts entre les triangles voisins qu'elle contient est évité. Pour cette raison, la méthode des cônes normaux permet d'atténuer ce problème. Cependant, comme le montre la figure 4.4, plusieurs de ces contacts superflus subsistent et ajoutent du temps à la simulation.

Pour résoudre ce problème, la détection de contacts devrait être évitée entre les triangles se trouvant à moins d'une certaine distance l'un de l'autre le long de la surface plutôt que seulement entre les triangles adjacents. Une façon simple et efficace d'y arriver serait d'ajouter une étape à l'initialisation du programme pour déterminer, pour chaque triangle, la liste des indices de triangles avoisinants avec lesquels les collisions doivent être ignorées. Pour un triangle donné, cette liste contiendrait les indices des autres triangles se trouvant à une distance inférieure à un seuil de contact lors de l'initialisation. Durant la simulation, pour chaque paire de triangles dans une cellule donnée, il suffirait de déterminer si l'index d'un des triangles est présent dans la liste de l'autre et d'ignorer cette paire le cas échéant.

Parcourir la surface pour construire cette liste pour chaque triangle serait coûteux, mais en effectuant cette opération à l'initialisation, cette méthode ajouterait un coût négligeable durant la simulation. Cependant, une limitation de cette solution est que les paires de triangles à ignorer seraient déterminées à partir de leur position initiale. Avec des objets déformables, les triangles d'une même surface peuvent se rapprocher lorsque les objets sont déformés. C'est pourquoi le seuil de contact devrait être plus grand que la distance de contact, mais plus petit que la largeur de quelques triangles, sans quoi des triangles qui devraient normalement être en collisions pourraient être ignorés. Sa valeur pourrait être établie comme un multiple de la distance de contact ou de la taille du plus grand triangle.

Une solution plus robuste serait d'utiliser la détection de collisions continues avec la méthode des contraintes pour la réponse aux collisions. Ainsi, les triangles seraient considérés en collision que s'ils se croisent réellement au cours de leur déplacement entre deux pas plutôt que lorsqu'ils se trouvent à une certaine distance. De cette manière, aucun contact ne serait détecté entre des triangles voisins à moins qu'ils ne soient réellement en collision.

5.3 Découpe

5.3.1 Détection de collisions lors de la découpe

L'expérience n'a pas réussi à démontrer le fonctionnement de la découpe des objets dans Scyther avec la méthode de détection de collisions implémentée. Lorsque l'objet est découpé, la simulation devient instable après quelques pas.

Ce phénomène est parfois observé dans d'autres circonstances, notamment lorsque des forces trop grandes sont appliquées sur un objet. Dans ce cas, le problème provient de la résolution du système d'équations qui calcule le déplacement des degrés de liberté du modèle. Certaines conditions peuvent faire en sorte que les calculs numériques ne convergent pas vers une solution. Les degrés de liberté semblent alors être dispersés en tout sens à chaque pas, tel qu'observé lors de la découpe.

Le problème est donc certainement dû à l'utilisation des forces de pénalité comme méthode de réponse aux collisions durant la découpe. Au fur et à mesure que la lame avance un peu plus dans l'objet à chaque pas, l'algorithme de découpe génère de nouveaux triangles de part et d'autre de la coupure. Dès qu'ils sont générés, des contacts sont immédiatement détectés entre ces nouveaux triangles. Puisque les surfaces de chaque côté du plan de découpe sont très rapprochées, les forces de pénalité qui y sont appliquées sont très élevées alors que ces deux surfaces sont toujours reliées le long de la lame de l'outil.

La réponse aux collisions par forces de pénalité et l'algorithme de découpe ne sont donc pas compatibles dans leur forme actuelle. Ce problème requiert de modifier au moins l'un des deux. Pour l'algorithme de découpe, une solution serait d'élargir la distance entre les surfaces générées de façon à réduire la magnitude des forces de pénalité. Cependant, cette distance risque de réduire le réalisme visuel de la découpe. Une meilleure solution serait d'améliorer la méthode de calcul des déformations pour qu'elle puisse traiter des forces externes plus élevées.

Pour la réponse aux collisions, une solution serait d'utiliser la détection de collisions continue et la réponse aux collisions par contraintes. Cette méthode éliminerait complètement l'addition de forces de pénalité sur la surface et, par conséquent, le problème d'instabilité qu'elles causent. De plus, elle permettrait d'éviter l'interpénétration des surfaces de part et d'autre de la coupure sans avoir à maintenir de distance entre elles contrairement aux forces de pénalité. Cet avantage améliorerait le réalisme de la découpe de tissus avec un scalpel, une étape où le réalisme doit être particulièrement important pour la simulation de plusieurs opérations chirurgicales.

5.3.2 Séparation des objets

Une fois que la découpe fonctionnera, une autre amélioration requise sera de détecter lorsqu'un objet est scindé en deux et de le séparer en deux objets distincts. Actuellement, même si deux parties d'un objet deviennent déconnectées à la suite d'une découpe, ils continuent de faire partie du même objet. Cela signifie qu'un seul volume englobant couvre les deux parties, même si elles deviennent très éloignées dans l'espace. Dans une scène avec plusieurs objets, certains d'entre eux risquent de chevaucher ce volume englobant même s'ils sont loin de chacune des deux parties, déclenchant ainsi la détection de collisions inutilement. Il serait donc utile de détecter lorsqu'un objet est scindé et de séparer ses différentes structures de données, dont le maillage de surface, les différentes structures du modèle mécanique et les mappages entre ces structures. Cette opération serait coûteuse, mais elle devrait généralement pouvoir être répartie sur plusieurs pas de simulation avant que la situation décrite plus haut devienne problématique.

5.4 Parallélisation

La recherche sur la détection de collisions est à présent surtout tournée vers les algorithmes parallèles. C'est l'une des motivations pour le choix de la méthode choisie pour ce projet. Comme le montrent les résultats obtenus avec OpenMP, l'algorithme se prête bien au parallélisme de données sur CPU multicœur. Sur un processeur à six cœurs, la simulation de la scène testée est accélérée de 2.03 à 2.92 fois par rapport au calcul en série. Les scènes ayant le plus grand nombre de triangles affichent la plus grande accélération.

Un avantage de la méthode choisie est que la presque totalité de la détection de collisions peut être exécutée en parallèle. Selon les données du tableau 4.7, les quatre étapes de l'algorithme qui ont été parallélisées totalisent entre 87% et 91% du temps de calcul en série pour les différentes scènes. La principale étape manquante est le tri de la liste des paires cellule-triangle qui a été implémentée avec la fonction `std::sort` de la librairie standard de C++. Un algorithme approprié qui pourrait être utilisé pour le tri en parallèle est le tri par base (*radix sort* en anglais). Le Grand [47] présente en détail une implémentation du tri par base pour le partitionnement de l'espace sur GPU. Si le tri était parallélisé, la charge de travail en parallèle totaliserait 98% à 99% de la charge totale. Selon la loi d'Amdahl, l'accélération théorique attendue en augmentant le nombre de fils d'exécution est principalement limitée par la proportion des calculs bénéficiant de l'amélioration des ressources. Ainsi, un gain important peut être attendu avec un plus grand nombre de cœurs ou en portant l'algorithme sur GPU.

5.4.1 Sections critiques

Un facteur qui limite l'accélération observée est l'utilisation de sections critiques pour combiner les résultats des différentes étapes dans un même vecteur. Les différents fils d'exécution passent donc une partie de leur temps en attente.

Quelques changements à l'algorithme pourraient améliorer son accélération en parallèle. Principalement, la mémoire requise pour enregistrer les résultats à chaque étape pourrait être allouée à l'avance. Toutefois, le nombre d'éléments produits à chaque pas n'est pas prévisible. Il faudrait donc calculer le nombre d'éléments maximal pouvant être produit et allouer la mémoire en conséquence.

Dans la première étape, celle de la détection des triangles dans la région d'intérêt, ce nombre correspond au nombre de triangles dans la surface des objets en collision. Toutefois, seulement une faible proportion de ces triangles se trouve généralement dans la région d'intérêt. Ce changement créerait donc un vecteur creux et de taille beaucoup plus grande et serait plus coûteux en mémoire.

Cette amélioration serait particulièrement bénéfique à la détection des auto-collisions. Dans la scène du nœud, l'exécution sur six cœurs n'a pas accéléré cette étape par rapport à l'exécution en série. Au contraire, son temps de calcul a presque doublé. Pour la détection des auto-collisions, la région d'intérêt couvre l'objet en entier. Par conséquent, les tests d'intersection des triangles avec la région d'intérêt ne sont pas nécessaires contrairement à la détection de collisions entre deux objets. Tous les triangles sont simplement ajoutés à la liste. Ainsi, une plus grande partie du temps est passée dans la section critique, ce qui explique le ralentissement observé. Puisque tous les triangles de l'objet doivent être pris en compte, le nombre d'éléments est prévisible et allouer la mémoire à l'avance réglerait ce problème sans coût additionnel.

Dans l'étape de la création de la liste des paires triangle-cellule, où les cellules intersectées par les triangles sont détectées et leurs valeurs de hachage calculées, le nombre maximal d'éléments est huit fois le nombre de triangles dans la région d'intérêt puisque chaque triangle peut intersecter au plus huit cellules. Les éléments vides peuvent être marqués comme invalide et ignorés lors du tri à l'étape suivante.

L'étape de l'identification des paires de triangles à tester a été parallélisée avec une tâche par cellule. Plutôt que d'enregistrer les paires de triangles à tester, le code pourrait être réusiné pour calculer directement les contacts entre les triangles dès qu'une paire est identifiée. Ceci éliminerait complètement le besoin d'une section critique pour cette étape.

5.4.2 Parallélisation sur GPU

L'implémentation a été parallélisée sur processeur multicœur, mais une meilleure accélération serait attendue en portant le code sur GPU. Certains changements pourraient alors être avantageux pour tirer profit du plus grand nombre de fils d'exécution.

L'étape de la détection des triangles dans la région d'intérêt occupe une grande partie du temps de calcul de l'algorithme. Une amélioration possible serait d'éliminer les tests d'intersection des triangles avec la région d'intérêt et de simplement inclure tous les triangles des surfaces, comme il est déjà fait pour les auto-collisions. L'étape ne consisterait alors qu'à mettre à jour les volumes englobants des triangles et à trouver la taille du plus grand triangle pour déterminer la taille des cellules de la grille. Les calculs supplémentaires causés par le plus grand nombre d'éléments seraient mitigés par le nombre de fils d'exécution beaucoup plus grand. Il faudrait tester si le temps gagné par l'élimination des tests d'intersections compense le nombre accru d'éléments.

De plus, les volumes englobants des objets ne seraient alors testés qu'avec ceux des autres objets et non plus avec les milliers de triangles qu'ils englobent. Des volumes englobants plus complexes, comme les k-DOP, pourraient élaguer davantage les paires d'objets pour un coût additionnel négligeable, contrairement à s'ils étaient toujours testés avec l'ensemble des triangles.

Une autre amélioration possible serait d'appliquer le partitionnement de l'espace à toute la scène et de détecter les collisions pour l'ensemble des objets simultanément. Ainsi, l'algorithme ne serait appelé qu'une seule fois par pas plutôt qu'une fois par paire d'objets et par objet pouvant subir des auto-collisions. Ce changement pourrait être bénéfique lorsque des objets sont en contact avec plusieurs autres objets à la fois. Ceci éviterait de répéter certains calculs comme celui des valeurs de hachages des triangles. Ce changement permettrait aussi de maximiser le nombre de fils d'exécution utilisés lorsque les surfaces de certains objets contiennent peu de triangles relativement au nombre de fils d'exécution.

À la section précédente, il a été suggéré de ne pas enregistrer les paires de triangles identifiées dans chaque cellule et de tester les contacts entre eux directement. Ceci implique de répartir le travail avec une cellule par fil d'exécution. Cependant, pour tirer profit du grand nombre de fils d'exécution d'un GPU, il faudrait au moins plusieurs milliers de cellules. Une meilleure utilisation du GPU serait accomplie en divisant le travail par paire de triangles plutôt que par cellule. Il faudrait alors connaître le nombre de paires de triangles à tester pour organiser la répartition du travail. Selon les critères décrits à la section 3.6.2, une paire de triangles est seulement considérée dans une cellule donnée si au moins un des deux triangles y possède son

centroïde. Ainsi, pour une cellule intersectée par n triangles ayant leur centroïde dans cette cellule et m triangles ayant leur centroïde dans une cellule adjacente, le nombre de paires à tester est donné par :

$$\frac{n(n-1)}{2} + nm \tag{5.1}$$

Le nombre total de paires de triangles à tester est la somme du nombre dans chaque cellule.

5.5 Performances

Comme spécifié au début du chapitre, les résultats à la section 4.4 montrent que la solution basée sur le partitionnement de l'espace implémentée pour ce travail se compare favorablement à l'algorithme de hiérarchie de volumes englobants de SOFA. Cependant, le principal critère de performance pour la simulation chirurgicale est sa capacité à être exécutée en temps réel. À la section 2.8.4, on avait émis comme hypothèse que la solution choisie pouvait satisfaire ce critère en permettant un temps de calcul d'au plus 50 ms par pas, soit un rendu d'au moins 20 Hz, pour des scènes de plusieurs dizaines de milliers de triangles.

Dans toutes les scènes utilisées pour les expériences, le temps de calcul de la détection de collisions est inférieur à 50 ms aussi bien en série qu'en parallèle. Cependant, l'exécution en temps réel dépend du temps de la simulation au complet et non seulement de la détection des collisions. Comme le montrent les résultats du tableau 4.7, le temps total de la simulation est inférieur à 50 ms pour toutes les scènes exécutées en parallèle sur un processeur à six cœurs. Pour le calcul en série, le temps est également inférieur à 50 ms pour la scène des deux tores, même avec la plus haute résolution des surfaces, soit 51 200 triangles. Par contre, la scène du nœud excède les 50 ms avec un temps d'exécution de 56,99 ms et ce malgré une surface de seulement 3960 triangles. Ceci démontre la difficulté de simuler les auto-collisions en temps réel pour des scènes de complexité requise pour la simulation chirurgicale.

Avec ces résultats, on pourrait conclure que la solution proposée permet la simulation en temps réel sur un processeur multicœur. Cependant, il y a plusieurs choses à considérer. Premièrement, le temps de calcul de la détection de collisions dépend non seulement du nombre de triangles dans la scène, mais aussi de la disposition des objets dans l'espace et le nombre de contacts qui en résultent. Pour les collisions entre objets, cela influence le nombre de triangles dans les régions d'intérêt, le nombre de paires de triangles à tester et le nombre de tests élémentaires à exécuter. La scène des deux tores avec 51 200 triangles donne un temps de calcul moyen de 27.85 ms. Il n'y a aucune garantie qu'une scène différente contenant le même nombre de triangles, mais avec un plus grand nombre de contacts, n'excéderait pas les 50 ms dans le pire cas.

Deuxièmement, comme mentionnée plus haut, la détection des auto-collisions est particulièrement coûteuse. De plus, contrairement aux collisions entre objets où la détection doit seulement être exécutée lorsque les volumes englobants s'intersectent, les auto-collisions doivent être détectées à chaque pas. L'expérience avec la scène du nœud en parallèle a donné un temps de calcul de 27.84 ms avec une surface de seulement 3960 triangles. Dans un scénario de simulation chirurgicale comprenant des dizaines de milliers de triangles, la détection des auto-collisions pourrait être requise pour un plus grand nombre d'objets et pour des objets plus complexes. Encore une fois, les résultats ne garantissent pas que le temps de calcul ne dépasse jamais 50 ms.

Troisièmement, pour régler le problème rencontré avec la découpe, on a proposé l'usage de la méthode de réponse aux collisions par la méthode des contraintes, au moins pour les objets subissant une découpe. Cependant, l'usage de cette méthode augmenterait le temps de calcul pour la résolution du système dynamique. Généralement, la boucle d'animation doit être modifiée pour diviser les pas de simulation en deux étapes. La première résout le système dynamique sans tenir compte des interactions alors que la deuxième corrige le système de façon à satisfaire les contraintes. Comme le montre le tableau 4.7, la détection des collisions représente une minorité du temps de calcul. La majorité est plutôt consacrée à la résolution du système dynamique. La méthode de réponse aux collisions par contraintes pourrait donc avoir un impact important sur la performance de la simulation.

Par conséquent, les résultats obtenus sont probablement trop près de la limite pour assurer l'exécution en temps réel, même en parallèle sur un processeur multicœur. Une façon d'assurer le rendu en temps réel en pire cas serait d'adapter l'algorithme pour le calcul sur GPU. Comme mentionné au chapitre 2, certains auteurs affirment réduire le temps de calcul de la détection de collisions sur GPU par un ordre de grandeur par rapport à l'exécution en série. C'est le cas de Pabst *et al.* [41] sur lequel la méthode de partitionnement spatial pour ce projet est partiellement inspirée. Puisque dans les scènes testées la majorité du temps est consacrée à la résolution du système dynamique, cette partie devrait aussi être portée sur GPU. Cependant, ce dernier est en dehors du cadre de ce projet.

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Cette étude avait pour but de développer un algorithme de détection de collisions dans la librairie Scyther. L'algorithme développé est basé sur le partitionnement uniforme de l'espace et permet de détecter les collisions entre objets de même que les auto-collisions. Il a été comparé avec la méthode de détection basée sur les hiérarchies de volumes englobants implémentée dans SOFA et s'est montré plus rapide, particulièrement pour la détection des auto-collisions.

L'analyse des résultats a permis de vérifier l'efficacité de différentes étapes d'élagage des triangles et, par le fait même, de déterminer quelques paramètres qui améliorent la performance de l'algorithme. D'abord, les boîtes englobantes alignées avec les axes se sont montrées plus efficaces que les sphères englobantes pour élaguer les triangles et réduire le temps de calcul au cours des différentes étapes de l'algorithme. De plus, utiliser des volumes englobants pour élaguer les primitives de collisions (faces et arêtes des triangles de la surface) s'est avéré ralentir la simulation plutôt que de l'accélérer. Enfin, la méthode des cônes normaux a démontré être efficace à réduire davantage le temps de calcul de la détection des auto-collisions en permettant d'élaguer des cellules entières.

Les résultats ont aussi permis de vérifier les hypothèses émises à la section 2.8.4. L'hypothèse **H1**, qui portait sur la parallélisation de l'algorithme sur processeur multicœur, a été confirmée. Sur un processeur à six cœurs, les différentes scènes ont été accélérées de 2.03 à 2.92 fois par rapport au calcul en série, avec les scènes les plus complexes affichant une plus grande accélération. De plus, l'accélération pourrait être augmentée avec les modifications proposées à la section 5.4.1 pour réduire le temps en attente des fils d'exécution lors de l'enregistrement des résultats aux différentes étapes.

L'hypothèse **H2**, qui portait sur la simulation en temps réel, a été confirmée, au moins avec le parallélisme sur un processeur multicœur. Cependant, avec des scènes de plusieurs dizaines de milliers de triangles, les résultats sont près de la limite de 50 ms par pas, soit une fréquence de 20 Hz, nécessaire pour le rendu en temps réel. Comme discuté à la section 5.5, des scènes différentes, particulièrement avec davantage d'auto-collisions, pourraient excéder ce seuil. Pour cette raison, porter Scyther sur GPU offrirait une meilleure garantie d'exécution en temps réel. Puisque la détection de collisions occupe qu'une minorité du temps de calcul, les calculs de déformations et de déplacements des objets devraient également être portés sur

GPU.

L'hypothèse **H3**, qui portait sur les changements de topologies, n'a pas pu être confirmée puisque le fonctionnement de l'algorithme lors de la découpe d'un objet n'a pas pu être démontré. Le problème ne provient pas directement de la détection, mais plutôt de la réponse aux collisions. Plus précisément, il est causé par l'interaction de la réponse aux collisions par forces de pénalité, la configuration des nouveaux triangles générés lors de la découpe et la résolution du système d'équations pour le calcul de déformation et de déplacement des objets. Essentiellement, les forces de pénalité ajoutées lors d'une coupure entraînent des instabilités numériques dans les calculs de déformations du modèle. Comme proposé à la section 5.3, utiliser la méthode des contraintes pour la réponse aux collisions réglerait ce problème. Une autre solution serait d'améliorer les calculs de déformation des objets pour que ceux-ci soient en mesure de traiter les forces de pénalité lors de la découpe. Une fois ce problème résolu, il serait nécessaire de pouvoir détecter lorsqu'un objet est coupé en deux et de séparer ses structures de données de façon à avoir deux objets distincts.

6.2 Améliorations futures

En plus des améliorations proposées au chapitre précédent pour répondre aux limitations observées dans cette étude, quelques autres sont envisageables pour faire face aux besoins de la simulation chirurgicale.

Une extension possible à ce travail serait d'étudier l'usage de grilles hiérarchiques pour le partitionnement de l'espace. Une limitation du partitionnement par grilles uniformes est que les triangles doivent être de taille semblable pour que la méthode soit efficace. Autrement, l'espace est partitionné en de plus grandes cellules. Celles-ci contiennent alors un plus grand nombre de paires de triangles, augmentant le nombre de tests de collisions à exécuter. Cette limitation n'était pas un problème pour ce travail puisque les surfaces utilisées étaient composées de triangles de taille relativement semblable. Cependant, simuler des scènes avec des triangles de taille très variables pourrait être utile dans l'avenir. Par exemple, de grandes surfaces planes et rigides, comme la surface d'une table, sont plus facilement modélisées par un petit nombre de grands triangles.

Une autre amélioration future serait d'élaborer la simulation de l'outil de découpe. Actuellement, l'outil est modélisé que par un fil de lame qui se déplace et tranche les objets sur sa trajectoire. L'outil devrait être représenté comme un objet plus complet avec des surfaces pouvant subir des collisions. Pour être plus réaliste, le mécanisme de découpe pourrait être mieux intégré à l'algorithme de détection de collisions pour que la lame puisse aussi bien

pousser sur l'objet que le trancher, dépendant de la pression appliquée. Finalement, pour l'entraînement de chirurgiens, l'outil devrait pouvoir être contrôlé par un appareil haptique.

RÉFÉRENCES

[1] N. E. Seymour, A. G. Gallagher, S. A. Roman, M. K. O'Brien, V. K. Bansal, D. K. Andersen et R. M. Satava, "Virtual Reality Training Improves Operating Room Performance," *Annals of Surgery*, vol. 236, n°. 4, p. 458–464, oct. 2002.

[2] I. Badash, K. Burtt, C. A. Solorzano et J. N. Carey, "Innovations in surgery simulation: A review of past, current and future techniques," *Annals of Translational Medicine*, vol. 4, n°. 23, déc. 2016.

[3] V. Magnoux et B. Ozell, "Real-time visual and physical cutting of a meshless model deformed on a background grid," *Computer Animation and Virtual Worlds*, vol. 31, n°. 6, p. e1929, 2020.

[4] ——, "Dynamic Cutting of a Meshless Model for Interactive Surgery Simulation," dans *Augmented Reality, Virtual Reality, and Computer Graphics*, ser. Lecture Notes in Computer Science, L. T. De Paolis et P. Bourdot, édit. Cham : Springer International Publishing, 2020, p. 114–130.

[5] ——, "GPU-friendly data structures for real time simulation," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 8, n°. 1, p. 7, mars 2021.

[6] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik et S. Cotin, "SOFA: A Multi-Model Framework for Interactive Physical Simulation," dans *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, ser. Studies in Mechanobiology, Tissue Engineering and Biomaterials, Y. Payan, édit. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 283–321.

[7] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser et P. Volino, "Collision Detection for Deformable Objects," *Computer Graphics Forum*, vol. 24, n°. 1, p. 61–81, mars 2005.

[8] P. Hubbard, "Collision detection for interactive graphics applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, n°. 3, p. 218–230, sept. 1995.

[9] C. Ericson, *Real-Time Collision Detection*. Boca Raton, FL, USA : CRC Press, Inc., 2004.

[10] H. Delingette et N. Ayache, "Soft Tissue Modeling for Surgery Simulation," dans *Handbook of Numerical Analysis*, ser. Computational Models for the Human Body. Elsevier, janv. 2004, vol. 12, p. 453–550.

[11] D. Baraff, “Dynamic Simulation of Non-Penetrating Rigid Bodies,” Thèse de doctorat, Cornell University, mars 1992.

[12] J. D. Cohen, M. C. Lin, D. Manocha et M. Ponamgi, “I-COLLIDE: An Interactive and Exact Collision Detection System for Large-scale Environments,” dans *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, ser. I3D ’95. New York, NY, USA : ACM, 1995, p. 189–196.

[13] P. M. Hubbard, “Approximating Polyhedra with Spheres for Time-critical Collision Detection,” *ACM Trans. Graph.*, vol. 15, n°. 3, p. 179–210, juill. 1996.

[14] I. J. Palmer et R. L. Grimsdale, “Collision Detection for Animation using Sphere-Trees,” *Computer Graphics Forum*, vol. 14, n°. 2, p. 105–116, 1995.

[15] G. van den Bergen, “Efficient Collision Detection of Complex Deformable Models using AABB Trees,” *Journal of Graphics Tools*, vol. 2, n°. 4, p. 1–13, janv. 1997.

[16] W. Xiao-rong, W. Meng et L. Chun-gui, “Research on Collision Detection Algorithm Based on AABB,” dans *2009 Fifth International Conference on Natural Computation*, vol. 6, août 2009, p. 422–424.

[17] S. Gottschalk, M. C. Lin et D. Manocha, “OBBTree: A hierarchical structure for rapid interference detection,” dans *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH ’96*. ACM Press, 1996, p. 171–180.

[18] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral et K. Zikan, “Efficient collision detection using bounding volume hierarchies of k-DOPs,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, n°. 1, p. 21–36, janv. 1998.

[19] M. Tang, D. Manocha, S.-E. Yoon, P. Du, J.-P. Heo et R.-F. Tong, “VolCCD: Fast continuous collision culling between deforming volume meshes,” *ACM Transactions on Graphics*, vol. 30, n°. 5, p. 1–15, oct. 2011.

[20] X. Provot, “Collision and self-collision handling in cloth model dedicated to design garments,” dans *Computer Animation and Simulation ’97*, 1er éd., ser. Eurographics, W. Hansmann, W. T. Hewitt, W. Purgathofer, D. Thalmann et M. van de Panne, édit. Vienna : Springer Vienna, 1997, p. 177–189.

[21] M. Tang, S. Curtis, S. Yoon et D. Manocha, “ICCD: Interactive Continuous Collision Detection between Deformable Models Using Connectivity-Based Culling,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, n°. 4, p. 544–557, juill. 2009.

[22] T. Larsson et T. Akenine-Möller, “Collision Detection for Continuously Deforming Bodies,” *Eurographics Conference*, p. 325–333, 2001.

[23] ——, “A dynamic bounding volume hierarchy for generalized collision detection,” *Computers & Graphics*, vol. 30, n°. 3, p. 450–459, juin 2006.

[24] J. Mezger, S. Kimmerle et O. Etzmuß, “Hierarchical techniques in collision detection for cloth animation,” *Journal of WSCG*, vol. 11, n°. 2, p. 322–329, 2003.

[25] M. A. Otaduy, O. Chassot, D. Steinemann et M. Gross, “Balanced Hierarchies for Collision Detection between Fracturing Objects,” dans *2007 IEEE Virtual Reality Conference*, mars 2007, p. 83–90.

[26] H. Jung et D. Y. Lee, “Real-time cutting simulation of meshless deformable object using dynamic bounding volume hierarchy,” *Computer Animation and Virtual Worlds*, vol. 23, n°. 5, p. 489–501, 2012.

[27] V. Kumar, V. N. Rao et K. Ramesh, “Parallel Depth First Search on the Ring Architecture,” University of Texas at Austin, Austin, TX, USA, Rapport technique, 1988.

[28] V. Kumar, A. Y. Grama et N. R. Vempaty, “Scalable Load Balancing Techniques for Parallel Computers,” *Journal of Parallel and Distributed Computing*, vol. 22, n°. 1, p. 60–79, juill. 1994.

[29] A. Reinefeld et V. Schnecke, “Work-load balancing in highly parallel depth-first search,” dans *Proceedings of IEEE Scalable High Performance Computing Conference*. Knoxville, TN, USA : IEEE Comput. Soc. Press, 1994, p. 773–780.

[30] O. Tropp, A. Tal, I. Shimshoni et D. P. Dobkin, “Temporal coherence in bounding volume hierarchies for collision detection,” *International Journal of Shape Modeling*, vol. 12, n°. 02, p. 159–178, déc. 2006.

[31] M. Tang, D. Manocha, J. Lin et R. Tong, “Collision-streams: Fast GPU-based collision detection for deformable models,” dans *Symposium on Interactive 3D Graphics and Games*. ACM Press, 2011, p. 63–70.

[32] X. Zhang et Y. J. Kim, “Scalable Collision Detection Using p-Partition Fronts on Many-Core Processors,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, n°. 3, p. 447–456, mars 2014.

[33] X. Wang, M. Tang, D. Manocha et R. Tong, “Efficient BVH-based Collision Detection Scheme with Ordering and Restructuring,” *Computer Graphics Forum*, vol. 37, n°. 2, p. 227–237, mai 2018.

[34] D. Kim, J.-P. Heo, J. Huh, J. Kim et S.-e. Yoon, “HPCCD: Hybrid Parallel Continuous Collision Detection using CPUs and GPUs,” *Computer Graphics Forum*, vol. 28, n°. 7, p. 1791–1800, 2009.

[35] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke et D. Manocha, “Fast BVH Construction on GPUs,” *Computer Graphics Forum*, vol. 28, n°. 2, p. 375–384, avr. 2009.

[36] C. Lauterbach, Q. Mo et D. Manocha, “gProximity: Hierarchical GPU-based Operations for Collision and Distance Queries,” *Computer Graphics Forum*, vol. 29, n°. 2, p. 419–428, 2010.

[37] a. M. M. F. Yuen, “Collision detection for clothed human animation,” dans *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*, oct. 2000, p. 328–337.

[38] M. Teschner, B. Heidelberger, M. Muller, D. Pomeranets et M. Gross, “Optimized Spatial Hashing for Collision Detection of Deformable Objects,” *Modeling, and Visualization : Proceedings*, vol. 3, p. 47–54, 2003.

[39] M. Eitz et G. Lixu, “Hierarchical Spatial Hashing for Real-time Collision Detection,” dans *IEEE International Conference on Shape Modeling and Applications 2007 (SMI ’07)*, juin 2007, p. 61–70.

[40] H. Jin, Z. Liu, T. Wu et Y. Wang, “The Research of Collision Detection Algorithm Based on Spatial Subdivision,” dans *2009 International Conference on Computer Engineering and Technology*, vol. 2, janv. 2009, p. 452–455.

[41] S. Pabst, A. Koch et W. Straßer, “Fast and Scalable CPU/GPU Collision Detection for Rigid and Deformable Surfaces,” *Computer Graphics Forum*, vol. 29, n°. 5, p. 1605–1612, juill. 2010.

[42] F. Ganovelli, J. Dingliana et C. O’Sullivan, “BucketTree : Improving Collision Detection Between Deformable Objects,” *Proc. of Spring Conference on Computer Graphics SCCG’00*, p. 8, 2000.

[43] T. H. Wong, G. Leach et F. Zambetta, “An adaptive octree grid for GPU-based collision detection of deformable objects,” *The Visual Computer*, vol. 30, n°. 6, p. 729–738, juin 2014.

[44] S. Melax, “Dynamic Plane Shifting BSP Traversal,” *Proceedings of the Graphics Interface 2000 Conference*, vol. 2000, p. 213–220, 2000.

[45] R. G. Luque, J. L. D. Comba et C. M. D. S. Freitas, “Broad-phase Collision Detection Using Semi-adjusting BSP-trees,” dans *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, ser. I3D ’05. New York, NY, USA : ACM, 2005, p. 179–186.

[46] K. Glass, “Analysis of Broad-Phase Spatial Partitioning Optimisations in Collision Detection,” Grahamstown, South Africa : Rhodes University, Rapport technique, 2005.

[47] S. Le Grand. (2007) Broad-phase collision detection with cuda. [En ligne]. Disponible : https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch32.html Accédé le 2019-03-08.

[48] F. Liu, T. Harada, Y. Lee et Y. J. Kim, "Real-time Collision Culling of a Million Bodies on Graphics Processing Units," dans *ACM SIGGRAPH Asia 2010 Papers*, ser. SIGGRAPH ASIA '10. New York, NY, USA : ACM, 2010, p. 154 :1–154 :8.

[49] T. H. Wong, G. Leach et F. Zambetta, "Virtual subdivision for GPU based collision detection of deformable objects using a uniform grid," *The Visual Computer*, vol. 28, nº. 6, p. 829–838, juin 2012.

[50] W. Fan, B. Wang, J.-C. Paul et J. Sun, "A Hierarchical Grid Based Framework for Fast Collision Detection," *Computer Graphics Forum*, vol. 30, nº. 5, p. 1451–1459, 2011.

[51] S. F. Frisken, R. N. Perry, A. P. Rockwood et T. R. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," dans *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '00. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 2000, p. 249–254.

[52] J. Wu et L. Kobbelt, "Piecewise Linear Approximation of Signed Distance Fields," *Modeling, and Visualization : Proceedings*, vol. 3, p. 513–520, 2003.

[53] M. W. Jones, J. A. Baerentzen et M. Sramek, "3D distance fields: A survey of techniques and applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, nº. 4, p. 581–599, juill. 2006.

[54] J. Lombardo, M. Cani et F. Neyret, "Real-time collision detection for virtual surgery," dans *Proceedings Computer Animation 1999*, mai 1999, p. 82–90.

[55] B. Heidelberger, M. Teschner et M. Gross, "Real–Time Volumetric Intersections of Deforming Objects," dans *Modeling, and Visualization : Proceedings*, vol. 3, 2003, p. 461–468.

[56] J. Shade, S. Gortler, L.-w. He et R. Szeliski, "Layered Depth Images," dans *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '98. New York, NY, USA : ACM, 1998, p. 231–242.

[57] B. Heidelberger, M. Teschner et M. Gross, "Detection of Collisions and Self-collisions Using Image-space Techniques," dans *Journal of WSCG*, 2004, p. 145–152.

[58] F. Faure, J. Allard, F. Falipou et S. Barbier, "Image-based Collision Detection and Response between Arbitrary Volumetric Objects," dans *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, juill. 2008, p. 155–162.

[59] J. Allard, F. Faure, H. Courtecuisse, F. Falipou, C. Duriez et P. G. Kry, "Volume contact constraints at arbitrary resolution," *ACM Transactions on Graphics*, vol. 29, nº. 4, p. 1, juill. 2010.

[60] E. Hermann, F. Faure et B. Raffin, “Ray-traced collision detection for deformable bodies,” dans *GRAPP 2008 - 3rd International Conference on Computer Graphics Theory and Applications*. INSTICC, janv. 2008, p. 293–299.

[61] D. Eberly. (1999) Distance Between Point and Triangle in 3D. [En ligne]. Disponible : https://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf Accédé le 2020-07-14.

[62] C. Ericson, “Chapter 5 - Basic Primitive Tests,” dans *Real-Time Collision Detection*, ser. The Morgan Kaufmann Series in Interactive 3D Technology, C. Ericson, édit. San Francisco : Morgan Kaufmann, janv. 2005, p. 125–233.