



Détection de collision par lancer de rayon : La quête de la performance

François Lehericey

► To cite this version:

François Lehericey. Détection de collision par lancer de rayon : La quête de la performance. Synthèse d'image et réalité virtuelle [cs.GR]. INSA de Rennes, 2016. Français. NNT : 2016ISAR0011 . tel-01427732

HAL Id: tel-01427732

<https://tel.archives-ouvertes.fr/tel-01427732>

Submitted on 6 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse



UNIVERSITE
BRETAGNE
LOIRE

THESE INSA Rennes
sous le sceau de l'Université Bretagne Loire
pour obtenir le titre de
DOCTEUR DE L'INSA RENNES
Spécialité : Informatique

présentée par
François Lehericey
ECOLE DOCTORALE : MATISSE
LABORATOIRE : IRISA

Détection de collision par lancer de rayon : la quête de la performance

Thèse soutenue le 20.09.2016
devant le jury composé de :

Jean-Louis Pazat

Professeur à l'INSA de Rennes / Président

François Faure

Professeur à l'université Joseph Fourier de Grenoble / Rapporteur

Laurent Grisoni

Professeur à l'université de Lille 1 / Rapporteur

Bruno Raffin

Directeur de recherche à Inria Grenoble Rhône-Alpes / Examinateur

Valérie Gouranton

Maître de conférences à l'INSA de Rennes / Co-encadrante de thèse

Bruno Arnaldi

Professeur à l'INSA de Rennes / Directeur de thèse

Détection de collision par lancer de rayon :
la quête de la performance

François Lehericey



En partenariat avec



"Il y a une théorie qui dit que si un jour quelqu'un découvre à quoi sert l'univers et pourquoi il est là, il disparaîtra immédiatement pour être remplacé par quelque chose d'encore plus bizarre et inexplicable.

Il y a une autre théorie qui dit que cela s'est déjà produit."

Douglas Adams,
Le dernier restaurant avant la fin du monde

"Dans notre monde, la réalité absolue n'existe pas. La plupart des choses qui passent pour réelles sont fictives. Tout ce qu'on voit semble réel dans la mesure où notre cerveau le conçoit ainsi."

Solid Snake,
Metal Gear Solid 2

Remerciements

Je souhaite remercier tous ceux qui ont participé de près ou de loin à cette thèse. Je remercie en premier Bruno Arnaldi, directeur de thèse et Valérie Gouranton, co-encadrante pour leur soutien et leurs conseils qui m'ont accompagné tout au long de cette thèse. Je voudrais aussi remercier le jury : François Faure, Laurent Grisoni, Bruno Raffin et Jean-louis Pazat pour leur participation et leur remarques constructives sur le manuscrit et la soutenance. Je remercie le ministère de l'enseignement supérieur et de la recherche pour avoir financé cette thèse.

Je voudrais aussi remercier l'ensemble de membres de l'équipe Hybrid dirigée habilement par Anatole Lecuyer pour m'avoir accueilli et pour toutes les discussions productives (ou pas) qui ont été faites en réunion d'équipe, aux séminaires au vert ou encore autour d'un café. Je remercie mes co-bureaux pour la bonne ambiance au quotidien : Guillaume Claude qui fait découvert le Wushu et Morgan Le Chenechal. La bonne ambiance était aussi présente lors des déjeuners Super Smash Bros. le mardi midi avec entre autres Florian Nouviale, Andeol Evain, Benoît Le Gouis, Gwendal Le Moulec, Quentin Petit, Kevin Jordao, Tristan Le Bouffant, Julian Josef et tous ceux que j'aurai pu oublier.

Je remercie aussi mes amis, notamment Romain Lebarbenchon et Olivier Leblay ainsi que ma famille pour avoir relu ce manuscrit en quête des fautes d'orthographe, mais aussi pour leur soutien tout au long de ces trois ans.

Table des matières

Introduction	1
1 Domaines d'application	2
2 Problématique et objectif de cette thèse	3
3 Organisation du mémoire	4
1 État de l'art	5
1 Approche préliminaire	5
1.1 Modèles de représentation	5
1.2 Types de requêtes	9
1.3 Dynamicité de l'environnement	11
1.4 Synthèse	13
2 Pipeline de détection de collision	13
2.1 Broad-phase	14
2.2 Narrow-Phase	17
3 Optimisations	28
3.1 Culling	29
3.2 Simulation de tissus multi-résolution	30
3.3 Modélisation de l'espace entre les objets	30
4 Solutions GPUs	30
4.1 Évolution des GPUs	31
4.2 Broad-phase GPU	32
4.3 Narrow-phase GPU	33
5 Synthèse	34
2 Cadre général des contributions	37
1 Lancer de rayon	38
1.1 Structures accélératrices	38
1.2 Cohérence spatiale entre les rayons	38
1.3 Cohérence temporelle de la caméra	39
1.4 Rendu sur GPU	39
1.5 Synthèse	39
2 Exploitation de la cohérence temporelle	40
3 Prédiction de collisions	42
4 Utilisation de différents algorithmes de lancer de rayon	45
5 Base algorithmique commune sur la détection de collision surfacique/volumique	45
6 Exploitation des processeurs graphiques et pipeline	46

7	Synthèse	47
3	Pipeline de détection de collision	49
1	Mise en place d'un pipeline générique	49
1.1	Éléments du pipeline	51
1.2	Organisation du pipeline	52
1.3	Prédiction de remplissage des tampons	55
1.4	Évaluation des performances	57
1.5	Synthèse et perspectives	61
2	Prédiction de collisions	62
2.1	Méthode de prédiction	62
2.2	Impact sur les performances	63
2.3	Synthèse	65
3	Re-projection des rayons	66
3.1	Méthode de re-projection des rayons	67
3.2	Évaluation de la re-projection des rayons	68
3.3	Synthèse	69
4	Détection de collision pour les objets surfaciques	70
4.1	Principe général de la méthode	70
4.2	Détection de collision pour les tissus	72
4.3	Évaluation des performances	79
4.4	Synthèse et perspectives	80
5	Conclusion	82
4	Exploitation de la cohérence temporelle	85
1	Lancer de rayon itératif	85
1.1	Intersection avec un maillage de triangles	86
1.2	Intersection avec un triangle	87
1.3	Factorisation des prédictions avec l'algorithme itératif	89
1.4	Stockage des données temporelles	90
1.5	Synthèse	90
2	Application aux objets rigides	91
2.1	Mesure de déplacements relatifs	92
2.2	Évaluation	92
2.3	Synthèse	96
3	Utilisation des prédictions pour la robustesse	97
3.1	Utilisation des prédictions avec la méthode itérative	98
3.2	Évaluation	98
3.3	Synthèse	99
4	Application aux objets déformables	100
4.1	Mesure de déplacement relatifs entre objets déformables	101
4.2	Évaluation	105
4.3	Synthèse	110
5	Conclusion	110
Conclusion		113

Publications de l'auteur	117
Liste des figures	119
Liste des algorithmes	121
Liste des tableaux	123
Bibliographie	125

Introduction

La simulation physique a pour but de reproduire, grâce à des simulations, les phénomènes physiques qui nous entourent. Il est possible de simuler des phénomènes que nous pouvons observer à notre échelle tels que la chute d'objets, la déformation d'un tissu, la fracture d'un objet, l'écoulement de l'eau dans une rivière ou encore un drapeau qui flotte avec le vent. Mais il est aussi possible de simuler des phénomènes physiques qui sont en dehors de nos expériences quotidiennes tels que la simulation du mouvement des galaxies ou des simulations de la relativité générale d'Einstein. La simulation physique est utilisée dans différents domaines et en particulier dans la réalité virtuelle. La réalité virtuelle est une science dont le but est de placer un ou plusieurs utilisateurs dans un monde virtuel qui lui fait office de réalité. Philippe Fuchs [Fuchs, 1996] propose de définir la réalité virtuelle par une définition de sa finalité :

La finalité de la réalité virtuelle est de permettre à une personne (ou à plusieurs) une activité sensori-motrice et cognitive dans un monde artificiel, créé numériquement, qui peut être imaginaire, symbolique ou une simulation de certains aspects du monde réel.

Cette définition montre le besoin de simulations qui doivent régir le monde virtuel. Habituellement, les simulations utilisées ont pour but de reproduire le monde réel de la manière la plus fidèle possible. Ceci permet à l'utilisateur de comprendre plus facilement les règles du monde virtuel en le rendant plus proche du monde réel. Une seconde définition plus technique proposée par Bruno Arnaldi et al. [Arnaldi et al., 2003] précise certaines problématiques :

La réalité virtuelle est un domaine scientifique et technique exploitant l'informatique et des interfaces comportementales en vue de simuler dans un monde virtuel le comportement d'entités 3D, qui sont en interaction en temps réel entre elles et avec un ou des utilisateurs en immersion pseudo-naturelle par l'intermédiaire de canaux sensori-moteurs.

Un point important de cette définition est l'interaction en temps réel. C'est cette interaction qui permet à l'utilisateur d'être acteur dans le monde virtuel et celle-ci est au cœur de la définition de la réalité virtuelle. Une problématique présente avec ce besoin d'interaction est la contrainte d'exécution en temps réel. Il est essentiel que les interactions soient réalisées en temps réel pour que l'utilisateur ait un retour direct de ses actions, sans ce retour la boucle sensori-motrice est incomplète et l'utilisateur n'est plus que spectateur du monde virtuel. L'interaction en temps réel est une contrainte technique forte qui requiert que la simulation physique du monde soit elle-même réalisée en temps réel. Les méthodes actuelles utilisées en simulations physiques sont incapables de reproduire l'intégralité des phénomènes physiques avec lesquels nous interagissons dans le monde réel en temps réel. Pour cette raison, de nombreux travaux de recherche

ont été proposés pour réaliser des simulations physiques les plus réalistes possibles en temps réel. L'une des tâches les plus complexes à réaliser lors de simulations physiques est la détection de collision. La détection de collision est responsable de localiser à tout moment quels objets sont en collision et où sont localisées les collisions sur les objets. Cette tâche constitue un goulet d'étranglement calculatoire et limite les performances des simulations physiques en général.

Dans cette thèse nous nous sommes intéressés à la détection de collision en temps-réel pour permettre à un utilisateur d'interagir avec des scènes complexes. Dans cette introduction nous allons, dans un premier temps, présenter les domaines d'application de la détection de collision, nous présenterons ensuite les problématiques auxquelles cette thèse s'est intéressée ainsi que l'organisation de ce manuscrit.

1 Domaines d'application

La détection de collision est une problématique très large et qui ne se limite pas au domaine de la réalité virtuelle. Déetecter si deux formes géométriques sont en collision est une opération qui est utile dans de nombreuses applications. Nous présentons ici certains domaines d'application majeurs de la détection de collision :

Simulations physiques : La détection de collision est utilisée dans la simulation physique pour calculer numériquement des phénomènes physiques trop complexes ou trop coûteux à reproduire tels que la simulation de crash test de voiture ou de simulation de tremblement de terre. Dans ce type de simulations, la quantité de données à traiter est très importante et la réalisation de la détection de collision reste aujourd'hui un défi majeur.

Robotique : Un des problèmes majeurs en robotique est la planification de mouvement. La planification de mouvement est responsable du calcul des mouvements que doit réaliser un robot pour se déplacer entre deux positions. La contrainte est que le robot ne doit pas entrer en collision avec son environnement (ou avec lui-même). La détection de collision est utilisée en robotique pour tester si un mouvement provoque des collisions et pour réaliser la recherche de mouvements libres de collisions.

Interactions en réalité virtuelle : La manipulation d'objets nécessite d'être capable de détecter à quels moments le ou les objets manipulés sont en collision pour permettre une interaction physiquement réaliste et empêcher l'utilisateur de placer les objets en interénétration. De plus, les applications haptiques permettent à un utilisateur de ressentir et manipuler des objets à l'aide du sens du toucher. Ces applications sont réalisées à l'aide de dispositifs haptiques tels que des bras à retour d'effort. Lors de manipulations haptiques, la détection de collision est utilisée pour déterminer quand l'utilisateur touche l'objet virtuel pour lui fournir une réponse adaptée. Une contrainte des manipulations haptiques est la sensibilité : pour obtenir une sensation de continuité lors de la manipulation, le retour d'effort doit être réalisé à des fréquences de l'ordre du kilo-hertz. Cela implique que la détection de collision doit être calculée dans le même ordre de fréquence. Pour atteindre cette fréquence de calcul le nombre d'objets et leur complexité sont généralement très limités.

Animation : La détection de collision est utilisée pour générer des animations libres de collisions. Ceci est, par exemple, le cas lors de la simulation de foules d'êtres humains virtuels où la détection de collision est utilisée pour planifier les déplacements qui ne produisent pas de collisions entre les humains virtuels. Ces méthodes sont aussi utilisées dans la simulation de vêtements, il peut être utile d'animer un vêtement porté par un avatar par simulation physique pour le faire correspondre aux mouvements pré-enregistrés de l'avatar (par exemple pour les films d'animation).

Les contraintes de temps de calculs ne sont pas les mêmes dans tous ces domaines. Dans le domaine de la simulation physique et de l'animation, les simulations sont généralement calculées hors-ligne. De nombreuses applications de simulation physique pré-calcule les simulations numériques avant de les présenter à un utilisateur. Dans le domaine de l'animation, celles-ci sont généralement pré-calculées pour être affichées plus tard (par exemple les effets spéciaux pour le cinéma). Dans d'autres domaines, les calculs sont réalisés en ligne, ceci est le cas lors d'interaction en réalité virtuelle. À cause des interactions en temps-réel avec les utilisateurs, il est impossible de réaliser la détection de collision en calcul hors-ligne.

2 Problématique et objectif de cette thèse

Dans les applications de réalité virtuelle, la contrainte d'exécution en temps-réel pose de fortes contraintes sur le temps de calcul de la détection de collision. La simulation physique doit être réalisée à un rythme régulier et celui-ci dépend de la fréquence propre du système simulé. La détection de collision doit être réalisée à cette fréquence pour permettre l'interactivité de la simulation. De nombreuses applications font correspondre à tort la fréquence d'affichage et la fréquence de la simulation physique et réalisent la détection de collision et la simulation à une fréquence variant entre 30 et 60 Hz.

De plus, pour que les environnements virtuels soient crédibles du point de vue de l'utilisateur, la simulation physique doit se comporter de manière la plus réaliste possible. Ces deux contraintes sont antagonistes et les méthodes actuelles cherchent à réaliser un compromis acceptable. Un compromis possible est de limiter le nombre d'objets simulés. De nombreuses applications de réalité virtuelle ne simulent que les éléments principaux de l'environnement pour pouvoir réaliser une simulation précise, le reste de l'environnement est statique (il n'est pas interactif que se soit de manière directe ou indirecte). Un second compromis possible est de limiter la complexité des objets. En remplaçant la forme visuelle (utilisée pour le rendu) des objets par une forme physique (utilisée pour la simulation) plus simple dans la simulation, il est possible de réduire fortement le coût des simulations et d'être capable de simuler de nombreux objets. Le problème est que le décalage entre la forme visuelle et la forme physique des objets peut provoquer des incohérences perceptibles par l'utilisateur.

Du point de vue calculatoire, la détection de collision est une tâche impliquant un grand nombre d'opérations géométriques en trois dimensions. Les GPU sont des processeurs dédiés aux calculs géométriques en trois dimensions et leur puissance calculatoire brute dépasse actuellement celles des CPUs. Pour ces raisons, de

nombreuses méthodes proposent d'utiliser les GPUs comme périphériques de calcul pour la détection de collision.

Dans ce contexte, notre problématique a été de proposer de nouvelles méthodes permettant de réaliser la détection de collision sur les formes géométriques complètes des objets (sans nécessiter de simplification) tout en permettant de simuler un grand nombre d'objets. Notre but est de proposer une méthode de détection de collision permettant une complexité plus accrue des scènes virtuelles (en termes de nombre d'objets et de complexité des objets) pour que les applications de réalité virtuelle puissent s'approcher de la complexité des scènes réelles attendues. Pour cela, nous proposons une méthode de détection de collision par lancer de rayon qui est performante sur GPU. Pour ce faire, nous revisitons le *pipeline* de détection de collision pour permettre d'exécuter notre méthode sur GPU. Puis, dans le cadre de ce *pipeline*, nous proposons une méthode permettant de fortement accélérer les calculs en exploitant la cohérence temporelle (cohérence du mouvement des objets au fil de temps) tout en permettant de rendre les calculs et le résultat de la détection plus fiables.

3 Organisation du mémoire

Ce manuscrit est organisé en quatre chapitres.

Le chapitre 1 présente un état de l'art des méthodes actuelles de détection de collision. Nous présentons le *pipeline* de détection qui est couramment utilisé avec les différentes familles d'algorithmes qui le composent. De plus, nous discutons de l'évolution des processeurs graphiques (GPU) et présentons les approches qui ont été proposées pour les exploiter dans le cadre de la détection de collision.

Le chapitre 2 présente le cadre général de nos contributions. Nous présentons un bref état de l'art sur les méthodes de lancer de rayon et nous introduisons ensuite l'ensemble de nos contributions et expliquons intuitivement leur motivation.

Les chapitres 3 et 4 présentent en profondeur nos contributions. Le chapitre 3 présente le *pipeline* de détection de collision que nous avons proposé. Celui-ci permet d'accueillir l'ensemble de nos contributions et est compatible avec une exécution sur GPU pour offrir des performances accrues. Nous présentons ensuite dans ce chapitre des éléments fonctionnels qui permettent de généraliser notre méthode de détection de collision et qui peuvent s'insérer dans notre *pipeline*. Le chapitre 4 présente une méthode qui permet de fortement accélérer la détection de collision par lancer de rayon en exploitant la cohérence temporelle.

Nous concluons ensuite nos travaux sur la détection de collision par lancer de rayon et sur l'exploitation de la cohérence temporelle et présentons différentes perspectives possibles issus de nos travaux.

État de l'art

1

La détection de collision a pour but de répondre à la question suivante : étant donné un ensemble d'objets en mouvement, lesquels sont en interpénétration ? Cette question peut paraître simple mais la détection de collision est une tâche extrêmement complexe. La première source de complexité est liée à l'environnement, on souhaite pouvoir simuler des environnements contenant un grand nombre d'objets avec des comportements complexes tels que des déformations ou des fractures. La seconde source de complexité est liée aux informations que doit produire la détection de collision. En effet, en plus de fournir une réponse binaire (collision ou non-collision), la détection de collision doit généralement fournir la localisation des collisions. La troisième source de complexité est la contrainte de calcul temps-réel. Dans le cas de simulations physiques interactives, la détection de collision doit être réalisée avec des contraintes de temps extrêmement courts (de l'ordre de quelques millisecondes).

Dans cet état de l'art, nous commençons par une approche préliminaire présentant le contexte de la détection de collision (cf. Section 1). Nous détaillons ensuite les deux étapes classiques de la détection de collision (cf. Section 2) ainsi que des optimisations proposées dans la littérature (cf. Section 3). Nous présentons ensuite les évolutions récentes liées à l'utilisation de processeurs graphiques (cf. Section 4).

1 Approche préliminaire

Avant de présenter les travaux existants, nous proposons d'introduire certaines notions liées à la détection de collision et plus généralement à la simulation physique. Dans un premier temps, nous présentons les différents modèles de représentation utilisés en 3D (cf. Section 1.1). Nous présentons ensuite les différents types de requêtes auxquelles la détection de collision doit faire face (cf. Section 1.2). Nous présentons enfin des éléments liés à la dynamicité des environnements (cf. Section 1.3).

1.1 Modèles de représentation

Les modèles de représentation permettent de structurer la manière dont les formes en trois dimensions sont représentées dans l'espace. La Figure 1.1 présente une taxonomie inspirée de différents travaux sur la classification des modèles de représentation pour la simulation physique [Lin and Gottschalk, 1998, Gottschalk, 1997] ou pour des usages plus génériques [Badler and Glassner, 1997, Smith, 2012]. Notre taxonomie décompose les modèles de représentation en trois catégories : les modèles ponctuels (cf. Section 1.1.1), les modèles surfaciques (cf. Section 1.1.2) et les modèles volumiques (cf. Section 1.1.3). De plus, une distinction peut être

réalisée entre deux classes d'objets : les objets convexes et les objets non-convexes (cf. Section 1.1.4).

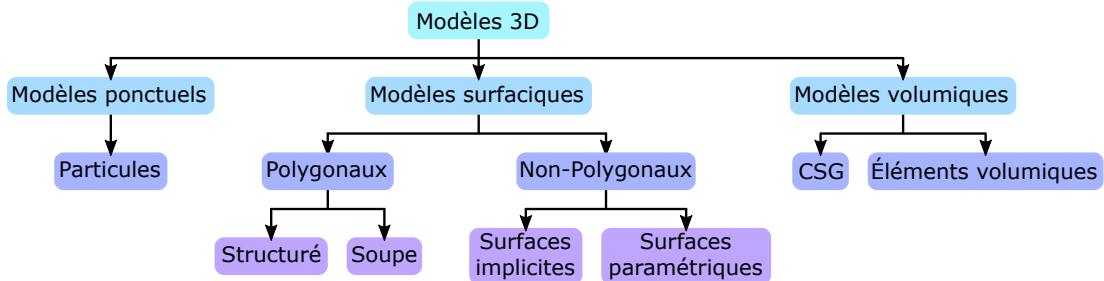


Figure 1.1 – Taxonomie des représentations des modèles 3D.

1.1.1 Modèles ponctuels

Les modèles ponctuels modélisent la matière avec un ensemble de points appelés particules. Cette modélisation est adaptée pour représenter des formes avec une forte dynamique interne telles que des objets déformables mais peut aussi être utilisée pour représenter des formes rigides [Macklin et al., 2014]. Leur mouvement peut être contrôlé à l'aide de modèles probabilistes pour modéliser certains phénomènes tels que le feu, des explosions ou de l'herbe. Les systèmes de particules peuvent aussi être contrôlés par une simulation physique pour modéliser des objets déformables tels que des fluides (cf. Figure 1.2).

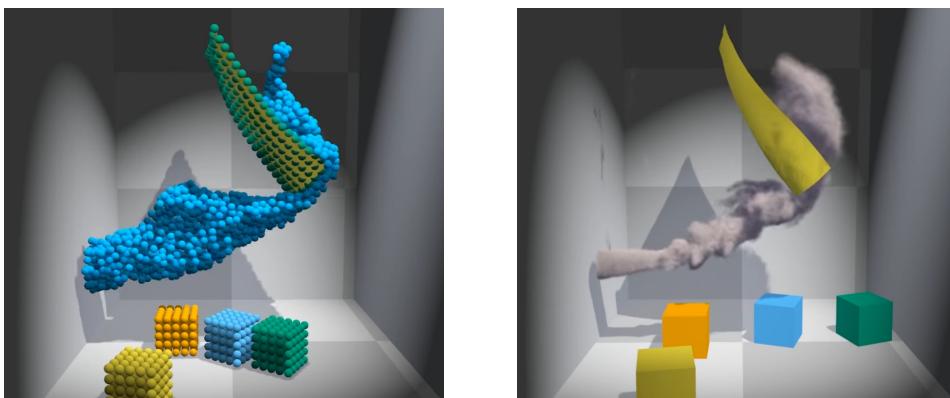


Figure 1.2 – Exemple de modélisation à l'aide de particules [Macklin et al., 2014]. Gauche : modèle physique à particules utilisé dans la simulation. Droite : rendu final de la scène.

1.1.2 Modèles surfaciques

Les modèles surfaciques modélisent un objet en 3D en définissant la frontière entre l'intérieur et l'extérieur de l'objet, cette frontière correspond à une surface en 2D. Ces surfaces peuvent être fermées, dans ce cas l'objet a un intérieur et un extérieur clairement définis (hors cas pathologiques tels que la bouteille de Klein). Les surfaces peuvent aussi être ouvertes, dans ce cas l'objet ne partitionne pas l'espace en deux

parties distinctes. Dans ce cas, nous pouvons définir pour chaque point de la surface une normale (vecteur qui pointe vers l'extérieur de l'objet), ce vecteur va permettre de définir dans le cas des surfaces ouvertes un côté intérieur et un côté extérieur (cependant l'intérieur et l'extérieur ne sont pas définis au niveau des ouvertures).

Les modèles surfaciques sont décomposés en deux catégories : les modèles polygonaux et les modèles non-polygonaux.

A Modèles polygonaux

Les modèles polygonaux sont les modèles les plus utilisés actuellement dans les applications de réalité virtuelle interactives. Les rendus de modèles polygonaux sont très efficaces car les processeurs graphiques (GPU : *Graphics Processing Unit*) ont été conçus spécifiquement pour traiter ce type de modèle. De plus, il est possible de convertir des objets 3D utilisant d'autres modèles de représentation en modèles polygonaux (éventuellement par approximation) ce qui permet de traiter tout type de modèles en utilisant une représentation polygonale.

La classe la plus générale de modèles polygonaux est la soupe de polygones. Dans cette classe, les modèles sont des ensembles de polygones non connectés, ce qui signifie qu'aucune information topologique n'est disponible et qu'il n'est pas possible de discerner l'intérieur et l'extérieur de l'objet. Contrairement aux soupes de polygones, les modèles structurés possèdent une information topologique : les polygones sont connectés entre eux avec une notion de voisin. Cette topologie permet de définir une surface complète, éventuellement fermée pour que l'objet possède un intérieur et un extérieur bien définis.

B Modèles non-polygonaux

Il existe deux types de modèles non-polygonaux (cf. Figure 1.3) : les surfaces implicites et les surfaces paramétriques.

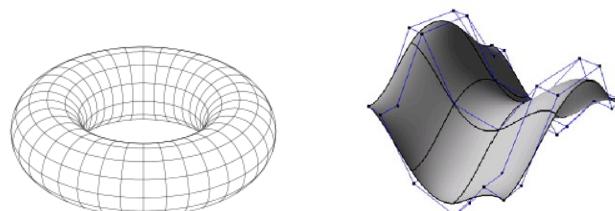


Figure 1.3 – Exemples de modèles non-polygonaux. Gauche : tore modélisé avec une surface implicite. Droite : surface paramétrique de type NURBS.

Une surface implicite est définie à l'aide d'une fonction implicite $f : \mathbb{R}^3 \mapsto \mathbb{R}$. La surface est définie par l'ensemble des points qui satisfont l'équation $f(x, y, z) = 0$. De plus, les inéquations $f(x, y, z) < 0$ et $f(x, y, z) > 0$ permettent de savoir si un point est respectivement à l'intérieur ou à l'extérieur de l'objet. Les travaux de Grisoni donnent une description plus approfondie de ce type de surface [Grisoni, 2005].

Les surfaces paramétriques utilisent des fonctions $f : \mathbb{R}^2 \mapsto \mathbb{R}^3$ pour faire la correspondance entre les coordonnées dans le plan et l'espace. Contrairement aux surfaces implicites, les surfaces paramétriques ne définissent généralement pas une

surface fermée. Elles permettent de faire une description de la frontière de l'objet. Les surfaces NURBS (Non-Uniform Rational B-Spline) et les surfaces de Bézier sont des types de surfaces paramétriques.

1.1.3 Modèles volumiques

Les modèles volumiques permettent de modéliser le volume occupé par l'intérieur des objets avec des informations sur propriétés physiques ou visuelles (telles que la densité ou la couleur). Les informations physiques permettent de simuler plus fidèlement les objets en prenant en compte le comportement interne des objets. Les informations visuelles peuvent être utilisées pour afficher correctement l'intérieur des objets lors de changements topologiques (en cas de fracture par exemple). Il existe deux types de modèles volumiques : les *Constructive Solid Geometry* (CSG) et les éléments volumiques.

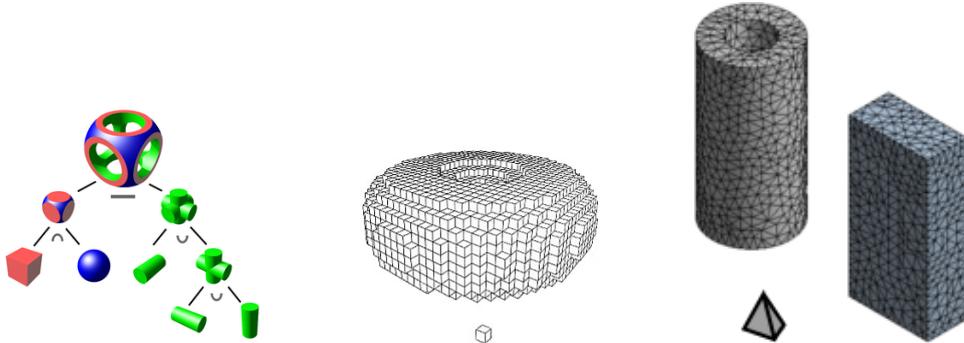


Figure 1.4 – Exemples de modèles volumiques avec leurs éléments unitaires. Gauche : objet construit à l'aide de *constructive solid geometry*. Milieu : modélisation en voxels. Droite : décomposition en éléments finis avec des tétraèdres.

Les CSG sont des objets composés de primitives telles que des pavés, cylindres, sphères ou cônes qui sont combinés à l'aide opérateurs booléens tels que l'union, l'intersection et la différence.

Les modèles volumiques utilisent des éléments unitaires volumiques pour modéliser l'ensemble des objets. Des informations physiques peuvent être stockées sur ces éléments (densité, élasticité, ...). Les éléments volumiques les plus courants sont les voxels et les tétraèdres. La modélisation en voxels est une généralisation en 3D des images 2D modélisées avec des pixels, les voxels se présentent sous la forme de cubes. Les tétraèdres sont une généralisation des triangles en 3D. Cette modélisation tétraédrique est utilisée pour modéliser des objets déformables avec la méthode des éléments finis, notamment pour modéliser des objets hétérogènes. La Figure 1.4 montre des exemples de d'objets modélisés à l'aide de voxels et de tétraèdres.

1.1.4 Objets convexe/non-convexe

Il existe deux types de classes d'objets lorsque un modèle de représentation permet de construire des objets complexes : les objets convexes et les objets non-convexes. Un objet O est convexe s'il respecte la propriété suivante : pour toute paire de points $(A, B) \in O$, le segment $[AB]$ est intégralement contenu dans O . En pratique les objets

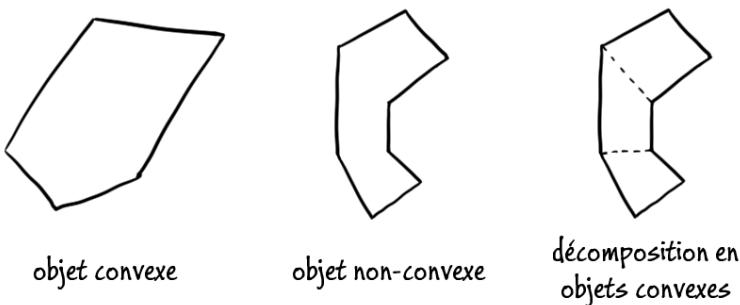


Figure 1.5 – Exemple avec des polygones d'un objet convexe, d'un objet non-convexe et de la décomposition en objets convexes d'un objet non-convexe

convexes ne possèdent pas de creux, ni de trous. La Figure 1.5 donne un exemple d'un objet convexe et d'un objet non-convexe.

Les objets convexes sont généralement plus simples à gérer que les objets non-convexes, certaines méthodes de détection de collision sont uniquement compatibles avec les objets convexes. Pour permettre leur utilisation avec des objets non-convexes, il est possible de chercher à décomposer un objet convexe en un ensemble d'objets non-convexes. Lors de l'utilisation de modèles polygonaux, cette décomposition est toujours possible (cf. Figure 1.5), mais risque de produire un grand nombre d'objets dans certains cas complexes. Dans le cas de modèles non-polygonaux, cette décomposition n'est parfois pas possible.

1.2 Types de requêtes

Différents types de requêtes peuvent être attendues de la détection de collision. Ces différences vont contraindre le choix de la méthode de détection de collision, notamment sur la précision de la localisation spatiale des collisions (cf. Section 1.2.1) et sur le type de mesure de distance (cf. Section 1.2.2).

1.2.1 Méthodes discrètes et méthodes continues

La détection de collision doit, en théorie, permettre de localiser précisément toutes les collisions aussi bien au niveau spatial que temporel. On doit pouvoir, étant données la géométrie et la trajectoire de déplacement des objets, détecter où les objets se touchent et précisément à quel instant afin d'adapter leurs trajectoires en conséquence. Ce type de méthode est appelé détection de collision continue et nécessite la connaissance de fonctions de déplacement régissant la position des objets en fonction du temps. L'avantage de ce type de méthode est la précision : les objets ne sont jamais en interpénétration et la détection de collision se résume à une détection de points ou surfaces de contacts. L'inconvénient de ce type de méthodes est de nécessiter la connaissance des fonctions de déplacement des objets ainsi que le coût élevé de la détection des points de contacts liés à ces fonctions temporelles.

Pour réduire le coût de la détection de collision, de nombreuses méthodes réalisent la détection de collision sur un échantillonnage temporel de la position des objets. Dans ces méthodes dites discrètes, la détection de collision est effectuée à des pas de temps

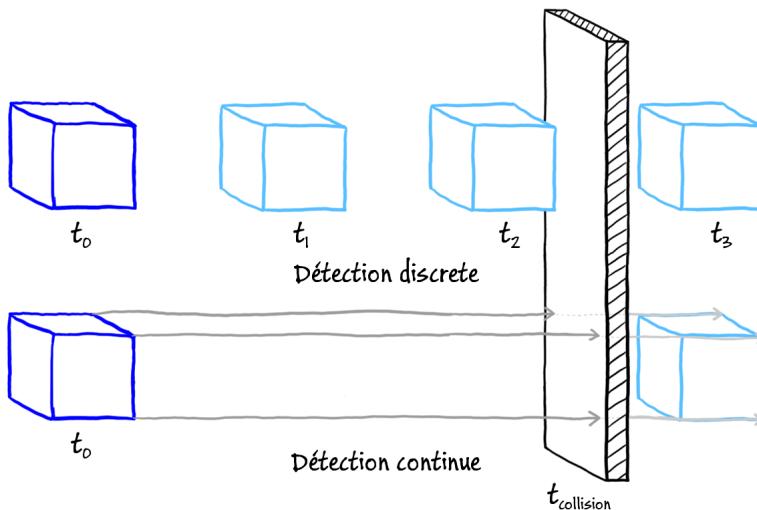


Figure 1.6 – Exemple d’erreur lors de la détection de collision en cas d’échantillonnage trop large. Un cube est déplacé de gauche à droite vers un mur fin. Avec la détection de collision discrète, si l’espace entre les pas de temps (t_0 à t_3) est trop important, le cube peut passer à travers le mur sans aucune détection. Au contraire, avec la détection de collision continue, on va calculer l’instant précis où le cube rentre en collision avec le mur ($t_{\text{collision}}$) sans erreur.

fixes (généralement régulièrement espacés dans le temps). Dans ce cas, la détection de collision doit détecter les objets qui sont en interpénétration. L’avantage de la détection de collision discrète est son plus faible coût de calcul car il n’est pas nécessaire de prendre en compte la trajectoire exacte des objets. Néanmoins, le problème du choix de l’échantillonnage est complexe. Un échantillonnage trop fin sera coûteux en temps de calcul tandis qu’un échantillonnage trop large peut conduire à manquer des collisions. La Figure 1.6 montre un exemple où un échantillonnage trop large conduit à un faux-négatif.

Il existe aussi une méthode intermédiaire consistant à échantillonner la position des objets dans le temps et à réaliser une détection de collision continue entre chaque pas de temps. La trajectoire entre deux pas de temps est approximée par interpolation linéaire à vitesse constante. Cette méthode permet de réaliser une détection de collision continue sans nécessiter de calculer de manière exacte la fonction de déplacement des objets.

1.2.2 Mesure de distance

En plus de savoir si deux objets sont en collision, on souhaite généralement savoir quelle est la profondeur de l’interpénétration. On peut aussi vouloir savoir, étant donnés deux objets qui ne sont pas en collision, quelle est la plus courte distance entre les deux objets. Cameron et al. [Cameron and Culley, 1986] ont proposé de définir une métrique appelée MTD (*Minimum Translational Distance*). $MTD(A, B) \in \mathbb{R}$ est défini par la longueur de la plus courte translation plaçant les deux objets A et B en contact (avec $MTD(A, B) < 0$ si A et B sont en collision). On a donc le choix de définir la détection de collision comme un test binaire (collision ou non-collision) ou une mesure qui fournit la

distance entre deux objets, qu'ils soient en collision (distance négative) ou non (distance positive). Mesurer la distance entre des objets qui ne sont pas en collision peut aussi être nécessaire dans certaines applications, ceci est le cas en robotique pour réaliser les tâches de planification de mouvement.

1.3 Dynamicité de l'environnement

Différentes propriétés liées à la dynamicité de l'environnement et des objets vont contraindre les calculs. La première contrainte est liée au nombre d'objets dynamiques en jeu (cf. Section 1.3.1). La seconde contrainte est liée aux types d'objets présents dans l'environnement. Les objets les plus simples traités dans la détection de collision sont les objets rigides. Nous pouvons ensuite ajouter des objets plus complexes tels que des objets déformables (cf. Section 1.3.2) et les changements topologiques (cf. Section 1.3.3). Malgré cette dynamicité, une certaine cohérence existe au sein de l'environnement. Cette cohérence peut être exploitée pour accélérer les calculs. Deux types de cohérence peuvent être exploitées : la cohérence spatiale (cf.. Section 1.3.4) et la cohérence temporelle (cf. Section 1.3.5).

1.3.1 Environnements 2-body et N-body

Selon le nombre d'objets on peut distinguer deux types d'environnements : *2-body* et *N-body* [Lin and Gottschalk, 1998]. Dans les environnements *2-body* (aussi *pair processing*), uniquement deux corps sont présents dans la scène : un objet mobile et un environnement statique. Ce type d'environnements est utilisé dans des tâches de manipulations ou dans les problèmes de planification de mouvement. Les tests de collision *2-body* se focalisant sur un objet, ils sont généralement plus précis car le nombre de requêtes de collision est plus faible. Dans les environnements *N-body*, un nombre d'objets mobiles non restreint est présent dans l'environnement. Ce type d'environnement est le plus répandu car plus générique. Les tests de collision sont plus coûteux et généralement moins précis que dans le cas *2-body*, ceci est expliqué par la combinatoire qui est bien plus élevée tandis que les contraintes d'exécution en temps-réel restent présentes.

1.3.2 Objets déformables

Les objets déformables subissent des déformations au cours du temps et leur présence engendre des calculs supplémentaires lors de la détection de collision. Prendre en compte les déformations des objets engendrent des calculs supplémentaires comparé aux objets rigides. La détection de collision en temps-réel dans ce type de situation reste aujourd'hui un défi dans le domaine de la détection de collision [Teschner et al., 2005]. De plus, par leurs déformations les objets déformables peuvent entrer en auto-collision. Il faut donc, en plus de tester les collisions avec les autres objets, tester les auto-collisions. Comme exemple d'objets déformables on peut citer les objets plastiques (ex : pâte à modeler) ou les tissus (cf. Figure 1.7).



Figure 1.7 – Exemples d’objets déformables et de changements topologiques. Gauche : simulation de tissu [Pabst et al., 2010]. Droite : fracture d’une tirelire en céramique [Glondu et al., 2012].

1.3.3 Changements topologiques

Les changements topologiques sont des modifications de la structure de l’objet (par exemple : fracture, déchirure ou fusion). Contrairement aux déformations, les changements topologiques sont généralement des phénomènes ponctuels dans le temps. Les changements topologiques peuvent aussi bien concerner les objets déformables (par exemple la déchirure d’un tissu) et les objets rigides (par exemple un verre qui se brise, cf. Figure 1.7). À l’instar des déformations, les changements topologiques vont aussi engendrer des calculs supplémentaires et poser des contraintes additionnelles dans les optimisations.

1.3.4 Cohérence spatiale

La cohérence spatiale est liée aux contraintes géométriques des objets. La position de chaque point constituant un objet est dépendant des autres points constituant ce même objet. Pour les objets rigides, la position de l’ensemble des points peut être calculée à partir de transformations locales (translation et rotation). Pour les objets déformables, la position de chaque point constituant un objet est en partie indépendante, mais deux points voisins seront toujours voisins. En cas de changements topologiques, la cohérence spatiale est généralement conservée au niveau local. En cas de fracture, la cohérence spatiale est conservée dans les fragments. Cette cohérence est largement exploitée dans la détection de collision sous la forme de structures accélératrices. De nombreuses méthodes construisent des structures de données (liste, arbre, graphe, ...) sur les objets. L’existence et l’utilité de telles structures proviennent de la cohérence spatiale.

Une seconde forme de cohérence spatiale est liée à l’étude des lieux. Dans une simulation où les contraintes physiques sont respectées, un point de l’espace ne peut être occupé que par un seul objet. On peut étendre cette propriété à de petits espaces. Une zone restreinte de l’espace ne peut être occupée que par un nombre limité d’objets (sans qu’ils soient en superposition). Cette observation justifie l’utilisation de structures pour partitionner l’espace complet (avec l’ensemble des objets) car cette structuration est discriminante entre les objets.

La cohérence spatiale (qu’elle soit sur les objets individuels ou sur l’ensemble des objets) est largement exploitée à l’aide de structures accélératrices. Dans la seconde partie de cet état de l’art, nous verrons les détails des structures accélératrices qui

permettent d'exploiter la cohérence spatiale.

1.3.5 Cohérence temporelle

La cohérence temporelle est liée à la continuité du déplacement des objets dans le temps. La détection de collision étant exécutée à instant réguliers sur une simulation, il existe une certaine cohérence dans la position des objets entre chaque pas de simulation successif [Lin and Gottschalk, 1998]. Cette cohérence peut être exploitée pour éliminer des tests unitaires entre des objets (deux objets très éloignés à un instant t ne seront probablement pas en collision à l'instant $t + 1$). Elle peut aussi être exploitée dans le cas d'algorithmes gloutons (deux points formant le minimum local de distance entre deux objets à un instant t sont de bons candidats pour un algorithme glouton chargé de localiser les deux points formant le minimum local de distance à un instant $t + 1$). Afin d'exploiter cette cohérence temporelle, certains algorithmes nécessitent de borner la vitesse ou l'accélération des objets ou de pouvoir exprimer le mouvement à l'aide d'équations temporelles.

1.4 Synthèse

Cette approche préliminaire montre l'étendue du problème de la détection de collision. Ce domaine est contraint par la représentation des objets de l'environnement et du type de requêtes. Les objets peuvent être représentés avec des modèles variés (polygones, CSG, surfaces implicites, particules, ...) et des propriétés diverses (convexe, non-convexe, rigide, déformable, ...). Le type de requêtes et leur précision ajoutent des contraintes supplémentaires. Les méthodes de détection de collision continues sont plus précises mais aussi plus coûteuses que les méthodes discrètes. En plus de détecter les collisions, il peut être utile de détecter les objets qui sont proches entre eux et de mesurer leur distance. Le nombre d'objets présents dans la scène pose des problèmes de complexité calculatoire supplémentaire, notamment en cas de simulations temps-réel. Toutes ces contraintes ont conduit à l'émergence d'un grand nombre de méthodes pour réaliser la détection de collision. Dans la suite de ce document, nous présentons les méthodes de détection de collision, ainsi que des méthodes accélératrices destinées à maximiser les performances.

2 Pipeline de détection de collision

Les environnements *N-body* sont les plus étudiés pour la détection de collision car ceux-ci sont les plus génériques et correspondent mieux aux cas réels d'utilisation lors de simulations physiques où un ensemble de n objets sont simulés. Pour effectuer les tests de collisions sur n objets, la méthode naïve est de tester toutes les paires d'objets. Cette méthode naïve possède une complexité quadratique en $O(n^2)$ en terme de nombre de paires testées. De nombreuses méthodes existent pour réduire cette complexité, Hubbard [Hubbard, 1993] a proposé de modéliser la détection de collision dans un pipeline pour formaliser ces "filtres" qui permettent de réduire la complexité.

Ce pipeline (cf. Figure 1.8) prend en entrée l'ensemble des données géométriques de la scène et fournit en sortie une liste de points de contact indiquant quels objets

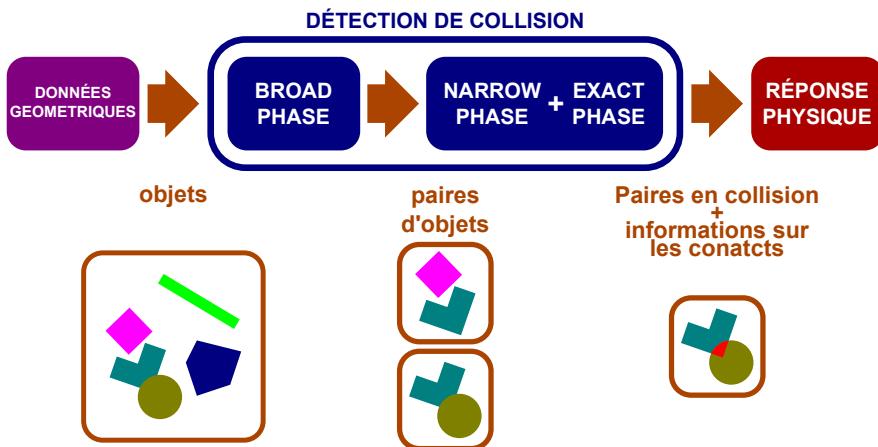


Figure 1.8 – Pipeline de détection de collision avec un exemple. Une liste d’objets est fournie en entrée du pipeline. La *broad-phase* liste les paires d’objets qui sont potentiellement en collision. La *narrow-phase* et l’*exact-phase* exécutent des tests plus précis sur les paires et liste les paires qui sont en collision avec des informations sur les zones en contact.

sont en collision et où se situent les collisions sur les objets. Le pipeline est divisé en trois phases : *broad-phase*, *narrow-phase* et *exact-phase*. La *broad-phase* a pour rôle de lister les paires d’objets qui sont potentiellement en collision. Les tests sont effectués à une large granularité, le but est d’éliminer rapidement et à faible coût le nombre maximum de paires d’objets qui ne sont pas en collision avant d’exécuter des tests plus précis. La *narrow-phase* exécute des tests avec une granularité plus fine sur les paires d’objets. Le but est de localiser sur les objets les parties qui sont potentiellement en collision. L’*exact-phase* calcule si les deux objets sont en collision et, le cas échéant, où sont localisées les collisions sur les objets. La majorité des méthodes actuelles réalisent la *narrow-phase* et l’*exact-phase* en une seule étape (que l’on nomme généralement *narrow-phase*).

Il est possible d’ajouter des phases intermédiaires à la *broad-phase* et la *narrow-phase*, notamment pour ajouter un élagage supplémentaire. Nous présentons par la suite les méthodes existantes pour la *broad-phase* (cf. Section 2.1) et pour la *narrow-phase* (cf. Section 2.2).

2.1 Broad-phase

Les calculs sont effectués sur une version simplifiée du modèle 3D (généralement avec des volumes englobants). Le but n’est pas de détecter avec certitude si des objets sont en collision mais, étant donnée une liste d’objets, quelles paires d’objets sont potentiellement en collision. Kockara et al. [Kockara et al., 2007] décomposent les algorithmes de *broad-phase* en trois catégories : force brute (cf. Section 2.1.1), découpage spatial (cf. Section 2.1.3) et *sweep and prune* (cf. Section 2.2).

2.1.1 Force brute

La méthode de force brute consiste à regarder toutes les paires d'objets et de tester si leurs volumes englobant sont en collision. Si ils sont en collision, alors on transmet la paire d'objets à la *narrow-phase*. Si ils ne sont pas en collision, on peut ignorer cette paire car les deux objets ne peuvent pas être en collision. Avec n objets, cette méthode réalise $\frac{n(n-1)}{2}$ tests de collision résultant en une complexité quadratique en $O(n^2)$. Il est en effet nécessaire de tester toutes les paires d'objets (a, b) possibles sans tester les auto-collisions (paires de type (a, a)) et les paires symétriques (il faut tester la paire (a, b) mais pas la paire (b, a)).

2.1.2 Découpage spatial

Les méthodes de découpage spatial partitionnent l'espace dans une structure de données. Cette structure peut ensuite être interrogée pour énumérer les objets qui partagent un lieu en commun. Le découpage spatial peut être absolu, il ne prend alors pas en compte la configuration de l'environnement [Bentley and Friedman, 1979]. Le découpage peut aussi être adaptatif pour prendre en compte les éléments fixes de l'environnement.

Il existe différentes structures de données pour réaliser le découpage spatial (cf. Figure 1.9). Le découpage peut être réalisé à l'aide d'une grille uniforme sur l'ensemble de l'espace [Turk, 1989, Overmars, 1992] ou hétérogène [Eitz and Lixu, 2007] pour mieux prendre en compte la configuration des objets dans l'environnement. Le découpage peut être fait dans une structure hiérarchique telles que des *quadtree* (arbre où chaque noeud a quatre fils) [Finkel and Bentley, 1974] et des *octree* (arbre où chaque noeud a huit fils) [Bandi and Thalmann, 1995, Vemuri et al., 1998]. Les *quadtree* et *octree* utilisent des plans de séparations alignés sur les axes et centrés qui ne sont pas toujours adaptés pour séparer les objets. Les *k-d tree* [Klosowski et al., 1998, Fünfzig and Fellner, 2003] permettent de placer plus librement les plans séparateurs pour mieux correspondre aux objets. Les *r-tree* [Guttman, 1984, Sellis et al., 1987, Beckmann et al., 1990] s'adaptent aux objets en autorisant des recouvrements entre les noeuds. Les BSP (*Binary Space Partitioning*) [Naylor, 1992, Naylor et al., 1990, Ar et al., 2000, Luque et al., 2005] permettent une plus grande liberté sur l'orientation des plans séparateurs.

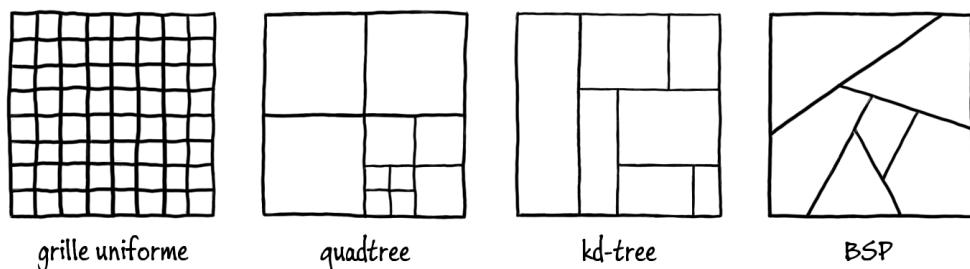


Figure 1.9 – Différents types de découpage spatial.

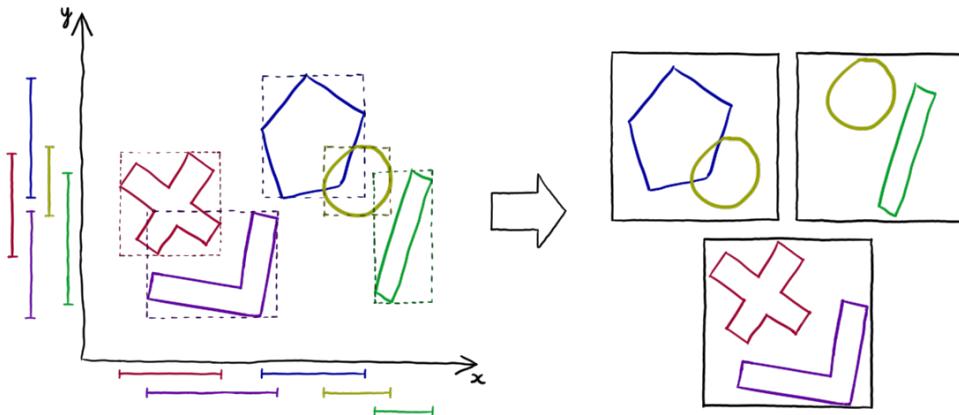


Figure 1.10 – Exemple en 2D de l'algorithme de *sweep and prune*. Les tests de collision sont réalisés sur deux axes (x et y), les résultats sont combinés pour produire une liste de trois paires d'objets qui seraient transmises à la *narrow-phase*.

2.1.3 Sweep and prune

La méthode de *sweep and prune* [Baraff, 1992] permet de réaliser la *broad-phase* sans faire un découpage explicite de l'espace. Cette méthode est l'une des plus utilisée car elle permet une élimination efficace des paires d'objets sans dépendre de la complexité des objets. Les tests sont réalisés sur les volumes englobants (généralement des boîtes alignées sur des axes). La méthode se décompose en deux parties. Premièrement, les limites supérieures et inférieures des volumes englobants sont projetées sur des axes (généralement sur les axes x , y et z) et ces listes sont triées. La seconde étape consiste à parcourir les listes pour repérer les collisions sur chaque axe. Étant donné n le nombre d'objets, le coût pour construire une liste triée sur chaque axe est en $O(n \log n)$ et le parcours peut être fait en $O(n)$. Ce qui donne une complexité totale de $O(n \log n)$, mais cette complexité peut être réduite en exploitant la cohérence temporelle. Au lieu de retrier les listes d'objets à chaque pas de temps, on peut mettre à jour le résultat du pas précédent avec un tri pas insertion. Cette utilisation de la cohérence temporelle permet de réduire la complexité total en $O(n + s)$ avec s le nombre d'échanges nécessaires pour trier la liste.

Dans le cas d'objets déformables, Larsson et al. [Larsson and Akenine-Möller, 2001] ont proposé d'élargir les volumes englobants pour contenir toutes les déformations et changement d'orientation possible des objets pour réduire le coût de mise à jour (au prix d'une baisse du taux de filtrage de la *broad-phase*). La méthode du *sweep and prune* a été adaptée pour les simulations continues [Coming and Staadt, 2006]. Dans cette adaptation, les auteurs proposent d'exécuter la mise à jour des listes triées lors d'événements liés aux objets contrairement à la méthode classique qui est exécutée à des pas de temps fixes. Dans le cas d'environnement large échelle, Tracy et al. [Tracy et al., 2009] ont amélioré la mise à l'échelle en utilisant des listes segmentées.

2.2 Narrow-Phase

La *narrow-phase* réalise des tests de collision plus précis sur les paires d'objets filtrés par la *broad-phase*. Son rôle est de déterminer précisément si deux objets sont en collision et le cas échéant où se situent les collisions. La position des collisions est une information primordiale pour pouvoir calculer une réponse physique correcte. Le coût de la *narrow-phase* est plus élevé que la *broad-phase* lorsque des objets complexes sont utilisés car les tests ne sont pas réalisés sur un modèle simplifié des objets mais sur leur modélisation complète.

Différentes classifications ont été proposées pour la *narrow-phase*, nous utilisons celle proposée par Kockara et al. [Kockara et al., 2007] ainsi que la classification proposée dans le cas des objets déformables par Teschner et al. [Teschner et al., 2005]. La classification complète liste cinq catégories : les algorithmes basés caractéristiques (cf. 2.2.1), les algorithmes basés simplex (cf. 2.2.2), les algorithmes utilisant des hiérarchies de volumes englobants (cf. 2.2.3), les algorithmes basés images (cf. 2.2.4) et les algorithmes utilisant des champs de distance (cf. 2.2.5).

2.2.1 Algorithmes basés caractéristiques

Les algorithmes basés caractéristiques travaillent directement sur les primitives géométriques des objets.

Dans le cas des modèles polygonaux, il existe trois types de caractéristiques : les sommets, les arêtes et les faces (généralement des triangles).

Moore et Wilhelms [Moore and Wilhelms, 1988] testent si des points représentatifs sont à l'intérieur d'un autre objet. Pour tester deux objets convexes A et B, chaque sommet de A est testé face à l'objet B et vice versa. Les objets A et B sont en collision si au moins un sommet d'un objet est à l'intérieur de l'autre objet.

L'algorithme de Lin-Canny [Lin and Canny, 1991] utilise une méthode de descente de gradient pour localiser la paire de caractéristiques la plus proche entre les deux objets. L'avantage de cette méthode est qu'elle permet d'exploiter la cohérence temporelle en utilisant la paire de caractéristiques trouvée au pas de temps précédent comme état initial pour la descente de gradient. Cette méthode ne peut être utilisée que sur des objets convexes en état de non-pénétration.

V-Clip (Voronoi-Clip) [Mirtich, 1998] est un algorithme basé sur l'algorithme de Lin-Canny. A l'instar de l'algorithme de Lin-Canny, V-Clip nécessite des objets convexes mais ne requiert pas d'avoir des objets en non-pénétration. V-Clip décompose l'espace à l'extérieur des objets avec des régions de voronoï, ces régions indiquent pour tout point de l'espace quelle est la caractéristique la plus proche de ce point. V-Clip repose sur le théorème suivant :

Théorème de V-Clip. *Si X et Y sont une paire de caractéristiques appartenant à deux polyèdres convexes, et $x \in X$ et $y \in Y$ les points les plus proches entre X et Y . Si x appartient à la région de voronoï de Y et y appartient à la région de voronoï de X , alors x et y sont les deux points les plus proches entre les deux objets.*

SWIFT [Ehmann and Lin, 2000] est une amélioration de l'algorithme de Lin-Canny dans lequel les objets utilisent des représentations avec différents niveaux de détail

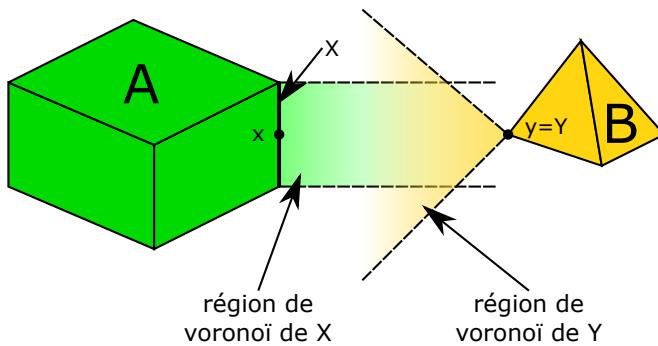


Figure 1.11 – Exemple de V-Clip. Les points x et y sont les deux points les plus proches entre A et B.

avec une structure hiérarchique. Cette méthode permet d'améliorer les performances particulièrement dans le cas où la cohérence temporelle est très faible.

Gottschalk [Gottschalk, 1996] a introduit le SAT (*Separating Axis Theorem*) basé sur le théorème de séparation des convexes. Le théorème de séparation des convexes énonce la propriété suivante :

Théorème de séparation des convexes. *Si A et B sont deux ensembles disjoints, non vides et convexes dont l'un des deux est un compact, alors il existe un hyperplan séparant A et B.*

Le théorème de séparation des convexes montre que si deux objets convexes ne sont pas en collision, alors il existe un hyperplan (ou par équivalence un axe) séparateur, mais ce théorème ne donne aucune indication pour trouver ce plan (ou axe). SAT énonce que si deux polygones ou polyèdres convexes ne sont pas en collision, alors il existe un axe séparateur qui est soit perpendiculaire à l'une des faces des deux objets ou perpendiculaire à deux arêtes prises sur chacun des objets. Le *Separating Axis Theorem* permet donc de restreindre la recherche d'un plan séparateur à un nombre fini de cas possibles. SAT a originellement été proposé pour résoudre les collisions entre des boîtes orientées lors du parcours de hiérarchies de volumes englobants (cf. Section 2.2.3), il peut en réalité être utilisé entre des polyèdres convexes arbitraires.

La cohérence temporelle peut être exploitée avec le SAT en commençant par tester l'axe séparateur trouvé au pas précédent. En effet, si un axe a est séparateur à un instant t , il est fortement probable que ce même axe a soit aussi séparateur à l'instant $t + 1$.

2.2.2 Algorithmes basés simplexes

Les algorithmes basés simplexes travaillent sur les enveloppes convexes d'ensembles de points. Par conséquence, ils ne fournissent un résultat correct qu'avec des objets convexes. Ils peuvent néanmoins être utilisés en tant que filtre dans le cas d'objets non-convexes.

GJK [Gilbert et al., 1988] est à l'origine de ce type d'algorithmes, il permet de calculer la distance euclidienne entre deux ensemble convexes. Cette méthode a été généralisée à tout ensemble arbitraire de points [Gilbert and Foo, 1990]. Le point important de l'algorithme GJK est qu'il ne travaille pas sur les deux objets fournis

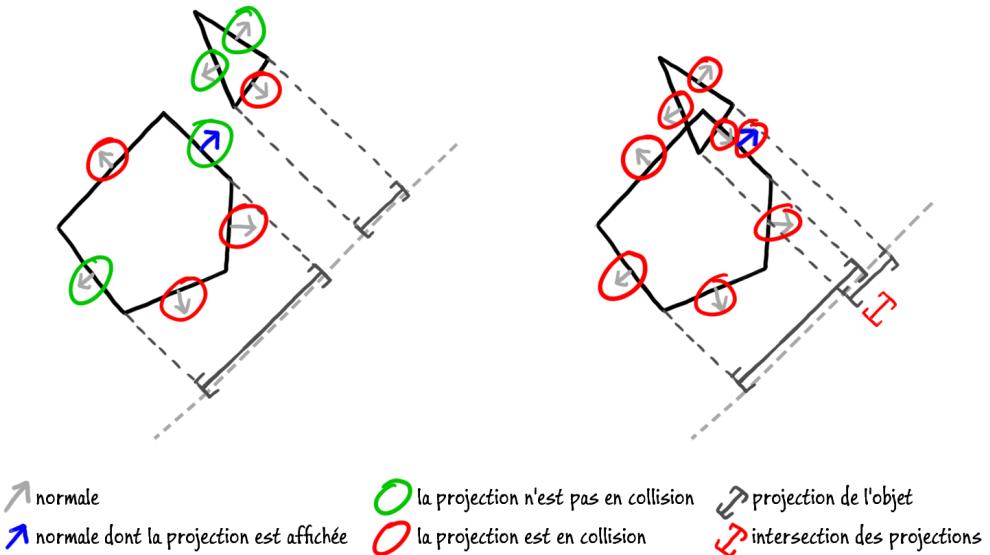


Figure 1.12 – Exemple de SAT. À gauche, deux objets ne sont pas en collision. Parmi les huit normales (flèches), il en a quatre (entouré en vert) qui donne des axes séparateurs. La flèche bleue montre un exemple d'axe où les deux projections ne sont pas en collision. À droite les deux objets sont en collision, il n'est pas possible de trouver parmi les normales un axe séparateur. La flèche bleue montre un exemple d'axe où les projections sont en collisions.

en entrée, mais sur leur différence de Minkowski. Pour deux objets A et B , la différence de Minkowski $A \Theta B$ est définie par la formule suivante : $A \Theta B = \{x - y : x \in A, y \in B\}$. Avec cette formulation, on peut remplacer la recherche d'un point de collision (i.e. $\exists x \in X, y \in Y$ tel que $x = y$) par tester si $\overline{0} \in A \Theta B$ est vraie.

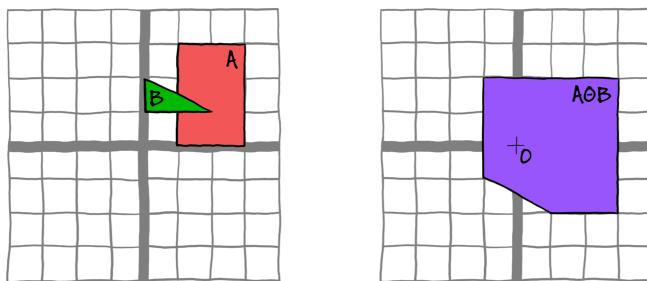


Figure 1.13 – Exemple d'une différence de Minkowski sur deux polygones. Les deux polygones A et B (à gauche) sont en collision, leur différence de Minkowski (à droite) contient donc $\overline{0}$.

GJK permet de mesurer la distance entre deux polyèdres qui ne sont pas en collision, cet algorithme a été complété par le *Expanding Polytope Algorithm* (EPA) qui permet de calculer la distance de pénétration lorsque les deux polyèdres sont en collision. Enhancing GJK [Cameron, 1997] utilise une technique de *Hill Climbing* en partant du résultat de pas de temps précédent pour exploiter la cohérence temporelle. ISA-GJK [Bergen, 1999] garde en mémoire certains résultats intermédiaires des calculs pour les ré-utiliser dans le pas de temps suivant, ce qui est une forme de cohérence temporelle.

De plus les calculs sont interrompus dès qu'un axe séparateur est trouvé pour arrêter au plus tôt les calculs. Sundaraj et al. [Sundaraj et al., 2000] ont montré que l'algorithme GJK peut être combiné à l'algorithme de Lin-Canny.

2.2.3 Hiérarchie de volumes englobants

Les hiérarchies de volumes englobants (BVH : *Bounding Volume Hierarchy*) partitionnent chaque objet de manière récursive. Chaque noeud est un volume englobant de chacun de ses fils, tandis que les feuilles contiennent une ou plusieurs primitives de l'objet. Suivant la structuration de l'arbre, les noeuds peuvent tous avoir exactement le même nombre de fils ou un nombre variable. Les arbres à nombre fixe de noeuds sont couramment utilisés en détection de collision, notamment les arbres binaires (deux fils), les quad-trees (quatre fils) et les octrees (huit fils). Mezger et al. [Mezger et al., 2003] ont montré que les quad-trees et les octrees sont plus adaptés que les arbres binaires pour la détection de collision.

Ehmann et al. [Ehmann and Lin, 2001] proposent de gérer les objets non-convexes à l'aide d'un BVH dont les primitives sont des objets convexes. De plus, cette méthode montre que les BVHs peuvent être utilisés pour réaliser des tests de proximité en plus de tests de collisions. Les BVHs peuvent être utilisés avec de nombreux types de représentations, y compris les modèles volumiques [Tang et al., 2011b] pour la simulation en éléments finis.

Quatre problématiques doivent être prises en compte lors de l'utilisation de BVHs : le type de volume englobant, la méthode de construction de l'arbre, la méthode de mise à jour de l'arbre et l'algorithme de test de collision entre deux BVHs.

A Types de volumes englobants

Chaque noeud dans la hiérarchie de volumes englobants est le volume englobant de l'ensemble de ses fils, il faut donc choisir le type de volume englobant à utiliser. De nombreux types de volumes englobants ont été proposés dans la littérature (cf. Figure 1.14). Le choix du type de volume englobant est un compromis à faire entre la précision du volume englobant et son coût calculatoire. La précision d'un volume englobant est défini par le pourcentage du volume qui est occupé par l'objet englobé, ce pourcentage devrait être le plus élevé possible. Le coût calculatoire est le coût des requêtes de collision entre les volumes englobants, il devrait être le plus faible possible. Le problème étant que les volumes englobants ayant les meilleures précisions sont les plus complexes (c'est-à-dire avec les coûts calculatoires les plus élevés). À cause de ce compromis à trouver, il existe un grand nombre de volumes englobants, le choix de ceux-ci va dépendre de l'environnement (la précision des volumes englobants dépendant de la complexité des objets), du type d'opérations et de leur nombre (qui aura un impact sur le coût de calcul). Parmi les volumes englobants existants, les plus utilisées sont :

Sphères : Il s'agit de la plus simple forme de volume englobant. Les tests d'intersection entre les sphères sont très rapides à réaliser. De plus, les sphères étant invariantes aux rotations, il n'est nécessaire de mettre à jour que leur position lors du déplacement des objets. Il peut être néanmoins difficile de faire correspondre une sphère à une forme arbitraire sans

gaspiller de l'espace vide entre la sphère et la géométrie. Les avantages des sphères les rendent très utilisées [Palmer and Grimsdale, 1995, Hubbard, 1996, O'SULLIVAN and Dingliana, 1999, Benitez et al., 2005].

AABBS (Axis Aligned Bounding Box) [Zachmann and Felger, 1995, Bergen, 1997, Larsson and Akenine-Möller, 2001] : Les AABBs sont des boites alignées sur les axes. L'avantage est qu'il est possible de stocker la géométrie d'un AABB avec 6 valeurs donnant le minimum et le maximum sur les axes x , y et z . Les tests d'intersection entre des AABBs sont rapides grâce à leur alignement. Il peut néanmoins être difficile de faire correspondre un AABB avec une forme arbitraire lorsque celle-ci n'est pas alignée sur les axes (cf. Figure 1.14).

OBBs (Oriented Bounding Box) [Gottschalk et al., 1996, Redon et al., 2002, Redon et al., 2005] : Contrairement aux AABBs, les OBBs sont des boites orientées qui permettent d'approcher plus finement les objets. Cette approximation plus fine permet aux OBBs d'être plus efficaces que les sphères et les AABBs. Malheureusement, les tests d'intersections sont plus coûteux. Il est nécessaire d'utiliser l'algorithme SAT (cf. Section 2.2.1) ce qui conduit à tester 15 axes pour effectuer un test d'intersection entre deux OBBs (3 faces de chaque OBB et 9 combinaisons deux à deux d'arêtes).

Coques sphériques [Krishnan et al., 1998] : Ce type de volume correspond à une portion du volume entre deux sphères concentriques. Celui-ci offre une forte convergence, en termes de minimisation de volume, entre chaque niveau de la hiérarchie [Krishnan et al., 1997].

k-DOPs (Discrete Oriented Polytope) [Klosowski et al., 1998, Zachmann, 1998, Fünfzig and Fellner, 2003] : Un k-DOP est un polyèdre convexe qui est construit par l'intersection de k demi-espaces. La construction est réalisée à partir d'une liste finie des plans qui sont placés pour être en contact avec les objets. Les k-DOPs sont une généralisation des boites englobantes, un AABB est un 6-DOP dont les 6 plans sont orthogonaux aux axes. Les k-DOPs les plus utilisés sont les 14-DOPs, 18-DOPs et 26-DOPs.

Enveloppes convexes [Bajaj and Dey, 1992, Ehmann and Lin, 2001] : L'enveloppe convexe d'un objet est le plus petit volume convexe qui englobe l'objet. Dans le cas de polyèdres ou de nuage de points, l'enveloppe convexe est un polyèdre.

Il existe d'autres types de volume englobants tels que des prismes [Ponce and Faugeras, 1987], des QuOSPO (*Quantized Orientation Slabs with Primary Orientations*) [He, 1999] qui sont une amélioration des k-DOPs permettant de maintenir une précision élevée en cas de rotation des objets, les VADOPs (*Velocity-Aligned Discrete Oriented Polytopes*) [Coming and Staadt, 2008] qui proposent d'utiliser les axes orthogonaux aux vecteurs vitesses des objets comme plans séparateurs ou des rectangles balayés par une sphère [Larsen et al., 2000]. Il existe donc un grand nombre de volumes englobants possibles lors de la construction de hiérarchies de volumes englobants.

Le choix de type de volume englobant à utiliser dépend de différents critères. Il faut prendre en compte le coût des tests d'intersection face à l'efficacité d'élagage. Il faut aussi prendre en compte le coût de mise à jour des volumes englobants, notamment dans le cas d'objets déformables. Selon Gottschalk [Gottschalk, 2000], les OBBs surpassent

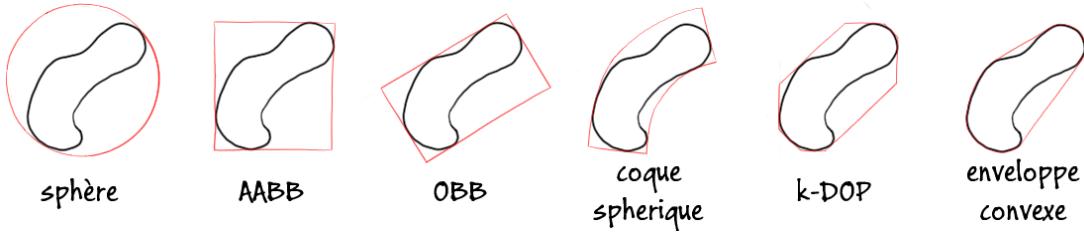


Figure 1.14 – Exemples de volumes englobants (en rouge) sur un objet en 2D.

les sphères et les AABBs pour des objets surfaciques en terme d'élagage. Néanmoins, dans le cas de surfaces déformables la mise à jour d'OBBs peut être coûteuse. Il peut donc être intéressant d'utiliser des AABBs dont la mise à jour est plus simple malgré leur moins bonne performance en élagage [Larsson and Akenine-Möller, 2001].

B Construction de l'arbre

Il existe trois stratégies principales pour construire un arbre de volumes englobants : bottom-up, top-down et insertion. Les méthodes bottom-up [Roussopoulos and Leifker, 1985, Volino and Thalmann, 1995] partent de chaque primitive (le dernier niveau de la hiérarchie) et cherchent à chaque niveau à regrouper les primitives ou les volumes englobants avec leurs voisins jusqu'à convergence vers un seul volume englobant (racine de l'arbre). Les méthodes top-down [Mezger et al., 2003] partent du volume englobant l'ensemble de l'objet et le divisent récursivement jusqu'à ce qu'un certain critère soit atteint. Les méthodes par insertion [Goldsmith and Salmon, 1987] partent d'un arbre vide et insèrent itérativement les primitives de l'objet en cherchant à minimiser l'inflation des volumes englobants.

C Mise à jour de l'arbre

Contrairement aux objets rigides, il est nécessaire de mettre à jour la hiérarchie de volumes englobants des objets déformables à chaque pas de temps. La mise à jour de l'arbre peut être réalisée en reconstruisant complètement l'arbre à chaque pas de temps ou en le mettant à jour de manière incrémentale. Bergen [Bergen, 1997] a constaté qu'une mise à jour incrémentale de l'arbre peut être dix fois plus rapide qu'une reconstruction complète et n'impacte pas de manière significative les performances des requêtes de collisions.

Diverses méthodes ont été proposées pour réduire le coût de stockage et de mise à jour de l'arbre. les BD-Tree (*Bounded Deformation Tree*) [James and Pai, 2004] utilisent des sphères comme volumes englobants et appliquent des champs de déplacements arbitraires sur l'ensemble de la hiérarchie de manière indépendante. Les RACBVHs (*Random-Accessible Compressed Bounding Volume Hierarchies*) [Kim et al., 2010] permettent de compresser les hiérarchies de volumes englobants tout en permettant des accès aléatoires sans nécessiter de décompresser entièrement la hiérarchie.

D Interrogation de l'arbre

Pour réaliser un test de collision entre deux objets, la hiérarchie de volumes englobants est parcourue récursivement à partir de la racine jusqu'aux feuilles des arbres. À chaque étape, si deux noeuds sont en collision alors toutes les paires de fils sont testées, dans le cas contraire le parcours s'arrête. Lorsque le parcours arrive aux feuilles, toutes les paires de primitives des feuilles sont testées. Pour réaliser un test d'auto-collision, il suffit de tester le BVH avec lui-même.

La cohérence temporelle peut être exploitée de diverses manières pour accélérer les calculs. Larsoson et al. [Larsson and Akenine-Möller, 2006] exploitent la cohérence temporelle lors de la mise à jour de la hiérarchie de volumes englobants d'objets déformables pour mettre à jour les parties les plus dynamiques des objets. Li et al. [Li and Chen, 1998] ont proposé de maintenir une "liste de séparation" entre chaque paire de BVH. Cette liste contient les paires de noeuds sur lesquelles la récursion s'est arrêtée au dernier pas de temps. La détection de collision reprend cette liste au pas suivant pour la mettre à jour. Mezger et al. [Mezger et al., 2003] ont proposé de représenter les mouvements dans une hiérarchie de volumes englobants à l'aide de "cônes de vitesse". Ces cônes représentent une approximation de la distribution de la vitesse dans la hiérarchie. L'angle d'ouverture du cône mesure la corrélation des vecteurs de vitesse de chaque primitive et la longueur du cône donne la vitesse maximale atteinte dans la sous-hiéarchie. Les cônes de vitesse sont utilisés pour détecter des noeuds qui ont une faible vitesse relative et pour lesquels les algorithmes de détection de collision incrémentaux sont les plus intéressants.

2.2.4 Algorithmes basés image

Les méthodes basées images utilisent des techniques de rendu pour réaliser la détection de collision. Ces méthodes ont l'avantage de pouvoir être exécutées sur un processeur graphique (GPU), elles peuvent aussi être utilisées sur CPU. La majorité de ces méthodes réalisent les requêtes de collision via des tests d'occlusion. Les méthodes basées images sont particulièrement adaptées aux objets déformables car ils ne nécessitent généralement pas de structures accélératrices particulières pour être utilisées. Nous avons classé ces méthodes en deux catégories principales : les méthodes de *depth peeling* et les méthodes utilisant le lancer de rayon. De plus, nous présentons quelques méthodes transversales à ces deux catégories.

A Méthodes de depth peeling

La première méthode dans la famille des algorithmes de détection de collision basée image consiste à calculer des LDI. Un LDI (*Layered Depth Image*) [Shade et al., 1998] est une représentation multi-couches d'une image. Au lieu de n'avoir qu'une seule valeur par pixel, un LDI possède pour chaque pixel une liste ordonnée correspondant aux différents objets (ou parties d'un objet) visibles dans une direction donnée (cf. Figure 1.15). Cette image multi-couches peut ensuite être post-traitée pour détecter des collisions. Les avantages de cette méthode sont multiples : elle permet de calculer les collisions entre différents objets quelque soit leur complexité. Les objets peuvent être convexes ou non-convexes, ils peuvent être rigides ou déformables (car aucune structure

accélératrice n'est utilisée) et cette méthode permet même de faire de la détection de collisions entre des objets utilisant des modèles de représentations différents (tant qu'ils peuvent être rendus sur une image). Le rendu peut être exécuté sur une carte graphique pour accélérer les calculs. L'inconvénient majeur de cette méthode est l'introduction d'approximations, les formes géométriques sont discrétisées sur un plan ce qui peut conduire à des erreurs. Un second inconvénient est le format du résultat, ce type de méthode produit des volumes d'interpénétration (cf. intersection dans la figure 1.15) qu'il faut post-traiter avant de pouvoir appliquer une réponse physique car ces volumes n'indiquent pas les directions pour séparer les objets.

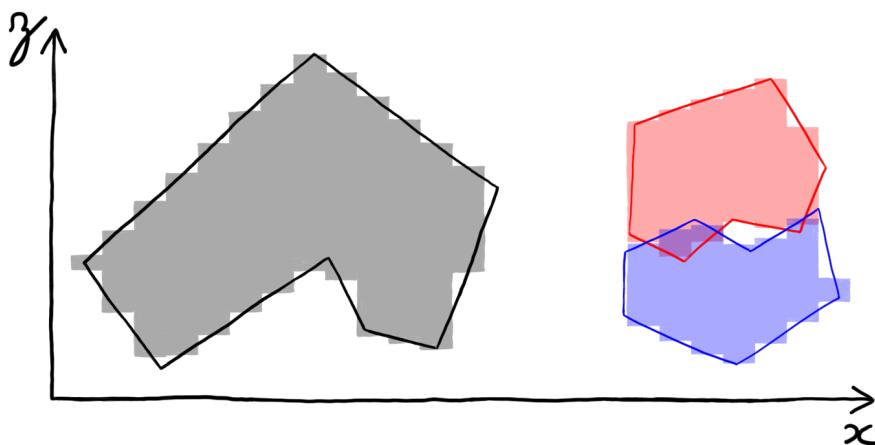


Figure 1.15 – Exemples de LDI en 2D, l'axe z est l'axe de profondeur. A gauche, un objet est discrétisé avec un LDI. A droite, un LDI est utilisé pour détecter les collisions entre deux objets.

L'une des premières approches en détection de collision basée image utilisant des LDI a été introduit par Shinya et al. [Shinya and Forgue, 1991]. Dans cette méthode, les objets sont rendus par rastérisation dans un tampon de profondeur multi-couches. Pour chaque pixel, on obtient les évènements d'entrée et de sortie des objets et ceux-ci sont triés par ordre de profondeur. Les collisions sont détectées en parcourant les listes de valeurs de profondeur pour chaque pixel à la recherche d'évènements en collision (tels que deux évènements successifs d'entrée dans un objet). Cette méthode a été testée dans le cas d'objets convexes car cela permet de n'avoir qu'à calculer la valeur minimum et le maximum de profondeur sur l'axe de projection pour chaque objet. Baciu et al. [Baciu et al., 1998] ont obtenu des résultats similaires en se limitant également aux objets convexes.

Myszkowski et al. [Myszkowski et al., 1995] ont proposé une approche similaire intégrant toutefois les objets non-convexes. Dans cette méthode, plusieurs algorithmes de détection sont disponibles pour chaque pixel suivant si les objets sont non-convexes ou convexes. Dans le cas d'objets non-convexes, un tri est nécessaire pour obtenir une détection correcte. Des limitations sur le tri dues aux contraintes techniques des GPUs de l'époque limitaient la profondeur du tri (ce qui posait des limites à la concavité des objets).

Heidelberger et al. [Heidelberger et al., 2003, Heidelberger et al., 2004] décomposent la détection de collision avec LDI en trois étapes. La première permet de localiser la

zone d'intérêt dans laquelle la LDI sera calculée. La seconde calcule la LDI dans la zone d'intérêt. La troisième détecte les collisions avec trois méthodes possibles en fonction du type de requête (collision, auto-collision ou test d'un point).

Govindaraju et al. [Govindaraju et al., 2005] proposent de réaliser une décomposition chromatique du maillage (avec une coloration de graphe) pour obtenir des listes de primitives non-adjacentes. La détection de collision dans un ensemble de primitives non-adjacentes étant plus performante.

Faure et al. [Faure et al., 2008, Allard et al., 2010] utilisent des LDI en réalisant des projections sur trois axes orthogonaux et proposent une méthode pour calculer la réponse physique appropriée. Pour éliminer les problèmes de choix de la discréétisation Wang et al. [Wang et al., 2012] ont proposé d'utiliser un échantillonnage adaptatif. Contrairement à un échantillonnage régulier, un échantillonnage adaptatif permet de cibler plus précisément les zones d'intérêts et ainsi éviter de sur-échantillonner des zones vides tout en évitant de sous-échantillonner des zones géométriquement complexes. Pour pouvoir calculer un LDI avec un échantillonnage irrégulier, la rastérisation classique est remplacée par du lancer de rayons.

Les méthodes de depth peeling ont rapidement montré leurs performances dans le cas de la simulation de tissus. Vassilev et al. [Vassilev et al., 2001] ont proposé les premiers travaux en simulations de vêtements utilisant le depth peeling. Leur méthode utilise un GPU pour calculer des cartes de profondeur sur un avatar virtuel qui est ensuite utilisé pour détecter les collisions avec un vêtement. Néanmoins, cette méthode ignore les auto-collisions générées par le vêtement.

B Tests de points à l'aide de rayons

Ces méthodes testent si un ensemble de points représentatifs de chaque objet appartient au second objet. Ces méthodes nécessitent que les objets soient des surfaces closes, ce qui empêche la simulation de tissus. Sans information d'orientation des faces, le théorème de Jordan peut être exploité pour discerner l'intérieur et l'extérieur des objets. La version 3D du théorème de Jordan pour les objets polygonaux [Devadoss and O'Rourke, 2011] permet de savoir si un point donné est à l'intérieur d'un polygone. Le théorème énonce la propriété suivante :

Théorème de Jordan. *Pour tout point x , polygone P et rayon r ayant pour origine x , si x a un nombre pair d'intersections avec la surface de P alors x est à l'extérieur de P , sinon x est à l'intérieur de P .*

La Figure 1.16 donne un exemple de classification d'un point à l'intérieur et à l'extérieur d'un polygone à l'aide du théorème de Jordan. Si des informations d'orientation sont disponibles sur les faces (dans ce cas les faces ont un côté extérieur et un côté intérieur), alors il est possible de savoir si un point est à l'intérieur d'un objet en regardant l'orientation de la première intersection entre un rayon ayant pour origine le point et la surface de l'objet.

CinDeR [Knott, 2003] utilise le théorème de Jordan pour réaliser les tests de collision. Dans le cas de CinDeR, les tests sont réalisés à partir des arêtes. Dans un premier temps, les arêtes sont dessinées dans une image. Un algorithme de lancer de rayon est ensuite appliqué pour compter le nombre de franchissements de faces sur

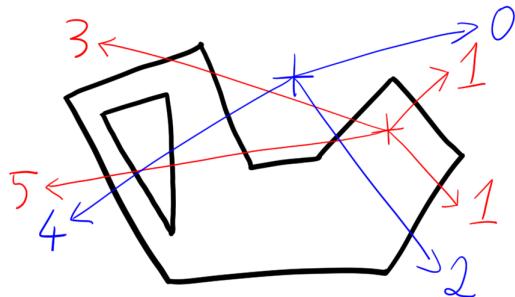


Figure 1.16 – Exemple de classification de points à l'aide du théorème de Jordan. Tous les rayons lancés à partir du point bleu ont un nombre d'intersections pair, ce point est à l'extérieur du polygone. Au contraire, tous les rayons lancés à partir du point rouge ont un nombre d'intersections impair, il est à l'intérieur du polygone.

chaque pixel. Le compteur est incrémenté lors du franchissement d'une face extérieure et décrémenté lors du franchissement d'une face intérieure. Si le compte final pour un pixel est impair, alors l'arête correspondante est en collision. Cet algorithme peut être implémenté efficacement sur GPU et ne nécessite pas de structures accélératrices particulières. Cette méthode nécessite que les objets soient des surfaces closes.

Wong et al. [Wong et al., 2013] exploitent le théorème de Jordan pour éliminer des paires de test entre primitives sans réaliser un lancer de rayon explicite lors des tests d'auto-collision. L'idée est que deux surfaces ne peuvent pas entrer en collision pendant un intervalle de temps Δt si depuis un point de vue q , les deux surfaces sont orientées dans la même direction par rapport à q durant l'intégralité de l'intervalle de temps Δt . Dans cette méthode, les objets sont décomposés en *clusters*, ce qui permet de grouper les primitives proches ensemble. Chacun de ces *clusters* a un point de vue q associé. De plus, lorsqu'il est possible de calculer un squelette sur un objet, les auteurs montrent que les noeuds de ce squelette sont d'excellent choix pour les points de vue q . Il faut cependant soit déjà avoir un squelette disponible (ce qui est le cas lors de l'animation d'humains virtuels), soit en calculer un (ce qui n'est possible que si les déformations sont limitées).

Hermann et al. [Hermann et al., 2008] utilisent les informations d'orientation des faces et proposent de réaliser les tests de collisions en lançant des rayons à partir de chaque sommet des objets vers l'intérieur de ceux-ci. Si un rayon touche l'intérieur d'un autre objet avant de quitter l'objet source, alors le sommet correspondant est en collision avec cet objet. La Figure 1.17 donne un exemple en deux dimensions. Cette méthode nécessite que les objets soient des surfaces closes avec des surfaces orientées (marquées avec un côté intérieur et un côté extérieur). Contrairement à la plupart des méthodes basées image, cette méthode fournit immédiatement les informations nécessaires pour calculer la réponse physique, les rayons fournissant la direction de la collision et sa profondeur. De plus, cette méthode ne discrétise pas l'espace ce qui évite les erreurs dues aux approximations contrairement aux méthodes précédentes. Cette méthode réalise néanmoins une approximation, les tests de collision ne sont réalisés que sur les sommets, cette méthode n'est donc pas capable de détecter les collisions entre arêtes.

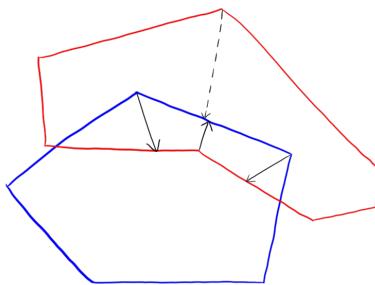


Figure 1.17 – Exemple de la méthode de Hermann et al. Les trois flèches noires sont des rayons détectant des collisions, les objets peuvent être séparés en appliquant des forces sur les sommets correspondants dans la direction des rayons. La flèche en pointillé est un rayon rejeté car il touche l’extérieur du polygone bleu, le sommet correspondant est à l’extérieur du polygone bleu.

C Autres méthodes

CULLIDE [Govindaraju et al., 2003] utilise des requêtes d’occlusion pour éliminer des tests de collisions entre des objets. CULLIDE utilise le lemme suivant : si un objet A est entièrement visible par rapport à un ensemble d’objets E , alors A n’est en collision avec aucun objet de l’ensemble E . Un objet est entièrement visible si aucun fragment (i.e. pixels) appartenant à l’objet n’est caché par un autre objet lors d’un rendu. Cette méthode commence par lister l’ensemble des objets et le lemme est utilisé pour retirer itérativement des objets de cette liste un à un avec des requêtes d’occlusion. Cette méthode est répétée ensuite en découplant les objets en sous-objets. Le résultat final est une liste de sous-objets qui sont potentiellement en collisions et pour lesquels une méthode de détection exacte de collision est exécutée. Boldt et al. [Boldt and Meyer, 2005] ont étendu CULLIDE pour gérer les tests d’auto collision.

Kim et al. [Kim et al., 2002b, Kim et al., 2002a] décomposent les objets en un ensemble de primitives convexes et réalisent la différence de Minkowski entre les primitives des objets. Les différences de Minkowski sont ensuite combinées à l’aide de techniques basées image. Le résultat final est un *cube-map* qui indique dans chaque direction la distance entre les deux objets. Cette méthode a l’avantage, en plus de détecter et mesurer les collisions, de pouvoir mesurer les distances entre les objets. Néanmoins, cette nécessité de décomposer les objets en ensembles de primitives convexes peut générer une importante combinatoire lorsque les objets sont fortement non-convexes.

2.2.5 Champs de distance

Un champ de distance est une application $D : \mathbb{R}^3 \rightarrow \mathbb{R}$ pour lequel $D(x)$ est la distance euclidienne entre x et la surface de l’objet. Un champ de distance peut être signé, dans ce cas $D(x)$ est positif si x est à l’extérieur de l’objet et négatif si x est à l’intérieur de l’objet. Le calcul et le stockage d’un champ de distance est extrêmement complexe, ils sont généralement trop coûteux pour être utilisés avec des objets déformables.

Les champs de distances sont été utilisés lorsque les collisions sont déjà connues pour estimer la profondeur de celles-ci dans des objets déformables [Fisher and Lin, 2001].

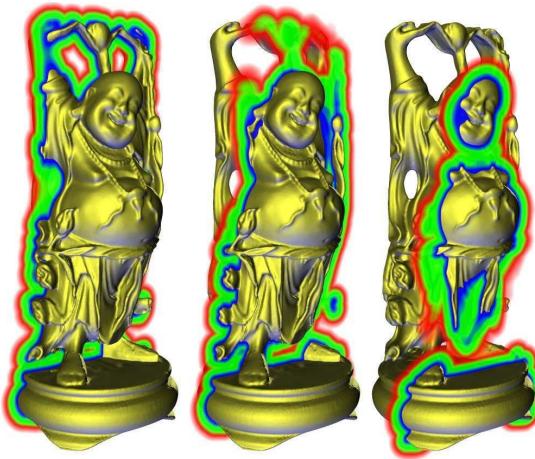


Figure 1.18 – Exemple d'un champ de distance [Teschner et al., 2005]. Trois plans de coupe du champ de distance sont affichés avec une coloration en fonction de la distance (bleu = faible distance, rouge = distance élevée).

Dans cette méthode le champ de distance est calculé uniquement à l'intérieur des objets avec une méthode des éléments finis, le coût des calculs empêchant toute utilisation en temps-réel.

DiFi [Sud et al., 2004] utilise des techniques basées image pour calculer et utiliser un champ de distance calculé sur des couches et la cohérence spatiale entre les couches adjacentes est exploitée. Les champs de distance sont calculés en temps réel ce qui permet de réaliser des tests de collision entre deux objets déformables.

Fuhrmann et al. [Fuhrmann et al., 2003] proposent de calculer les collisions entre un objet rigide et un objet déformable en utilisant uniquement le champ de distance de l'objet rigide (qui est constant et ne nécessite pas de mise à jour). Cette méthode est capable d'animer un vêtement en temps-réel sur un mannequin rigide. Pour pouvoir animer le mannequin, les auteurs proposent de le modéliser à l'aide d'une combinaison d'objets rigides. Cette méthode n'est pas applicable entre deux objets déformables car il serait nécessaire de recalculer le champ de distance de l'un des deux objets après chaque déformation. Morvan et al. [Morvan et al., 2008] proposent une implémentation GPU pour le calcul de distance entre un objet rigide (avec un champ de distance pré-calculé) et un autre objet éventuellement déformable.

Glondu et al. [Glondu et al., 2012, Glondu et al., 2014] proposent de combiner des champs de distance avec un arbre de sphères pour réaliser la détection de collision entre des objets rigides subissant des fractures. Les tests de collision sont réalisés en comparant l'arbre de sphère de chaque objet avec le champ de distance de l'autre objet. Les champs de distance sont mis à jour localement en cas de fracture.

3 Optimisations

Certaines méthodes et optimisation de la détection de collision sont plus génériques que les méthodes déjà présentées et peuvent s'appliquer en dehors du schéma *broad-phase/narrow-phase*. Ces méthodes comprennent le *culling* qui est utilisé pour réduire

les calculs (cf. Section 3.1), les méthodes spécifiques de simulation qui ont été mises en œuvre pour la simulation de tissus et de vêtements qui impactent le déroulement de la détection de collision (cf. Section 3.2) et les méthodes qui modélisent l'espace entre les objets pour détecter les collisions (cf. Section 3.3).

3.1 Culling

Le *culling* est l'élimination de tests de collision entre des parties des objets. Le *culling* peut être appliqué entre la *broad-phase* et la *narrow-phase* ou être appliqué pendant la *narrow-phase*. Le but est d'éliminer des tests de collision à l'aide de critères de haut niveau. Pour être rentable, le coût de calcul d'une méthode de *culling* doit être inférieur au coût des tests de collision éliminés.

Les vitesses relatives entre les objets peuvent être exploitées dans le *culling*. Dans le cas des polyèdres, Vaněkček [Vaněkček, 1994] propose de comparer le vecteur vitesse d'une face à sa normale. Si le vecteur vitesse et la normale sont opposés, alors la face est en train de s'éloigner et il n'est donc pas nécessaire de tester les collisions sur cette face. Cette méthode est appliquée sur les objets rigides pour lesquels les vecteurs vitesses ont de bonne propriétés de linéarité. Mezger et al. [Mezger et al., 2003] proposent de représenter la vitesse d'un ensemble de primitives à l'aide de cônes de vitesses. Le cône représente une approximation de la distribution des vecteurs vitesses des primitives, la longueur du cône mesure la vitesse maximum atteinte par les primitives et l'angle d'ouverture du cône mesure la corrélation entre les vecteurs vitesses des primitives. Deux ensembles de primitives sont testées entre elles si leur cônes de vitesse sont en collision.

Des volumes englobants supplémentaires peuvent être attribués aux primitives. Hutter et al. [Hutter and Fuhrmann, 2007] utilisent des volumes englobants sur les sommets et arêtes des triangles (en plus des volumes englobants utilisés sur les triangles) pour réduire le nombre de tests unitaires. Pour gérer plus efficacement les volumes englobants des sommets et des arêtes, Curtis et al. [Curtis et al., 2008] introduisent la notion de triangles représentatifs. Chaque sommet et arête sont représentés par un unique triangle dit représentatif qui a pour charge de stocker les volumes englobants de ceux-ci. Cette méthode permet d'éviter de dupliquer certains tests de collision entre primitives (tels que les tests entre arêtes) en leur attribuant un triangle responsable.

Dans le cadre de l'auto-collision, Barbič et al. [Barbič and James, 2010] proposent de calculer des certificats entre des sous ensembles d'un objet. Les certificats sont calculés dans l'état initial de l'objet et sont évalués lors des déformations. Tant que les certificats sont valides il n'est pas nécessaire de réaliser les tests de collision. Les certificats sont calculés en terme de déplacement modaux ce qui restreint le type des déformations. Zheng et al. [Zheng and James, 2012] proposent d'utiliser une mesure d'énergie en tant que certificat pour permettre des déformations arbitraires. Ces certificats mesurent, pour chaque partie de l'objet, la quantité d'énergie que celle-ci doit recevoir avant de pouvoir entrer en auto-collision. Ceux-ci sont couplés à une hiérarchie de volumes englobants et cette méthode requiert que la structure de la hiérarchie soit fixe, ceci limite fortement l'utilisation de techniques d'optimisation dynamique des hiérarchies pendant les simulations.

3.2 Simulation de tissus multi-résolution

La simulation de tissus et de vêtements pose de nombreux problèmes aussi bien au niveau de la détection de collision que de la réponse physique. Certains problèmes liés à la réponse physique ajoutent des contraintes à la détection de collision.

Lorsque des objets complexes sont simulés, une optimisation possible est d'utiliser des représentations avec différents niveaux de détails disponibles. Cette méthode est utilisée dans la simulation de tissus pour laquelle la réponse physique est extrêmement coûteuse. Kang et al. [Kang and Cho, 2002] proposent de simuler les tissus à deux niveaux de détails. Le premier niveau simule le comportement global du tissu avec la détection de collision. Le second niveau simule le comportement local du tissu, tels que la formation de plis. Müller et al. [Müller and Chentanez, 2010] proposent une méthode pour gérer les collisions au second niveau de détail mais sans prendre en compte les auto-collisions.

3.3 Modélisation de l'espace entre les objets

Au lieu de modéliser les objets et leur intérieur, il est possible de modéliser l'espace présent entre les objets. Cette méthode de décomposition peut s'avérer difficile, Held et al. [Held et al., 1995] ont proposé de décomposer l'espace libre avec des tétraèdres dont les faces coïncident avec les faces des objets. La détection de collision est réalisée si certains tétraèdres sont en intersection avec les faces de certains objets. Comparés à d'autres méthodes, les résultats de celle-ci sont mitigés, cette méthode est plus performante uniquement dans un nombre limités de cas.

Müller et al. [Müller et al., 2015] réalisent la détection de collision et la réponse physique en une seule étape en utilisant un maillage tétraédrique entre les objets (rigide et déformables). Les collisions sont empêchées en forçant les volumes des tétraèdres à rester positifs. La méthode requiert de recalculer le maillage tétraédrique de l'espace en cas de trop grands déplacements des objets. Le recalcule du maillage tétraédrique est une opération coûteuse qui empêche d'utiliser cette méthode en temps-réel de manière générique. Néanmoins, celle-ci peut être utilisée dans des situations où les déplacements maximaux sont contraints, notamment lors de la simulation de vêtements sur un humain virtuel.

4 Solutions GPUs

Les GPUs (*Graphics Processing Units*) sont des processeurs dédiés au rendu d'image. Originellement non-programmables, les GPUs actuels sont grandement programmables ce qui a donné naissance au GPGPU (*General-Purpose Processing on Graphics Processing Units*) qui est un domaine de l'informatique visant à utiliser les GPUs pour réaliser des calculs génériques, pas nécessairement liés à des calculs graphiques (cf. Section 4.1). Les GPUs sont aujourd'hui exploités pour réaliser la détection de collision de manière plus performante que ce soit pour la *broad-phase* ou pour la *narrow-phase* (cf. Sections 4.2 et 4.3).

4.1 Évolution des GPUs

L'évolution des GPUs a été pilotée par la demande du marché pour le rendu temps-réel en haute définition. Cette demande a poussé les fondeurs à produire des GPUs de plus en plus puissants, creusant l'écart avec les CPUs en termes de puissance de calcul théorique et taille de bande passante vers la mémoire (cf. Figure 1.19). Les GPU ont aussi une consommation électrique plus faible à puissance de calcul équivalente. Cette différence de puissance de calcul en terme d'opérations numériques s'explique par la spécialisation des GPUs. Ceux-ci ont été conçus pour les calculs destinés au rendu d'image à partir de données géométriques en 3D, ils ont donc été spécialisés pour les calculs à virgule flottante en dimension 3 ou 4.

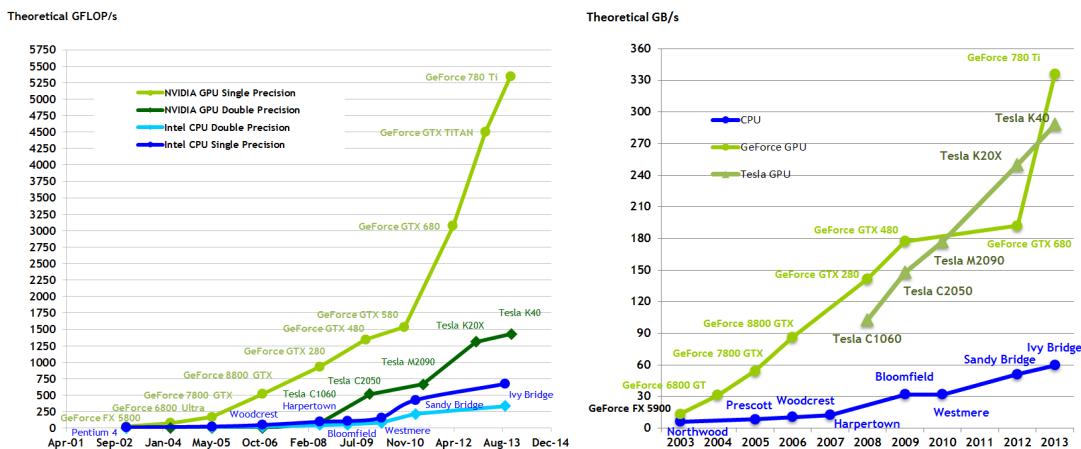


Figure 1.19 – Évolution de la puissance de calcul théorique des GPUs et CPUs (gauche) et du débit maximum entre le processeur et leur mémoire (droite). FLOPS (*Floating point Operations Per Second*) = nombre d'opérations en virgule flottante par seconde.

Les CPUs et les GPUs possèdent des différences de conception qui ont été héritées de leur utilisation historique. Les CPUs sont des processeurs généralistes capables de traiter les flux de données génériques à forte interdépendance. Les évolutions récentes en parallélisme leur permettent de traiter de manière indépendante des flux de données en parallèle. Les GPUs sont des processeurs spécialisés en traitement de données parallèles présentant peu d'interdépendances. Historiquement, les GPUs étaient capables de réaliser uniquement du rendu d'image, les évolutions récentes permettent aujourd'hui de les programmer pour réaliser des calculs génériques. Cependant, les contraintes techniques liées au GPUs peuvent limiter considérablement les performances de nombreux algorithmes rendant leur implémentation GPU non viable. Parmi les contraintes, les plus importantes sont le parallélisme, la divergence d'exécution, l'ordonnancement des accès mémoires et le coût des transferts de mémoire entre GPU et CPU.

Parallélisme : Les GPUs actuels peuvent contenir plusieurs milliers de coeurs, et ces coeurs ne peuvent pas exécuter des fonctions indépendantes. Lorsque un calcul est exécuté sur GPU, tous les coeurs exécutent en parallèle la même fonction. Il est donc nécessaire d'exécuter les algorithmes intrinsèquement

parallèles, les algorithmes séquentiels ne seront pas capable d'utiliser l'intégralité des cœurs.

Divergence d'exécution : Chaque cœur ne possède pas sa propre unité de contrôle dédiée. Les unités de contrôle sont partagées entre des groupes de cœurs appelés *work-group*. Tous les cœurs appartenant à un *work-group* avancent de manière synchronisée dans le flux d'instructions. Si le flux d'exécution diverge dans un *work-group*, chaque flux divergeant sera exécuté séquentiellement. Il faut donc réduire au maximum la divergence à l'exécution qui est typiquement dûe à l'utilisation d'instructions conditionnelles.

Ordonnancement des accès mémoires : Les accès mémoires sur GPUs doivent être ordonnés pour maximiser les performances. Pour réduire le nombre de transactions avec la mémoire, les cœurs successifs doivent accéder à des adresses mémoire successives. Ce problème est exacerbé par la plus faible quantité de caches présents entre les cœurs de GPU et la mémoire comparée aux CPUs. Une méthode simple pour ordonner les accès mémoire est de fournir en entrée des calculs un buffer ordonné et d'établir une correspondance un à un entre chaque élément du buffer d'entrée et chaque cœur.

Coût des transferts de mémoire : Les coûts des transferts de mémoire entre GPUs et CPUs sont actuellement élevés. Ce coût est dû aux architectures actuelles dans lesquelles chaque GPU a sa propre mémoire distincte de la mémoire centrale (qui est accessible depuis les CPUs). Pour accéder à des données entre un CPU et un GPU, il est nécessaire de recopier l'intégralité des données. Dans ce contexte il faut minimiser les quantités de données à transférer entre le CPU et les GPUs. Il est à noter que des architectures à mémoire unifiée sont en train d'émerger, ces architectures éliminent les problèmes de copie de mémoire car le CPU et le GPU partagent une mémoire unique.

Malgré toutes ces contraintes, les GPUs sont d'excellents candidats pour être utilisés pour réaliser la détection de collision. La détection de collision est une tâche impliquant de nombreux calculs géométriques en 3D, ce pour lesquels les GPUs ont été conçus. La taille et complexité des modèles 3D complets entraînent des problématiques de vitesse d'accès mémoire aux données géométriques. Les GPUs ont été conçus pour pouvoir accéder à de gros volumes de données rapidement à l'aide d'une bande passante élevée vers la mémoire. La contrainte de parallélisme est plus problématique, certaines méthodes ne sont pas parallèle par nature, il est donc nécessaire de les modifier pour les rendre parallèles. D'autres méthodes sont intrinsèquement parallèles, mais peuvent contenir de trop grandes interdépendances entre les calculs rendant leur utilisation sur GPUs difficile.

Nous nous intéressons maintenant aux travaux qui ont été proposés dans la littérature pour prendre en charge la détection de collision sur GPU pour la *broad-phase* et la *narrow-phase*.

4.2 Broad-phase GPU

La *broad-phase* doit tester un ensemble de n objets pour déterminer ceux qui sont potentiellement en collision. L'approche naïve testant toutes les combinaisons d'objets

(ayant une complexité de $O(n^2)$) est facilement parallélisable sur GPU car le test de collision de chaque paire est complètement indépendante. Cette méthode est équivalente à tester toutes les cases d'une matrice dont les lignes et les colonnes correspondent aux numéros des objets. Il n'est alors pas nécessaire de tester la diagonale de la matrice (qui correspond à l'auto-collision), ni la partie triangulaire supérieure de la matrice (car la partie triangulaire inférieure réalise les mêmes tests). Avril et al. [Avril et al., 2012] ont proposé d'utiliser un *thread* par paire et une fonction de *mapping* est utilisée pour aider chaque *thread* à déterminer quelle paire tester en fonction de son identifiant. Cette fonction de *mapping* permet de générer toutes les paires d'identifiants correspondant à la matrice triangulaire inférieure. Ce problème de parcours de matrice triangulaire sur GPU a été généralisé par Navarro et al. [Navarro and Hitschfeld, 2013].

Le Grand [Le Grand, 2007] a proposé d'utiliser une subdivision spatiale avec une grille régulière combinée avec une fonction de hachage. Chaque objet stocke les clés de hachage des cases le contenant. Les correspondances entre les clés de hachage pour des objets différents sont ensuite recherchées.

L'algorithme de *sweep-and-prune* est lui plus complexe à mettre en place sur GPU à cause de la présence de tris. Liu et al. [Liu et al., 2010] ont proposé de réaliser le *sweep-and-prune* sur GPU sur un seul axe séparateur et repose sur un algorithme de tri adapté pour GPU (*radix-sort*).

4.3 Narrow-phase GPU

La *narrow-phase* étant l'étape la plus consommatrice en temps de calculs, l'utilisation de la puissance de calculs des GPUs permet d'alléger la charge de travail. Nous ne rediscutons pas des méthodes basées image car celles-ci sont déjà adaptées pour les GPUs.

Les méthodes hiérarchiques (BVH) ont vu un effort récent pour les adapter sur GPU. Avec *gProximity*, Lauterbach et al. [Lauterbach et al., 2010] proposent une méthode complète pour construire, mettre à jour et utiliser des hiérarchies de volumes englobants sur GPU. La détection de collision est réalisée de manière discrète ou continue. La construction de la hiérarchie est décomposée en deux phases, une phase de découpage et une phase d'ajustement. Dans la phase de découpage, les primitives sont groupées en noeuds et ceux-ci sont placés dans la hiérarchie avec un algorithme hautement parallèle [Lauterbach et al., 2009]. La phase d'ajustement calcule les volumes englobants des noeuds avec une approche *bottom-up*. Cette méthode est également parallèle, les calculs des volumes englobants à un niveau donné de la hiérarchie sont tous indépendants, chaque niveau étant calculé de manière itérative. La mise à jour de la hiérarchie pour les objets déformables est réalisée en recalculant les volumes englobants des noeuds en utilisant la méthode de la phase d'ajustement. Les tests de collisions sont réalisés en comparant deux BVH, les tests entre chaque paire de noeuds peuvent être parallélisés mais en début du parcours le nombre de paires de noeuds parcouru est trop faible pour occuper tous les coeurs d'un GPU. Pour obtenir un haut parallélisme dès le début du parcours la cohérence temporelle est exploitée. Le parcours démarre avec la "liste de séparation" (cf. Section 2.2.3) calculée au pas de temps précédent. Cette liste contient un grand nombre d'éléments lorsque les deux objets sont proches et permet de paralléliser le parcours de l'arbre très rapidement. Ce parcours est associé à une

méthode de distribution de tâches basée sur des listes d'attente locales. Chaque *work-group* possède sa liste de tâches et l'exécution de celles-ci est entrecoupée par des phases de redistribution des tâches lorsque trop de coeurs sont en attente. Les *Collision-Streams* [Tang et al., 2011a] permettent d'améliorer cette méthode en proposant une méthode de distribution de tâches sur GPU sous forme de flux pour mieux gérer l'enregistrement de nouvelles tâches pendant le parcours de la hiérarchie.

Pabst et al. [Pabst et al., 2010] proposent d'utiliser les méthodes de subdivision spatiale avec du hachage spatial. Cette méthode utilisée pour la *broad-phase* obtient des performances remarquables pour la *narrow-phase* avec des objets déformables grâce à la parallélisation massive sur GPU.

5 Synthèse

Au fil de cet état de l'art nous avons vu un grand nombre de méthodes permettant de réaliser la détection de collision. L'approche divisant le problème en deux étapes (*broad-phase* et *narrow-phase*) est l'approche la plus utilisée aujourd'hui. Cependant, les méthodes actuelles sont incapables de traiter le problème dans son intégralité en temps-réel, de nombreuses méthodes reposent sur des approximations sur les objets (en les simplifiant ou en ajoutant des contraintes) ou sur une limitation du nombre d'objets pour réduire la charge calculatoire. Les méthodes proposées sont alors non-génériques, elles sont spécialisées sur des cas d'utilisations précis. Aujourd'hui, nous cherchons à simuler des environnements complexes étant capable de prendre en compte un ensemble de contraintes antagonistes :

Gérer un grand nombre d'objets : Nous souhaitons être capables de simuler des environnements comprenant un grand nombre d'objets indépendants.

Gérer des objets complexes : Nous voulons être capables de travailler avec des objets ayant n'importe quelle complexité sans poser des contraintes sur la convexité et la rigidité des objets.

Fonctionner en temps-réel : Notre but est d'être capable d'avoir des interactions directes entre les utilisateurs et la simulation, les algorithmes doivent alors travailler en temps-réel.

En ce qui concerne la *broad-phase*, les méthodes actuelles peuvent atteindre le temps-réel avec une centaine de milliers d'objets et s'approchent du million d'objets en temps-réel. Elles respectent donc les contraintes sur le nombre d'objets et sur le fonctionnement en temps réel. Concernant la complexité des objets, la *broad-phase* travaille avec les volumes englobants des objets et ne s'occupe pas de leur forme réelle. L'augmentation de la complexité des objets n'a donc pas d'impact sur les performances de la *broad-phase*. Notre état de l'art montre donc que la *broad-phase* n'est pas la phase la plus critique en cas de simulations complexes.

Au contraire, la *narrow-phase* travaille avec la forme réelle des objets. Cette phase est donc plus sensible à la complexité des objets. Pour résoudre la détection de collision, certaines familles de méthodes posent des contraintes sur les objets. Les méthodes basés caractéristiques sont limitées en matière de dynamité des objets, elles sont trop coûteuses pour être utilisées avec des objets déformables ou avec des changements

topologiques et certaines de ces méthodes se limitent aux objets convexes. Les méthodes basées simplexes sont capables de traiter des objets déformables et des changements topologiques efficacement mais sont limitées aux objets convexes. Les hiérarchies de volumes englobants sont aujourd’hui largement utilisées en cas de simulations d’objets complexes. Ces méthodes nécessitent en revanche des algorithmes d’optimisations d’arbres en cas de trop forte déformation ou en cas de changements topologiques des objets. Il est également complexe de mettre en œuvre ces méthodes sur GPU. Les méthodes basées images peuvent travailler avec des objets ayant n’importe quelles propriétés avec un faible sur-coût en utilisant les modèles de représentations utilisés lors du rendu d’image. Une grande partie de ces méthodes réalisent une discréétisation de l’espace pouvant poser des problèmes de précision. Les méthodes utilisant des champs de distances peuvent travailler en temps réel entre des objets rigides et des objets déformables, mais sont difficilement applicables entre des objets déformables à cause du calcul du champ de distance.

Dans ce contexte nous avons décidé de nous intéresser à la *narrow-phase*, qui est la phase la plus complexe et la plus consommatrice en ressources actuellement, et de chercher à proposer une méthode répondant au mieux aux contraintes que nous avons proposées. Dans la littérature actuelle, les méthodes les plus génériques en termes de type d’objets sont les méthodes utilisant des hiérarchies de volumes englobants et les méthodes basées images qui sont capables de traiter aussi bien des objets rigides (convexes et non-convexes) que des objets déformables et avec des changements topologiques.

La puissance de calculs des GPUs actuels dépassant largement la puissance des CPUs, il est intéressant de chercher à implémenter les problèmes hautement calculatoires sur GPUs. Dans le cadre de la *narrow-phase*, les méthodes basées images sont les meilleurs candidats pour être implémentées sur GPU car ces méthodes utilisent des algorithmes de rendu d’images pour lesquels les GPUs ont été conçus.

Avec toutes ces observations sur la générnicité des méthodes actuelles de détection de collisions et sur l’intérêt à utiliser les GPUs comme périphériques de calculs, il nous semble intéressant de nous intéresser aux méthodes basées images. Ces méthodes sont génériques en termes de types d’objets utilisables (rigide/déformable, convexe/non-convexe, changements topologiques) et sont compatibles avec une implémentation sur GPU. En particulier dans les méthodes basées images, nous avons décidé de nous intéresser à la méthode proposée par Hermann et al. [Hermann et al., 2008]. Il a été montré que cette méthode est capable de réaliser la détection de collision correctement pour des objets rigides et déformables. Contrairement à la majorité des méthodes basées images, cette méthode ne réalise pas de discréétisation de l’espace, ce qui élimine les problèmes d’approximations. Cette méthode n’a en revanche pas été étudiée en cas d’environnement large échelle temps-réel avec un grand nombre d’objets complexes.

Dans ce contexte nous proposons dans cette thèse une méthode de détection de collision basée image reposant sur des algorithmes de lancer de rayon. Dans notre méthode nous généralisons la méthode de Hermann et al. tout en améliorant ses performances. Nous décrivons dans le prochain chapitre l’approche que nous avons proposée dans cette thèse.

Cadre général des contributions

2

Suite à l'état de l'art, ces perspectives et les contraintes de la détection de collision, nous avons décidé de nous intéresser à la détection de collision par lancer de rayon sur GPU.

Les évolutions actuelles des GPUs en termes de puissance de calcul et d'efficacité énergétique justifient leur utilisation pour exécuter la détection de collision. De nombreux travaux ont été proposés pour adapter les méthodes existantes à une utilisation sur GPU en les parallélisant. D'autres travaux tentent d'utiliser les méthodes de rendu d'image pour directement réaliser la détection de collision au prix de discrétisations. Nous avons décidé de nous orienter vers l'utilisation du calcul sur GPU, il nous semble extrêmement intéressant d'exploiter les matériels de calculs les plus performants à ce jour pour résoudre ce problème calculatoire.

Nous nous sommes intéressés à la méthode de Hermann et al. [Hermann et al., 2008] qui réalise la détection de collision avec des algorithmes de lancer de rayon. La détection de collision est réalisée en lançant des rayons à partir des sommets des objets vers leurs intérieurs (cf. Figure 1.17). Celle-ci présente plusieurs avantages qui la rendent générique et performante :

- Cette méthode est capable de réaliser la détection de collision en temps-réel.
- Aucune contrainte n'est posée sur la convexité des objets ni sur leur rigidité.
- Il est possible d'utiliser n'importe quel type de modèle de représentation. Il est juste nécessaire de posséder un algorithme de lancer de rayon pour chaque modèle et d'une discrétisation pour choisir les points d'origine des rayons.
- Cette méthode fournit donc toutes les informations nécessaires pour réaliser une réponse physique sans nécessiter de post-traitement complexes (contrairement à de nombreuses méthodes basées image). Elle permet donc un fonctionnement en synergie avec la réponse physique sans nécessiter de calculs intermédiaires supplémentaires.
- Les calculs réalisés sont intrinsèquement parallèles ce qui permet une implémentation facile sur GPU. De plus, de nombreux travaux de recherche ont été proposés pour exécuter les algorithmes de lancer de rayon sur GPU de manière efficace.

Nous proposons dans cette thèse des contributions pour améliorer les performances et la générnicité de cette méthode. Dans ce chapitre, nous présentons le cadre général de nos contributions et expliquons intuitivement leur motivation. Ces contributions seront ensuite détaillées en profondeur dans les chapitres suivants. Nous commençons par discuter du lien entre détection de collision et lancer de rayon (cf. Section 1). Nous présentons ensuite les contributions de cette thèse de manière générale en donnant l'idée ou les propriétés qu'elles exploitent. Ces contributions sont liées à l'exploitation de la

cohérence temporelle en vue d'améliorer les performances (cf. Section 2), à l'estimation de distances inter-objets à l'aide de lancer de rayon (cf. Section 3), à la combinaison d'algorithmes de lancer de rayon (cf. Section 4), à la généralisation de la méthode de Hermann et al. à des surfaces non-closes (cf. Section 5) et à l'exploitation des processeurs graphiques en tant qu'unités de calculs (cf. Section 6).

1 Lancer de rayon

En nous focalisant sur la détection de collision utilisant des techniques de lancer de rayon, il est utile de nous intéresser aux méthodes de lancer de rayon en général. Nous n'allons cependant pas faire un état de l'art exhaustif sur les techniques de lancer de rayon. Pour une vue plus approfondie, de nombreux état de l'art sont disponibles sur les techniques de rendu par lancer de rayon [Wald et al., 2009, Wald and Slusallek, 2001]. Nous nous intéressons uniquement aux problématiques liées à la détection de collision, en particulier les structures accélératrices utilisées (cf. Section 1.1), les problématiques de cohérence entre les rayons et de cohérence temporelle de point de vue de la caméra (cf. Sections 1.2 et 1.3) et l'utilisation de GPUs (cf. Section 1.4).

1.1 Structures accélératrices

Les structures accélératrices de lancer de rayon ont le même rôle que les structures accélératrices pour la détection de collision : exploiter la cohérence spatiale. Les structures accélératrices les plus utilisées pour le lancer de rayon sont les grilles uniformes [Ize et al., 2006], les *kd-tree* [Foley and Sugerman, 2005], et les BVH [Wald et al., 2007]. Ces structures sont les mêmes que celles utilisées en détection de collision et les problématiques sont proches. Parmi ces problèmes communs on peut citer les problèmes de construction et de mise à jour des structures accélératrices pour les objets déformables.

Les algorithmes de parcours des structures accélératrices sont en revanche différents entre le lancer de rayon et la détection de collision. Contrairement aux algorithmes de détections de collision qui comparent les structures accélératrices deux à deux, les algorithmes de lancer de rayon parcourront une seule structure accélératrice à la fois pour tracer des rayons.

1.2 Cohérence spatiale entre les rayons

Dans le rendu par lancer de rayons, l'image est calculée à partir d'une caméra en suivant le trajet inverse de la lumière. Avec ce type de méthodes, il est possible de distinguer deux types de rayons : les rayons primaires et les rayons secondaires. Les rayons primaires sont les premiers rayons lancés à partir de la caméra. Les rayons secondaires sont les autres rayons calculés et sont issus de "rebonds" des rayons primaires. Les rayons secondaires sont entre autre utilisés pour calculer les ombres, les réflexions et les réfractions.

Il existe une forte cohérence spatiale entre les rayons primaires, ils ont tous une origine commune (la camera) et partagent tous la même direction générale (direction

de la caméra). Cette cohérence est exploitée en groupant les rayons lors du parcours des structures accélératrices [Lauterbach et al., 2006]. Cependant, dans le cas de la détection de collision par lancer de rayon, cette cohérence entre les rayons n'est pas toujours présente. Par exemple la méthode de [Faure et al., 2008] possède une cohérence entre les rayons mais la méthode de [Hermann et al., 2008] n'en possède pas. Ce cas sans cohérence est aussi présent dans le rendu par lancer de rayon avec les rayons secondaires, dans cette situation les méthodes qui groupent les rayons sont inutiles.

1.3 Cohérence temporelle de la caméra

De même que dans la détection de collision, le rendu par lancer de rayon est généralement effectué sur des scènes qui possèdent une certaine cohérence temporelle. Cette cohérence temporelle est visible dans le rendu d'image sous la forme de similarités entre les images successives. *Render Cache* [Walter et al., 1999] est une méthode destinée à exploiter la cohérence temporelle. L'idée est de réaliser une reprojecion de l'image calculée à l'instant $t - 1$ sur l'image à l'instant t , il ne reste ensuite plus qu'à calculer les parties manquantes de l'image pour obtenir un rendu complet. Cette exploitation de la cohérence temporelle est difficilement compatible avec les problématiques de détection de collision. Avec les méthodes de détection de collision basées images utilisant des LDIs, il est difficile de faire la correspondance entre les images et de localiser les zones nécessitant un nouveau rendu car il faut calculer les correspondances entre les images multi-couches. Avec la méthode de [Hermann et al., 2008], la reprojecion de l'image est impossible car aucune image cohérente n'est calculée.

1.4 Rendu sur GPU

De même que pour la détection de collision, le rendu par lancer de rayon peut être réalisé sur GPU. Les structures des données accélératrices de lancer de rayon étant proches de celles utilisées par la détection de collision. Les algorithmes de construction et de mise à jour de celles-ci sont très proches voir identiques. Concernant leur parcours, les requêtes étant différentes, des méthodes spécialisées dans le traçage des rayons sur GPU ont été proposées. L'une des optimisations les plus importantes à ce jour est liée au parcours des structures accélératrices. Le coût mémoire du parcours des structures accélératrices avec une pile est trop élevé pour les GPUs. Par conséquent, des méthodes permettant de parcourir les structures accélératrices sans pile ont été mises au point, que ce soit pour les *kd-tree* [Popov et al., 2007] ou les BVHs [Hapala et al., 2011].

1.5 Synthèse

Avec cette brève discussion sur le lancer de rayon, nous remarquons que les structures accélératrices utilisées en lancer de rayon sont proches de celles utilisées en détection de collision et exploitent les mêmes propriétés sur la cohérence spatiale. Les autres formes de cohérences ne sont pas présentes ou pas exploitées de manières compatibles avec les méthodes que nous étudions. La cohérence entre les rayons qui est exploitée par les méthodes de rendu par lancer de rayon n'est pas applicable dans notre

cas car le lancer de rayon tel qu'utilisé par Hermann et al. ne présente pas de cohérence entre les rayons. La cohérence temporelle n'est pas exploitée de manière compatible avec la méthode de détection de collision que nous utilisons car elle repose sur la présence d'une certaine cohérence entre les rayons. Cependant, la cohérence temporelle existe dans le cas de la méthode de Hermann et al. mais devrait être exploitée sous une autre forme.

2 Exploitation de la cohérence temporelle

La détection de collision est effectuée à intervalle de temps fixe, entre ces pas de temps la variation de la configuration géométrique de la scène est assez faible, il existe donc une cohérence entre chaque pas de temps que l'on appelle cohérence temporelle. Cette cohérence temporelle justifie la réutilisation de certains résultats de la détection de collisions d'un pas de temps au pas suivant (que ces résultats soient finaux ou intermédiaires). Cette réutilisation de résultats permet de réduire le temps de calcul et ainsi améliorer les performances. Il n'est donc pas étonnant de constater que la cohérence temporelle est utilisée de manière récurrente dans la résolution de la détection de collision.

La cohérence temporelle n'est en revanche pas exploitée dans le cas des méthodes basées image, et notamment dans les méthodes utilisant des algorithmes de lancer de rayon. L'utilisation de la cohérence temporelle a pourtant été proposée dans les méthodes de rendu par lancer de rayon (cf. Section 1.3), mais ces méthodes ne sont pas adaptées à la détection de collision. Les méthodes exploitant la cohérence temporelle dans le rendu par lancer de rayon cherchent à créer des correspondances entre les images rendues successivement. Conceptuellement, cette méthode est équivalente à chercher à trouver des correspondances entre les rayons en cours de calcul (au pas de temps t) et les rayons précédents (au pas de temps $t - 1$). Les correspondances trouvées ne sont généralement pas entre les mêmes rayons de la caméra (ie. rayons lancés dans la même direction dans le repère de la caméra) mais avec des rayons voisins. Ceci est expliqué par deux propriétés des rayons lancés par la caméra (illustré sur la Figure 2.1) :

1. Les rayons sont longs, un faible mouvement de caméra aura tendance à fortement déplacer l'extrémité du rayon. Il est alors fortement improbable que l'extrémité du rayon touche des lieux proches à des pas de temps successifs.
2. Il existe une cohérence entre les rayons, ils sont tous lancés dans une direction commune. Il est donc fortement probable qu'un lieu visible à l'image le soit de manière successive mais avec des rayons différents. La figure 2.2 montre un exemple où une partie d'un objet est visible sur des images successives et sur des pixels différents de l'image, le rayon qui détecte cette partie de l'objet est alors différent à chaque image.

Dans le cas de la méthode proposée par Hermann et al., ces deux propriétés ne sont pas présentes (cf. Figure 2.1) :

1. Les rayons sont très courts et sont dirigées vers l'intérieur des objets. De faibles mouvements ne vont provoquer que de faibles mouvements des extrémités. De plus, les déplacements relatifs entre les surfaces de contacts des objets ont

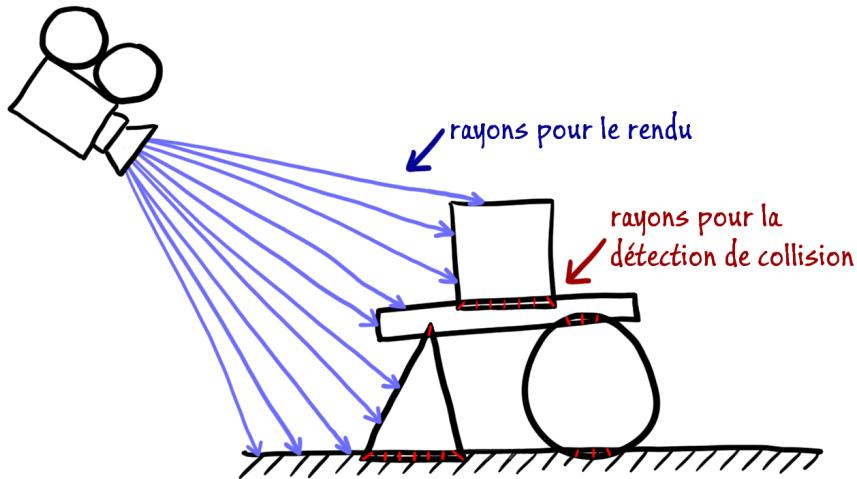


Figure 2.1 – Comparaison entre des rayons utilisés pour le rendu et des rayons utilisés pour la détection de collision. Les rayons de rendu ont tendance à être longs, un faible mouvement de caméra va causer un déplacement important des extrémités des rayons. Les rayons de détection de collisions sont très courts, un faible déplacement relatif d'un objet ne va déplacer que faiblement les extrémités des rayons.

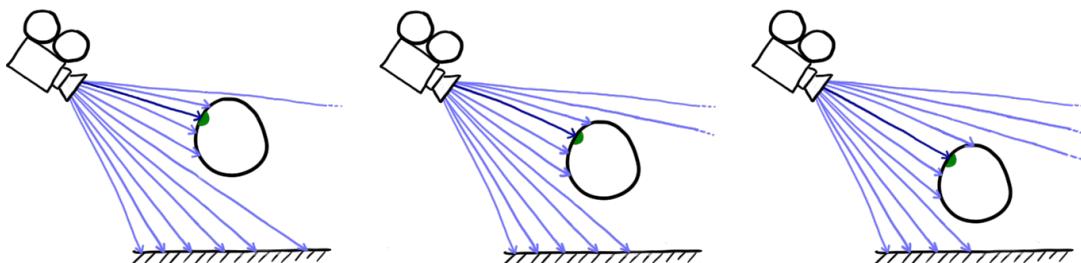


Figure 2.2 – Exemple de cohérence temporelle lors du rendu par lancer de rayons sur des pas de temps successifs. La partie verte de l'objet est touchée à chaque pas de temps par un rayon, mais celui-ci est à chaque fois différent. Pour réutiliser le résultat du lancer de rayon du pas précédent, il est alors nécessaire, pour chaque rayon, de regarder le résultat des rayons voisins.

tendance à être faibles, notamment à cause de la friction entre les objets. Il est alors fortement probable que les extrémités des rayons touchent des lieux proches à des pas de temps successifs (cf. Figure 2.3).

2. Il n'existe pas de cohérence entre les rayons. Il est donc impossible de chercher à faire des correspondances entre différents rayons lors de la détection de collision contrairement à ce qui peut être fait avec les rayons utilisés pour le rendu.

Ces observations montrent que les méthodes utilisées dans le rendu par lancer de rayon pour exploiter la cohérence temporelle ne sont pas applicables avec la détection de collision par lancer de rayon de Hermann et al.

Dans ce contexte, nous proposons un algorithme de **lancer de rayon itératif** qui exploite la cohérence temporelle (cf. Chapitre 4 Section 1). Cet algorithme utilise une méthode itérative pour localiser le nouveau point d'intersection à partir du résultat d'une ancienne intersection. Cet algorithme est valide tant que la quantité

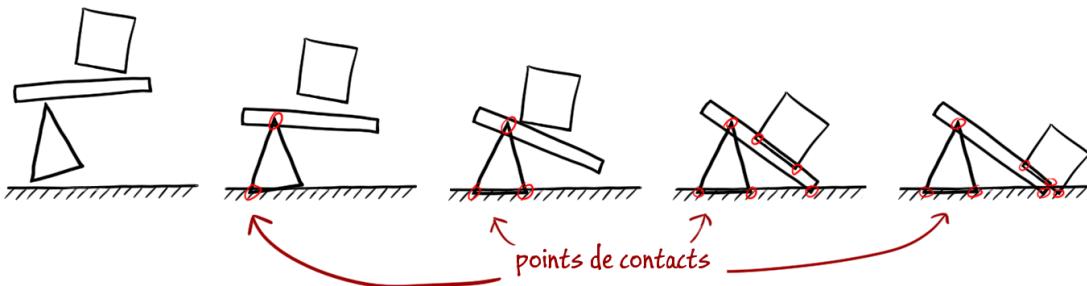


Figure 2.3 – Cohérence des points de contacts entre des pas de temps successifs. À cause de la cohérence temporelle et de la friction entre les objets, les points de contacts ne se déplacent que faiblement relativement aux surfaces.

de déplacement du rayon par rapport à l'objet cible est faible. En effet, plus la distance entre le point de départ de la recherche et le nouveau point d'intersection est élevée, plus le risque d'être piégé dans un minimum local est élevé. Néanmoins, comme nous venons de le voir, les extrémités des rayons ont tendance à avoir de faibles déplacements avec la méthode de Hermann et al.

Cette méthode va nous permettre de réduire fortement le coût du lancer de rayon pour les sommets en état de collision en exploitant la cohérence temporelle. Or, la détection de collision est appliquée sur tous les sommets, qu'ils soient en collision ou non. Il nous faut donc trouver une méthode pour exploiter la cohérence temporelle pour les sommets qui ne sont pas en collision.

3 Prédiction de collisions

La méthode proposée par Hermann et al. permet de détecter les collisions entre différents objets en lançant des rayons à partir du sommet des objets vers l'intérieur de ceux-ci. Lancer un second rayon vers l'extérieur de l'objet permettrait de réaliser une mesure de distance (ou du moins une approximation) entre les objets (cf. Figure 2.4). Dans le contexte d'exploitation de la cohérence temporelle, une mesure de distance peut être utilisée de différentes manières : éliminer des tests et localiser les surfaces candidats à des collisions.

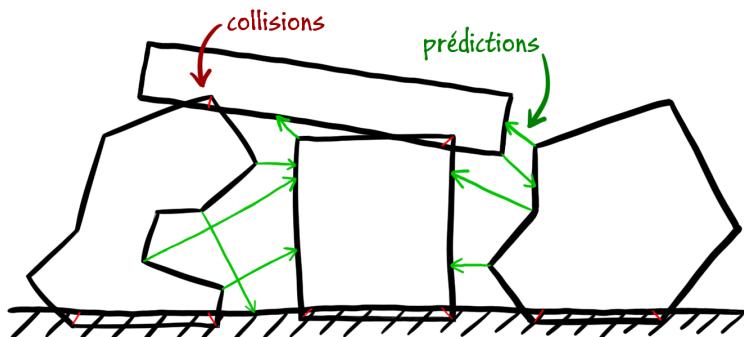


Figure 2.4 – Exemples de rayons détectant des collisions (en rouge) et des prédictions (en vert).

A Éliminer des tests

Une mesure de distance effectuée à un instant t entre deux objets A et B peut être utilisée pour chercher à éliminer des tests de collision entre ces deux objets aux instants suivants ($t + \Delta t$). Une mesure de distance globale entre deux objets indique la quantité de déplacement relatif que les deux objets doivent subir avant que ceux-ci puissent entrer en collision. Nous pouvons alors énoncer la propriété suivante :

Propriété 1. *Si $d \in \mathbb{R}$ est la distance entre deux objets A et B à un instant t et si le déplacement relatif entre A et B n'excède pas d entre toute paire points appartenant respectivement à A et B entre l'instant t et l'instant $t + \Delta t$, alors A et B ne sont pas en collision.*

Cette propriété permet d'éliminer intégralement les tests de collision entre les deux objets A et B à tout pas $t + \Delta t$ tant que le déplacement relatif entre A et B n'excède pas d . La partie gauche de la figure 2.5 montre un exemple où la mesure est appliquée sur les sommets. Cette propriété peut être étendue à des points appartenant à A ou B pour réaliser une mesure locale et obtenir la propriété suivante :

Propriété 2. *Pour tout $a \in A$ (ou $b \in B$) et d la distance entre a et B (ou b et A) à un instant t et si le déplacement relatif entre a et B (ou b et A) est inférieur à d entre l'instant t et l'instant $t + \Delta t$, alors a et B (ou b et A) ne sont pas en collision.*

Cette seconde version permet d'éliminer des tests de collisions sur des points appartenant à A ou B au lieu des objets entiers. L'intérêt de mesurer localement la mesure de distance est de permettre la modélisation d'un plus grand ensemble de mouvements relatifs libres de collisions. La partie droite de la figure 2.5 compare une mesure locale appliquée sur les sommets avec une mesure globale.

Cette seconde propriété peut être appliquée sur la méthode de Hermann et al., en particulier sur les sommets à partir desquels sont lancés des rayons. Il est donc possible de complètement éliminer certains lancers de rayon en se basant uniquement sur une mesure de distance et une mesure de déplacements relatifs.

Il est à noter que ces propriétés ne font pas intervenir les positions absolues des objets mais uniquement des déplacements relatifs. Ces propriétés sont donc valides dans n'importe quel repère, y compris des repères en mouvement.

B Localiser les surfaces candidats à des collisions

En plus de mesurer la distance, un rayon extérieur peut aussi identifier une surface proche d'un sommet donné. De par sa proximité, cette surface est un bon candidat pour être en collision dans les pas de temps suivants, de même que les voisins de cette surface.

Il serait donc possible d'utiliser une telle prédiction pour restreindre la recherche de points de contacts à un sous-ensemble de l'objet cible. Cette méthode permettrait de réduire le nombre de tests unitaires. Il serait aussi possible d'utiliser une telle prédiction dans des algorithmes à optimisation locale. La prédiction serait alors utilisée comme point de départ pour une recherche locale avec les voisins de cette surface.

Dans tous les cas, ces méthodes ne seraient valides qu'en cas de déplacements relatifs faibles entre les deux objets. En effet, la taille de la zone de recherche dépend de la

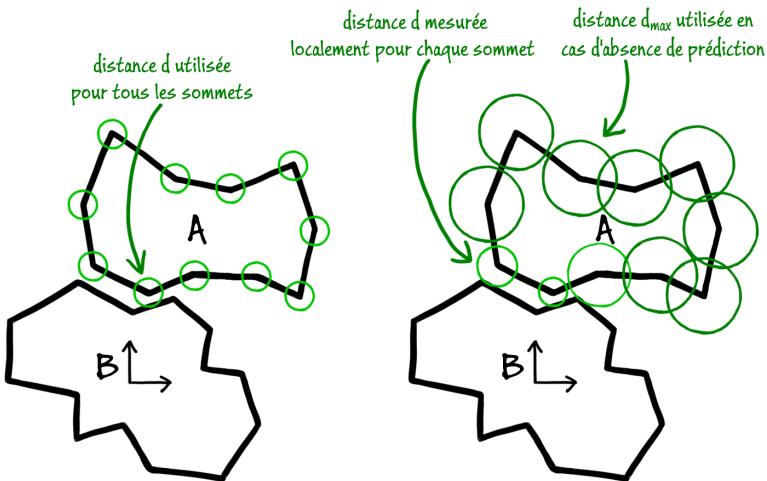


Figure 2.5 – Exemple de mesure de distance entre deux objets et de leur utilisation pour éliminer des tests de collisions. Deux objets A et B ne sont pas en collision. Les cercles verts représentent les zones de déplacement possibles des sommets de A relativement à B pour lesquels A et B ne sont pas en collision. À gauche une mesure de distance globale est utilisée, les deux objets A et B sont séparés d'une distance d et chaque sommet de A peut se déplacer d'une distance d relativement à B sans rentrer en collision. À droite une mesure locale est utilisée, chaque sommet de A peut se déplacer relativement à B d'une amplitude d correspondant à sa distance locale avec B (majorée par une distance max d_{max}). Cette seconde solution permet de modéliser un plus grand nombre de déplacements relatifs libres de collisions.

quantité de déplacement relatif entre les deux objets. Plus la quantité de déplacement relatif entre les deux objets depuis l'instant où la prédiction a été effectuée est élevée, plus il sera nécessaire d'agrandir la zone de recherche. Et en cas d'utilisation d'algorithmes à optimisation locale, le risque d'être piégé dans un minimum local grandit avec la taille de la zone de recherche.

Conclusion

Suite à ces observations, nous proposons une méthode pour réaliser des **prédictions** avec un rayon extérieur de manière efficace (cf Chapitre 3 Section 2). Grâce à plusieurs optimisations, nous montrons que le sur-coût de ce rayon extérieur est négligeable. Cette méthode est une approximation d'une mesure de distance entre objets. Néanmoins, cette mesure nous permet de réaliser deux optimisations :

1. Si la prédiction ne détecte pas le second objet ou si elle détecte le second à une distance élevée, alors nous pouvons ignorer la détection de collision pour ce sommet tant que celui-ci ne subit aucun déplacement significatif relativement au second objet.
2. Si la prédiction détecte le second objet à une distance faible, nous pouvons enregistrer la zone de prédiction sur le second objet et restreindre la détection de collision à cette zone tant qu'aucun déplacement relatif significatif avec le second objet n'est présent.

Les prédictions nous permettent donc de résoudre le problème posé précédemment :

comment exploiter la cohérence temporelle sur les sommets qui ne sont pas en collision ? En utilisant les prédictions pour évaluer la probabilité de collision de chaque sommet dans un futur proche et en mettant à jour ces prédictions avec l'algorithme de lancer de rayon itératif.

4 Utilisation de différents algorithmes de lancer de rayon

Dans l'état de l'art, les méthodes de détection avec lancer de rayon utilisent un algorithme de lancer de rayon qui est choisi en fonction des propriétés des objets et de la dynamité de la scène. Ce choix est fait de manière globale et le même algorithme de lancer de rayon est utilisé sur tous les objets. Ainsi, une scène contenant des objets rigides pourra utiliser des structures accélératrices avec une construction hors-ligne tandis qu'une scène contenant des objets déformables utilisera des structures accélératrices permettant une mise à jour incrémentale.

Dans un environnement générique qui contient aussi bien des objets rigides que des objets déformables, nous proposons de choisir dynamiquement les algorithmes de lancer de rayon en fonction des propriétés des objets cibles (cf. Chapitre 3 Section 1).

De plus, nous proposons un algorithme de lancer de rayon itératif. Cet algorithme est valide tant que la quantité de déplacement entre l'origine du rayon et la cible est faible. Nous avons donc besoin d'un critère pour mesurer la quantité de déplacement entre deux objets et ainsi décider quand utiliser l'algorithme de lancer de rayon itératif.

Pour réaliser cette mesure nous proposons deux méthodes pour mesurer la quantité de déplacement relatif entre deux objets. La première méthode mesure la quantité de déplacement globale entre deux objets rigides en utilisant uniquement les positions globales des deux objets (cf. Chapitre 4 Section 2). Nous proposons ensuite une mesure plus fine qui mesure la quantité de déplacement entre les deux objets de manière locale. Cette mesure peut être appliquée aux objets déformables et prend en compte les déformations internes (cf. Chapitre 4 Section 4).

Notre critère final de choix d'algorithme de lancer de rayon permet de choisir individuellement, pour chaque rayon, l'algorithme de lancer de rayon à utiliser en fonction de la nature de l'objet cible (rigide ou déformable) et de la quantité de déplacement relatif que les objets ont subi.

5 Base algorithmique commune sur la détection de collision surfacique/volumique

La méthode proposée par Hermann et al. peut être utilisée aussi bien sur des objets rigides que sur des objets déformables. Cependant, elle requiert que les objets soient des surfaces closes avec un côté intérieur et un côté extérieur (cette surface modélise un volume). Or, pour représenter des objets extrêmement fins, certains modèles de représentation surfaciques définissent une surface pour laquelle chaque côté est extérieur. Ce type de surfaces est couramment utilisé pour représenter des vêtements et peut être utilisé pour représenter d'autres types d'objets fins tels que des feuilles de

papier ou de la tôle métallique. Deux points posent problème pour utiliser la méthode de Hermann et al. avec ce type de représentation :

1. Les rayons sont lancés à partir de la surface vers l'intérieur des objets, ce qui n'est pas possible si la surface de l'objet ne possède pas un côté intérieur.
2. Seuls les rayons touchant l'intérieur du second objet sont conservés, ce qui n'est également pas possible si la surface du second objet ne possède pas un côté intérieur.

Pour prendre en compte les modèles surfaciques, nous proposons d'étendre la méthode de Hermann et al. en ajoutant trois tests unitaires : volumique → surfacique, surfacique → volumique et surfacique → surfacique (cf. Chapitre 3 Section 4). Ces ajouts permettent la possibilité de tester les collisions entre des objets surfacique et de tester les collisions entre un objet surfacique et un objet volumique.

6 Exploitation des processeurs graphiques et pipeline

Les processeurs graphiques (GPU) ont aujourd'hui une puissance brute bien plus élevée que les processeurs génériques (CPU). De plus, les architectures actuelles permettent de programmer librement les fonctions exécutées sur GPU et il est possible d'exécuter des instructions qui ne sont pas destinées à réaliser un rendu graphique. Il est donc logique que de nombreuses méthodes tentent aujourd'hui de tirer parti de cette puissance supplémentaire. Les GPUs ayant été conçus pour exécuter des commandes de rendu, ils semblent parfaitement adaptés pour exécuter des algorithmes de détections de collision basés images. Il est à noter que même si les GPUs ont été conçus pour exécuter des algorithmes de rendu basés rastérisation, de nombreux travaux de recherche permettent d'exécuter des algorithmes de lancer de rayon de manière efficace sur GPU.

Étant données les contraintes liées aux GPUs (cf. Chapitre 1), il paraît raisonnable d'implémenter la méthode de Hermann et al. sur GPU. Le lancer de rayon peut être implémenté sur GPU à l'aide de méthodes existantes et chaque rayon lancé est complètement indépendant ce qui permet de paralléliser le calcul de chaque rayon. Nous avons réalisé cette implémentation relativement triviale et observé une amélioration significative des performances.

Notre problématique dans cette thèse a été de s'assurer que toutes nos contributions maintiennent ce parallélisme et cette facilité d'implémentation sur GPU. S'assurer que l'intégralité de nos méthodes respecte les contraintes liées aux GPUs lors de leur conception permet de pouvoir ensuite tirer parti de la puissance de calcul des GPUs sans nécessiter un effort supplémentaire.

Pour s'assurer que toutes nos contributions respectent les contraintes liées aux GPUs nous proposons de modéliser l'ensemble de nos calculs dans un *pipeline* (cf. Chapitre 3 Section 1). Ce *pipeline* s'insère dans le *pipeline* général de détection de collision en prenant la place de la narrow-phase et exact-phase. Ce *pipeline* permet de modéliser l'agencement des calculs en prenant en compte les contraintes liées aux calculs sur GPUs.

7 Synthèse

Suite à cette étude, nous proposons les contributions suivantes :

Un algorithme de lancer de rayon itératif : Cet algorithme de lancer de rayon permet d'exploiter la cohérence temporelle. Pour être utilisé sur un rayon, il nécessite de connaître le résultat du pas précédent.

Une méthode pour prédire les collisions : Cette méthode permet de savoir si un sommet donné risque de rentrer en collision dans un futur proche. Le cas échéant, cette méthode fournit la localisation de la zone qui risque d'entrer en collision.

Un algorithme de détection de collision pour les objets surfaciques : Cet algorithme permet d'exécuter la détection de collision avec des objets fins tels que des tissus.

Combiner des algorithmes de lancer de rayon : Nous proposons d'utiliser plusieurs algorithmes de lancer de rayon (dont l'algorithme itératif que nous proposons) pour réaliser la détection de collision. Le choix de l'algorithme utilisé est réalisé dynamiquement lors de l'exécution.

Exploiter les GPUs : Nous proposons d'exécuter tous ces calculs sur GPUs pour maximiser les performances.

Pour Formaliser l'agencement de toutes ces contributions, nous proposons un **pipeline de détection de collision par lancer de rayon**. Le rôle de ce pipeline est d'intégrer toutes ces contributions dans une séquence de calculs adaptée aux GPUs.

Les deux prochains chapitres vont détailler ces contributions. Le chapitre 3 présente notre pipeline de détection de collision ainsi que les éléments majeurs permettant de le faire fonctionner. Le chapitre 4 se focalise sur les optimisations liées à la cohérence temporelle.

Pipeline de détection de collision

3

Dans ce chapitre, nous présentons un nouveau *pipeline* [Lehericey et al., 2015b] permettant de réaliser la détection de collision par lancer de rayon dans la *narrow-phase*. Nous commençons par présenter notre *pipeline* qui permet de réaliser la méthode de détection de collision de Hermann et al. tout en permettant d'utiliser plusieurs algorithmes de lancer de rayon différents (cf. Section 1).

Nous présentons ensuite plusieurs contributions visant à généraliser les fonctionnalités de notre *pipeline* et à le rendre plus fiable [Lehericey et al., 2013b, Lehericey et al., 2015c] : Nous présentons une méthode permettant de réaliser une mesure de distance inter-objets lorsque les objets ne sont pas en collision (cf. Section 2). Nous proposons de réaliser un post-traitement sur les points de contacts pour améliorer leur fiabilité (cf. Section 3). Nous généralisons la méthode de détection de collision de Hermann et al. afin de réaliser la détection de collision sur des tissus (cf. Section 4).

1 Mise en place d'un pipeline générique

Cette partie s'intéresse à des problématiques propres aux GPUs et pour cela nous allons utiliser la modélisation et la nomenclature utilisées dans OpenCL¹. Cette section presuppose un minimum de connaissances sur les architectures et les usages des GPUs de la part du lecteur. Dans le cas contraire, nous proposons au lecteur une introduction au modèle d'OpenCL [Tompson and Schlachter, 2012] ainsi que des cas d'utilisation de la programmation générique sur GPU présentés dans les livres *GPU Gems* [Fernando, 2004, Pharr and Fernando, 2005, Nguyen, 2007]. Dans cette section nous allons utiliser les éléments de nomenclature suivants :

work-item : Un *work-item* est l'équivalent d'un *thread* sur CPU. La différence majeure avec les *threads* est l'impossibilité de lancer des *work-items* indépendants. Les *work-items* sont toujours lancés en très grand nombre et exécutent tous en parallèle la même fonction sur des coeurs différents.

work-group : Les *work-item* sont regroupés dans des groupes appelés *work-group*. Tous les *work-items* appartenant au même *work-group* partagent des ressources communes (tels que la mémoire locale ou le compteur ordinal).

kernel : Un *kernel* est une fonction implémentée pour GPU. Un *kernel* est une fonction qui sera exécutée à l'identique par tous les *work-item* (même fonction et mêmes paramètres). La seule variable entre chaque *work-item* est le numéro de l'appel (il permet à chaque *work-item* de réaliser des tâches différentes en servant, par exemple, d'indice de lecture dans un tableau).

1. <https://www.khronos.org/opencl/>

Nous proposons de modéliser l'ensemble des étapes de notre méthode dans un *pipeline*. La contrainte majeure de ce *pipeline* est d'obtenir des performances maximales sur GPU tout en restant assez flexible pour permettre d'intégrer l'ensemble de nos contributions. Les GPU sont des processeurs hautement parallèles, les cartes actuelles comportent plusieurs milliers de coeurs. Pour tirer parti de ce parallélisme, il est nécessaire de décomposer notre méthode en un ensemble de phases dans lesquelles chaque calcul est parallèle. De plus, les coeurs de GPU ne sont pas entièrement indépendants, ils sont organisés en *work-group*. Les coeurs au sein d'un *work-group* ne partagent qu'un seul compteur ordinal, cela signifie que les coeurs avancent de manière synchronisée dans le flot d'instructions. Dans ce contexte, les programmes ayant des embranchements divergents à l'exécution sont particulièrement inefficaces car chaque branche est exécutée séquentiellement dans un *work-group*. Pour éviter cette perte, il est nécessaire de minimiser le nombre de divergences dans un programme destiné à être exécuté sur GPU. Nous réduisons la divergence en décomposant les calculs en des étapes non-divergentes.

Le but final du *pipeline* est d'exécuter le lancer de rayon sur les sommets des objets. Plusieurs algorithmes de lancer de rayon sont disponibles et différents critères vont permettre de choisir s'il est nécessaire de lancer un rayon et le cas échéant quel algorithme de lancer de rayon doit être utilisé pour chaque sommet. Les critères comprennent les éléments suivants :

- Un moyen de classer les objets en fonction de leur nature (rigide ou déformable).
- Un moyen de classer les paires d'objets en fonction de la quantité de déplacements relatif accumulés entre les objets (que ce soit globalement ou localement).
- L'introduction de critères de *culling* permettant de réduire le nombre de tests.

Pour combiner plusieurs algorithmes de lancer de rayon sur GPU, une méthode naïve consisterait à implémenter tous les algorithmes dans une seule fonction avec des instructions d'embranchements pour choisir quel algorithme utiliser pour chaque rayon. Cette implémentation naïve causerait une forte divergence à l'exécution pour deux raisons. Premièrement, tous les algorithmes de lancer de rayon seraient exécutés par la même fonction, ceci ajouterait une divergence à l'exécution de chaque tracé de rayon. Chaque *work-group* qui exécute plus d'un algorithme de lancer de rayon les exécuteraient séquentiellement. Secondelement, dans un contexte où il est possible d'élaguer certains lancers de rayon avec des critères de *culling*, cette implémentation créerait des *work-items* qui s'achèvent immédiatement. Ces *work-items* seraient ensuite en attente pendant que les autres *work-items* appartenant au même *work-group* terminent, éliminant le gain obtenu par le *culling*. Cette observation justifie la création d'un *pipeline* permettant de gérer plus efficacement la divergence à l'exécution.

En plus du *pipeline*, nous proposons de prédire la charge de travail lors de l'exécution du *pipeline* pour éviter des synchronisations superflues entre le CPU et le GPU. Éviter des synchronisations permet de retirer des coûts fixes et d'améliorer les performances.

La prochaine section (Section 1.1) détaille les éléments qui doivent être présents dans le *pipeline*. Nous présentons ensuite le *pipeline* (cf. Section 1.2). Nous expliquerons ensuite comment le fait de prédire la charge peut conduire à améliorer les performances (cf. Section 1.3). Puis nous évaluerons les performances de notre *pipeline* et de l'utilisation des prédictions (cf. Section 1.4).

1.1 Éléments du pipeline

Cette partie présente les éléments majeurs qui doivent être présents dans notre *pipeline* : les algorithmes de lancer de rayon, la mesure de déplacement relatif et le *culling*.

1.1.1 Algorithmes de lancer de rayon

Dans ce *pipeline*, nous allons utiliser trois algorithmes de lancer de rayon. Chaque algorithme de lancer de rayon a un ou des cas d'utilisations optimaux mais aucun n'est optimal dans tous les cas d'utilisation. C'est pour cela que nous utilisons plusieurs algorithmes de lancer de rayon pour utiliser en permanence l'algorithme de lancer de rayon le plus optimal à chaque situation. Les deux premiers algorithmes que nous utilisons sont qualifiés de complets, ils peuvent être utilisés sans nécessiter d'autres informations que la géométrie des objets ciblés. Le troisième algorithme est qualifié d'itératif, il nécessite des informations supplémentaires (appelées données temporelles).

Le premier algorithme est un **algorithme de lancer de rayon avec parcours sans pile d'une hiérarchie de volumes englobants (BVH)**. Cet algorithme utilise une hiérarchie de volumes englobants [Wald et al., 2007]. Le parcours est réalisé sans pile [Popov et al., 2007] pour permettre un parallélisme massif sur GPU. Le coût de construction et de maintenance des BVH est élevé, ils sont donc utilisés pour les objets rigides pour lesquels la construction des BVH peut être réalisée hors-ligne.

Le deuxième algorithme est un **algorithme basique de lancer de rayon**. Cet algorithme n'utilise pas de structures accélératrices, il est donc parfaitement adapté aux objets déformables. Pour obtenir des performances maximales, cet algorithme est massivement parallélisé sur GPU.

Le troisième algorithme est l'**algorithme de lancer de rayon itératif** que nous présenterons dans le chapitre 4. Celui-ci est adapté aussi bien pour les objets rigides que pour les objets déformables, cependant, cet algorithme ne peut fonctionner seul car il nécessite de connaître le résultat de l'intersection précédente du même rayon (que nous appellerons données temporelles dans la suite de ce document). L'algorithme itératif ne peut pas fonctionner seul et nécessite l'utilisation d'un algorithme complet pour calculer la première intersection. L'algorithme itératif peut ensuite être utilisé pour remplacer l'algorithme complet tant que le déplacement relatif entre le rayon et la cible est faible. Lorsque des données temporelles sont disponibles, l'utilisation de l'algorithme itératif devrait être préféré car son coût est plus faible que les algorithmes complets (cf. les évaluations réalisées dans le chapitre 4).

1.1.2 Mesure de déplacement relatifs

L'algorithme de lancer de rayon itératif est valide uniquement en cas de faibles déplacements relatifs entre le rayon et la cible. La source du rayon et la cible appartiennent respectivement aux deux objets de la paire testée. Nous proposons de mesurer le déplacement relatif entre les deux objets de la paire pour décider si l'algorithme itératif peut être utilisé. Pour réaliser cette mesure, nous avons mis au point deux méthodes dont le choix dépend de la présence ou non d'objets déformables.

Le fonctionnement exact de ces mesures est présenté dans le chapitre 4. La première méthode est destinée aux paires d'objets rigides, la mesure est réalisée de manière globale sur les objets. Le résultat de la classification est donc global à la paire d'objets, le résultat indique si toute la paire est en déplacement relatif faible ou si toute la paire est en déplacement relatif important. La seconde méthode est destinée aux paires contenant au moins un objet déformable, la mesure est réalisée de manière locale car il est nécessaire de prendre en compte les déformations locales des objets. Avec cette seconde mesure, la classification est fournie pour chaque sommet et des sommets appartenant à une même paire d'objets peuvent avoir des classifications différentes.

1.1.3 Culling

Avant d'exécuter le lancer de rayon, il est possible de réaliser un simple élagage pour réduire le nombre de tests. Nous proposons d'utiliser deux critères de *culling*. Le premier critère proposé par Hermann et al. teste si l'origine du rayon est à l'extérieur du volume englobant de la cible, si c'est le cas alors nous pouvons éliminer ce rayon car il ne peut pas être à l'intérieur de la cible. Le second critère est réservé aux sommets ayant subi de faibles déplacements, dans ce cas nous pouvons tester la présence de données temporelles. Si aucune donnée temporelle n'est présente pour un sommet, alors nous pouvons éliminer ce rayon. L'absence de données temporelles indique que, au pas de temps précédent, aucune collision (ou prédition de collision) n'a été détectée et que, étant donné un faible déplacement, il est très fortement probable que le sommet ne soit pas en collision.

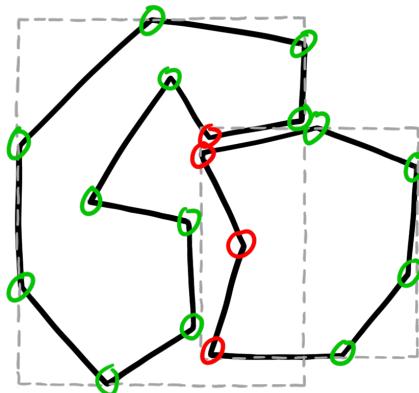


Figure 3.1 – Exemple de *culling* proposé par Hermann et al., les sommets entourés en vert n'ont pas besoin d'être testés car ils sont à l'extérieur du volume englobant du second objet (en gris).

1.2 Organisation du pipeline

Pour réaliser la détection de collision, notre *narrow-phase* doit donc réaliser plusieurs tâches :

Appliquer une mesure de déplacements relatifs entre des paires d'objets pour décider si l'algorithme de lancer de rayon itératif peut être utilisé.

Réaliser un *culling* sur les sommets pour éliminer des tests inutiles.

Exécuter le lancer de rayon sur les sommets restants, l'algorithme utilisé dépendant de la nature des objets et du résultat de la classification des mesures de déplacements.

On peut remarquer que les tâches sont réalisées à des granularités différentes. La mesure de déplacement relatif peut être appliquée globalement sur les paires d'objets rigides tandis que la mesure de déplacement relatif est appliquée sur les sommets en cas de présence d'objets déformables. Le *culling* est appliqué sur les sommets pour les deux critères (test de volume englobant ou test de présence de données temporelles). Le lancer de rayon est exécuté sur les sommets restants. Ces tâches montrent trois niveaux de granularité : par-paire, par-sommet et par-rayon. Nous proposons en conséquence un pipeline qui modélise les calculs à ces trois niveaux de granularité successifs. La figure 3.2 donne une vue d'ensemble du *pipeline*.

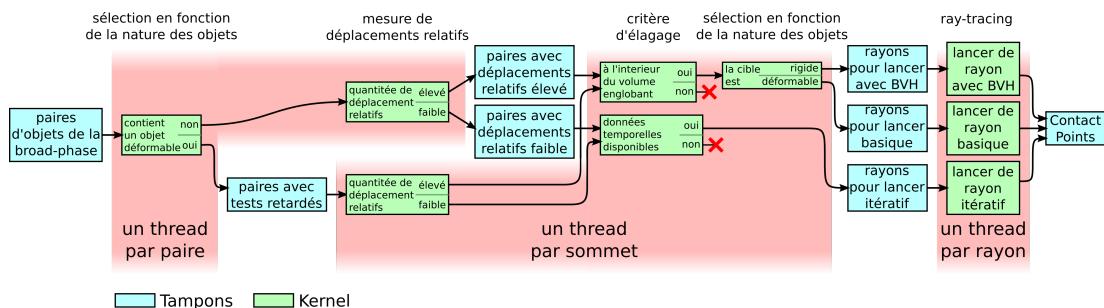


Figure 3.2 – Organisation en *pipeline* de la *narrow-phase*. La *narrow-phase* prend en entrée la liste des paires d'objets détectées par la *broad-phase*. La première étape est appliquée sur les paires d'objets et les sépare en trois catégories : paires rigides avec un faible déplacement relatif, paires rigides avec un haut déplacement relatif et paires contenant un objet déformable. La seconde étape est appliquée sur les sommets des objets des paires, cette étape sélectionne quel algorithme de lancer de rayon utiliser en fonction de la nature des objets et du résultat de leur mesure de déplacement relatif. La dernière étape est appliquée sur les rayons et exécute les algorithmes de lancer de rayon.

Des mémoires tampons sont utilisées à la fin de chaque étape pour stocker les résultats intermédiaires avant de les redistribuer aux étapes suivantes. Celles-ci fonctionnent à l'aide d'ajouts atomiques. Les *kernels* qui précèdent un tampon les remplissent. Chaque entrée du tampon va générer un pour plusieurs *work-item* dans les *kernels* suivants. Le but principal des tampons est de garantir une entrée dense dans les *kernels* suivants pour éviter d'avoir des coeurs en attente. Une entrée dense pour un *kernel* est une entrée qui occupe chaque cœur du GPU, si une entrée n'est pas dense alors le GPU aura des coeurs en attente pendant l'exécution du *kernel*. Les trois sections suivantes détaillent chaque étape.

1.2.1 Étape par-paire

L'étape par-paire prend en entrée la liste des paires d'objets fournie par la *broad-phase*. Dans cette étape chaque *work-item* correspond à une paire.

Premièrement nous décomposons les paires en deux catégories : celles qui contiennent uniquement des objets rigides et celles qui contiennent au moins un objet

déformable. Cette séparation est réalisée car la mesure de déplacement n'est pas appliquée au même niveau de granularité lorsque des objets déformables sont présents.

Pour les paires contenant uniquement des objets rigides, nous appliquons à cette étape la mesure de déplacement relatifs pour objets rigides (cf. Chapitre 4 Section 2) pour séparer les paires possédant un haut niveau de déplacement relatif (qui vont utiliser un algorithme de lancer de rayon complet) des paires ayant un faible déplacement relatif (qui vont utiliser l'algorithme de lancer de rayon itératif). La liste des paires appartenant à chaque catégorie est stockée dans un tampon séparé.

La liste des paires contenant au moins un objet déformable est directement stockée dans un tampon, il n'est pas possible d'appliquer la mesure de déplacement relatif car elle soit être mesurée localement.

1.2.2 Étape par-sommet

L'étape par-sommet prend en entrée la liste des paires classées par l'étape précédente. Dans cette étape un *work-item* est exécuté pour chaque sommet de chaque objet de chaque paire.

Pour les paires contenant au moins un objet déformable nous pouvons enfin appliquer la mesure de déplacement relatif pour objets déformables (cf. Chapitre 4 Section 4). Cette mesure va séparer les sommets classifiés avec un faible déplacement relatif des sommets classés avec un déplacement relatif élevé.

Nous pouvons ensuite appliquer les critères d'élagage sur les sommets pour réduire le nombre de tests unitaires. Le critère est différent pour les sommets utilisant un algorithme de lancer de rayon complet et pour les sommets utilisant l'algorithme de lancer de rayon itératif. Pour les sommets utilisant un algorithme complet on teste si les sommets sont à l'intérieur du volume englobant du second objet. Si ce n'est pas le cas, nous pouvons retirer ce sommet en toute sécurité car il est impossible qu'il soit en collision. Pour les sommets utilisant l'algorithme itératif, on teste la présence de données temporelles. Si aucune donnée temporelle n'est disponible pour un sommet, celui-ci peut être retiré. L'absence de données temporelles indique qu'aucune collision n'est à mettre à jour et qu'aucune prédiction indique la possibilité d'une collision prochaine.

Après avoir appliqué les critères d'élagage, chaque sommet restant génère un rayon qui sera lancé sur l'autre objet de la paire. Les paramètres nécessaires à réaliser le lancer de rayon sont stockés dans trois tampons séparés. Chacun de ces tampons représente un algorithme de lancer de rayon différent.

1.2.3 Étape par-rayon

L'étape par-rayon prend en entrée les liste de rayons calculés par l'étape précédente. Dans cette étape un *work-item* est exécuté pour chaque rayon. Les algorithmes utilisés sont ceux présentés dans la section 1.1.1.

Chaque algorithme de lancer de rayon est implémenté dans des *kernels* séparés pour éviter des divergences de branches à l'exécution. Les rayons de prédiction de collision sont aussi calculés à cette étape (cf. Section 2). Chaque algorithme de lancer de rayon produit les points de contacts en cas de collision, ceux-ci seront post-traités (cf. Section 3) et ensuite utilisés par la réponse physique pour séparer les objets.

1.3 Prédiction de remplissage des tampons

Tout au long de l'exécution du *pipeline* nous avons besoin de connaître le nombre d'éléments contenus dans chaque mémoire tampon (nous appellerons ce nombre remplissage par la suite). Pour chaque tampon, nous avons besoin de savoir a priori le remplissage pour s'assurer que le tampon est assez grand et a posteriori pour savoir combien de *work-items* lancer dans les *kernels* suivants. Le problème est que le remplissage n'est pas connu a priori et qu'il est stocké dans la mémoire du GPU a posteriori. Pour éviter des calculs supplémentaires ou des synchronisations entre le CPU et le GPU qui limiteraient les performances nous proposons d'utiliser des prédictions.

Dans cette section nous commençons par expliquer notre méthode de prédiction puis nous l'appliquons au dimensionnement des tampons et à la prédiction de la charge de travail.

1.3.1 Prédiction générique

Posons N_t le nombre d'éléments contenus dans un tampon au pas de temps courant, N_{t-1} le nombre d'éléments au pas de temps précédent et $p(N_t)$ la valeur prédite de N_t .

Nous avons accès à N_{t-1} , c'est-à-dire le remplissage exact qui aurait du être utilisé au pas de temps précédent. Cette valeur peut être lue de manière asynchrone avec un coût très faible. Pour prédire l'évolution de N , on peut s'appuyer sur l'évolution du nombre de paires produites par la *broad-phase* qui donne une idée de l'évolution générale de la charge de travail. Posons $nbPaires_t$ (et $nbPaires_{t-1}$) le nombre de paires détectées par la *broad-phase* à l'instant t (et $t - 1$).

L'équation 3.1 donne une prédiction de la valeur $p(N_t)$. Cette équation prend en entrée la valeur réelle du remplissage de tampon à l'instant précédent N_{t-1} . Cette valeur est pondérée par l'évolution du nombre de paires qui donne une indication sur l'évolution de la charge de travail globale. Le tout est multiplié par un facteur de confiance d'au moins 1 pour prendre en compte la variabilité.

$$p(N_t) = N_{t-1} \times \frac{nbPaires_t}{nbPaires_{t-1}} \times \text{confiance} \quad (3.1)$$

À la fin du pas de temps, la valeur de N_t peut être lue de manière asynchrone et comparée à la prédiction $p(N_t)$ pour vérifier si la prédiction était correcte (la prédiction est correcte si $P(N_t) \geq N_t$). Si la prédiction était trop faible, il existe deux stratégies pour corriger cette erreur. La plus simple est d'ignorer les erreurs et de continuer la simulation. Cette solution va provoquer des erreurs dans la simulation mais sera appropriée pour les applications temps-réels où les contraintes de temps de calculs sont fortes. Ces erreurs seront des faux-négatifs dans la détection de collision mais ceux-ci ne provoqueront pas d'erreurs graves dans la simulation car ils seront détectés et corrigés dans les pas de temps suivants. Ces erreurs pourront néanmoins être visibles de l'utilisateur notamment à cause du changement de comportement de la simulation physique (par exemple plusieurs objets identiques tombant de la même manière pourront rebondir de manière différents si certains points de contacts sont manquants sur certains objets). La seconde solution consiste à revenir en arrière et

ré-exécuter la simulation avec les valeurs correctes. Cette solution permettra d'avoir une simulation correcte et sera préférée pour les simulations hors-ligne pour lesquelles il n'y a pas de contraintes de régularité des temps de calculs.

1.3.2 Prédiction de la taille des tampons

Avant d'exécuter un calcul qui va remplir un tampon nous avons besoin de savoir a priori le nombre d'éléments qui vont lui être insérés pour vérifier que le tampon est assez grand pour contenir tous les éléments. Pour obtenir ce nombre il serait possible d'exécuter les calculs deux fois, une première fois pour obtenir le nombre d'éléments et une seconde fois pour remplir le tampon, cette méthode serait sûre mais très coûteuse.

On peut utiliser des prédictions pour éviter la première exécution. Nous proposons d'appliquer notre équation de prédiction pour évaluer la taille nécessaire du tampon. Nous pouvons utiliser une grande valeur de confiance dans ce type de prédiction car la valeur prédite n'est pas utilisée pour exécuter des calculs mais uniquement pour allouer de la mémoire.

1.3.3 Prédiction de la charge de travail

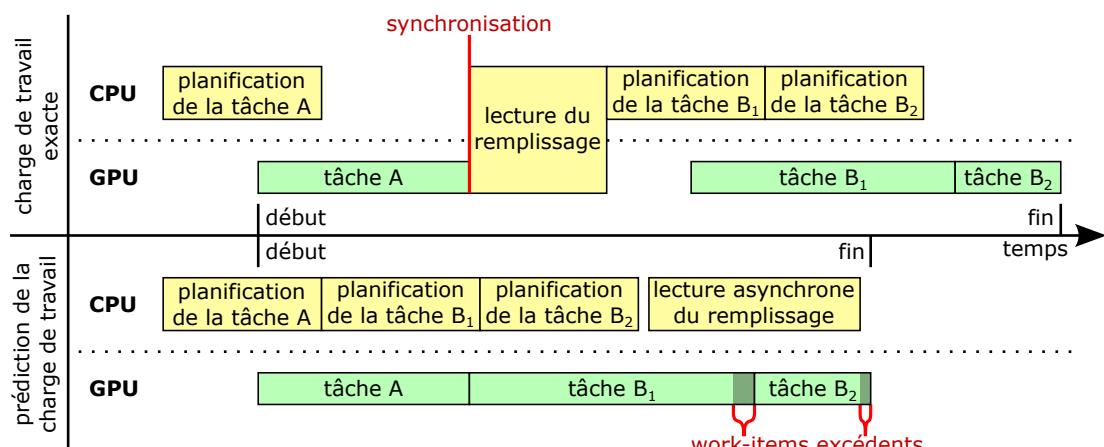


Figure 3.3 – Comparaison de la chronologie des évènements entre les deux stratégies de gestion de charge de travail. Dans cet exemple, une tâche *A* remplit une mémoire tampon et deux tâches *B*₁ et *B*₂ consomment ce tampon. Dans la partie supérieure la charge de travail exacte est utilisée pour les tâches *B*₁ et *B*₂ mais le GPU est en attente pendant que le CPU récupère le remplissage du tampon et planifie l'exécution des tâches suivantes. Dans la partie inférieure, la charge de travail est prédictive, l'exécution des tâches *B*₁ et *B*₂ est alors instantanée mais des *work-items* excédants sont lancés (c-à-d des *work-items* qui ne réalisent aucun calcul).

La planification de l'exécution de chaque *kernel* sur GPU est réalisée par le CPU et celui-ci a besoin de connaître la charge de travail du *kernel*, c'est-à-dire le nombre de *work-items* qui doivent être exécutés. Lors de la planification de l'exécution d'un *kernel* qui prend en entrée un tampon nous avons besoin de connaître le remplissage de celui-ci pour savoir combien de *work-items* lancer. Cette valeur de remplissage est stockée dans la mémoire du GPU et elle doit être connue sur le CPU pour planifier les

exécutions des calculs utilisant en entrée le tampon avec un nombre adéquat de *work-items*. Si des valeurs de remplissage trop faibles sont utilisées pour planifier l'exécution du *pipeline*, les derniers éléments des tampons ne seront pas utilisés. Avec notre méthode de détection de collision, cela va provoquer des erreurs de détection sous la forme de faux-négatifs. Pour résoudre ce problème de dépendance, nous avons deux solutions :

La première solution utilise la valeur exacte de remplissage. Dans cette solution, nous lisons le remplissage du tampon stocké sur le GPU à partir du CPU. Il est ensuite possible de planifier l'exécution des calculs suivants avec le nombre exact de *work-items*. La partie supérieure de la figure 3.3 donne un exemple de la chronologie des événements entre un CPU et un GPU dans une telle situation. L'inconvénient majeur de cette méthode est la synchronisation entre le CPU et le GPU. Le GPU est en attente pendant que le CPU est en train de planifier l'exécution des calculs suivants.

La seconde solution consiste à utiliser une prédiction de la valeur de remplissage à la place de la valeur réelle. Pour prédire le remplissage d'un tampon, nous pouvons appliquer notre équation de prédiction avec une valeur élevée de confiance pour minimiser l'introduction de faux-négatifs dans notre méthode. Si trop de *work-items* sont exécutés, ceux-ci s'achèveront immédiatement après avoir testé si leur indice dépasse le remplissage du tampon. L'inconvénient de cette méthode est que de larges prédictions vont entraîner le lancement de nombreux *work-items* ne réalisant aucune tâche sur le GPU, mais ces *work-items* seront tous groupés dans les derniers *work-groups* et donc ne provoqueront pas de divergences critiques. L'avantage de cette solution est qu'elle ne nécessite de pas synchronisation entre le CPU et le GPU comme l'illustre la partie inférieure de la figure 3.3. Dans cet exemple, le temps d'exécution final est plus faible avec des prédictions car le coût supplémentaire causé par les *work-items* excédants est inférieur au coût d'une synchronisation entre le CPU et le GPU. Nous avons utilisé cette solution car son plus faible coût est plus adaptée aux contraintes de temps-réel.

1.4 Évaluation des performances

Dans cette partie nous présentons notre environnement expérimental et nos résultats. Nous présentons, en premier, nos conditions expérimentales puis nous mesurons les gains obtenus avec l'utilisation de notre *pipeline* et des prédictions des charges de travail et enfin nous mesurons la quantité de faux-négatifs introduits par la prédiction des charges de travail.

1.4.1 Conditions expérimentales

Pour mesurer le gain en performance de notre *pipeline* pour la *narrow-phase*, nous avons mesuré le temps de calcul de la détection de collision sur GPU de simulations durant 10 secondes exécutées à 60 Hz. Deux scènes ont été testés : une scène "objets rigides" et une scène "objets rigides et déformables".

Dans la scène "objets rigides" (cf. Figure 3.5), 1.000 objets rigides non-convexes tombent sur un sol irrégulier. Quatre objets différents présentés dans la figure 3.4 sont utilisés en quantités égales. Jusqu'à 10.000 paires d'objets sont testées dans la *narrow-phase*.

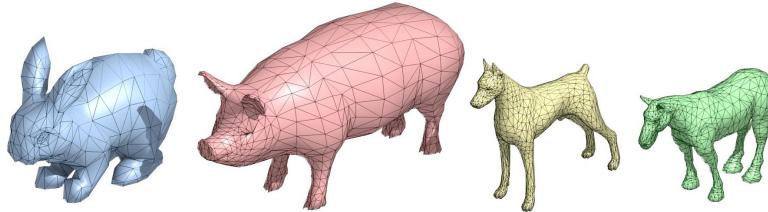


Figure 3.4 – Quatre formes d'objets rigides utilisés dans les simulations. Ces objets sont respectivement composés de 902, 2.130, 3.264 et 3.458 triangles.

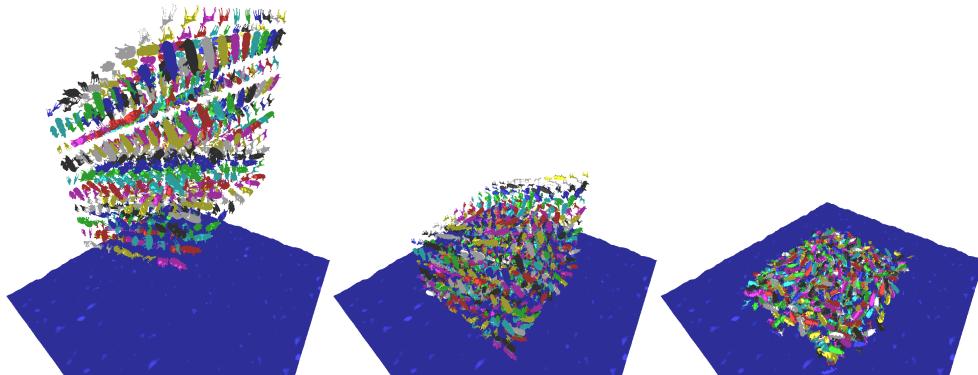


Figure 3.5 – Scène "objets rigides", 1.000 objets rigides non-convexes tombent sur un sol irrégulier

Dans la scène "objets rigides et déformables" (cf. Figure 3.6), 216 objets rigides non-convexes tombent sur un sol irrégulier et 36 tissus déformables tombent aux dessus des objets rigides. Les objets rigides sont les mêmes que dans la scène précédente. Les tissus déformables sont des draps de forme carrée et sont composés de 8.192 triangles chacun. Jusqu'à 860 paires d'objets rigides et 600 paires rigides/déformables sont testées dans la *narrow-phase*.

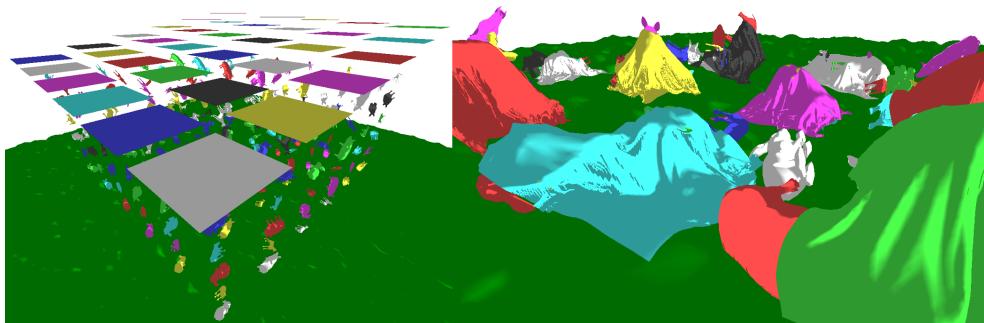


Figure 3.6 – Scène "objets rigides et déformables", 216 objets rigides non-convexes tombent sur un sol irrégulier et 36 tissus déformables tombent aux dessus des objets rigides.

Nos scènes expérimentales ont été développées avec Bullet Physics 3.x² sur GPU. Notre *narrow-phase* a été implémentée sur GPU avec OpenCL. Tous les tests ont été exécutés sur une carte AMD HD 7990 (1/2 GPU utilisé, 2048 coeurs, 6 Go de mémoire).

2. <http://bulletphysics.org/>

1.4.2 Performances du pipeline

Pour mesurer l'accélération obtenue avec notre *pipeline* pour la *narrow-phase*, nous avons comparé le temps de calcul sur GPU de la *narrow-phase* en utilisant notre pipeline et sans l'utiliser.

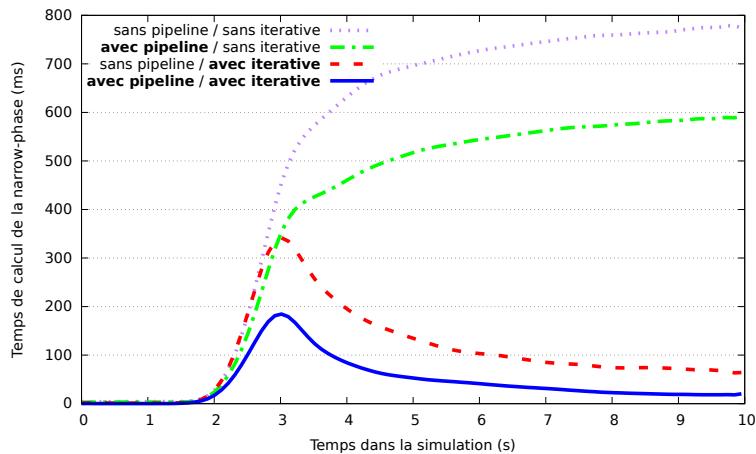


Figure 3.7 – Comparaison des performances sur GPU de la détection de collision avec et sans notre pipeline dans la scène "objets rigides". Avec uniquement l'algorithme de lancer de rayon complet notre *pipeline* permet de diviser le temps de calcul par 1,31 en moyenne. Avec la combinaison de l'algorithme complet et de l'algorithme itératif de lancer de rayon notre *pipeline* divise en moyenne par 2,73 le temps de calcul.

Dans la scène "objets rigides", nous avons mesuré les performances de notre *pipeline* en utilisant uniquement l'algorithme complet de lancer de rayon (parcours sans pile de BVH) et en utilisant la combinaison de l'algorithme complet et de l'algorithme itératif de lancer de rayon. Les résultats sont présentés dans la figure 3.7.

Dans le premier test, nous avons utilisé uniquement un algorithme de lancer de rayon complet. Le but de ce test est de montrer que même avec un seul algorithme de lancer de rayon, notre *pipeline* améliore les performances de l'exécution des calculs sur GPU en fournissant des entrées denses aux *kernels*. Avec ce test, les résultats montrent que le temps de calcul est divisé en moyenne par 1,31 en utilisant notre *pipeline*. Ce gain est expliqué par l'étape par-sommet qui garantit que l'étape par-rayon est exécutée sur un ensemble d'entrées denses même si l'entrée de l'étape par-sommet n'est pas dense.

Dans le second test, nous avons utilisé deux algorithmes de lancer de rayon (algorithme de lancer de rayon complet et algorithme itératif). Le but de ce second test est de montrer que la séparation des exécutions des algorithmes de lancer de rayon dans des *kernels* différents réduit la divergence à l'exécution et améliore les performances. Les résultats montrent que le temps de calcul est divisé en moyenne par 2,73 en utilisant le *pipeline* et que la séparation de l'exécution des algorithmes de lancer de rayon permet d'améliorer les performances.

Nous avons ensuite réalisé un test avec trois algorithmes de lancer de rayon avec la scène "objets rigides et déformables". Les trois algorithmes de lancer de rayon utilisés sont les deux algorithmes complets qui sont utilisés soit pour les objets rigides (lancer de rayon avec parcours sans pile de BVH) soit pour les objets déformables (lancer de rayon basique) ainsi que l'algorithme de lancer de rayon itératif qui est utilisé avec tous

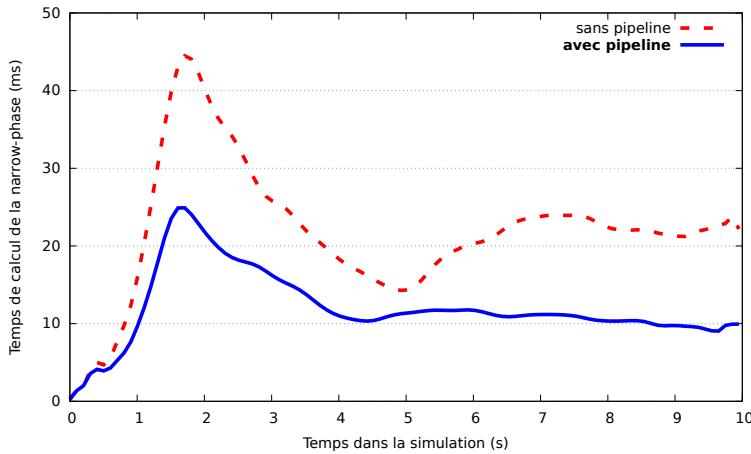


Figure 3.8 – Comparaison des performances sur GPU de la détection de collision avec et sans notre pipeline dans la scène "objets rigides et déformables". L'utilisation de pipeline permet de diviser les temps de calculs par 1,84 en moyenne.

les types d'objets. Les résultats sont présentés dans la figure 3.8 et montrent que notre *pipeline* divise le temps de calcul par 1,84 en moyenne.

1.4.3 Prédiction de la charge de travail

Pour monter le gain de performances lors de l'utilisation de prédictions pour l'estimation de la charge de travail nous avons comparé les temps de calcul sur GPU de simulations en utilisant des prédictions de charge de travail et en utilisant la charge de travail exacte.

Scénario	Gain moyen en temps de calcul
"Objets rigides"	2,84 ms/pas de temps
Lancer de rayon complet uniquement	(4,5% du temps de calcul en moyenne)
"Objets rigides"	1,97 ms/pas de temps
Lancer de rayon complet et itératif	(19,7% du temps de calcul en moyenne)
"Objets rigides et déformables"	2,05 ms/pas de temps
Lancer de rayon complet et itératif	(14,9% du temps de calcul en moyenne)

Table 3.1 – Gain moyen en temps de calcul sur GPU lors de l'utilisation de prédictions pour estimer la charge de travail comparé à l'utilisation de la charge exacte. Utiliser des prédictions est jusqu'à 2,84 ms plus rapide par pas de temps.

Le tableau 3.1 donne le gain moyen par pas de temps lors de l'utilisation de prédictions comparé à l'utilisation de la charge de travail exacte. Nous avons mesuré le gain moyen dans les deux scènes. Pour la scène "objets rigides", nous avons mesuré le gain moyen en utilisant uniquement l'algorithme complet de lancer de rayon et en utilisant l'algorithme complet et itératif de lancer de rayon. Les résultats montrent qu'utiliser des prédictions pour estimer la charge de travail est entre 2,0 et 2,8 ms plus rapide par pas de temps dans tous les cas testés. Ce gain provient de l'élimination des coûts fixes de synchronisations entre le CPU et le GPU pendant la détection de collision (le coût d'une synchronisation entre le CPU et le GPU est fixe et ne dépend

pas de la quantité de travail). Cette réduction n'est pas négligeable dans un contexte temps-réel où nous n'avons que quelques millisecondes pour réaliser la détection de collision. Dans nos exemples, ce gain réduit le temps de calcul entre 4,5% et 19,7% en moyenne à chaque pas de temps. Ce pourcentage de réduction dépend fortement du temps de calcul moyen de la détection de collision, plus celui-ci est faible, plus le gain sera significatif. Dans notre cas, le premier scénario à un temps de calcul élevé (~ 380 ms par pas de temps en moyenne), un gain de quelques millisecondes est alors négligeable. Au contraire, lorsque la détection de collision est réalisée de manière interactive avec des temps de calculs plus faibles (~ 43 et 13 ms par pas de temps en moyenne pour le deuxième et troisième scénario), le gain devient significatif.

1.4.4 Évaluation de la précision des prédictions

Pour évaluer la précision de notre prédition générique (cf. Équation 3.1), nous avons mesuré le pourcentage de rayons manqués causés par une sous-estimation des prédictions. Nous avons comparé ce pourcentage de rayons manqués avec une équation de prédition simpliste (cf. Équation 3.2) qui ne prend pas en compte l'évolution du nombre de paires détectées par la *broad-phase*. Dans les deux cas nous avons utilisé un taux de variabilité de 2% (confiance = 1.02).

$$p(N_t) = N_{t-1} \times \text{confiance} \quad (3.2)$$

Scénario	Taux de réduction du nombre d'erreurs
"Objets rigides" Lancer de rayon complet et itératif	79%
"Objets rigides et déformables" Lancer de rayon complet et itératif	28%

Table 3.2 – Mesure de la réduction du nombre de rayons manqués. Prendre en compte l'évolution du nombre de paires détectées par la *broad-phase* permet de réduire le nombre de rayons manqués jusqu'à 79%.

Le tableau 3.2 montre la réduction du nombre d'erreurs en utilisant notre méthode de prédition face à une méthode plus simple. Nous n'avons pas inclus la scène "objets rigides" avec l'algorithme de lancer de rayon complet seul car avec un seul algorithme le pourcentage d'erreur était trop faible pour que les variations du taux d'erreur soient significatives (moins de 0.1%). Les résultats montrent que prendre en compte l'évolution de nombre de paires réduit le pourcentage d'erreur de 28 à 79%.

1.5 Synthèse et perspectives

En organisant la détection de collision en *pipeline*, nous avons montré un gain de performances non-négligeable. Le *pipeline* permet d'intégrer de multiples algorithmes de lancer de rayon dans notre *narrow-phase* pour gérer plus efficacement des objets de natures différentes. En décomposant la *narrow-phase* en trois étapes, nous sommes capables de maintenir des entrées denses tout au long du *pipeline* pour maximiser

l'utilisation des cœurs des GPUs. Notre implémentation permet de diviser le temps de calcul moyen par un facteur pouvant atteindre 2,73.

Ce *pipeline* pourrait être généralisé pour prendre en compte des objets avec des natures plus variées tels que des objets subissant des changements topologiques ou des fluides. Ce *pipeline* pourrait aussi être généralisé pour prendre en compte des architectures plus larges telles que les architectures multi-GPU et les architectures CPU/GPU hybrides.

Dans la suite de ce chapitre, nous allons introduire différents éléments qui vont enrichir ce *pipeline*. Nous allons expliquer comment réaliser et gérer les prédictions de collisions, comment améliorer la réponse physique en post-traitant le résultat de la détection de collision et comment réaliser la détection de collision par lancer de rayon avec des tissus.

2 Prédiction de collisions

Le principal problème pour exploiter la cohérence temporelle est d'exploiter la cohérence temporelle sur les sommets qui ne sont pas en collision. Pour cela nous proposons de réaliser des prédictions de collision à l'aide de lancer de rayon. Dans cette section, nous allons nous intéresser à la méthode qui permet de réaliser des prédictions et à son coût. L'exploitation des prédictions pour la cohérence temporelle sera introduite dans le chapitre 4.

2.1 Méthode de prédiction

La détection de collision, fondée sur l'algorithme de Hermann et al., est réalisée en lançant des rayons à partir des sommets vers l'intérieur des objets. Nous proposons de lancer un second rayon dans la direction opposée, c'est-à-dire vers l'extérieur des objets. Si le rayon extérieur touche un triangle et que la longueur du rayon est faible, cela signifie que le sommet pourrait entrer en collision avec ce triangle (ou un triangle voisin) dans un futur proche. En pratique, nous allons conserver les prédictions pour lesquelles la longueur du rayon ne dépasse pas un seuil appelé *distPrediction*. Le choix de la valeur de ce seuil dépend de l'application. La figure 3.9 montre des exemples de prédictions.

Une optimisation présentée dans le *pipeline* liée au *culling* (cf Section 1.1.3) va provoquer une erreur de non-estimation des prédictions. En effet, l'une des méthodes de *culling* teste si les sommets sont à l'intérieur du volume englobant de l'autre objet, les sommets qui sont en dehors du volume englobant ne sont pas testés car ils ne peuvent pas être en collision. Le problème est que les rayons extérieurs peuvent détecter des prédictions à partir de sommets qui sont à l'extérieur du volume englobant de la cible. Nous avons besoin de détecter ces prédictions au moins pour celles ayant une longueur inférieure à notre seuil *distPrediction*. La figure 3.10 montre un exemple dans lequel chaque sommet est à l'extérieur du volume englobant de l'autre objet, il est pourtant possible de détecter des prédictions à l'aide de rayons extérieurs avec certains de ces sommets. Pour être certain d'obtenir toutes les prédictions qui sont en dessous de la distance *distPrediction*, il est nécessaire (et suffisant) d'étendre tous les volumes

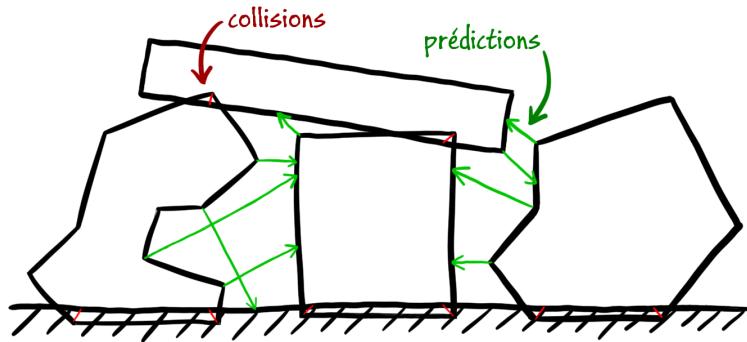


Figure 3.9 – Exemples de rayons détectant des collisions (en rouge) et des prédictions (en vert).

englobants d'une distance $distPrediction$ lors du *culling*.

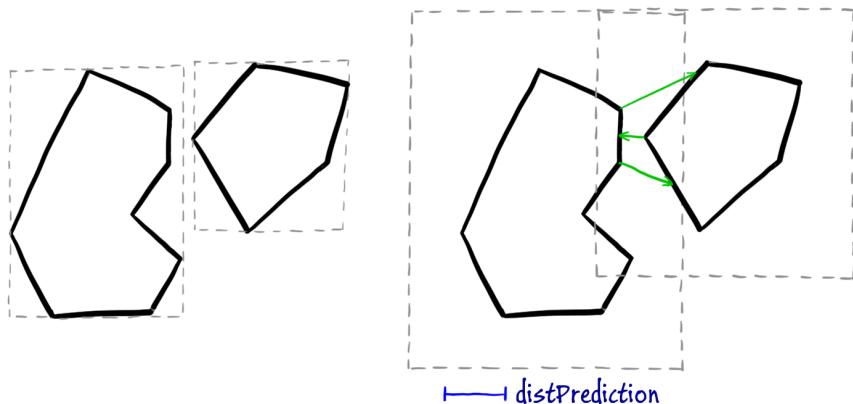


Figure 3.10 – Exemples de prédictions pouvant avoir lieu en dehors des volumes englobants standards. À gauche : deux objets (en noir) avec leurs volumes englobants (en gris) ne sont pas en collision. Aucune prédiction n'est détectée car chaque sommet est à l'extérieur du volume englobant de l'autre objet. À droite : les volumes englobants ont été étendus d'une distance $distPrediction$ dans toutes les directions, toutes les prédictions (en vert) ayant une longueur ne dépassant pas $distPrediction$ sont correctement détectées.

2.2 Impact sur les performances

En théorie, lancer un second rayon pour chaque sommet double le coût du lancer de rayon. Or, nous pouvons utiliser deux propriétés pour réduire le coût du rayon secondaire. Premièrement, les rayons partagent plusieurs paramètres en commun, ils ont la même origine et sont sur la même droite (dans des directions opposées). Ces paramètres communs permettent de factoriser une partie de lancer de rayon des deux rayons en fonction de l'algorithme utilisé. La seconde propriété est qu'il n'est pas nécessaire de tracer le rayon au delà de la distance $distPrediction$. Ceci permet de raccourcir le parcours des structures accélératrices et donc de le rendre moins coûteux.

En pratique, l'augmentation des volumes englobants et le lancer de rayons extérieurs

peuvent être réalisés à un coût très faible. Dans notre *pipeline*, il est possible d'exploiter ces propriétés avec tous les algorithmes de lancer de rayon que nous utilisons. Avec l'algorithme de lancer de rayon par parcours d'un BVH, nous pouvons factoriser la première descente dans l'arbre qui ne dépend que du point d'origine du rayon. Il est ensuite possible d'arrêter le parcours de l'arbre dès que le rayon dépasse la longueur *distPrediction*. Avec l'algorithme basique et l'algorithme itératif, il est possible de factoriser entièrement les deux lancers de rayon. Cette factorisation est réalisée en combinant l'algorithme d'intersection rayon/triangle proposé par [Möller and Trumbore, 2005] pour le rayon intérieur et le rayon extérieur. Il suffit juste de regarder si la coordonnée *t* est positive (collision) ou négative (prédition).

Pour évaluer le coût des rayons extérieurs, nous avons comparé les temps de calculs de notre *pipeline* complet avec et sans prédition sur deux scènes (cf. Figure 3.11). Dans la première scène une avalanche de 512 objets non-convexes tombent sur un sol plan. Dans la seconde scène 500 objets sont insérés progressivement, un nouvel objet est ajouté tous les dixièmes de secondes. Les deux scènes sont calculées à 60Hz et les objets utilisés sont les quatre objets utilisés dans les tests du *pipeline* (cf. Figure 3.4). Dans la première scène, le temps de calcul culmine au moment où l'ensemble des objets touchent le sol, à ce moment environ 7.300 paires d'objets sont testées par la *narrow-phase*. Dans la seconde scène, le temps de calcul culmine à la fin de la simulation lorsque tous les objets sont présents. À ce moment, environ 10.000 paires d'objets sont testées par la *narrow-phase*.

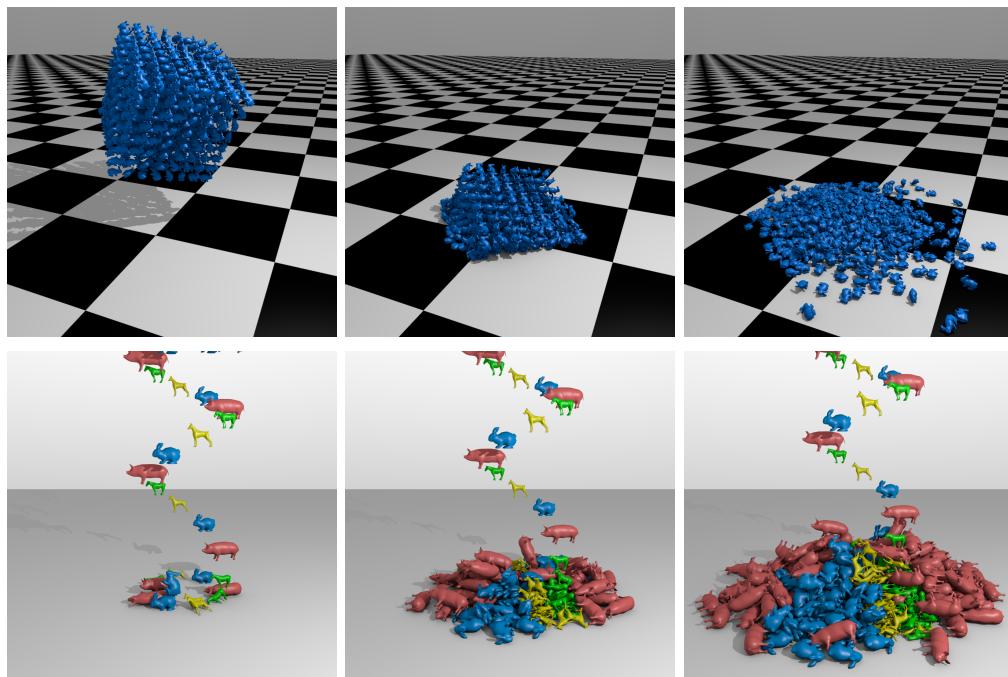


Figure 3.11 – Scènes expérimentales. Haut : 512 objets non-convexes tombent sur un sol plan. Bas : 500 objets sont insérés progressivement.

Les figures 3.12 et 3.13 donnent le temps de calcul de la *narrow-phase* dans les deux scènes. Les calculs ont été réalisés sur une carte graphique Nvidia GTX 660 (960 coeurs, 2 Go de mémoire). Dans l'ensemble des résultats, l'ajout de rayons prédictifs fait varier

les temps de calcul de -21% à +28%. De manière générale le coût l'utilisation de rayons prédictifs ne fait pas doubler le temps de calcul.

Les deux cas extrêmes de temps de calculs ont tous les deux lieu dans la partie B de la figure 3.12. Dans cette partie, les objets ont des mouvements très rapides. Le *pipeline* choisit, en conséquence, d'utiliser de manière majoritaire les algorithmes de lancer de rayon complets. Dans le cas de l'utilisation du lancer de rayon avec BVH, la factorisation partielle du parcours du rayon intérieur et du rayon extérieur permet de réduire le coût mais pas entièrement, dans ce cas l'utilisation de prédictions augmente le temps de calcul de 28%. Dans le cas de l'utilisation de l'algorithme de lancer de rayon basique, le parcours du rayon intérieur et du rayon extérieur est complètement factorisé, dans ce cas l'utilisation de prédictions réduit le temps de calcul de 21%. Ce résultat est contre-intuitif, mais est expliqué par les critères de *culling* utilisés dans le *pipeline*. L'un des critères de *culling* est de ne pas réaliser de lancer de rayon pour un sommet donné si celui-ci n'était pas en collision dans les pas de temps précédents et qu'aucune prédition n'indiquait une future collision possible. L'utilisation de prédictions permet donc d'éliminer des tests unitaires de collision et de réduire la quantité de rayons calculés. Dans le cas présent, le coût de la prédition est plus faible que le gain occasionné par les critères de *culling* les utilisant.

Dans les autres cas (c'est-à-dire en dehors de la partie B de la figure 3.12), le *pipeline* repose plus fortement sur l'utilisation de l'algorithme de lancer de rayon itératif. Dans ce cas la variation de performance engendrée par l'utilisation de prédictions est plus faible voir négligeable.

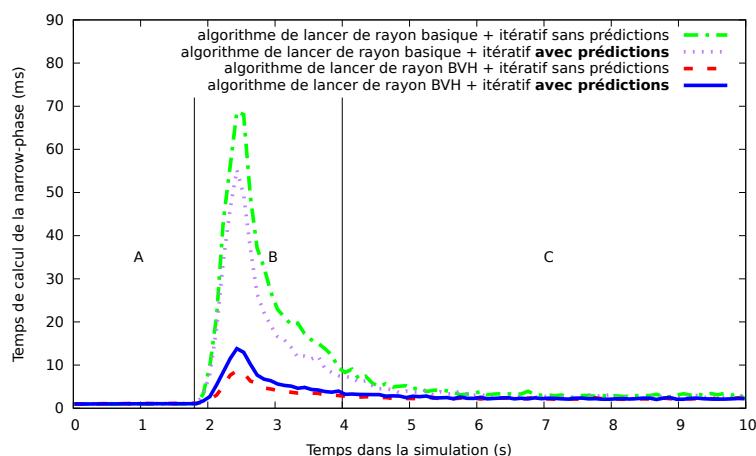


Figure 3.12 – Comparaison du temps de calcul de la *narrow-phase* avec et sans prédictions dans la première scène.

2.3 Synthèse

Ces résultats montrent que l'utilisation de rayons extérieurs pour réaliser une mesure de distance inter-objets est une méthode fiable avec un faible coût. Cette mesure n'est en revanche applicable que si les objets sont proches. Plus les objets sont à des distances élevées, plus le risque que les rayons extérieurs ratent les objets est élevé. Dans notre

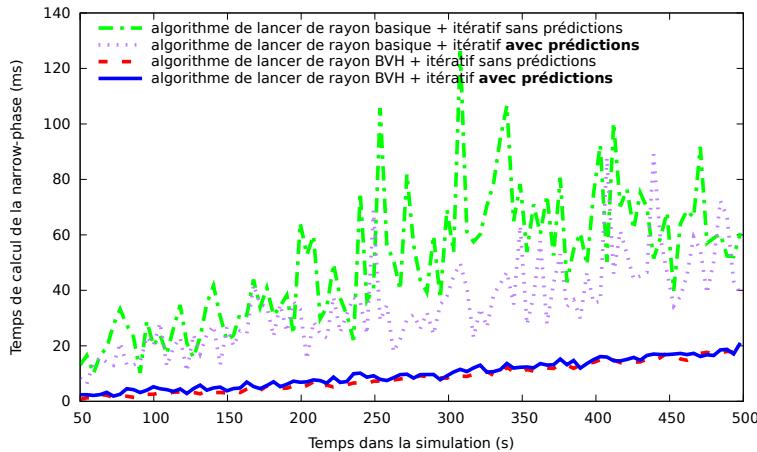


Figure 3.13 – Comparaison du temps de calcul de la *narrow-phase* avec et sans prédictions dans la seconde scène.

cas d'utilisation, les objets seront proches car les prédictions ne seront réalisées qu'entre les objets détectés par la *broad-phase*.

En théorie, lancer un rayon extérieur devrait doubler le coût du lancer de rayon. En pratique, les mesures de performance montrent que le rayon extérieur a un coût bien inférieur au coût théorique. Dans le pire cas, le temps de calcul n'est augmenté que de 28%. Contre-intuitivement, l'utilisation de rayons extérieurs peut diminuer le temps de calcul (jusqu'à -21% dans nos tests) en permettant d'éliminer certains tests unitaires lorsque les prédictions montrent que les objets sont à des distances assez élevées.

Les prédictions sont aussi utilisées pour réduire le nombre d'erreurs de détection de collision lors de l'utilisation de notre algorithme de lancer de rayon itératif. Nous reviendrons dans le chapitre 4 sur la manière d'utiliser les prédictions de collision pour réduire le nombre d'erreurs dans notre méthode après avoir présenté notre méthode itérative.

3 Re-projection des rayons

Avec la méthode de [Hermann et al., 2008] que nous utilisons dans notre *pipeline*, les rayons qui mettent en évidence les collisions sont utilisés pour déterminer les points de contact pour réaliser la réponse physique, la direction du rayon donnant la direction dans laquelle séparer les objets. Cependant, la direction des rayons ne correspond pas toujours à la direction optimale pour séparer les objets. La figure 3.14.a montre des exemples de rayons qui détectent des collisions mais dont la direction ne correspond pas aux besoins de la réponse physique présentée dans la figure 3.14.b.

Pour résoudre ce problème, Hermann et al. ont proposé de pondérer les forces de répulsions des rayons par le cosinus de l'angle entre le rayon et la normale de la surface. Cette méthode ne fait que réduire les erreurs sans les éliminer et n'est applicable qu'avec les algorithmes de réponse physique basés forces. Dans cette section, nous proposons une nouvelle méthode permettant de fournir la direction optimale pour la réponse physique par rapport à la surface en collision et nous la comparons à la méthode de

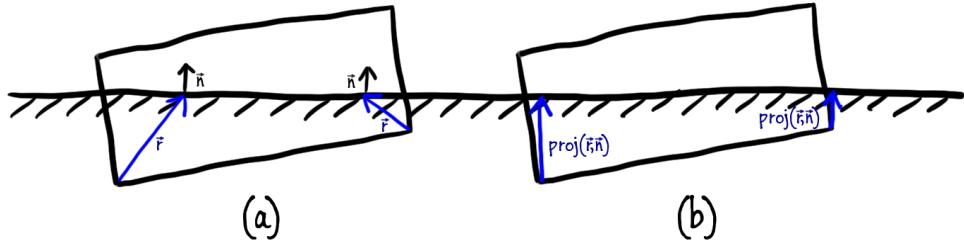


Figure 3.14 – Collision entre un objet et le sol. En (a) les rayons ne donnent pas les directions optimales pour séparer les objets. En (b) la re-projection a été appliquée et les rayons donnent la direction optimale pour séparer les objets.

Hermann et al.

3.1 Méthode de re-projection des rayons

Lors de l'exécution du lancer de rayon, nous enregistrons pour chaque rayon la normale de la surface touchée. Nous proposons ensuite de post-traiter les rayons en les re-projetant sur la normale de la surface qu'ils ont touché. Cette re-projection minimise la longueur des rayons relativement à la surface touchée. Dans le domaine de la simulation physique, ceci correspond à la Minimum Translational Distance (MTD) [Cameron and Culley, 1986]. La MTD indique que la direction la plus adaptée pour séparer deux objets en collision est la direction minimisant le déplacement entre les deux objets. Cette méthode correspond à une heuristique pour la réponse physique à une collision. Le résultat de la re-projection correspond au résultat théorique attendu dans la figure 3.14.b. De manière générale la re-projection des rayons fournit le résultat théorique attendu de manière locale à la surface touchée. Le résultat théorique global sera atteint si la surface touchée contient celui-ci.

Projeter un rayon sur un vecteur peut être facilement réalisé à l'aide de l'équation 3.3, avec \vec{r} le rayon (vecteur du point d'origine du rayon jusqu'au point touché) et \vec{n} la normale unitaire de la surface touchée. Cette méthode ne fait aucune hypothèse sur la réponse physique et peut donc être utilisée avec n'importe quel algorithme de réponse physique.

$$\text{proj}(\vec{r}, \vec{n}) = \vec{n} \times (\vec{r} \cdot \vec{n}) \quad (3.3)$$

En pratique, la réponse physique ne représente pas les collisions avec des rayons mais avec des points de contacts. Un point de contact est représenté par un point en collision P avec sa distance de pénétration pd et sa direction séparatrice \vec{d} (un exemple est illustré sur la figure 3.15). Avec cette représentation, nous pouvons éviter de calculer explicitement la re-projection de rayon en prenant :

- P = sommet en collision
- $pd = \vec{r} \cdot \vec{n}$
- $\vec{d} = \vec{n}$.

Cette méthode nous permet de réaliser la re-projection des rayons de manière implicite.

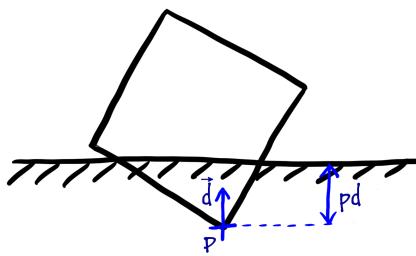


Figure 3.15 – Un point de contact est composé d'un point P , d'une direction séparatrice d et d'une distance de pénétration pd .

3.2 Évaluation de la re-projection des rayons

Pour évaluer notre méthode, nous avons premièrement considéré le cas d'une collision entre un objet arbitraire et un sol plan, dans un tel cas la réponse physique devrait être orthogonale au sol. Ce test a été utilisé par Hermann et al. pour évaluer leur méthode de pondération des forces. Nous avons testé trois objets : une sphère discrétisée et deux objets irréguliers (le lapin et le cochon présentés dans la figure 3.4). Chaque objet a été testé avec 400 orientations différentes et 20 niveaux de pénétration différents avec le sol (comme illustré dans la figure 3.16).

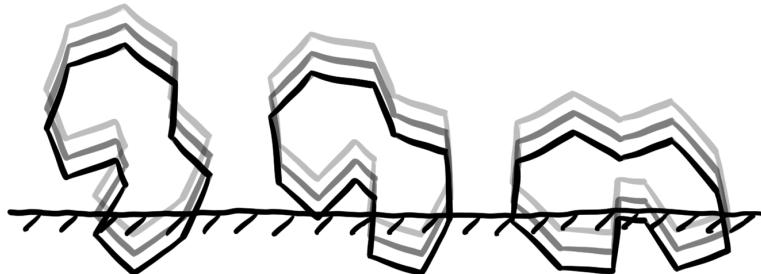


Figure 3.16 – Pour tester la déviation moyenne de la réaction physique entre un objet et le sol, nous avons réalisé des tests de collision avec différents niveaux de pénétration et avec différentes orientations.

La table 3.3 donne la déviation moyenne définie par le ratio entre les forces tangentielles et les forces normales. En théorie, cette déviation devrait être de zéro. La méthode standard correspond à ne pas modifier ni pondérer les rayons.

Méthode	sphère	lapin	cochon
nombre de sommets	70	453	1085
standard	3.97%	22.0%	27.0%
pondération [Hermann et al., 2008]	2.66%	16.5%	20.6%
Re-projection	0.00%	0.00%	0.00%

Table 3.3 – Déviation moyenne des forces de pénalités pour trois objets.

La déviation est faible avec la sphère, ceci est expliqué par la régularité de la forme. Avec les symétries, les forces tangentielles ont tendance à s'annuler. Avec des

formes irrégulières, les forces tangentielles ne s'annulent pas et la déviation n'est pas négligeable. La méthode de pondération proposée par Hermann et al. réduit les forces tangentielles, mais la réduction n'est environ que de 25% pour les formes irrégulières. Notre méthode de re-projection permet de garantir une déviation de 0% car les forces sont réorientées dans la direction optimale.

Le test entre un objet arbitraire et un sol plan n'étant pas représentatif des situations réelles, nous avons évalué notre méthode de re-projection sur deux scènes tests. Les deux scènes tests sont les scènes utilisées dans la section précédente (cf. Figure 3.11). La réponse physique est obtenue en utilisant le *solver* de contraintes de Bullet Physics qui réduit les points de contacts entre les objets (jusqu'à quatre points de contacts sont conservés) puis applique des impulsions de vitesse. Nous avons exécuté chaque scène avec et sans re-projection des rayons. La table 3.4 donne la somme des longueurs des rayons tout au long des simulations. Cette somme devrait être la plus petite possible car elle mesure la quantité d'inter-pénétrations présentes entre les objets. Cette somme de longueurs n'est pas nulle car le *solver* utilisé cherche à faire un compromis entre précision et temps de calcul. Les résultats montrent que l'utilisation de la re-projection des rayons diminue jusqu'à 59% la quantité d'interpénétration. Cette réduction montre que la re-projection des rayons permet de fournir à la réponse physique des points de contacts plus adaptés, ceci va se traduire par des simulations comportant moins d'erreurs et plus fidèles à la réalité.

	Standard	Re-projection	Réduction
Scène 1	16 115	10 793	59%
Scène 2	66 094	38 398	42%

Table 3.4 – Comparaison de la somme des longueurs des rayons tout au long de la simulation avec et sans re-projection des rayons.

3.3 Synthèse

La re-projection des rayons est une méthode simple qui permet d'obtenir une réponse physique correcte. Cette méthode est appliquée après le lancer de rayon et modifie les directions associées aux points de contacts avec un coût de calcul négligeable. Nos tests ont montré que cette méthode est plus performante que la méthode proposée par Hermann et al.

Cette méthode réalise une approximation de la recherche du déplacement le plus court séparant les objets en limitant la recherche au triangle ayant été détecté en collision. Cette approximation correspond à réaliser une recherche locale. Cependant, la méthode de détection de collision que nous utilisons fournit des correspondances sommet/triangles très proches et il est donc fortement probable que le déplacement minimum séparant le sommet détecté avec le second objet se situe sur le triangle détecté. Si ce n'est pas le cas la réponse physique sera tout de même capable de calculer une réponse pour séparer les objets, mais celle-ci ne sera pas toujours réaliste (cf. figure 3.17).

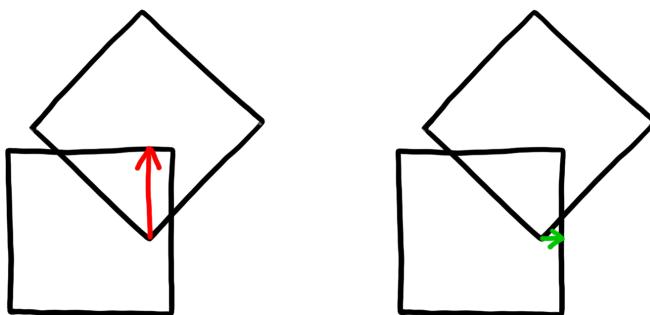


Figure 3.17 – Exemple où la re-projection des rayons (à gauche) ne fournit pas le déplacement minimum global (à droite).

4 Détection de collision pour les objets surfaciques

La méthode de détection proposée par [Hermann et al., 2008] permet de réaliser la détection de collision sur des surfaces closes, c'est-à-dire des objets volumiques représentés par leur surface (que nous appellerons par la suite simplement objet volumique). Cependant, certains types d'objets sont difficilement représentables avec ce type de représentation. C'est le cas des objets extrêmement fins et pouvant être représentés par une surface simple tels que des tissus ou de la tôle (que nous appellerons par la suite objet surfacique).

Nous proposons une méthode de détection de collision pour les tissus (et plus généralement les objets surfaciques) pour la *narrow-phase* qui utilise les méthodes de lancer de rayon pour réaliser des tests de collision entre deux objets. Notre méthode est capable de réaliser la détection de collision discrète entre un tissu et un objet volumique (incluant des objets non-convexes) ainsi que la détection de collision discrète entre deux tissus (en incluant les auto-collisions). Notre méthode est couplée avec une méthode permettant de corriger les artefacts de collision entre des tissus causées par les simulations discrètes. En effet, avec des simulations discrètes, des artefacts peuvent apparaître lorsque des objets ont des mouvements avec une amplitude supérieure à leur taille en un seul pas de temps. Ce problème est critique avec des tissus à cause de leur faible épaisseur. Notre méthode détecte ces inversions et les corrige.

Dans cette section, nous commençons par présenter le principe général de notre méthode de détection de collision pour les tissus (cf. Section 4.1). Nous détaillons ensuite les différents cas à traiter (cf. Section 4.2). Puis nous évaluons les performances de notre méthode (cf. Section 4.3).

4.1 Principe général de la méthode

Avec des objets volumiques (rigides ou déformables), la détection de collision peut être réalisée en lançant des rayons depuis les sommets vers l'intérieur des objets. Dans cette méthode, les objets volumiques sont représentés par leur surface (généralement un maillage de triangles).

Dans notre méthode, les tissus sont représentés par une surface, mais au lieu de représenter les limites de l'objet, cette surface représente la couche centrale du tissu.

L'intérieur du tissu s'étend des deux côtés de la surface jusqu'à une distance e ($e = \text{demi épaisseur du tissu}$), avec e petit face aux dimensions du tissu. La figure 3.18 compare notre modèle de représentation des tissus face au modèle de représentation des objets volumiques.

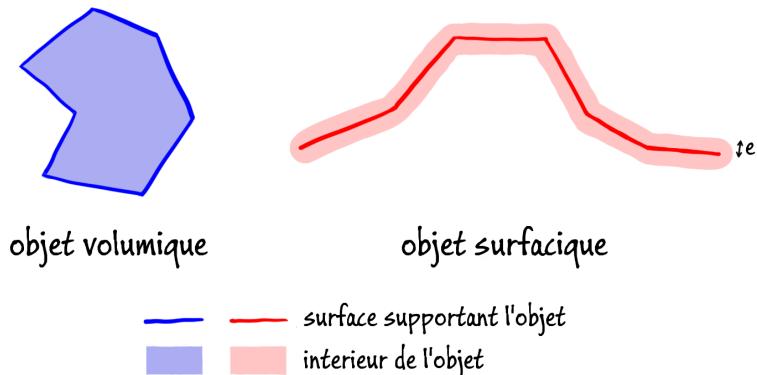


Figure 3.18 – Différences de représentation entre les objets volumiques et les objets surfaciques.

Pour réaliser la détection de collision entre une paire d'objets, notre méthode lance des rayons depuis chaque sommet de chaque objet. Chaque paire $(A; B)$ est décomposée en deux tests $(A \rightarrow B)$ et $(B \rightarrow A)$ dans lesquels les rayons sont lancés respectivement de A vers B et B vers A . Chaque test $(A \rightarrow B)$ exécute une requête de collision pour chaque sommet de A face à l'objet B . La combinaison des deux tests permet de tester tous les sommets de chaque objet. Exécuter seulement un test pourrait mener à de faux-négatifs. La figure 3.19 donne un exemple où les deux tests sont nécessaires pour détecter toutes les collisions.

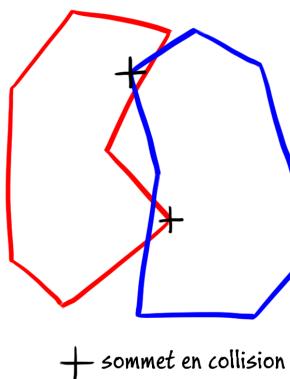


Figure 3.19 – Exemple de deux objets en collision. Si les tests de collision n'avait été réalisés que sur un seul objet, seulement l'un des deux sommets en collision aurait été détecté. Pour détecter toutes les collisions, les sommets de chaque objet doivent être testés.

Décomposer les paires $(A; B)$ en deux tests séparés permet de prendre en compte les spécificités de chaque objet. A et B peuvent avoir différentes propriétés, chacun de ces objets peut être un objet volumique ou un tissu. Étant donné les différences de représentations possibles, tous les types de tests doivent être spécialisés. Les tests

possibles sont :

- volumique → volumique
- volumique → tissu
- tissu → volumique
- tissu → tissu

Le test volumique → volumique correspond à la méthode proposée par [Hermann et al., 2008]. De plus, suivant la nature de l'objet ciblé par les rayons (rigide ou déformable), différents algorithmes de lancer de rayon peuvent être utilisés pour améliorer les performances comme nous l'avons montré avec notre *pipeline*. Les objets rigides peuvent utiliser des structures accélératrices complexes pour améliorer les performances du lancer de rayon, tandis que les objets déformables vont devoir utiliser des structures accélératrices plus simples qui peuvent être mises à jour avec un coût plus faible.

Les tests d'auto-collision pour les objets déformables sont gérés de la même manière que des tests de collisions entre deux objets différents. Pour gérer les auto-collisions, il suffit simplement d'ajouter pour chaque objet déformable D un test ($D \rightarrow D$).

Notre méthode va produire des points de contacts qui vont être utilisés par la réponse physique. Un point de contact donne une paire de caractéristiques, une appartenant à chaque objet, qui sont en collision. Dans notre méthode, ces caractéristiques sont un sommet (origine du rayon) et un triangle (qui a été touché par le rayon). Chacune de ces caractéristiques est accompagnée d'une normale, celle-ci donne la direction optimale pour déplacer les objets afin de les séparer.

4.2 Détection de collision pour les tissus

Nous allons présenter les trois étapes de notre méthode :

- Nous lançons des rayons à partir des sommets des objets. Ces rayons sont responsables de la détection des caractéristiques en collisions (et en auto-collision) et de la détection des collisions futures possibles (prédictions).
- Pour les tests de collisions entre tissus, nous appliquons un algorithme de traitement des inversions pour les détecter et les corriger. Ce test détecte les artefacts de collision entre les tissus qui sont introduits lorsque les déplacements sont trop importants dans un pas de temps.
- Nous générerons les points de contacts pour la réponse physique. Les points de contacts sont générés à partir du résultat du lancer de rayon (et potentiellement corrigés par l'algorithme de traitement des inversions). Un post-traitement est appliqué pour s'assurer d'une réponse physique correcte.

4.2.1 Détection par lancer de rayon

La première étape pour réaliser la détection de collision avec le lancer de rayon est de lancer des rayons à partir de chaque sommet de chaque objet pour chaque paire testée. Deux rayons sont lancés de chaque sommet, un dans la direction de la normale et un dans la direction opposée de la normale.

L'un des problèmes à résoudre est de ne pas détecter de faux points de contacts. En effet, les points de contacts doivent être à l'intérieur des objets en collision et

la normale de point du contact doit indiquer la direction permettant de séparer les objets avec le plus faible déplacement possible. Les faux points de contacts sont des points de contacts non fiables qui sont détectés soit en dehors des objets, soit avec des normales qui n'indiquent pas la bonne direction pour séparer les objets. Ces faux points de contacts vont causer des erreurs dans la réponse physique en générant des forces contradictoires qui peuvent verrouiller les objets entre eux ou créer des instabilités.

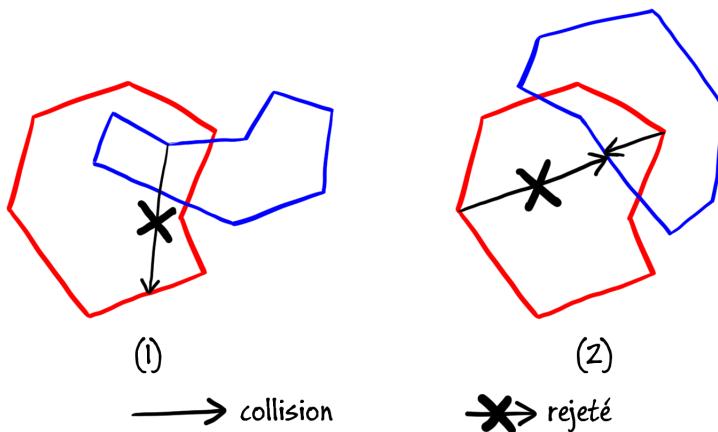


Figure 3.20 – Illustration des conditions proposées par Hermann et al. Les flèches barrées ont été rejetées car elles ne satisfaisaient pas les conditions. En (1) le rayon touche l'objet cible après avoir quitté l'objet source. En (2) le rayon touche la face extérieure de la cible (contrairement à l'autre flèche). Si les rayons barrés étaient utilisés comme points de contacts, des contraintes opposées verrouilleraient les objets entre eux dans la réponse physique.

Pour éviter ces conflits, Hermann et al. ont proposé d'utiliser deux conditions sur les rayons lors des tests entre deux objets volumiques :

1. Le rayon doit toucher la cible avant de quitter l'objet source.
2. Le rayon doit toucher l'intérieur de la cible.

Ces deux conditions éliminent les faux points de contacts et garantissent qu'il n'y aura pas de conflit entre les points de contacts restants. Des exemples sont présentés dans la figure 3.20 dans lesquels les flèches barrées sont éliminées car elles ne satisfisaient pas les conditions.

Notre but est de permettre la réalisation de la détection de collision entre des paires d'objets comprenant des tissus sans détecter de faux points de contacts. Pour cela nous proposons trois tests élémentaires, ainsi que les conditions adéquates qui permettent l'introduction de tissus dans les tests de collision : volumique → tissu, tissu → volumique et tissu → tissu.

volumique → tissu

Pour les objets volumiques, le rayon qui est lancé dans la direction de la normale va à l'extérieur de l'objet (ce rayon est appelé rayon extérieur) et le rayon qui est lancé dans la direction opposée de la normale va à l'intérieur de l'objet (ce rayon est appelé rayon intérieur). La figure 3.21 donne des exemples de rayons classés comme des collisions, prédictions ou rejettés par les rayons intérieurs et extérieurs.

Le rayon intérieur est uniquement utilisé pour détecter des collisions. Avec les tissus, les conditions proposées par Hermann et al. ne sont pas applicables. Les tests de collisions sont traités avec la surface supportant le tissu (la couche centrale), ce qui rend impossible le fait de savoir si un rayon touche l'intérieur ou l'extérieur du tissu. De plus, exiger qu'un sommet soit dans la gamme d'épaisseur d'un tissu pour pouvoir détecter des collisions exigerait une fréquence de simulation extrêmement élevée pour fonctionner avec des tissus fins.

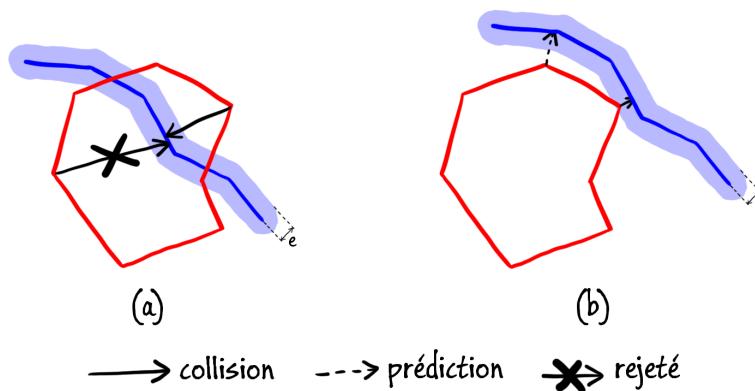


Figure 3.21 – Exemples de rayons détectant des collisions, des prédictions et de rayons rejetés dans les tests volumique → tissu. En (a) une partie du tissu est entièrement à l'intérieur de l'objet volumique. Un rayon détecte une collision si et uniquement si le rayon a une longueur inférieure à la moitié de la distance de sortie. En (b) le tissu touche l'objet volumique. Un rayon détecte une collision si le rayon a une longueur inférieure à e , autrement le rayon détecte une prédition.

Pour réaliser la détection de collision tout en évitant de générer de faux points de contacts nous proposons d'utiliser une version modifiée de la condition (1). De plus, pour éviter les faux-négatifs avec des tissus fins, nous abandonnons la condition (2). L'idée principale est de choisir le chemin le plus court pour séparer les objets, ce qui est la solution la plus probable. Avec notre méthode, un rayon est conservé si sa longueur ne dépasse pas la moitié de la distance de sortie (longueur du rayon lorsqu'il quitte l'objet source), dans le cas contraire il est ignoré. Si le rayon a une longueur supérieure à la moitié de la distance de sortie alors le tissu est plus proche de l'autre côté de l'objet, la collision sera donc détectée de l'autre côté. La figure 3.21.a donne un exemple : la flèche pleine satisfait la condition tandis que la flèche barrée a une longueur dépassant la moitié de la distance de sortie.

Dans la méthode de prédiction de collision que nous avons proposée (cf. Section 2), le rayon extérieur est uniquement utilisé pour prédire des collisions. Avec les tissus nous devons prendre en compte leur épaisseur. Notre solution est de classer les rayons extérieurs lancés sur des tissus comme des collisions si la longueur du rayon est inférieure à e (demi-épaisseur du tissu). Dans les autres cas, les rayons extérieurs sont toujours classés comme des prédictions. La figure 3.21.b donne un exemple : la flèche pleine a une longueur inférieure à e et est classée comme une collision (contrairement à la flèche en pointillés).

tissu → volumique

Avec les tissus, les deux rayons lancés à partir d'un sommet sont à l'intérieur de l'objet et le quitte après avoir parcouru la distance e . Contrairement aux objets volumiques, les deux rayons lancés à partir d'un tissu se comportent de la même manière. La figure 3.22 donne un exemple de collisions détectées à partir d'un tissu avec un objet volumique.

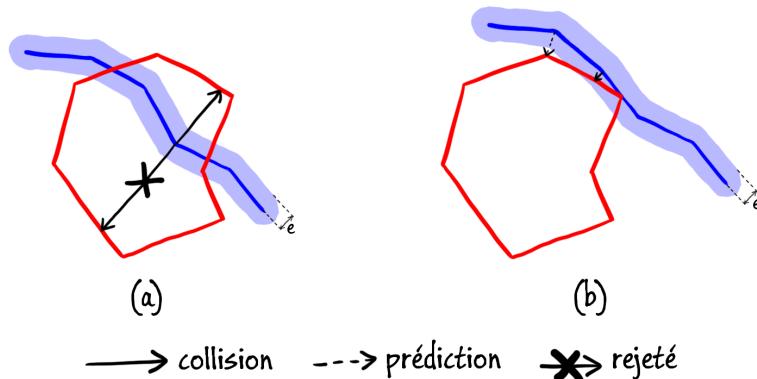


Figure 3.22 – Exemples de rayons détectant des collisions, des prédictions et de rayons rejetés dans les tests tissu → volumique. En (a) une partie du tissu est entièrement à l’intérieur de l’objet volumique. Pour une paire de rayons touchant l’intérieur de l’objet volumique, uniquement le rayon le plus court est utilisé pour détecter une collision. En (b) le tissu touche l’objet volumique. Un rayon détecte une collision si le rayon a une longueur inférieure à e , autrement le rayon détecte une prédiction.

Lorsqu'un rayon issu d'un sommet d'un tissu touche un objet volumique, nous avons deux cas possibles : (a) le rayon touche l'intérieur de l'objet volumique, (b) le rayon touche l'extérieur de l'objet volumique.

Dans le cas (a), les deux rayons lancés du même sommet vont toucher l'intérieur de l'objet volumique (car le sommet est à l'intérieur de l'objet). Nous ne pouvons pas utiliser les deux rayons pour générer des points de contacts car ceux-ci seraient en conflits, les deux rayons donnent des directions opposées et il faut choisir une unique direction pour séparer le tissu de l'objet volumique. Dans ce cas, nous classons le rayon qui a la longueur la plus courte comme une collision et nous rejetons l'autre rayon. Ce choix correspond à la sélection du chemin le plus court pour séparer les deux objets. La figure 3.22.a donne un exemple d'un tel cas : la flèche pleine est préférée à la flèche barrée car elle est plus courte.

Dans le cas (b), nous détectons une collision si le rayon a une longueur inférieure à e pour prendre en compte l'épaisseur du tissu. La figure 3.22.b donne un exemple : la flèche pleine a une longueur inférieure à e et est classée comme une collision tandis que la flèche en pointillés est classée comme une prédiction.

tissu → **tissu**

Comme pour le cas tissu → volumique, les deux rayons sont lancés à partir de l'intérieur du tissu les quittent après avoir parcouru une distance e .

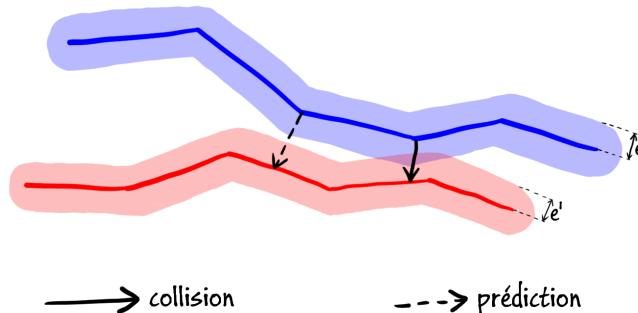


Figure 3.23 – Exemples de rayons détectant des collisions et des prédictions dans les tests tissu → tissu. La flèche pleine a une longueur inférieure à $e + e'$, elle détecte donc une collision. La flèche en pointillés est trop longue, elle détecte une prédiction.

Lorsque nous réalisons la détection de collision entre deux tissus, nous devons prendre en compte l'épaisseur des deux tissus (illustré dans la figure 3.23). Lorsque un rayon lancé par un tissu touche un autre tissu, nous classons le rayon comme une collision si la longueur de rayon est inférieure à $e + e'$ (avec e et e' les demi-épaisseurs de chaque tissu). Dans les autres cas, nous classons les rayons comme des prédictions.

4.2.2 Détection des inversions

Avec des simulations physiques discrètes, la détection de collision est exécutée après que les objets aient bougé d'un delta de temps (pas de temps). Dans ce cas, la détection de collision est réalisée lorsque les objets sont en interpénétration et la réponse physique correcte doit être trouvée. Lorsque de petits objets sont présents dans la simulation ils peuvent se traverser l'un à travers l'autre pendant un seul pas de temps même s'ils ne vont pas très vite. Ce problème est particulièrement présent avec les tissus à cause de leur faible épaisseur et de la relative indépendance des mouvements des sommets. La figure 3.24 montre un exemple de comment une inversion entre deux sommets peut avoir lieu entre deux tissus.

Lorsqu'une inversion se produit entre deux tissus (ou en auto-collision), les deux tissus se comportent comme s'ils étaient collés ensemble. Ceci s'explique par les contraintes en conflits entre les tissus. La réponse physique essaye de maintenir les sommets du côté auxquels ils ont été détectés, elle applique donc des forces dans les directions de séparation locale à chaque sommet. Avec des inversions, ces forces sont contradictoires et vont chercher à séparer les deux tissus dans des directions opposées. La somme de ces forces va finalement maintenir ensemble les deux tissus. Les inversions provoquent aussi des artefacts visuels, les sommets en inversion ne sont pas rendus du bon côté des tissus et provoquent les erreurs visuelles (cf. Figure 3.25).

Pour détecter les inversions, nous ajoutons un post-traitement sur les sommets pour lesquels un rayon a détecté une collision ou une prédiction. Notre solution est de tester de quel côté est localisé le sommet relativement au triangle détecté et le comparons avec le côté auquel était localisé le même sommet avec le même triangle avant l'intégration temporelle (cf. figure 3.26). Notre but étant de détecter si le sommet a franchi le triangle pendant cet intervalle de temps. Si le sommet a changé de côté relativement au triangle, nous considérons que nous avons une inversion. Dans un tel cas, nous inversons la

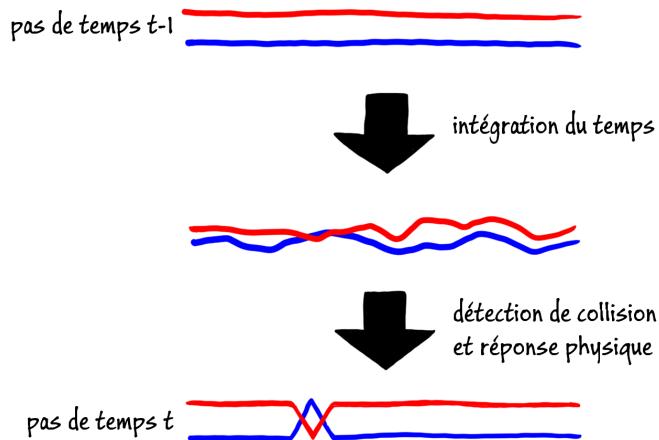


Figure 3.24 – Exemple d'inversion de deux sommets entre deux tissus. Au pas de temps $t - 1$, aucune inversion n'est présente. Après avoir appliqué l'intégration du temps (intégration des forces et des vitesses pour calculer les nouvelles positions des objets), une inversion est introduite dans la simulation. Si aucune méthode permettant de corriger les inversions n'est appliquée, la réponse physique va maintenir cette inversion et agir comme si les deux objets étaient collés entre eux.

direction de la normale du point de contact. En effet, l'inversion de la direction de la normale va forcer la réponse physique à pousser le sommet et le triangle dans les directions opposées et va donc dé-entremêler les tissus.

Lorsque une inversion est détectée sur un sommet, si des prédictions de collision ont été détectées à partir de celui-ci il faut les reclasser comme une collision (cf. figure 3.26). Ceci peut arriver lorsque un sommet bouge d'une distance supérieure à l'épaisseur des tissus en un seul pas de temps. Si les points de contacts n'étaient pas reclassés, ils seraient ignorés par la réponse physique, ce qui laisserait les tissus dans un état entremêlé. Si les mêmes sommets sont ensuite redétectés dans les pas de temps suivants, l'algorithme de détection d'inversion ne serait pas capable de corriger les inversions car il ne compare la situation actuelle qu'avec la situation du pas précédent pour détecter les inversions.

Cette méthode peut être vue comme proche de la détection d'inversion de tétraèdres telle que celle utilisée dans [Müller et al., 2015]. Dans cette méthode, l'espace entre les objets est modélisé avec un maillage de tétraèdres et les inversions sont évitées en contraignant les tétraèdres à conserver un volume positif. Dans notre méthode, au lieu de reposer sur une décomposition complète de l'espace entre les objets, nous travaillons sur des paires (sommets ; triangle) détectées par notre méthode de détection de collision. Nous empêchons ensuite ces paires de s'inverser en un seul pas de temps, ce qui est équivalent à contraindre le volume du tétraèdre formé par le sommet et le triangle à rester positif. Contrairement à la méthode de Müller et al., notre méthode n'a pas besoin d'appliquer une optimisation du maillage de tétraèdres lorsque de trop grands déplacements sont présents dans la simulation. Mais notre méthode n'est capable de détecter que les inversions qui se produisent dans un seul pas de temps.

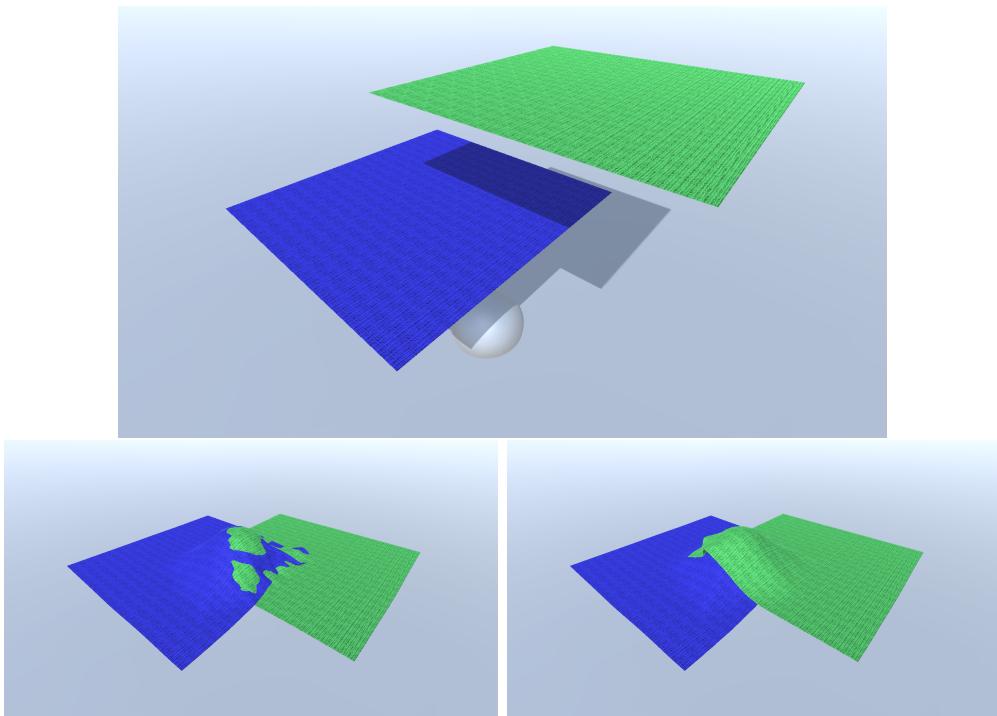


Figure 3.25 – Deux draps tombent sur une boule (image supérieure). Sans détection d'inversion, les tissus s'entremêlent (image gauche). Avec une détection d'inversion, la détection de collision rend un résultat correct et les tissus ne n'entremêlent pas (image droite).

4.2.3 Génération des points de contacts

Après que les collisions soient détectées et que les inversions soient corrigées nous devons générer les points de contacts pour la réponse physique. Pour améliorer la fiabilité de la réponse physique nous utilisons la re-projection des rayons (cf. Section 3) pour nous assurer que les points de contacts fournissent la direction du déplacement le plus court pour séparer les objets. Ceci est réalisé en projetant le rayon sur la normale de la surface en collision.

La projection du rayon modifie la longueur de celui-ci en le raccourcissant. Avec notre méthode de détection de collision, la classification des rayons dépend de leur longueur. Modifier la longueur des rayons peut donc changer le résultat de la classification. Par conséquence, il faut réaliser la re-projection des rayons avant la classification. La figure 3.27 donne un exemple où la re-projection des rayons change le résultat de la détection de collision, si la re-projection des rayons n'était pas appliquée alors le rayon serait faussement classé comme une prédiction. Ces erreurs, lorsque présentes dans la simulation, provoquent des instabilités sur les positions des objets. Sans re-projection, les points de contacts ne sont pas détectés à la distance où la réponse physique place les sommets ce qui conduit à une alternance entre un état où les collisions sont détectées et un état où les collisions ne sont pas détectées, ce qui conduit la simulation à un état oscillatoire. Ce problème est particulièrement présent sur les tissus et se traduit visuellement par des sommets qui vibrent.

Les points de contacts sont détectés à la surface supportant l'objet. Pour les tissus

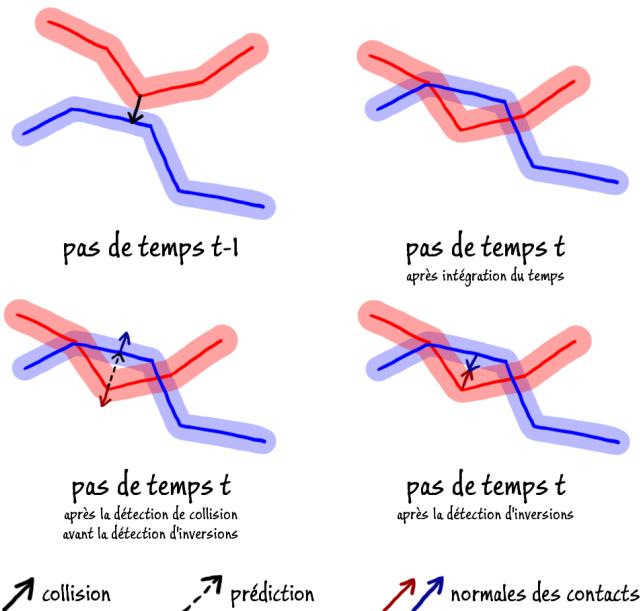


Figure 3.26 – Exemple d’une prédition de collision qui a besoin d’être reclassée après la détection d’inversion. Au pas de temps $t - 1$, deux tissus sont en contacts. Après l’intégration du temps, les deux tissus entrent en collision. Après la détection de collision, un sommet n’est pas détecté en collision mais en prédiction de collision car il est trop éloigné de l’autre tissu. La détection d’inversion est capable de corriger les normales des contacts et le point de contact a besoin d’être reclassé comme une collision pour ne pas être ignoré par la réponse physique.

cette surface ne représente pas une surface extérieure du tissu mais la couche centrale. Pour placer les points de contacts situés sur un tissu à sa surface, les points de contacts doivent être translatés dans la direction opposée de la normale du contact par une distance e (cf. Figure 3.28).

4.3 Évaluation des performances

Nous avons évalué les performances de notre algorithme de détection de collision pour les tissus. Nos simulations reposent sur une réponse physique basée positon [Bender et al., 2013]. La détection de collision est intégrée à notre *pipeline* et est exécutée sur une carte graphique AMD FirePro W9100 (2816 coeurs, 16 Go de mémoire) à 60 images par seconde. Une vidéo montrant certaines des scènes que nous avons testées est disponible à l’adresse suivante : <https://youtu.be/Qj6KC57Jhcw>. L’ensemble des trois algorithmes de détection de collision présentés dans le *pipeline* sont utilisés (basique, parcours sans pile de BVH et itératif).

Dans la scène présentée dans la figure 3.29, huit draps (composés chacun de 4.200 sommets) tombent sur une scène statique (composée de 9.000 sommets). La détection de collision prend en moyenne 5,3 millisecondes par pas de temps. Ce temps de calcul comprend les tests de collision entre les draps ainsi que les tests d’auto-collision sur les draps.

Dans la scène présentée dans la figure 3.30, un mannequin statique (composé de

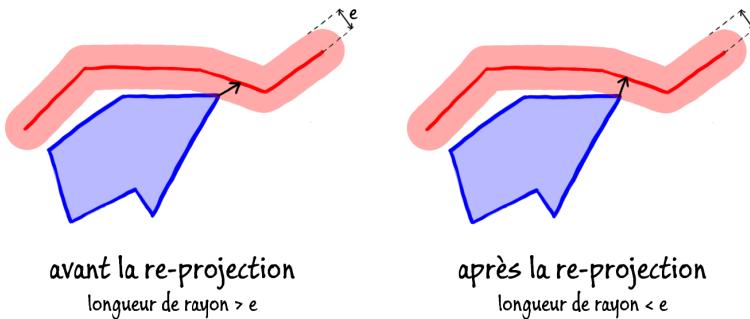


Figure 3.27 – Effet de la re-projection sur la longueur des rayons. Dans cet exemple la longueur de rayon est supérieure à e avant la re-projection, le rayon est donc classé comme une prédiction. Après avoir appliqué le re-projection la longueur de rayon devient inférieure à e , le rayon est classé comme une collision.

9.800 sommets) porte une robe (11.000 sommets), une veste (2.700 sommets) et une écharpe (3.800) sommets). La détection de collision prend en moyenne 4,3 millisecondes par pas de temps. Dans cette scène notre méthode est capable de gérer une scène complexe où trois couches de vêtements sont empilés. Jusqu'à 16.000 points de contacts sont détectés à chaque pas de temps entre les vêtements et avec le mannequin. L'écharpe repose sur la veste et touche la robe, la veste repose sur la robe et le mannequin et aucune interpénétration n'est présente durant la simulation. Les auto-collision sont correctement détectées sur la robe qui est composée de trois couches, ce qui résulte en une prise de volume entre les couches et la formation de plis.

La figure 3.31 présente un test de stress pour l'auto-collision. Un drap Carré tombe sur sa tranche sur un sol irrégulier. Notre méthode est capable de détecter les auto-collisions ayant lieu sur le drap pendant qu'il touche le sol ce qui provoque l'apparition de plis. Jusqu'à 3.800 points de contacts sont détectés par pas de temps (en incluant les collisions entre le drap et le sol et les auto-collision sur le drap). La détection de collision et la réponse physique tournent en temps-réel ce qui permet une interaction directe avec un utilisateur. Dans cet exemple, une interaction bi-manielle est réalisée avec un Razer Hydra permettant des déplacements en trois dimensions.

4.4 Synthèse et perspectives

Nous avons présenté une méthode permettant de réaliser la détection de collision avec des tissus en utilisant des méthodes de ray-tracing. Notre méthode est capable de réaliser la détection de collision entre des tissus et des objets volumiques ainsi que la détection de collision entre des tissus (dont l'auto-collision). De plus, une méthode de détection d'inversion permet de corriger les erreurs introduites par les simulations discrètes. Cette méthode s'insère parfaitement dans le *pipeline* sur GPU que nous avons présenté et est capable de produire des résultats en temps-réel. Dans nos cas d'utilisations, des utilisateurs sont capables d'interagir en direct avec les simulations.

Comme dans toute simulation discrète, les vitesses des objets ont besoin d'être limitées en fonction de la fréquence de la simulation (même avec la présence de l'algorithme de détection d'inversions). Il serait possible d'ajuster la direction des rayons pour réaliser une détection de collision continue. En lançant chaque rayon dans la

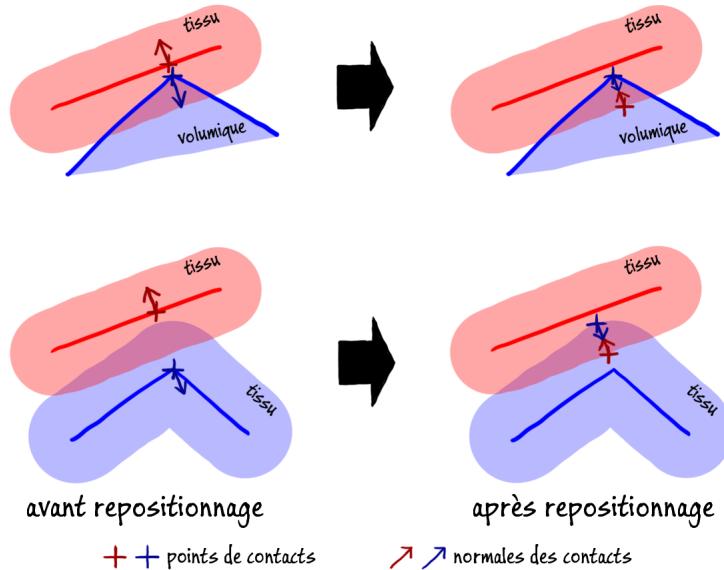


Figure 3.28 – Exemple de repositionnement des points de contacts sur le cas tissu/volumique (haut) et tissu/tissu (bas). Les points de contacts sont déplacés dans la direction opposées des normales pour les faire correspondre aux surfaces extérieures.

direction de la trajectoire des sommets, il serait possible de localiser dans le temps le moment exact des collisions. Cependant, nous serions alors forcés à utiliser des tests unitaires continus entre les primitives (algorithme d'intersection rayon/triangle continu), ce qui augmenterait le coût de notre méthode et limiterait ses cas d'utilisations en temps réels.

La méthode de détection d'inversion pourrait être améliorée. En effet, notre méthode presuppose que la configuration géométrique des objets est correcte au pas de temps précédent. Ceci classe notre méthode dans la catégorie des méthodes basées historique pour la résolution des inversions. Le problème est que des inversions peuvent rester incorrigées entre deux pas de temps s'il n'existe pas de solution globale (ce qui est possible en cas d'environnements hyper-constraints) ou si la réponse physique échoue (par contrainte de temps par exemple). Un exemple où notre méthode va échouer est le cas de la simulation de vêtements sur un humain virtuel dont les animations contiennent des auto-intersections (par exemple aux niveaux des genoux ou des coudes). Avec de telles animations les vêtements sont forcés à être en état d'auto-collision durant un certain temps, notre méthode ne sera ensuite pas capable de séparer les tissus. Pour résoudre ce type de problème Baraff et al. [Baraff et al., 2003] ont proposé une méthode pour calculer un état libre d'auto-pénétration à partir de n'importe quel état arbitraire (c'est à dire sans historique). Cette méthode utilise le résultat de la détection de collision et corrige les auto-collision dont la surface est la plus faible. Pour ne pas reposer sur un état global, [Volino and Magnenat-Thalmann, 2006] ont proposé de minimiser la longueur des intersections. Il serait intéressant de chercher à combiner notre méthode de détection d'inversion basée historique qui est extrêmement peu coûteuse avec une méthode n'utilisant pas l'historique qui ne serait utilisée que ponctuellement pour obtenir une méthode de détection de collision tolérante aux erreurs tout en restant

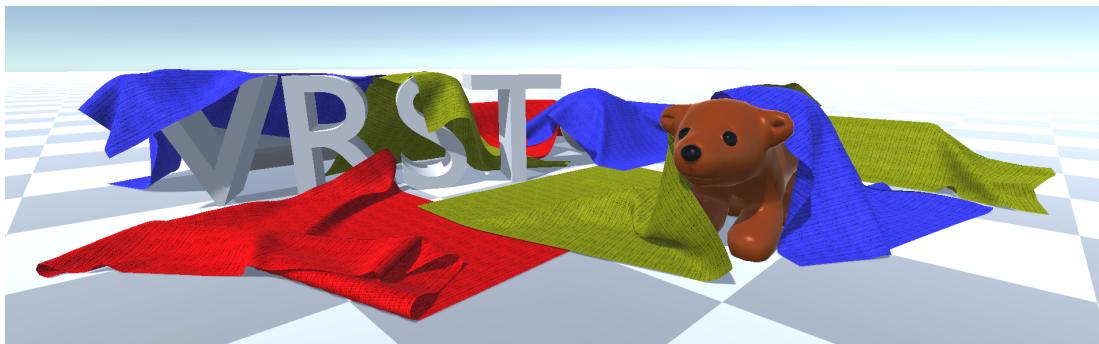


Figure 3.29 – Huit draps (chacun composé de 4.200 sommets) tombent sur une scène statique (comprenant des objets non-convexes). La détection de collision prend en moyenne 5,3 millisecondes par pas de temps.

performante.

Il serait aussi intéressant de gérer des tissus à épaisseurs variables. Dans notre implémentation, les tissus ont une taille constante (sur l'intégralité du tissu et dans le temps). Il serait possible de gérer des tissus avec des épaisseurs variables. Dans le contexte de simulation de tissus multi-échelles, une épaisseur variable pourrait être utilisée au niveau de simulation macroscopique pour représenter la compression du tissu. Dans [Müller and Chentanez, 2010], les auteurs proposent de simuler les tissus à deux niveaux : au niveau global et au niveau pli. Le niveau global simule le comportement général du tissu et le niveau pli simule la formation des plis sur le tissu. Une contrainte est ajoutée sur le niveau pli pour empêcher les sommets de dépasser l'épaisseur du tissu simulé au niveau global pour éviter les auto-collisions. Il serait possible avec notre méthode de faire varier l'épaisseur des tissus au niveau global en fonction des plis formés au niveau pli (cf. Figure 3.32). Ceci permettrait la formation de plis avec une plus grande amplitude.

5 Conclusion

Nous avons présenté dans ce chapitre un *pipeline* permettant de réaliser la *narrow-phase* sur GPU à l'aide d'algorithmes de lancer de rayon. Notre *pipeline* permet de modéliser la chaîne de traitement qui sélectionne quels rayons lancer avec quel algorithmes et permet d'implémenter la méthode de détection de collision de Hermann et al. Nous avons ensuite proposé de multiples améliorations permettant de généraliser ou d'améliorer la précision du *pipeline*.

Nous avons proposé une méthode de prédiction qui permet d'ajouter la mesure de distance inter-objets à notre *pipeline*. Notre méthode de prédiction de collision permet de réaliser des mesures de distances entre les objets sur les sommets. Cette mesure s'intègre parfaitement dans le *pipeline* au moment du lancer de rayon avec un faible coût. Ces prédictions peuvent ensuite être utilisées pour réaliser un *culling* afin de réduire la quantité de calcul.

La re-projection des rayons permet d'améliorer la fiabilité des points de contacts générés par notre *pipeline*. Ce traitement s'insère immédiatement après le lancer de rayon dans le *pipeline*. En modifiant la direction des normales des points de



Figure 3.30 – Une robe à trois couches, une veste et une écharpe reposent sur un mannequin. La détection de collision prend en moyenne 4,3 millisecondes par pas de temps.

contacts, nous garantissons le respect de la MTD à un niveau local. Nos tests ont montré que l'utilisation de la re-projection des rayons permet de réduire la quantité d'interpénétration globale au sein de nos simulations.

Nous avons généralisé la méthode de Hermann et al. pour y intégrer un autre modèle de représentation : la représentation surfacique. Ce type de représentation est largement utilisé pour la simulation de tissus. Cette généralisation permet d'intégrer à notre *pipeline* la détection de collision pour les tissus et autres types d'objets surfaciques. Grâce à cette méthode, nous sommes capables de réaliser la détection de collision entre des objets volumiques et des tissus ainsi de la détection de collision entre des tissus et l'auto-collision en temps-réel.

Toutes ces contributions ont permis de généraliser et de rendre plus fiable notre *pipeline* de détection de collision. Dans le chapitre suivant, nous nous intéressons à l'amélioration des performances en exploitant la cohérence temporelle.

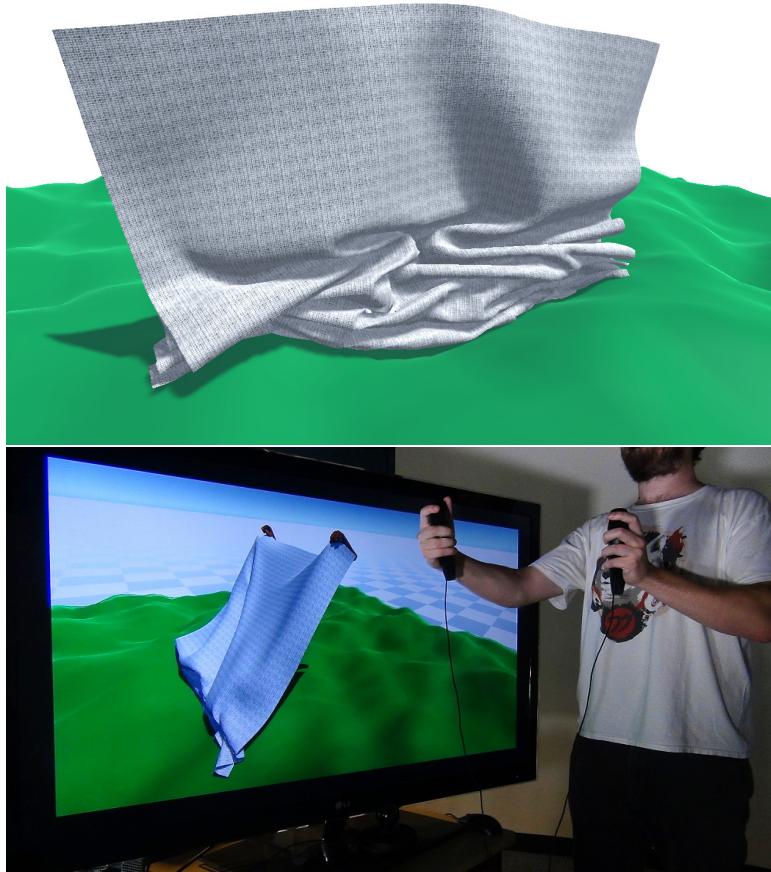


Figure 3.31 – Test de stress, un drap tombe sur sa tranche sur un sol irrégulier (haut). Dans ce scénario, des auto-collisions complexes ont lieu et notre méthode les détecte correctement. Cette scène peut être manipulée par un utilisateur (bas). Notre méthode tourne en temps-réel et permet donc l’interaction directe avec un utilisateur.

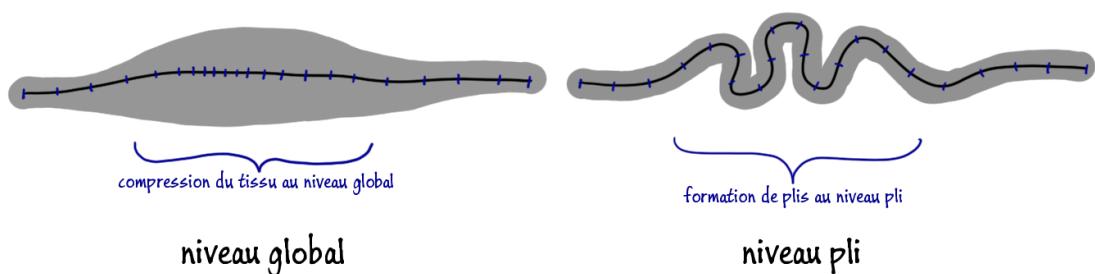


Figure 3.32 – Exemple d’épaisseur variable sur une simulation multi-échelle. Au niveau global le tissu subit une compression, nous pouvons alors augmenter l’épaisseur du tissu pour prendre en compte l’espace qui sera occupé par les plis qui seront simulés au niveau pli.

Exploitation de la cohérence temporelle

4

Dans ce chapitre, nous présentons notre méthode permettant d'exploiter la cohérence temporelle [Lehericey et al., 2013a, Lehericey et al., 2013b]. Nous commençons par présenter notre algorithme de lancer de rayon itératif (cf. Section 1) à bas coût qui est valide tant que les objets ont subi de faibles déplacements relatifs. Nous présentons ensuite une mesure de déplacement relatif pour les objets rigides (cf. Section 2) qui nous permet de décider quand nous pouvons utiliser notre algorithme de lancer de rayon itératif. Nous montrons ensuite comment l'utilisation de prédiction permet de réduire le nombre d'erreurs de détection lors de l'utilisation de notre méthode (cf. Section 3). Nous présentons enfin une mesure de déplacement relatif pour les objets déformables (cf. Section 4) qui nous permet d'utiliser notre méthode itérative avec des objets déformables.

1 Lancer de rayon itératif

L'idée principale de notre algorithme de lancer de rayon itératif est de chercher le nouveau point d'intersection entre le rayon et l'objet en démarrant la recherche à partir d'un ancien point d'intersection. Dans notre cas, cet ancien point d'intersection provient du résultat du lancer de rayon réalisé pour la détection de collision au pas de temps précédent. En pratique, il suffit de considérer que l'ancien résultat est l'identifiant du triangle appartenant au maillage de triangles qui a été touché par le rayon.

Notre méthode de lancer de rayon itérative peut commettre des erreurs. Ces erreurs sont principalement dues au fait que notre méthode réalise une recherche locale du point d'intersection. Nous considérons que notre algorithme de lancer de rayon donne un résultat erroné lorsque celui-ci diffère d'un algorithme de lancer de rayon complet (non-itératif) utilisé comme référence. Nous classons les erreurs en trois types :

- Faux-négatif : notre algorithme itératif n'a détecté aucune collision alors qu'il y en a une.
- Faux-positif : notre algorithme itératif a détecté une collision alors qu'il n'y en a pas.
- Erreur : notre algorithme itératif a détecté une collision mais celle-ci n'est pas localisée au bon endroit.

En mettant à jour de manière itérative les rayons au fur et à mesure de la simulation, les erreurs risquent de s'accumuler. Pour un rayon donné, si une erreur s'insère dans le processus de mise à jour, cette erreur sera conservée tant que le rayon est mis à jour par l'algorithme itératif au lieu d'être recalculé. Pour éviter de maintenir les erreurs

trop longtemps, nous proposons d'utiliser notre algorithme itératif de lancer de rayon lorsque les déplacements relatifs entre les deux objets ne dépassent pas un certain seuil depuis la dernière utilisation d'un algorithme complet de lancer de rayon. Nous appellerons ce seuil *seuilDéplacement*. Si les déplacements relatifs entre les objets dépassent *seuilDéplacement*, alors un algorithme complet de lancer de rayon est utilisé et la mesure de déplacement est remise à zéro.

Dans cette section, nous présentons notre algorithme de lancer de rayon itératif. Nous commençons par montrer comment il est possible de parcourir un maillage de triangles en navigant à travers les arêtes (cf. Section 1.1). Nous présentons ensuite un algorithme d'intersection rayon/triangle (cf. Section 1.2) permettant de localiser l'arête la plus proche d'un rayon donné pour pouvoir réaliser le parcours à travers les arêtes. Nous présentons ensuite une optimisation permettant de réduire le coût des rayons prédictifs (cf. Section 1.3) dans notre *pipeline* pour améliorer les performances. Enfin, nous montrons comment réduire le coût du stockage de notre méthode (cf. Section 1.4) pour éviter d'utiliser un stockage quadratique.

1.1 Intersection avec un maillage de triangles

Notre méthode d'intersection entre un rayon et un maillage de triangles repose sur une liste d'adjacence entre les triangles via les arêtes. Pour chaque triangle, chaque arête stocke l'identifiant du triangle se situant de l'autre côté de cette arête. Pour un rayon donné, l'algorithme itératif démarre du triangle en intersection au pas précédent et cherche à suivre le chemin entre l'ancien point d'intersection et le nouveau point d'intersection sur la surface du maillage.

L'algorithme 1 présente notre algorithme itératif de lancer de rayon. Cet algorithme prend en entrée le rayon à lancer et l'identifiant du triangle touché par ce rayon au pas de temps précédent. Le rayon est lancé sur l'ancien triangle et deux cas se présentent :

- Le rayon touche le triangle. L'algorithme s'achève et le nouveau point d'intersection est trouvé.
- Le rayon ne touche pas le triangle. Nous sélectionnons l'arête la plus proche du rayon (cf. Section 1.2) et récupérons le triangle se trouvant de l'autre côté de cette arête avec la liste d'adjacence. Nous réitérons ensuite l'algorithme.

Tant que le rayon ne touche pas un triangle, l'algorithme va itérer sur les triangles voisins jusqu'à ce qu'une intersection soit trouvée.

La figure 4.1 montre un exemple d'exécution de l'algorithme. En (a), l'intersection entre un rayon et un maillage de triangles est localisée avec un algorithme de lancer de rayon complet. En (b), nous démarrons la recherche à partir de l'intersection précédente, nous suivons ensuite les triangles voisins (en traversant les arêtes marquées en vert) jusqu'à ce que la nouvelle intersection soit trouvée.

Cet algorithme peut entrer dans une boucle infinie si le rayon ne touche plus l'objet ou si le chemin entre l'ancien point d'intersection et le nouveau point d'intersection traverse une concavité. Un exemple de rayon piégé dans une concavité est présenté dans la figure 4.2. En (a), l'intersection est calculée à l'aide d'un algorithme de lancer de rayon complet. En (b), l'algorithme itératif de lancer de rayon est utilisé mais une concavité dans le chemin entre l'ancien et le nouveau point d'intersection empêche l'algorithme de converger.

Algorithme 1 Intersection rayon/maillage de triangles itératif

```

1: fonction ITRAYTRIINTERSECTION(Rayon rayon, Triangle triangle)
2:   pour i = 1 → maxIt faire                                ▷ limite le nombre d'indirections
3:     intersection ← lance rayon sur triangle
4:     si rayon touche triangle alors
5:       retourner collision(triangle)
6:     sinon
7:       arête ← arêteLaPlusProche(triangle, rayon)
8:       triangle ← triangleAdjacent(triangle, arête)
9:     fin si
10:   fin pour
11:   retourner intersectionNonTrouvée
12: fin fonction

```

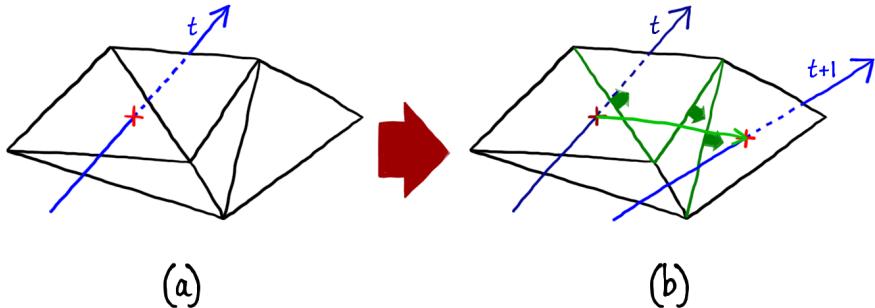


Figure 4.1 – Exemple de lancer de rayon itératif sur un maillage de triangles.

La solution exacte pour empêcher l'algorithme de rentrer dans une boucle infinie est de tester si le triangle courant a déjà été visité. Cette solution détecterait aussitôt la possible entrée dans une boucle infinie mais cette solution serait coûteuse (en temps de calcul et en mémoire). Une solution approchée plus légère que nous avons adoptée est de limiter le nombre d'itérations avec une constante *maxIt*. L'équation 4.1 donne la valeur de *maxIt* en fonction de la taille moyenne des triangles, du seuil de déplacement utilisé et un seuil de confiance d'au moins 1. Le seuil de confiance est présent pour prendre en compte la variabilité des distances parcourues sur chaque triangle, chaque parcours sur un triangle est généralement plus court que sa longueur. En cas de sous-estimation de la constante *maxIt* notre algorithme de lancer de rayon va produire des erreurs sous forme de faux-négatifs, c'est-à-dire ne pas détecter de collision alors qu'il y en a une. Pour éviter ce type d'erreurs, le seuil de confiance peut être fixé arbitrairement grand.

$$maxIt = confiance \times \frac{seuilD\acute{e}placement}{tailleMoyenneDesTriangles} \quad (4.1)$$

1.2 Intersection avec un triangle

Pour parcourir le maillage de triangles, nous avons besoin d'un algorithme pour calculer le point d'intersection entre un rayon et un triangle. Pour réaliser ce calcul, nous pouvons utiliser l'algorithme de Möller et al. [Möller and Trumbore, 2005]. Cependant,

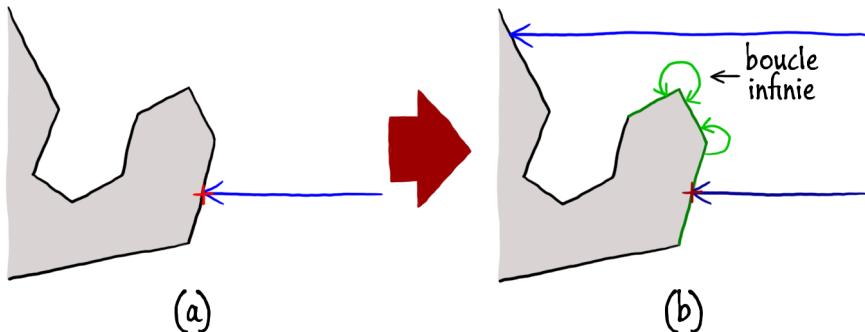


Figure 4.2 – L’algorithme itératif est piégé dans une concavité dans cet exemple.

lorsque le rayon rate le triangle, nous voulons savoir quelle est l’arête la plus proche du rayon. Nous pouvons simplifier ce problème en nous intéressant à l’intersection entre le rayon et le plan contenant le triangle, nous avons alors deux cas :

- Le rayon est parallèle au plan. Ceci est un cas dégénéré et dans ce cas le rayon est abandonné.
- Le rayon n’est pas parallèle au plan. L’intersection entre le rayon et le plan est un point.

Si le point d’intersection est à l’intérieur du triangle, alors le rayon et le triangle sont en intersection. En cas de non-intersection, nous considérons que l’arête la plus proche du point d’intersection est l’arrête la plus proche du rayon. La figure 4.3.a montre les différentes zones présentes sur le plan d’un triangle. La zone A correspond à l’intérieur du triangle. Si le point d’intersection est dans la zone A, le rayon et le triangle sont en intersection. Si le point d’intersection est à l’extérieur de la zone A, nous devons sélectionner l’arête la plus proche (numérotée ici 1, 2 ou 3). Dans les zones B_x, l’arête la plus proche est l’arête x (par exemple tous les points de la zone B₂ sont plus proches de l’arête 2 que des arêtes 1 et 3). Dans les zones C_{xy}, il existe une ambiguïté, les arêtes x et y sont à équidistance de tous les points appartenant à C_{xy}. Dans ce cas, nous pouvons choisir arbitrairement l’une des deux arêtes (par exemple on peut attribuer tous les points appartenant à la zone C₂₃ à l’arête 2).

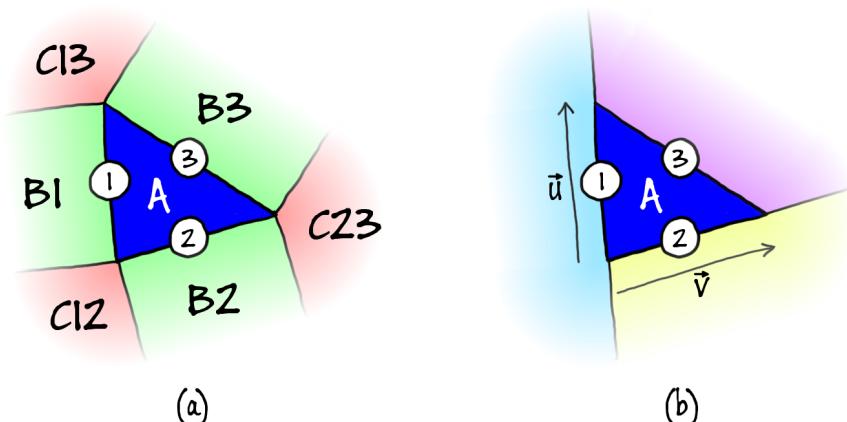


Figure 4.3 – Plan d’un triangle avec les arêtes numérotées.

Pour réaliser le test d’intersection entre un rayon et un triangle et, en cas de non-

intersection, localiser l'arête la plus proche, nous proposons une méthode basée sur l'algorithme de Möller et Trumbore permettant de réaliser ces deux tâches en une seule opération. L'algorithme de Möller et Trumbore calcule le point d'intersection dans l'espace des objets. Le résultat est un vecteur (t, u, v) où t est la coordonnée du point d'intersection sur la droite supportant le rayon et u, v est la coordonnée du point d'intersection sur le plan contenant le triangle. Ce vecteur doit satisfaire quatre inégalités pour que l'intersection soit sur le rayon et sur le triangle :

- $u \geq 0, v \geq 0$ et $u + v \leq 0$ testent si le point d'intersection est à l'intérieur du triangle.
- $t \geq 0$ teste sur la droite supportant le rayon si le point d'intersection est sur le rayon (t correspond à la longueur du rayon).

Nous proposons d'utiliser l'algorithme 2 pour savoir si le rayon est en intersection avec le triangle et en cas de non-intersection, quelle est l'arête la plus proche. Cet algorithme s'insère à la fin de l'algorithme de Möller et Trumbore et remplace les conditions finales permettant de savoir si il y a intersection ou non. Cet algorithme réalise un choix arbitraire en cas d'ambiguïté de l'arête la plus proche qui correspond à la figure 4.3.b.

Algorithme 2 Intersection rayon/triangle

```

1: fonction LOCALISERINTERSECTION( $u, v, t$ )
2:   si  $u < 0$  alors
3:     le rayon est plus proche de l'arête 1
4:   sinon si  $v < 0$  alors
5:     le rayon est plus proche de l'arête 2
6:   sinon si  $u + v > 1$  alors
7:     le rayon est plus proche de l'arête 3
8:   sinon si  $t >= 0$  alors
9:     le rayon est en intersection avec le triangle           ▷ collision
10:    sinon
11:      le triangle est derrière le rayon, arrêter le parcours      ▷ prédition
12:    fin si
13: fin fonction

```

1.3 Factorisation des prédictions avec l'algorithme itératif

Dans notre *pipeline* de détection de collision, nous lançons deux rayons par sommet : un rayon intérieur pour détecter les collisions et un rayon extérieur pour réaliser des prédictions. Le rayon intérieur et le rayon extérieur partagent le même point d'origine et sont contenus dans la même droite, la seule différence est la direction qui est opposée.

Dans le test d'intersection rayon/triangle, pour un même triangle, le rayon intérieur a pour coordonnée d'intersection (t_i, u_i, v_i) et le rayon extérieur a pour coordonnée d'intersection (t_e, u_e, v_e) . Les contraintes de symétries entre les deux rayons réalisent l'ensemble d'égalités suivantes :

$$\begin{cases} t_i = -t_e \\ u_i = u_e \\ v_i = u_e \end{cases}$$

Ce qui signifie que l'algorithme 2 classifie identiquement les deux rayons sauf dans les deux dernières conditions. Ces deux dernières conditions étant les cas d'arrêt de l'algorithme 1, cela signifie que le parcours dans cet algorithme des rayons intérieurs et extérieur sera le même et seul le résultat final change. Suite à ces observations, nous proposons de factoriser les deux lancers de rayons itératifs avec un seul lancer. Nous réalisons uniquement le lancer du rayon intérieur. Si le parcours du maillage se termine par le cas **collision** dans l'algorithme 2, cela signifie que le sommet est en collision avec le triangle. Si le parcours se termine par le cas **prédition** dans l'algorithme 2, le sommet n'est pas en collision avec le triangle mais celui-ci est conservé comme prédition. Cette factorisation permet de diviser par deux le nombre de lancers de rayons à réaliser avec l'algorithme itératif.

1.4 Stockage des données temporelles

Cette méthode nécessite de stocker le résultat du lancer de rayon du pas précédent. Ce résultat correspond à l'identifiant d'un triangle et doit être stocké pour chaque sommet de chaque objet pour chaque paire produite par la *broad-phase*. Cette méthode nécessiterait un stockage quadratique en mémoire car il faut prendre en compte toutes les combinaisons d'objets possibles.

Pour réduire cette complexité de stockage, il est possible d'utiliser la cohérence spatiale. La cohérence spatiale indique que chaque point de l'espace n'est occupé que par un seul objet (si la simulation est correcte). Par conséquent, il n'y a pour chaque sommet qu'un seul objet en face de celui-ci. Suite à cette observation, il n'est pas justifié de stocker pour chaque sommet les identifiants des triangles les plus proches pour chaque objet, il suffit de conserver l'identifiant du triangle le plus proche (correspondant à l'objet le plus proche).

Nous avons utilisé cette idée et, dans notre méthode, nous n'avons conservé qu'un seul résultat de lancer de rayon par sommet (deux dans le cas des tissus pour prendre en compte chaque côté). Ceci permet de réduire le coût de stockage du résultat précédent à un coût linéaire en fonction du nombre de sommets.

1.5 Synthèse

Nous avons présenté un algorithme de lancer de rayon itératif qui permet de mettre à jour de manière incrémentale le résultat d'un lancer de rayon au pas de temps précédent. Cet algorithme prend en entrée le résultat du lancer de rayon au pas précédent et le met à jour en suivant sur la surface de l'objet le déplacement du point d'intersection. Cette méthode ne fonctionne que si le chemin entre l'ancien point d'intersection et le nouveau point d'intersection ne contient pas de concavité. Dans le cas contraire, notre méthode de lancer de rayon ne sera pas capable de localiser le nouveau point d'intersection et indiquera de manière erronée que le rayon n'est pas en intersection avec l'objet. Nous appelons ce phénomène la perte de rayon car celui-ci empêche notre algorithme de lancer de rayon itératif de réaliser la détection de collision dans les pas de

temps futurs car notre méthode a besoin de connaître en permanence le résultat du pas de temps précédent pour fonctionner. La perte de rayon a pour conséquence de produire des faux-négatifs dans la détection de collision. Cependant notre utilisation de lancer de rayon est très spécifique, nous lançons des rayons dans les zones d'interpénétration entre deux objets et dans leur voisinage. De plus, dans le contexte de la simulation physique, les zones d'interpénétration sont en permanence minimisées. Cela signifie que les zones d'interpénétrations sont petites et que les rayons sont courts. Dans ce contexte, même si deux objets sont non-convexes, il est fortement probable que les zones en interpénétration soient convexes. La figure 4.4 montre un exemple d'intersections convexes entre deux objets non-convexes. Dans ce contexte la perte de rayons sera faible.

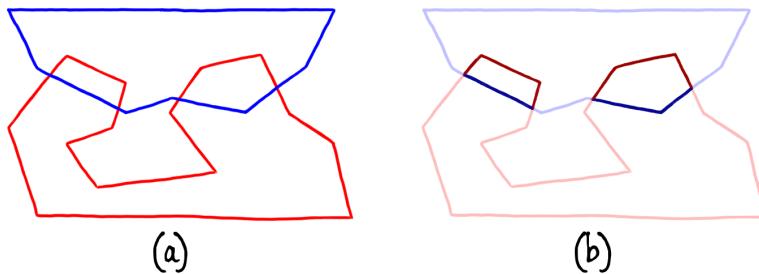


Figure 4.4 – L’intersection entre deux objets non-convexes peut être convexe. Ici, deux objets non-convexes sont en collision (a) et leur intersections sont convexes (b).

Une problématique restante est la perte de rayons, même si elle est faible, la perte s’accumule avec le temps. Celle-ci s’accumule car une fois que un rayon est perdu, l’algorithme de lancer de rayon itératif n’est pas capable de détecter quoi que ce soit avec ce rayon tant qu’un algorithme de lancer de rayon complet n’a pas été utilisé à nouveau. Nous pouvons tolérer un faible taux d’erreur lors de la détection de collision tant que suffisamment de points de contacts sont détectés pour réaliser une réponse physique acceptable. Pour éviter d’atteindre un taux d’erreurs trop élevé, nous proposons de restreindre l’utilisation de l’algorithme itératif de lancer de rayon aux cas où les déplacements relatifs entre les objets sont faibles.

Dans les prochaines sections, nous proposons deux métriques permettant de réaliser une mesure de déplacement relatif entre les objets. La première métrique se limite aux objets rigides tandis que la seconde métrique accepte les objets déformables. Avec ces métriques, nous évaluons l’apport en performances de l’algorithme lancer de rayon itératif dans notre *pipeline*. Nous montrons aussi l’apport de l’utilisation de prédictions dans la réduction du taux d’erreur.

2 Application aux objets rigides

Pour appliquer notre méthode de détection avec l’algorithme de lancer de rayon itératif sur des objets rigides, nous avons besoin d’une mesure de déplacement relatif entre objets rigides. Dans cette section, nous commençons par présenter une métrique permettant de mesurer la quantité de déplacements relatifs entre deux objets dans un intervalle de temps donné. Nous évaluons ensuite le gain en performance lors de

l'utilisation de l'algorithme de lancer de rayon itératif piloté par cette mesure.

2.1 Mesure de déplacements relatifs

Avec des objets rigides, aucune déformation interne n'est possible à l'intérieur d'un objet. Nous pouvons donc nous contenter de mesurer la quantité de déplacement uniquement en nous basant sur l'évolution de la transformation (translation et rotation) entre les deux repères locaux des objets.

$$\maxDis = \|t_{ni} - t\| + \text{angle}(q_{ni}, q) \times \text{rayonMax} \quad (4.2)$$

L'équation 4.2 donne une borne supérieure au déplacement relatif entre deux objets dans un intervalle de temps donné. L'intervalle de temps débute à l'instant de la dernière utilisation d'un algorithme de lancer de rayon complet (c'est-à-dire non-itératif) et se prolonge jusqu'au pas de temps courant. La mesure est un scalaire qui donne une borne supérieure de la plus grande amplitude de déplacement atteinte entre deux points appartenant respectivement à chaque objet de la paire d'objets. Dans cette équation, les variables sont :

- (t, q) est la transformation entre le repère du premier objet de la paire et le repère du second objet de la paire au pas de temps courant. t est le vecteur de translation et q est le quaternion de rotation.
- (t_{ni}, q_{ni}) est la transformation entre les deux objets de la paire au moment de la dernière utilisation d'un algorithme de lancer de rayon non-itératif.
- $\text{angle}(q_1, q_2)$ est l'angle entre les quaternions q_1 et q_2 .
- maxRadius est le rayon du plus grand objet de la paire (le rayon d'un objet correspond à la distance entre le point d'origine du repère local de l'objet et le sommet le plus éloigné du point d'origine du repère local de l'objet).

$\|t_{ni} - t\|$ correspond à la quantité de déplacement due aux translations et $\text{angle}(q_{ni}, q) \times \text{rayonMax}$ est une borne max au déplacement due aux rotations. Lorsque \maxDis dépasse le seuil seuilDéplacement un algorithme de lancer de rayon non-itératif est utilisé, dans les autres cas l'algorithme de lancer de rayon itératif est utilisé pour réaliser le lancer de rayon.

2.2 Évaluation

Nous avons évalué les performances de la détection de collision à l'aide de l'algorithme de lancer de rayon itératif, les deux scènes tests que nous avons utilisées dans le chapitre précédent lors de l'évaluation des prédictions de collisions (cf. Figure 3.11). Dans la première scène, 512 objets non-convexes tombent simultanément sur un sol plat et jusqu'à 7.300 paires d'objets sont testées dans la *narrow-phase*. Dans la seconde scène 500 objets sont ajoutés progressivement dans la simulation et jusqu'à 10.000 paires d'objets sont testées dans la *narrow-phase* en fin de simulation. Les deux scènes sont simulées à 60 Hz, la première scène dure 10 secondes tandis que la seconde dure 50 secondes. Une vidéo de ces deux scènes est disponible à l'adresse suivante : <https://youtu.be/H6HUiSrLhXk>.

L'environnement expérimental a été développé avec Bullet Physics¹, la *broad-phase* et la réponse physique sont exécutées sur CPU. Notre *narrow-phase* est exécutée sur GPU à l'aide d'OpenCL². Cet environnement génère de nombreux transferts de mémoire entre le CPU et le GPU car l'ensemble des données nécessaires à la simulation physique doit être copié à chaque pas de temps entre le CPU et le GPU. Avec une simulation physique entièrement implémentée sur GPU, ces coûts de transferts mémoire seraient entièrement éliminés. Pour ne pas inclure le coût des transferts mémoire dans nos mesures de temps de calcul, nous avons exclusivement mesuré le temps de calcul de lancer de rayon sur GPU. Les tests ont été réalisés sur une carte graphique Nvidia GTX 660 (960 coeurs, 2 Go de mémoire) et un processeur central Intel Core i5 (4 coeurs).

Premièrement, nous avons évalué le gain obtenu en utilisant notre algorithme de lancer de rayon itératif. Pour cela, nous avons comparé le temps de calcul du lancer de rayon dans nos scènes en faisant varier deux conditions. La première condition est sur la présence de notre algorithme de lancer de rayon itératif, nous avons testé deux cas :

- Sans notre algorithme de lancer de rayon itératif.
- Avec notre algorithme de lancer de rayon itératif.

La seconde condition est sur les algorithmes de lancer de rayon complets utilisés (cf. Chapitre 3), nous avons testé deux algorithmes de lancer de rayon :

- L'algorithme de lancer de rayon basique.
- L'algorithme de lancer de rayon avec parcours sans pile de BVH.

Les figures 4.5 et 4.6 donnent les mesures de performance pour les deux scènes.

Dans les résultats de la première scène nous pouvons observer trois phases différentes notées A, B et C dans la figure 4.5, les résultats dans ces phases sont présentés dans le tableau 4.1 :

Phase A : Les objets sont en chute libre, il n'y a pas de collision mais les objets sont assez proches pour que la *broad-phase* détecte un grand nombre de paires. Dans ce cas l'utilisation de l'algorithme de lancer de rayon itératif accélère par un facteur 3,6 l'algorithme de lancer de rayon basique et a un impact négligeable sur l'algorithme de lancer de rayon avec parcours sans pile de BVH.

Phase B : Les objets touchent le sol. Cette phase est un test de stress, les mesures de performances montrent toutes un pic du temps de calcul. À ce moment la cohérence temporelle est plus faible car les objets ne réagissent pas tous de manière identique au choc ce qui augmente les quantités de déplacements relatifs entre les paires d'objets. Cette faible cohérence est mise en évidence par une baisse du pourcentage d'utilisation de l'algorithme de lancer de rayon itératif. Dans ce contexte nous avons analysé le pic du temps de calcul et avons constaté que l'utilisation de l'algorithme itératif accélère les calculs d'un facteur 1,5 et 1,3 respectivement avec l'algorithme de lancer de rayon basique et l'algorithme de lancer de rayon avec parcours sans pile de BVH. Notre méthode accélère donc les calculs même si le taux d'utilisation de l'algorithme de lancer de rayon itératif chute à 50%.

Phase C : Les objets sont tous tombés, ils ont maintenant de plus faibles vitesses et la cohérence temporelle est plus élevée (le taux d'utilisation de l'algorithme de

1. <http://bulletphysics.org/>
 2. <http://www.khronos.org/opencl/>

lancer de rayon itératif remonte). le facteur d'accélération moyen est 21,9 et 6,2 respectivement avec l'algorithme de lancer de rayon basique et l'algorithme de lancer de rayon avec parcours sans pile de BVH. Avec une cohérence temporelle plus élevée, nous obtenons un facteur d'accélération plus élevé

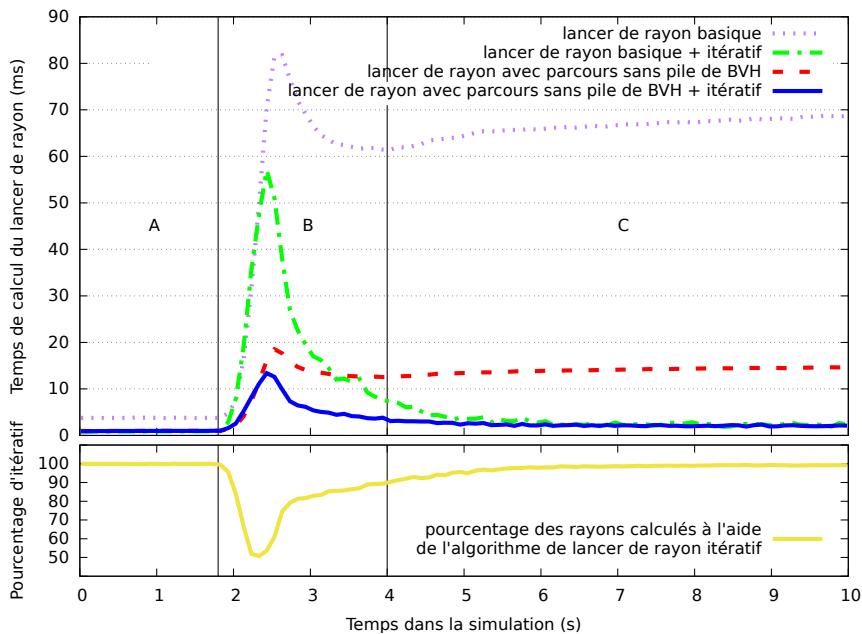


Figure 4.5 – Temps de calcul du lancer de rayon par pas de temps dans la première scène avec différents algorithmes de lancer de rayon et taux d'utilisation de l'algorithme de lancer de rayon itératif en cas d'utilisation ce celui-ci.

Algorithme de lancer de rayon	Moyenne en A	Pic en B	Moyenne en C
basique	3.78 ms	82.6 ms	66.2 ms
basique + itératif	1.04 ms	56.7 ms	3.0 ms
Facteur d'accélération	3.6	1.5	21.9
parcours sans pile de BVH	9.43 ms	18.5 ms	14.0 ms
parcours sans pile de BVH + itératif	9.71 ms	14.1 ms	2.27 ms
Facteur d'accélération	1.0	1.3	6.2

Table 4.1 – Temps moyen ou temps max d'exécution du lancer de rayon par pas de temps pour les trois phases de la scène 1

Dans la seconde scène les objets sont ajoutés progressivement et le temps de calcul augmente en fonction du nombre d'objets présents (cf. Figure 4.6). L'utilisation de l'algorithme de lancer de rayon itératif réduit fortement les temps de calculs. En fin de simulation le temps de calcul est divisé par 33,0 lorsque l'algorithme itératif est ajouté à l'algorithme de lancer de rayon basique et le temps de calcul est divisé par 19,0 lorsque l'algorithme de lancer de rayon itératif est ajouté à l'algorithme de lancer de rayon avec parcours sans pile de BVH. Dans tous les cas, les algorithmes de lancer de rayon complets utilisés seuls ne permettent que de simuler un nombre d'objets limité

de manière interactive et l'utilisation de notre algorithme de lancer de rayon itératif permet de fortement repousser cette limite.

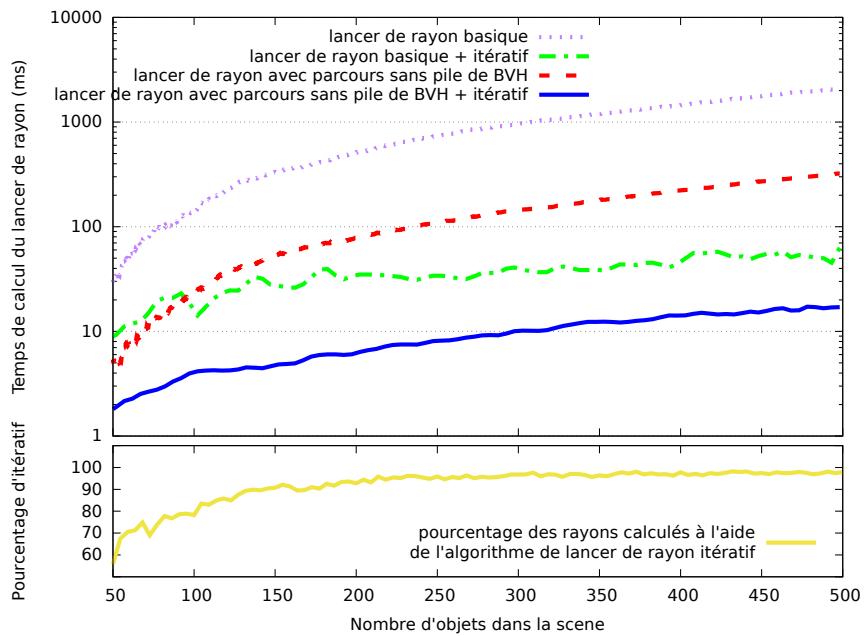


Figure 4.6 – Temps de calcul du lancer de rayon (échelle logarithmique) par pas de temps dans la seconde scène avec différents algorithmes de lancer de rayon et taux d'utilisation de l'algorithme de lancer de rayon itératif en cas d'utilisation ce celui-ci.

Secondement, nous avons comparé les performances de notre méthode de détection de collision avec une méthode existante qui est exécutée sur CPU : *GIMPACT*³. Cette comparaison a été réalisée uniquement avec la première scène test. Notre but est de montrer que notre méthode de détection de collision est plus performante que les méthodes existantes. La figure 4.7 donne les temps moyens de calcul par pas de temps de l'intégralité de la simulation physique. Le temps de calcul de l'intégralité de la simulation a été utilisé pour prendre en compte tous les impacts de notre *narrow-phase* sur le moteur physique (notre méthode produit par exemple un grand nombre de points de contacts qui doit être réduit dans la réponse physique). *GIMPACT* est un algorithme que ne peut être exécuté que sur CPU et notre méthode est conçue pour GPU. Pour pouvoir comparer les performances de *GIMPACT* et de notre méthode sans avoir des résultats biaisés par la différence de puissance de calcul des CPU et GPU, nous avons exécuté notre méthode sur CPU. Nous avons aussi mesuré les performances de notre méthode sur GPU pour montrer le gain obtenu avec ceux-ci. Nous avons testé notre méthode avec la combinaison de l'algorithme itératif et de l'algorithme basique de lancer de rayon ainsi qu'avec la combinaison de l'algorithme itératif et de l'algorithme de lancer de rayon avec parcours de BVH. Dans le cas de notre méthode, nous avons mesuré séparément le temps passé dans le calcul du lancer de rayon et dans le cas où notre méthode est exécutée sur GPU nous avons mesuré séparément le temps passé dans les transferts mémoire entre le CPU et le GPU. Les résultats mettent en évidence

3. <http://gimpact.sourceforge.net/>

les points-clés suivants :

- Notre méthode de détection de collision est bien plus rapide que *GIMPACT* que ce soit sur CPU ou GPU. La simulation finale est 8,2 fois plus rapide avec notre méthode itérative + lancer de rayon basique sur CPU que *GIMPACT* et 8,9 fois plus rapide avec notre méthode itérative + lancer de rayon avec BVH sur CPU que *GIMPACT*.
- L'implémentation GPU est plus rapide que l'implémentation CPU même en prenant en compte les coûts des transferts mémoire. Dans les deux cas le temps de calcul est divisé par 2,6.
- Dans le cas de notre méthode sur GPU, les performances de la simulation physique sont dominées par le reste de la simulation et par les coûts des transferts mémoire.

Il faut noter que les performances ont été mesurées sur l'intégralité de la simulation physique, elles montrent que notre méthode est plus rapide que *GIMPACT* mais ne font pas une comparaison complète car les performances précises de la *narrow-phase* n'ont pas été mesurées.

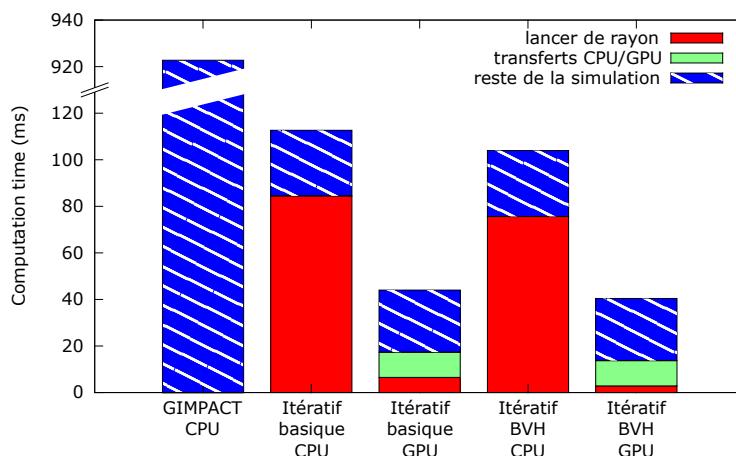


Figure 4.7 – Temps moyen passé dans les différentes étapes de la simulation à chaque pas de temps.

2.3 Synthèse

Nous avons proposé une mesure de déplacement entre objets rigides qui mesure une borne supérieure à la quantité de déplacements relatifs entre deux objets dans un intervalle de temps donné. Dans notre contexte de la détection de collision par lancer de rayon où nous avons un algorithme de lancer de rayon itératif qui ne peut être utilisé qu'en cas de faibles déplacements relatifs, nous utilisons cette mesure pour décider si l'algorithme itératif peut être utilisé ou non. À l'aide de cette mesure, nous montrons que notre algorithme de lancer de rayon itératif permet d'améliorer les performances. Dans nos tests, l'utilisation de l'algorithme de lancer de rayon itératif permet de diviser jusqu'à 19 et 33 fois le temps de calcul de lancer de rayon en fonction de l'algorithme de lancer de rayon complet utilisé. Comparé à *GIMPACT*, notre méthode résulte en

une simulation physique qui est entre 8 et 9 fois plus rapide.

Concernant la précision de la simulation physique, notre méthode introduit des erreurs dans la simulation. Ces erreurs sont principalement dues au fait que notre algorithme itératif n'est pas capable de détecter de nouvelles collisions, il ne peut mettre à jour que des collisions existantes. Cela signifie que toute collision apparaissant pendant l'utilisation de notre algorithme itératif sera ignorée jusqu'à ce qu'un algorithme complet soit utilisé. Pour réduire ce type d'erreurs, nous montrons dans la prochaine section comment utiliser les prédictions de collision pour éliminer ces erreurs.

3 Utilisation des prédictions pour la robustesse

Le principal problème de l'algorithme itératif est l'incapacité à détecter de nouvelles collisions. L'algorithme itératif met à jour les rayons du pas précédent mais ne peut pas en trouver de nouveaux. Lorsque de nouveaux sommets entrent en collisions, ils ne seront détectés que lors du calcul complet suivant. Ceci peut retarder la détection de collision dans une paire d'objets pendant plusieurs pas de temps, ce qui entraînera de plus grandes interpénétrations entre les objets. La première ligne de la figure 4.8 montre un exemple d'une telle situation. À l'instant $t = 0$, la détection de collision est réalisée avec un algorithme de lancer de rayon complet et aucune collision n'est détectée. Aux pas de temps suivants ($t = 1$ et $t = 2$) les rayons détectés précédemment sont mis à jour, comme les objets n'étaient pas en collision au pas de temps $t = 0$ il n'y a aucun rayon à mettre à jour. Dans ce cas notre méthode échoue à détecter les collisions. Au pas de temps $t = 3$ la quantité de déplacements relatifs entre les deux objets devient trop élevée et un algorithme de lancer de rayon complet est utilisé. Les collisions sont finalement détectées mais en retard et les objets sont maintenant en fortes interpénétrations.

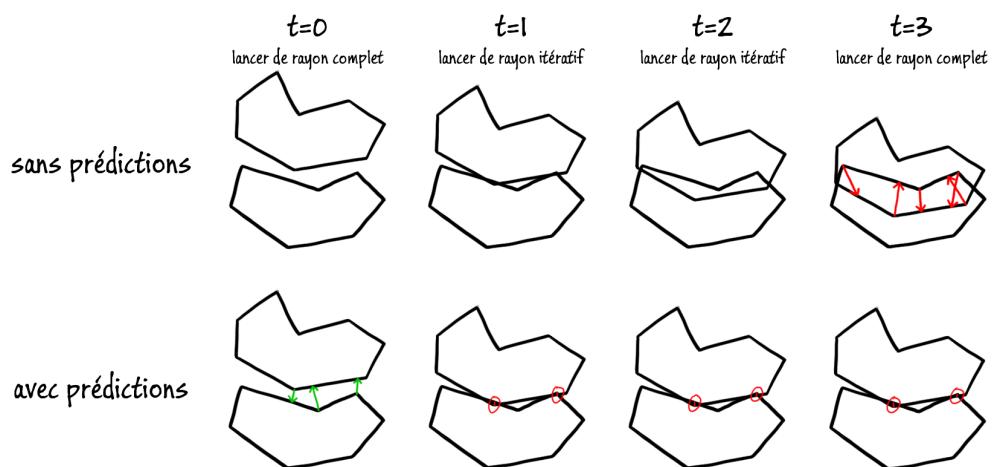


Figure 4.8 – Comparaisons du comportement de deux objets lors de la détection de collision avec et sans prédictions de collisions. En cas d'utilisation des prédictions, les collisions sont détectées plus tôt ce qui résulte en de plus faibles interpénétrations entre les objets.

3.1 Utilisation des prédictions avec la méthode itérative

Nous proposons de résoudre ce problème en utilisant les prédictions de collisions (cf. Chapitre 3). Les rayons détectant des prédictions peuvent être mis à jour à l'aide de l'algorithme de lancer de rayon itératif de la même manière que les rayons détectant des collisions sont mis à jour. Lorsque la longueur d'un rayon prédictif devient négative (ie. lorsque la surface touchée par un rayon passe derrière celui-ci), alors une nouvelle collision se produit à partir de la prédition. Nous proposons donc d'utiliser notre algorithme de lancer de rayon itératif avec les rayons de prédition pour pouvoir détecter les nouvelles collisions entre objets pendant les pas de temps itératifs. Pour chaque sommet, nous conservons en permanence soit un rayon prédictif soit un rayon de collision. Dans les pas de temps complets, les deux rayons sont calculés en permanence. Dans les pas de temps itératifs, seul un rayon est calculé (celui qui a été conservé au pas de temps précédent), celui-ci est mis à jour et si sa longueur devient négative alors sa classification est changée (les prédictions sont transformées en collisions et les collisions sont transformées en prédictions).

La seconde ligne de la figure 4.8 montre comment les prédictions peuvent être utilisées pour éviter de retarder la détection de collision. À l'instant $t = 0$ un algorithme de lancer de rayon complet est utilisé et aucune collision n'est détectée, des rayons de prédictions sont lancés à l'extérieur des objets et trois prédictions sont détectées. À l'instant $t = 1$ l'algorithme de lancer de rayon itératif est utilisé, les rayons ayant détectés des collisions et des prédictions sont mis à jour. Dans le cas présent deux des trois prédictions deviennent des collisions ce qui permet de détecter celles-ci correctement. Une réponse physique peut ensuite être appliquée pour empêcher les deux objets d'entrer en interpénétration plus profonde dans les pas de temps suivants.

En pratique nous pouvons factoriser le lancer de rayon itératif entre le rayon de collision et le rayon de prédition en une seule opération. L'algorithme 3 présente les modifications à apporter à l'algorithme de lancer de rayon itératif pour factoriser le lancer du rayon prédictif. Les lignes 6 et 7 sont ajoutées pour arrêter le parcours de la surface de l'objet lorsqu'une prédition est détectée.

3.2 Évaluation

Pour évaluer l'intérêt de l'utilisation des prédictions dans notre méthode, nous avons comparé le nombre d'erreurs dans deux simulations avec et sans utiliser les prédictions. Nous avons réalisé cette évaluation sur les deux scènes utilisées précédemment (cf. Figure 3.11). Dans la première scène, 512 objets non-convexes tombent simultanément et dans la seconde scène 500 objets non-convexes sont ajoutés progressivement. Pour chaque scène, nous avons mesuré le pourcentage de rayons erronés lors de l'utilisation de notre méthode de lancer de rayon itératif sans utiliser les prédictions et en utilisant les prédictions. Nous considérons qu'un rayon est erroné si le résultat de l'algorithme de lancer de rayon itératif diffère du résultat calculé par un algorithme de lancer de rayon standard (dans cet environnement de tests les rayons sont donc tous calculés deux fois).

Les Figures 4.9 et 4.10 donnent le pourcentage de rayon erroné pour chaque scène. Les résultats montrent que l'utilisation des prédictions de collision dans notre méthode

Algorithme 3 Intersection rayon/maillage de triangles itératif

```

1: fonction ITRAYTRIINTERSECTION(Rayon rayon, Triangle triangle)
2:   pour i = 1 → maxIt faire                                ▷ limite le nombre d'indirections
3:     intersection ← lancer rayon sur triangle
4:     si rayon touche triangle alors
5:       retourner collision(triangle)
6:     sinon si opposé(rayon) touche triangle alors
7:       retourner prediction(triangle)
8:     sinon
9:       arête ← arêteLaPlusProche(triangle, rayon)
10:      triangle ← triangleAdjacent(triangle, arête)
11:      fin si
12:    fin pour
13:    retourner intersectionNonTrouvée
14:  fin fonction

```

itératif permet de fortement réduire le nombre d'erreurs (jusqu'à 91%). Bien que le taux d'erreurs chute fortement, des erreurs sont toujours présentes dans les simulations. Les erreurs restantes sont majoritairement des faux-négatifs (~90%) et ceux-ci sont dues à plusieurs facteurs :

- Les prédictions de collision ne fournissent pas toujours de prédictions ou peuvent produire des prédictions erronées, il est possible que par exemple qu'un petit objet passe entre les rayons prédictifs.
- Notre algorithme de lancer de rayon itératif est soumis à une perte de rayons, il peut perdre des rayons dans certaines circonstances ce qui conduit à l'introduction de faux-négatifs.
- Pour chaque sommet, nous ne conservons que la prédition la plus courte (ou deux pour les tissus) pour réaliser le lancer de rayon itératif (cf. Section 1.4 sur le stockage des données temporelles). Si les prédictions abandonnées entrent en collision, elles ne sont pas détectées. Ces prédictions abandonnées peuvent donc conduire à des faux-négatifs.

Pour évaluer l'impact sur les simulations physiques de cette réduction du nombre d'erreurs, nous avons mesuré la quantité d'interpénétration présente tout au long des simulations. La mesure de la quantité d'interpénétration a été réalisé en mesurant la somme totale des tous les rayons détectant des collisions tout au long des simulations. Cette mesure montre que l'utilisation des prédictions de collisions dans notre méthode permet de réduire de respectivement 45% et 56% la quantité d'interpénétrations dans la première et la seconde scène. Cela signifie que l'utilisation des prédictions a bien un effet notable sur la simulation physique.

3.3 Synthèse

Avec notre algorithme itératif nous avons introduit des erreurs lors de la détection de collision principalement sous forme de faux-négatifs (environ 95% des erreurs). En injectant les prédictions de collisions issues des rayons extérieurs dans la détection

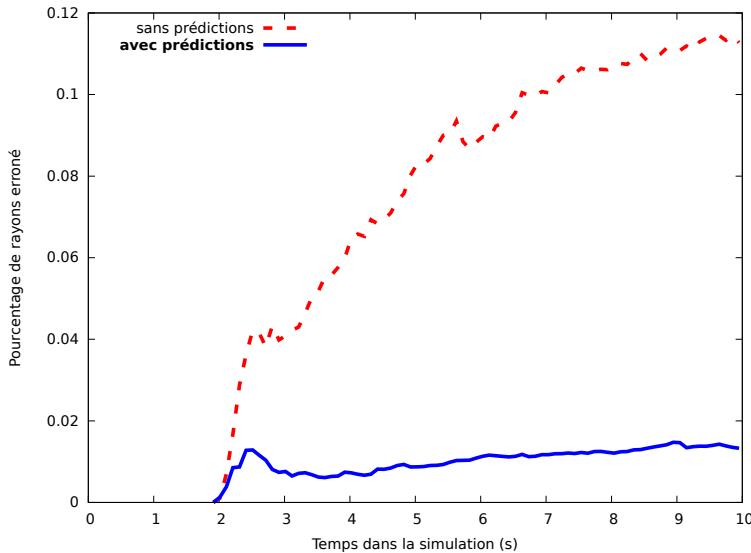


Figure 4.9 – Pourcentage de rayons erronés dans la première scène.

de collision, nous sommes capables de réduire considérablement la quantité d’erreurs lors de la détection de collision. En reposant sur des prédictions de collision dont le coût calculatoire est relativement faible (cf. Chapitre 3 Section 2.2) cette méthode est capable de rendre notre *pipeline* plus fiable sans dégradation des performances.

Les erreurs restantes sont toujours majoritairement des faux-négatifs (environ 90% des erreurs). Leur faible taux d’occurrence ($<0.05\%$) et la redondance des tests (les objets sont testés sur tous leurs sommets) laisse un risque d’erreur complet entre deux objets (c’est-à-dire une détection erronée sur tous les sommets) extrêmement faible. Si de telles erreurs arrivent, elles sont dans tous les cas limitées à de faibles mouvements car en cas de déplacements relatifs entre les objets dépassant le seuil *seuilDéplacement*, des algorithmes de lancer de rayon complets sont utilisés et ceux-ci ne sont pas sujets à ces erreurs de détection. Les erreurs seront donc, au pire, détectées dans les pas de temps suivants.

4 Application aux objets déformables

La mesure de déplacement que nous avons proposé précédemment n’est pas applicable aux objets déformables. Elle repose sur le mouvement des repères locaux des objets, cependant ceux-ci ne donnent aucune information sur les déformations internes des objets. Pour permettre de mesurer la quantité de déplacements relatifs entre deux objets déformables ou un objet déformable et un objet rigide, nous proposons une seconde mesure de déplacement relatif. Nous allons dans cette section présenter un algorithme qui permet de mesurer localement les déplacements relatifs entre des objets et nous réaliserons une évaluation de cette méthode.

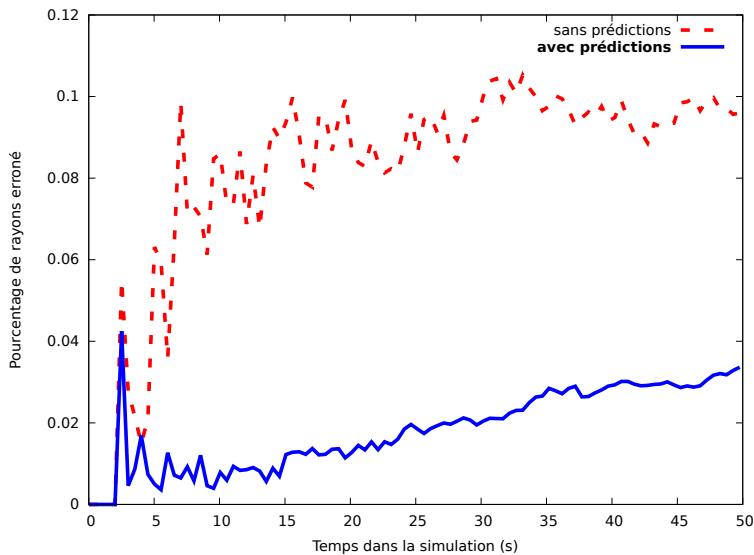


Figure 4.10 – Pourcentage de rayons erronés dans la seconde scène.

4.1 Mesure de déplacement relatifs entre objets déformables

Plusieurs problématiques supplémentaires apparaissent avec les objets déformables :

- Il faut mesurer les déplacements de chaque sommet de chaque objet car leurs déplacements sont libres.
- Il faut ensuite conserver l'historique des déplacements relatifs entre chaque paire de sommets en collision (ou en collision potentielle) depuis la dernière utilisation d'un algorithme de lancer de rayon complet.

Cette combinatoire est trop élevée pour être calculée et stockée. Pour la réduire, nous proposons de ne pas mesurer les déplacements relatifs sur les sommets individuels mais sur des *clusters* de sommets sur chaque objet (ie. groupes de sommets). La motivation principale de l'utilisation *clusters* est que les déplacements des sommets sont localement cohérents, un ensemble de sommets proches sur un objet a tendance à avoir des mouvements semblables. La décomposition d'objets en *clusters* a déjà été utilisée dans les méthodes de *culling* pour la détection de collision [Wong et al., 2013].

Notre méthode de mesure de déplacements relatifs se décompose en quatre étapes : le partitionnement des objets en *clusters*, les mesures des déplacements des *clusters*, la localisation des paires de *clusters* proches et les mesures de déplacements relatifs entre les paires de *clusters* proches.

Partitionnement des objets en clusters

Le partitionnement des objets est réalisé en calcul hors-ligne et ne varie pas ensuite lors de la simulation. Nous utilisons un partitionnement en K-moyennes pour décomposer les objets en *clusters*. Le nombre de *clusters* est déterminé à partir d'une constante *nbSommetsParClusters* qui indique combien de sommets doivent, en moyenne, être contenus dans un *cluster*. Pour savoir combien de *clusters* doit contenir

un objet, il suffit de diviser le nombre de sommets de cet objet par la valeur de cette constante.

Une métrique de distance doit être fournie à la méthode des K-moyennes pour mesurer la distance entre les sommets pour constituer les *clusters*. En utilisant la distance standard dans l'espace entre les sommets dans la configuration initiale des objets, nous obtiendrions des *clusters* cohérents localement. Cependant, les *clusters* doivent présenter une forte cohérence spatiale y compris après que des déformations aient été appliquées sur les objets. La distance standard ne garantit pas cette cohérence. Nous avons décidé d'utiliser une métrique basée sur une mesure de distance sur la surface des objets car celle-ci reste valide après que des déformations aient été appliquées sur les objets. La métrique de distance utilisée entre les sommets est la longueur du plus court chemin entre les sommets à travers les arêtes. De plus, cette mesure garantit qu'aucun *cluster* ne sera disjoint sur la surface des objets. La figure 4.11 montre un exemple de décomposition en *clusters*.

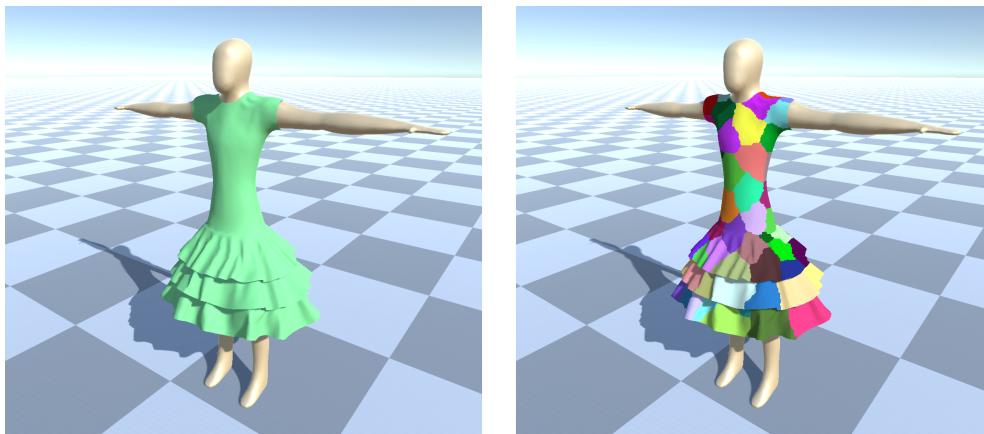


Figure 4.11 – Exemple de partitionnement d'un vêtement (à gauche) en *clusters* (à droite).

Mesure de déplacement des *clusters*

Nous avons besoin de mesurer le déplacement de chaque *cluster* à chaque pas de temps. Cette mesure n'est pas relative, elle est réalisée dans le repère global du monde. Cette mesure ne conserve pas d'historique, elle mesure le déplacement du *cluster* uniquement dans le pas de temps courant.

Pour éviter de travailler avec les mouvements individuels des sommets, nous proposons d'appliquer une opération de réduction sur l'ensemble des sommets des *clusters*. Une contrainte importante sur l'opération de réduction est qu'elle doit :

1. Mesurer l'amplitude du mouvement. Nous aurons besoin de savoir par la suite si les mouvements des sommets dépassent certains seuils.
2. Mesurer la direction du mouvement. Lors du calcul des mouvements relatifs entre paires de *clusters*, nous aurons besoin de savoir si les *clusters* se déplacent dans la même direction ou dans des directions opposées.

Nous notons cette mesure de déplacement $dep(c)$. Elle mesure le déplacement d'un seul *cluster* c à l'instant présent. Le déplacement n'est pas relatif, $dep(c)$ mesure le

déplacement absolu de c dans l'espace. Cette mesure n'est pas accumulée, elle mesure la quantité de déplacement subit par le *cluster* uniquement au pas de temps actuel (ie. entre l'instant $t - 1$ et l'instant t). Nous proposons plusieurs mesures de déplacement possibles qui satisfont les deux conditions que nous nous sommes posées :

- $dep_{moyenne}(c) \in \mathbb{R}^3$, la moyenne des déplacements des sommets.
- $dep_{moyenne+écart}(c) \in \mathbb{R}^4$, Cette mesure est constituée de deux variables :
- $dep_{moyenne}(c) \in \mathbb{R}^3$ qui est la moyenne des déplacements des sommets.
- $dep_{écart}(c) \in \mathbb{R}$ qui est l'écart type de l'amplitude des déplacements des sommets par rapport à $dep_{moyenne}(c)$.
- $dep_{max}(c) \in \mathbb{R}^3$ le mouvement du sommet ayant la plus grande amplitude de mouvement.

Localisation de paires de clusters proches

Avant de pouvoir mesurer les déplacements relatifs entre *clusters*, il est nécessaire de savoir quels *clusters* sont proches. Cette information est utilisée pour savoir entre quelles paires de *clusters* nous devons réaliser la mesure de déplacement relatifs (et la détection de collision.)

Nous utilisons un algorithme de détection de collision de *broad-phase* pour déterminer quels *clusters* sont proches. Dans cette phase, les tests de collisions sont réalisés sur des volumes englobants. Nous avons choisi d'utiliser des AABBs comme volumes englobants pour les *clusters* car ceux-ci se déforment en continu et les AABBs sont un bon compromis entre précision et temps de calculs pour les objets déformables [Larsson and Akenine-Möller, 2001]. La figure 4.12 présente un exemple de recherche de *clusters* proches sur un seul objet.

Lorsqu'une nouvelle paire de *clusters* proches est détectée, sa mesure de déplacement relatif est initialisée à zéro. Un algorithme de lancer de rayon complet sera utilisé entre les deux clusters lorsque ceux-ci auront accumulé une quantité de déplacements relatifs supérieurs à $seuilDéplacement$. Pour s'assurer que les deux *clusters* doivent réaliser un tel déplacement avant de pouvoir entrer en collision, nous agrandissons les volumes englobants d'une distance $\frac{seuilDéplacement}{2}$ dans toutes les directions. Ceci permet de s'assurer que les deux *clusters* ne vont pas entrer en collision avant qu'un algorithme de lancer de rayon complet ait été appelé au moins une fois.

Les déplacements relatifs ne seront mesurés que sur les paires de *clusters* dont les volumes englobants sont en collision. Si les volumes englobants ne sont pas en collision alors les *clusters* ne sont pas en collision, il n'est donc pas utile de mesurer leurs déplacements relatifs.

Mesure de déplacement relatifs entre clusters proches

La mesure de déplacement relatif est réalisée sur les *clusters* proches. Cette mesure nous donne une estimation de la quantité de déplacements relatifs entre les deux *clusters* depuis le dernier pas de temps complet. Il y a donc une notion de mémoire car, tant que l'algorithme de lancer de rayon itératif est utilisé, la quantité de déplacements relatifs doit être accumulée.

La mesure de déplacement relative se base sur la combinaison des mesures de déplacement $dep(c)$ des *clusters* et est constituée de plusieurs éléments :

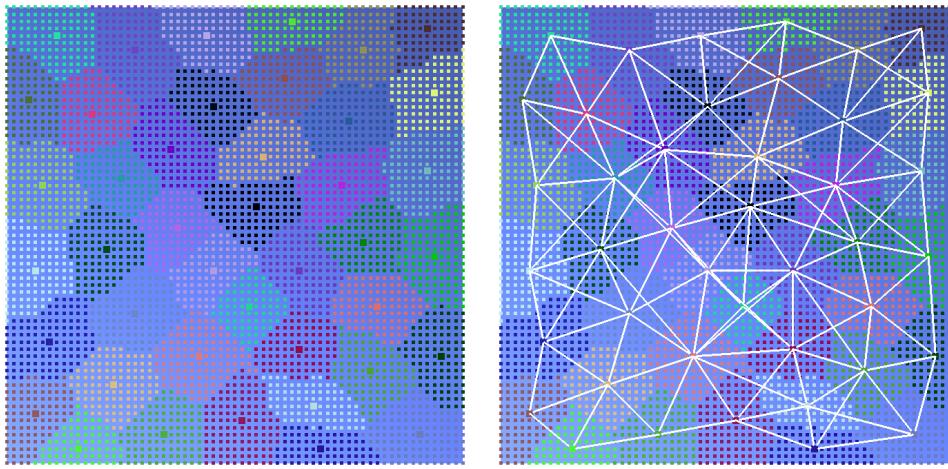


Figure 4.12 – Exemple de recherche de *clusters* voisins. À gauche un carré de tissu est affiché, les sommets sont colorés par *clusters* (le sommet au centre du *cluster* est affiché avec un carré plus grand). À droite les relations de proximités sont affichées, les *clusters* dont les AABBs sont en collision dans cette configuration géométrique sont reliés avec des lignes blanches.

Un opérateur de combinaison $depRelatif(dep_1, dep_2)$: elle permet de combiner deux mesures de déplacement $dep(c_1)$ et $dep(c_2)$ pour obtenir la valeur $depRelatif(dep(c_1), dep(c_2))$ qui est une mesure de déplacement relative entre deux *clusters* c_1 et c_2 . Cette valeur sera accumulée au cours de la simulation pour mesurer la quantité de déplacements relatifs accumulés entre deux *clusters*.

Un opérateur d'évaluation de longueur de déplacement $\|depRelatif\|$: à partir de la valeur de la quantité de déplacements relatifs accumulés $depRelatif$, le scalaire $\|depRelatif\|$ donne l'amplitude de déplacement qui sera comparé avec notre seuil $seuilDéplacement$.

Lorsque la mesure de déplacement est réalisée avec plusieurs variables (par exemple avec la mesure moyenne+écart), plusieurs opérateurs de combinaison et d'addition seront utilisés (un par variable). L'opérateur d'évaluation de la longueur de déplacement $\|depRelatif\|$ est unique, même dans le cas d'utilisation de plusieurs variables, il n'y a qu'un seul opérateur $\|\dots\|$ qui prend en paramètre toutes les variables. La méthode de mesure de déplacement relatif se déroule ensuite toujours de la même manière :

- Lors de l'apparition d'une nouvelle paire de *clusters* (c_1, c_2) , la quantité de déplacements relatifs initiale est nulle : $depRelatif(dep_1, dep_2) = \vec{0}$.
- À chaque pas de temps, le déplacement relatif entre chaque paire de *cluster* est accumulé : $depRelatifAccumulé_t(c_1, c_2) = depRelatifAccumulé_{t-1}(c_1, c_2) + depRelatif(dep(c_1) - dep(c_2))$.
- Lorsque le seuil de déplacement est atteint ($\|depRelatifAccumulé_t(c_1, c_2)\| > seuilDéplacement$), un algorithme de lancer de rayon complet est utilisé et la quantité de déplacements relatifs est réinitialisée : $depRelatifAccumulé_t(c_1, c_2) = \vec{0}$.
- La valeur de déplacement relatif $depRelatifAccumulé_t(c_1, c_2)$ est stockée pour être utilisée dans le pas de temps suivant.

Le tableau 4.2 donne les valeurs des opérateurs de déplacement relatif pour les trois mesures que nous avons proposées. Le point notable est que deux d'entre elles sont composé que d'une seule variable (moyenne et max), le déplacement relatif est alors calculé en réalisant la différence des mesures de déplacement de chaque *cluster*. La mesure moyenne+écart utilise deux variables : la moyenne du déplacement et l'écart type par rapport à cette moyenne. Dans ce cas le déplacement relatif est alors calculé en réalisant la différence des moyennes et en additionnant les écarts types.

Type de mesure : Moyenne des déplacements des sommets
Variables :
$dep_{moyenne}(c) \in \mathbb{R}^3$
Opérateur de combinaison :
$depRelatif_{moyenne}(dep1, dep2) = dep1_{moyenne} - dep2_{moyenne}$
Opérateur de longueur :
$\ depRelatif\ = \ depRelatif_{moyenne}\ $
Type de mesure : Moyenne des déplacements des sommets + écart type
Variables :
$dep_{moyenne}(c) \in \mathbb{R}^3$
$dep_{écart}(c) \in \mathbb{R}$
Opérateur de combinaison :
$depRelatif_{moyenne}(dep1, dep2) = dep1_{moyenne} - dep2_{moyenne}$
$depRelatif_{écart}(dep1, dep2) = dep1_{écart} + dep2_{écart}$
Opérateur de longueur :
$\ depRelatif\ = \ depRelatif_{moyenne}\ + depRelatif_{écart}$
Type de mesure : Mouvement du sommet ayant la plus grande amplitude
Variables :
$dep_{max}(c) \in \mathbb{R}^3$
Opérateur de combinaison :
$depRelatif_{max}(dep1, dep2) = dep1_{max} - dep2_{max}$
Opérateur de longueur :
$\ depRelatif\ = \ depRelatif_{max}\ $

Table 4.2 – Calcul de la mesure de déplacements relatifs en fonction du type de mesures de déplacements.

4.2 Évaluation

Nous avons évalué les performances de la détection de collision à l'aide d'algorithmes itératifs sur une scène test. Dans cette scène, 64 draps déformables tombent sur un sol irrégulier sur lequel des sphères fixes ont été ajoutées (cf. Figure 4.13). La simulation est exécutée sur une carte graphique AMD FirePro W9100 (2816 coeurs, 16 Go de mémoire) à 60 images par seconde pendant 5 secondes.

Pour comparer et évaluer l'intérêt d'une métrique mesurant les déplacements relatifs, nous avons implémenté une mesure de déplacement absolue. Ceci va nous permettre de monter qu'il est nécessaire de mesurer des déplacements de manière relative pour obtenir une métrique fiable. Cette mesure absolue mesure le déplacement

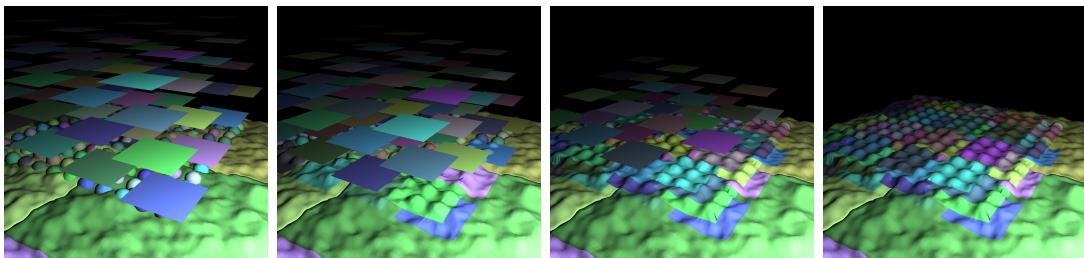


Figure 4.13 – Scène expérimentale utilisée dans l'évaluation de la mesure de déplacement relatif appliqués aux objets déformables.

individuel des sommets sans prendre en compte leur environnement voisin et fonctionne comme il suit :

- La décision d'utiliser l'algorithme de lancer de rayon itératif ou un algorithme de lancer de rayon complet est prise individuellement pour chaque sommet.
- La position du sommet lors de la dernière utilisation d'un algorithme de lancer de rayon complet est conservée.
- L'algorithme de lancer de rayon itératif est utilisé tant que la distance entre la position actuelle du sommet et la position du sommet lors de la dernière utilisation d'un algorithme de lancer de rayon complet est inférieure à $seuilDéplacement$.
- Lorsque la distance entre la position actuelle du sommet et la position du sommet lors de la dernière utilisation d'un algorithme de lancer de rayon complet est supérieure à $seuilDéplacement$, un algorithme de lancer de rayon complet est utilisé.

Nous avons réalisé deux évaluations : nous avons dans un premier temps comparé les performances et le taux d'erreur obtenu avec les trois mesures de déplacements relatifs que nous avons proposées (moyenne, moyenne+écart et maximum). Dans un second temps nous avons sélectionné une mesure de déplacement relatif et l'avons comparé à la mesure de déplacement absolue. Dans ces évaluations, nous avons mesuré trois éléments clés dans notre méthode :

- Le temps de calcul de la *narrow-phase* : notre but est de minimiser celui-ci.
- Le taux d'utilisation de l'algorithme de lancer de rayon itératif : l'algorithme de lancer de rayon itératif étant plus rapide à calculer que les autres algorithmes de lancer de rayon, son taux d'utilisation aura un impact majeur sur le temps de calcul de la *narrow-phase*. Nous cherchons donc de manière indirecte à maximiser ce taux pour minimiser le temps de calcul.
- Le pourcentage de rayons erronés (cf. Section 1) : nous cherchons à minimiser celui-ci car les erreurs lors de la détection de collision causent des artefacts dans la simulation. Les erreurs sont causées par l'algorithme de lancer de rayon itératif lorsqu'il est utilisé dans de mauvaises conditions. Le pourcentage de rayons erronés est donc fortement lié au taux d'utilisation de l'algorithme de lancer de rayon itératif.

Nous remarquons que nous avons deux objectifs antagonistes : minimiser le temps de calcul et minimiser le pourcentage de rayons erronés. Nous cherchons donc à trouver le meilleur compromis entre le temps de calcul et taux d'erreur.

4.2.1 Comparaison des types de mesures de déplacement

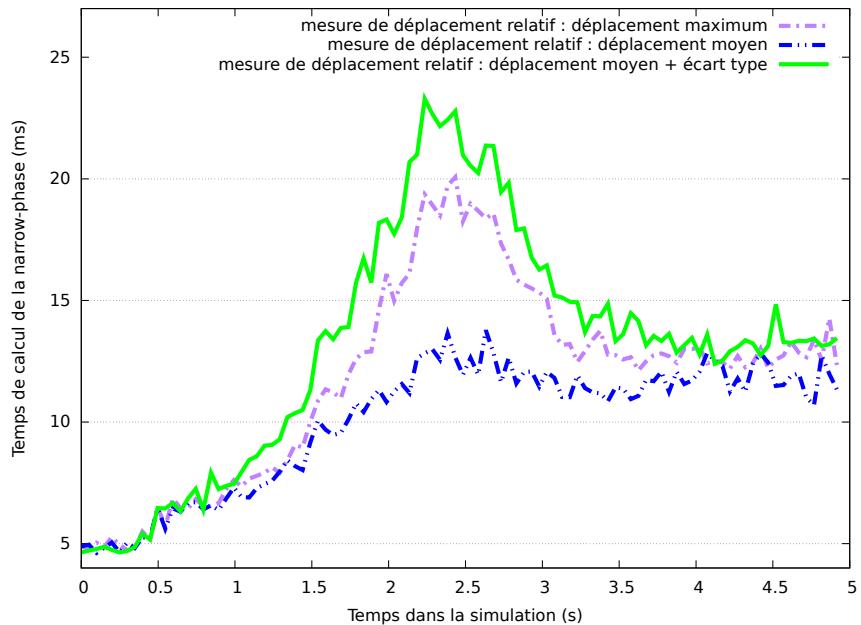


Figure 4.14 – Temps de calcul de la *narrow-phase* à chaque pas de temps avec les trois mesures de déplacement relatif.

La figure 4.14 donne les mesures de performances de la détection de collision avec les trois mesures de déplacement relatif. Les mesures moyenne et moyenne+écart présentent toutes les deux un pic de temps de calcul à la moitié de la simulation qui n'est pas présent avec la mesure maximum. Cette dernière semble alors être la plus adaptée car elle permet d'atteindre les meilleures performances.

La figure 4.15 donne une mesure du taux d'erreurs, celui-ci est défini comme le pourcentage de rayons erronés par rapport au nombre de rayons calculés au total. Cette mesure est accompagnée de la mesure du taux de rayons calculés avec l'algorithme de lancer de rayon itératif (par rapport à l'ensemble des rayons lancés). C'est l'algorithme de lancer de rayon itératif qui peut produire des erreurs, il est donc intéressant de comparer le taux d'erreurs dans la simulation avec l'utilisation de l'algorithme itératif pour voir si il existe un lien. Nous pouvons faire plusieurs observations :

- La mesure de déplacement absolu provoque un nombre élevé d'erreurs (jusqu'à ~4%) alors qu'elle utilise de manière beaucoup moins importante l'algorithme itératif de lancer de rayon. Ceci montre qu'une mesure non-relative utilise de manière inadéquate l'algorithme de lancer de rayon itératif.
- La mesure relative moyenne provoque le plus grand nombre d'erreurs (jusqu'à ~6,5%). Ces plus grandes erreurs s'expliquent par l'incapacité de la moyenne à mesurer les rotations. En effet, un objet qui tourne sur un axe passant par son centre a un déplacement moyen nul. Cette mesure utilise donc à tort l'algorithme itératif lorsque des rotations sont présentes (le taux d'utilisation de l'algorithme itératif est plus élevé) ce qui provoque un plus grand nombre d'erreurs.
- La mesure relative maximum a un taux d'erreurs plus élevé que la mesure

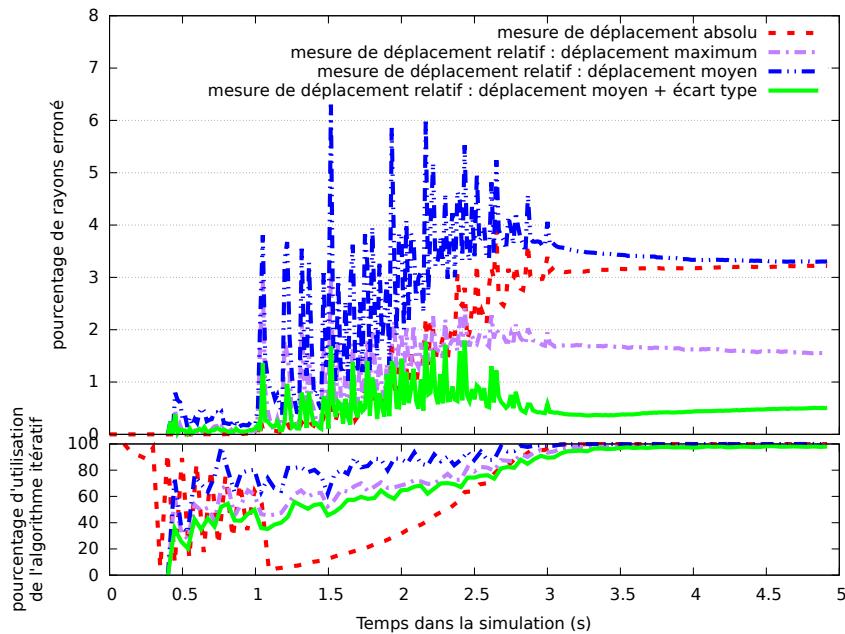


Figure 4.15 – Taux d’erreur et pourcentage de rayon utilisant l’algorithme de lancer de rayon itératif avec les trois mesures de déplacement relatif et la mesure de déplacement absolue.

moyenne+écart et utilise de manière plus fréquente l’algorithme itératif. Ce résultat est surprenant car nous pourrions nous attendre à ce que la mesure maximum déclenche plus souvent l’utilisation de l’algorithme complet de lancer de rayon (y compris dans des cas où ce n’est pas nécessaire). Nous pensons que l’explication provient du fait que le vecteur de déplacement maximum n’est pas mesuré à chaque fois sur le même sommet, ceci peut conduire à des cas où la somme des vecteurs maximum est faible même si l’objet se déplace. Nous n’avons pas confirmé cette théorie et des expériences supplémentaires seraient nécessaires pour confirmer cette explication.

- La mesure relative moyenne+écart donne les meilleurs résultats en matière de taux d’erreurs, cette mesure provoque un taux d’erreurs moyen d’environ 0.5% tout au long de la simulation.

Au vu de ces observations, la mesure de déplacement relatif moyenne+écart est la plus adaptée. Nous écartons la mesure de déplacement relatif moyenne car son taux d’erreurs est trop élevé, et ce, même si cette mesure permet d’obtenir les meilleures performances. La mesure de déplacement relatif maximum est légèrement plus rapide que la mesure moyenne+écart mais le taux d’erreurs plus faible de cette seconde mesure la rend plus attractive. En effet, la mesure moyenne+écart permet d’obtenir un taux d’erreurs total trois fois plus faible que la mesure maximum et son sur-coût n’est d’environ que de 10% en terme de temps de calcul. Néanmoins, la mesure maximum peut être utilisée dans certaines applications où le temps de calcul est extrêmement critique car elle permet un temps de calcul plus faible et plus stable.

4.2.2 Comparaison avec une mesure de distance non-relative

Nous avons comparé les performances de notre *narrow-phase* en faisant varier deux paramètres :

- Utilisation de la décomposition en *clusters* et de notre mesure de déplacement relatif : avec ou sans.
- Utilisation de l'algorithme de lancer de rayon itératif : avec ou sans.

Les résultats des mesures sont présentés sur la figure 4.16 et nous pouvons faire plusieurs observations. L'ajout de la mesure de déplacement relatif sans utiliser l'algorithme de lancer de rayon itératif améliore grandement les performances, le temps de calcul est divisé en moyenne par 28. Ceci s'explique par la structure spatiale qui accompagne la mesure de déplacement, les objets (rigides ou déformables) sont décomposés en *clusters* et les tests de collision sont réalisés entre ceux-ci au lieu de les réaliser sur les objets entiers. Ceci permet de diminuer fortement le nombre de tests unitaires de collision. Les résultats montrent cependant que la combinaison de la décomposition en *clusters*, de la mesure de déplacement relatif et de l'utilisation de l'algorithme de lancer de rayon itératif permettent d'obtenir des performances encore plus élevées. L'ajout de l'algorithme de lancer de rayon itératif lors de l'utilisation de la mesure de déplacement relatif permet de diviser le temps de calcul par un facteur de 4. En combinant ces deux facteurs, l'utilisation de la mesure de déplacement relatif et l'utilisation de l'algorithme de lancer de rayon itératif permet de diviser le temps de calcul par 99 en moyenne.

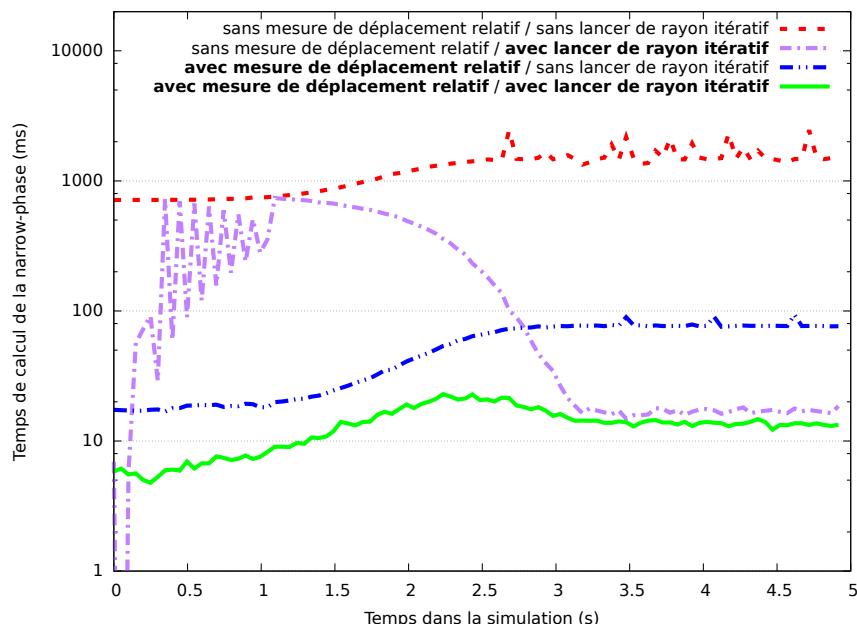


Figure 4.16 – Temps de calcul de la *narrow-phase* sur une échelle logarithmique tout au long de la simulation.

4.3 Synthèse

Nous avons proposé une mesure de déplacement relatif entre objets déformables. Dans notre contexte de détection de collision par lancer de rayon, cette mesure permet de décider dans quels cas notre algorithme de lancer de rayon itératif peut être utilisé. Nous avons montré que les performances obtenues dans notre *pipeline* sont meilleures en terme de performance et de précision lors de l'utilisation de cette mesure de déplacement relatif.

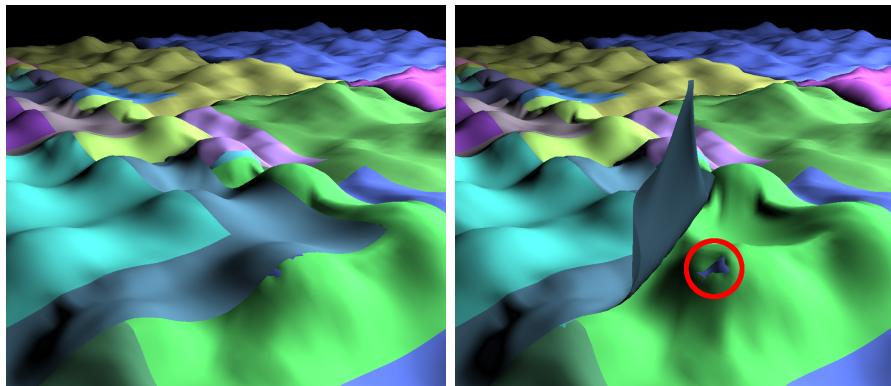


Figure 4.17 – Exemple d'inversion présente dans la simulation finale. Gauche : une inversion s'est produite entre plusieurs couches de tissus. Droite : une fois la première couche soulevée, l'inversion est plus visible (entourée en rouge).

Cependant, notre méthode produit des erreurs de détection. Même si celles-ci sont faibles elles peuvent être visibles, la figure 4.17 donne un tel exemple. Ces erreurs ne proviennent pas uniquement de notre méthode de détection de collision mais aussi de la réponse physique. Simuler en temps réel un grand nombre d'objets déformables est aussi un défi dans le domaine de la réponse physique. Lorsque la réponse physique n'arrive pas à séparer deux objets en collision, ceux-ci seront détectés de manière incorrecte dans les pas de temps suivants.

5 Conclusion

Nous avons présenté dans ce chapitre une méthode permettant d'exploiter la cohérence temporelle lors de la détection de collision par lancer de rayon. Cette méthode s'insère parfaitement dans notre *pipeline* et est capable d'être exécutée sur GPU. Cette méthode se décompose en deux éléments majeurs :

- Un algorithme de lancer de rayon itératif qui permet d'exploiter la cohérence temporelle. Cet algorithme est capable de mettre à jour le résultat de lancer de rayon du pas précédent. Cependant, cet algorithme ne doit être utilisé qu'en cas de faible déplacement entre l'origine du rayon et la cible du rayon.
- Une mesure de déplacement relatif pour décider quand utiliser notre algorithme de lancer de rayon itératif. Nous avons proposé deux mesures différentes pour réaliser cette tâche. Une première mesure est destinée à l'évaluation de la quantité de déplacement relatif entre deux objets rigides. Une seconde mesure

permet d'évaluer la quantité de déplacement relatif entre objets déformables en travaillant sur des sous-parties des objets.

De plus, nous avons montré que les prédictions de collision peuvent être utilisées pour réduire fortement les erreurs sans nuire de manière importante aux performances.

Conclusion

Nous avons présenté, dans cette thèse, nos travaux sur la détection de collision temps-réel destinés à la simulation physique de scènes larges et complexes. Notre objectif a été de permettre de réaliser des simulations physique temps-réel avec un grand nombre d'objets et avec leur formes géométriques complexes. Nous présentons dans cette conclusion un bilan de nos contributions ainsi que leurs différentes perspectives.

Bilan des contributions

L'étude de l'état de l'art nous a montré que les méthodes de détection de collision ne sont pas capables aujourd'hui de gérer l'intégralité du problème de la détection de collision. Les méthodes actuelles simplifient le problème et/ou réalisent des approximations. Ceci a un impact sur l'expérience des utilisateurs dans leur perception et interaction avec des applications de réalité virtuelle. Le nombre d'objets simulés est limité, rendant certaines parties des mondes virtuels non-interactives. Les objets peuvent être simplifiés, rendant leur comportement physique incohérent avec leur aspect visuel.

Pour permettre des simulations plus complètes et plus réalistes du point de vue de l'utilisateur, nous nous sommes intéressés aux méthodes de détection de collision par lancer de rayon. Nous avons proposé un certain nombre de contributions sur différents axes :

Généricité des calculs

Nous avons généralisé la méthode de détection de collision de Hermann et al. [Hermann et al., 2008] pour être capable de réaliser des requêtes supplémentaires et améliorer le résultat de la détection de collision.

Nous avons introduit une méthode permettant de réaliser la détection de collision en temps-réel avec une représentation surfacique de tissus. Cette méthode permet de réaliser la détection de collision entre un tissu et un objet volumique ainsi que la détection de collision entre tissus en intégrant l'auto-collision. Nous avons montré que cette méthode permet de réaliser des simulations physiques interactives, c'est-à-dire permettant à un utilisateur de manipuler des tissus en temps-réel.

Nous avons proposé une mesure de distance entre objets appelée prédition de collision qui est réalisée à l'aide de lancer de rayon. Elle permet de mesurer la distance entre deux objets de manière individuelle à chaque sommet et de fournir, en plus de la distance, la localisation des zones proches. Cette méthode s'insère parfaitement dans notre méthode de détection de collision et nous avons montré que son coût calculatoire est faible.

Nous avons amélioré la qualité de la réponse physique obtenue à partir du résultat de la détection de collision en re-projetant les rayons. Cette méthode permet à la réponse physique de séparer les objets dans les directions optimales.

Exploitation de la cohérence temporelle

Pour améliorer les performances et ainsi permettre la simulation de scènes plus complexes, nous avons proposé d'exploiter la cohérence temporelle.

Nous avons proposé un algorithme de lancer de rayon itératif capable d'exploiter la cohérence temporelle. Cet algorithme utilise le résultat du lancer de rayon du pas précédent et le met à jour avec un algorithme itératif jusqu'à ce que le nouveau point d'intersection soit trouvé. Cet algorithme peut être utilisé lorsque les objets subissent de faibles mouvements relatifs, c'est-à-dire lorsqu'il existe une forte cohérence temporelle. Dans les autres cas, des algorithmes de lancer de rayon complets doivent être utilisés.

Pour pouvoir décider à quels moments notre algorithme de lancer de rayon itératif peut être utilisé dans la détection de collision, nous avons proposé deux mesures de déplacement relatifs entre objets. Ces deux mesures permettent de savoir à quels moments les objets subissent de faibles mouvements relatifs et par conséquent, permettent de savoir à quels moments nous pouvons utiliser notre algorithme de lancer de rayon itératif. La première mesure proposée permet de mesurer la quantité de déplacements relatifs entre deux objets rigides. La seconde mesure permet de mesurer la quantité de déplacements relatifs en cas de présence d'objets déformables.

Nous avons aussi montré que les prédictions de collision peuvent être utilisées avec notre algorithme de lancer de rayon itératif pour considérablement réduire les erreurs de détection. Les rayons prédisant des collisions peuvent être mis à jour avec notre algorithme de lancer de rayon itératif de la même manière que les rayons détectant des collisions peuvent être mis à jour.

Utilisation des GPUs

La détection de collision est une tâche extrêmement consommatrice en puissance de calcul. Nous avons donc proposé d'utiliser les GPU comme périphérique de calcul pour améliorer les performances. Nous avons revisité le *pipeline* pour organiser les calculs pour qu'ils s'exécutent de manière efficace sur GPU. L'ensemble de nos contributions ont ensuite été pensées pour prendre en compte les contraintes des GPUs, par conséquent, l'ensemble de notre méthode peut être implémentée sur GPU à l'aide de notre *pipeline*.

Perspectives

La détection de collision est un problème qui nécessite encore beaucoup de recherches pour permettre des simulations génériques, réalistes, large-échelle et en temps-réel. Nous présentons ici certaines perspectives possibles à nos travaux qui s'orientent dans cette direction.

Simulations tolérantes aux erreurs

Notre méthode de détection de collision produit un faible taux d'erreurs lors de l'utilisation de l'algorithme de lancer de rayon itératif, ces erreurs sont principalement des faux-négatifs. Ceux-ci ne posent pas de problèmes critiques grâce à la redondance des tests. Les tests de collisions étant réalisés sur chaque sommet, lorsqu'un sommet n'est pas détecté en collision alors qu'il aurait dû l'être, les sommets voisins peuvent être détectés et empêcher le sommet non-détecté d'entrer en trop forte interpénétration. De plus, les détections de faux-négatifs sont limitées en terme de déplacements, dès que la quantité de déplacements relatifs entre deux objets devient trop importante, l'algorithme de lancer de rayon itératif est remplacé par un algorithme complet et tous les faux-négatifs sont corrigés.

Cependant, la non-détection de certains points de contacts pose des problèmes locaux. Lorsqu'un point de contact non-détecté est finalement détecté, la réponse physique corrige l'interpénétration qui a été accumulée. Ceci peut provoquer des comportements incohérents (comme produire le rebond spontané d'un objet). Une amélioration possible pourrait être apportée à l'aide de réponse physique. Il serait possible de marquer les points de contact ayant été détectés en retard lors de la détection de collision et de les traiter différemment dans la réponse physique, il serait ainsi possible d'obtenir un comportement pseudo-réaliste des objets qui semblerait plus cohérent du point de vue des utilisateurs.

Les erreurs sont aussi plus problématiques dans le cas de la simulation de tissus. De par le mouvement relativement indépendant des sommets, de la faible épaisseur des tissus et de la difficulté à réaliser une simulation physique, des inversions peuvent toujours se glisser dans nos simulations. Lorsqu'une inversion se produit, notre méthode est capable de la détecter, cependant, si les inversions ne sont pas corrigées immédiatement, elles ne seront plus détectées et resteront dans la suite de la simulation (cf. Figure 4.17). Il serait intéressant de rendre notre méthode tolérante aux erreurs en la permettant de détecter et corriger les inversions qui se sont accumulées en cours de simulation. Ceci pourrait être réalisé en exécutant ponctuellement des tests plus complets (et plus coûteux).

Généralisation à de nouveaux types d'objets

Avec notre méthode, nous sommes capables de gérer des objets rigides et déformables avec des représentations surfaciques et volumiques. Il serait intéressant de généraliser notre méthode à d'autres types d'objets.

Il serait possible de prendre en compte les changements topologiques (tels que la fracture ou la déchirure). Une problématique importante lors de la simulation de ce type de phénomènes est le coût additionnel en temps de calcul. Ce coût additionnel est généralement ponctuel (par exemple lors de la fracture d'une vitre) et pose un défi pour maintenir une simulation temps-réel. Il faut éviter de rompre l'interactivité de la simulation au moment de la fracture à cause du surcoût calculatoire, ce surcoût est généralement dû à la reconstruction de structures accélératrices. Pour maintenir un temps de calculs constant, il serait possible, avec notre méthode, de lancer de manière asynchrone la reconstruction des structures accélératrices de lancer de rayon. Tant que

les structures n'ont pas été reconstruites, notre *pipeline* pourrait utiliser un algorithme de lancer de rayon plus simple pour maintenir l'interactivité de la simulation.

Il serait aussi intéressant de généraliser notre méthode pour intégrer la simulation de fluides à l'aide de particules. Il ne serait pas intéressant de chercher à réaliser la détection de collision entre les particules de fluides à l'aide de lancer de rayon. Cependant, les collisions entre les particules de fluides et le reste de l'environnement pourrait être calculées avec des méthodes de lancer de rayon. Ceci permettrait de ne pas avoir à représenter les objets solides qui interagissent avec des fluides avec des modèles particulaires.

Calcul hors-ligne

La simulation physique, et par extension la détection de collision, doivent réaliser une quantité importante de calculs. Pour réduire le coût calculatoire, de nouvelles méthodes et algorithmes sont sans cesse proposés pour réaliser les calculs de la manière la plus efficace possible. Néanmoins, les calculs les moins coûteux sont ceux qui ne sont pas calculés lors de la simulation mais en amont de celle-ci : les calculs hors-ligne. Un grand nombre de méthodes actuelles réalisent des calculs hors-ligne, notamment lors de la construction des structures accélératrices. Certaines méthodes plus originales cherchent à réaliser la simulation physique entière en calcul hors-ligne et permettent d'interagir avec celle-ci à l'aide de méthodes basées données (par exemple pour l'animation de vêtements [Kim et al., 2013]).

Une perspective à plus long terme serait de chercher à utiliser de manière plus intensive les calculs hors-ligne. Cela pourrait être accompli en combinant des simulations pré-calculées et des simulations calculées en temps-réel. Mais il serait aussi intéressant d'appliquer cette idée à la détection de collision. Combiner l'exploration de bases de données de collisions calculées hors-ligne avec une méthode de détection de collision en ligne pourrait par exemple être utilisé pour permettre d'accélérer un grand nombre de requêtes. Il sera nécessaire d'étudier comment choisir et stocker les configurations de collision les plus utiles car il est impossible de calculer l'ensemble des simulations possibles.

Publications de l'auteur

- [Lehericey et al., 2013a] Lehericey, F., Gouranton, V., and Arnaldi, B. (2013a). New iterative ray-traced collision detection algorithm for gpu architectures. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, pages 215–218. ACM.
- [Lehericey et al., 2013b] Lehericey, F., Gouranton, V., and Arnaldi, B. (2013b). Ray-traced collision detection : interpenetration control and multi-gpu performance. In *Proceedings of the 5th Joint Virtual Reality Conference*, pages 33–40. Eurographics Association.
- [Lehericey et al., 2015a] Lehericey, F., Gouranton, V., and Arnaldi, B. (2015a). Extended version : Gpu ray-traced collision detection for cloth simulation. In *Journées de l'AFRV*.
- [Lehericey et al., 2015b] Lehericey, F., Gouranton, V., and Arnaldi, B. (2015b). Gpu ray-traced collision detection : Fine pipeline reorganization. In *Proceedings of 10th International Conference on Computer Graphics Theory and Applications (GRAPP'15)*.
- [Lehericey et al., 2015c] Lehericey, F., Gouranton, V., and Arnaldi, B. (2015c). Gpu ray-traced collision detection for cloth simulation. In *Proceedings of the 21st ACM Symposium on Virtual Reality Software and Technology*, pages 47–50. ACM.

Table des figures

1.1	Taxonomie des représentations des modèles 3D	6
1.2	Exemple de modélisation à l'aide de particules	6
1.3	Exemples de modèles non-polygonaux	7
1.4	Exemples de modèles volumiques	8
1.5	Exemple d'objet convexe et non-convexe	9
1.6	Exemple d'erreur d'échantillonnage trop large	10
1.7	Exemples d'objets déformables et de changements topologiques	12
1.8	Pipeline de détection de collision	14
1.9	Différents types de découpage spatial.	15
1.10	Exemple en 2D de l'algorithme de <i>sweep and prune</i>	16
1.11	Exemple de V-Clip	18
1.12	Exemple de SAT	19
1.13	Exemple d'une différence de Minkowski	19
1.14	Exemples de volumes englobants	22
1.15	Exemples de LDI	24
1.16	Exemple du théorème de Jordan	26
1.17	Exemple de la méthode de Hermann et al.	27
1.18	Exemple d'un champ de distance	28
1.19	Évolution de la puissance de calcul des GPUs et CPUs	31
2.1	Comparaison entre des rayons pour le rendu et pour la détection de collision	41
2.2	Exemple de cohérence temporelle lors du rendu par lancer de rayons	41
2.3	Cohérence des points de contacts	42
2.4	Exemples de rayons détectant des collisions et des prédictions	42
2.5	Exemple de mesure de distance entre deux objets	44
3.1	Exemple de <i>culling</i> proposé par Hermann et al.	52
3.2	Organisation en <i>pipeline</i> de la <i>narrow-phase</i>	53
3.3	Comparaison de la chronologie des évènements avec et sans prédictions	56
3.4	Quatre formes d'objets rigides utilisés dans les simulations	58
3.5	Scène "objets rigides"	58
3.6	Scène "objets rigides et déformables"	58
3.7	Pipeline : performance scène "objets rigides"	59
3.8	Pipeline : performance scène "objets rigides et déformables"	60
3.9	Exemples de rayons détectant des collisions et des prédictions	63
3.10	Exemples de prédictions pouvant avoir lieu en dehors des volumes englobants	63
3.11	Scènes expérimentales	64

Table des figures

3.12	Prédictions : performance première scène	65
3.13	Prédictions : performance seconde scène	66
3.14	Collision entre un objet et le sol	67
3.15	Composition d'un point de contact	68
3.16	Protocole test de déviation	68
3.17	Exemple où la re-projection ne fournit pas le déplacement minimum global	70
3.18	Définitions de représentation entre les objets volumiques et surfaciques	71
3.19	Exemple de deux objets en collision	71
3.20	Illustration des conditions proposées par Hermann et al.	73
3.21	Exemple volumique → tissu	74
3.22	Exemple tissu → volumique	75
3.23	Exemple tissu → tissu	76
3.24	Exemple d'inversion	77
3.25	Deux draps tombent sur une boule	78
3.26	Exemple d'une prédition de collision qui a besoin d'être reclassé	79
3.27	Effet de la re-projection sur la longueur des rayons	80
3.28	Exemple de repositionnement des points de contacts	81
3.29	Huit draps tombent sur une scène statique	82
3.30	Une robe à trois couches	83
3.31	Test de stress	84
3.32	Exemple d'épaisseur variable sur une simulation multi-échelle	84
4.1	Exemple de lancer de rayon itératif sur un maillage de triangles.	87
4.2	L'algorithme itératif est piégé dans une concavité dans cet exemple.	88
4.3	Plan d'un triangle avec les arêtes numérotées.	88
4.4	L'intersection entre deux objets non-convexes peut être convexe	91
4.5	Itératif rigide : performances dans la première scène	94
4.6	Itératif rigide : performances dans la seconde scène	95
4.7	Temps moyen passé dans les différentes étapes de la simulation	96
4.8	Comparaisons du comportement de deux objets avec et sans prédition	97
4.9	Pourcentage de rayons erronés dans la première scène.	100
4.10	Pourcentage de rayons erronés dans la seconde scène.	101
4.11	Exemple de partitionnement d'un vêtement	102
4.12	Exemple de recherche de <i>clusters</i> voisins	104
4.13	Scène utilisée dans l'évaluation de la mesure de déplacement relatif	106
4.14	Temps de calcul avec les trois mesures de déplacement relatif.	107
4.15	Taux d'erreur avec les trois mesures de déplacement relatif.	108
4.16	Temps de calcul de la <i>narrow-phase</i> tout au long de la simulation	109
4.17	Exemple d'inversion présente dans la simulation finale.	110

Liste des algorithmes

1	Intersection rayon/maillage de triangles itératif	87
2	Intersection rayon/triangle	89
3	Intersection rayon/maillage de triangles itératif	99

Liste des tableaux

3.1	Gain moyen en temps de calcul sur GPU lors de l'utilisation de prédictions pour estimer la charge de travail comparé à l'utilisation de la charge exacte. Utiliser des prédictions est jusqu'à 2,84 ms plus rapide par pas de temps.	60
3.2	Mesure de la réduction du nombre de rayons manqués. Prendre en compte l'évolution du nombre de paires détectées par la <i>broad-phase</i> permet de réduire le nombre de rayons manqués jusqu'à 79%.	61
3.3	Déviation moyenne des forces de pénalités pour trois objets.	68
3.4	Comparaison de la somme des longueurs des rayons tout au long de la simulation avec et sans re-projection des rayons.	69
4.1	Temps moyen ou temps max d'exécution du lancer de rayon par pas de temps pour les trois phases de la scène 1	94
4.2	Calcul de la mesure de déplacements relatifs en fonction du type de mesures de déplacements.	105

Bibliographie

- [Allard et al., 2010] Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C., and Kry, P. G. (2010). Volume contact constraints at arbitrary resolution. In *ACM Transactions on Graphics (TOG)*, volume 29, page 82. ACM. [25](#)
- [Ar et al., 2000] Ar, S., Chazelle, B., and Tal, A. (2000). Self-customized bsp trees for collision detection. *Computational Geometry*, 15(1) :91–102. [15](#)
- [Arnaldi et al., 2003] Arnaldi, B., Fuchs, P., and Tisseau, J. (2003). Chapitre 1 du volume 1 du traité de la réalité virtuelle. *Les Presses de l'Ecole des Mines de Paris*. [1](#)
- [Avril et al., 2012] Avril, Q., Gouranton, V., and Arnaldi, B. (2012). Fast collision culling in large-scale environments using gpu mapping function. In *Eurographics Symposium Proceedings*, pages 71–80. [33](#)
- [Baciu et al., 1998] Baciu, G., Wong, W. S.-K., and Sun, H. (1998). Recode : an image-based collision detection algorithm. In *Computer Graphics and Applications, 1998. Pacific Graphics' 98. Sixth Pacific Conference on*, pages 125–133. IEEE. [24](#)
- [Badler and Glassner, 1997] Badler, N. I. and Glassner, A. S. (1997). 3d object modeling. *SIGGRAPH 97 Introduction to Computer Graphics Course Notes*. [5](#)
- [Bajaj and Dey, 1992] Bajaj, C. L. and Dey, T. K. (1992). Convex decomposition of polyhedra and robustness. *SIAM Journal on Computing*, 21(2) :339–364. [21](#)
- [Bandi and Thalmann, 1995] Bandi, S. and Thalmann, D. (1995). An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. In *Computer Graphics Forum*, volume 14, pages 259–270. Wiley Online Library. [15](#)
- [Baraff, 1992] Baraff, D. (1992). Dynamic simulation of non-penetrating rigid bodies. Technical report, Cornell University. [16](#)
- [Baraff et al., 2003] Baraff, D., Witkin, A., and Kass, M. (2003). Untangling cloth. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 862–870. ACM. [81](#)
- [Barbič and James, 2010] Barbič, J. and James, D. L. (2010). Subspace self-collision culling. In *ACM Transactions on Graphics (TOG)*, volume 29, page 81. ACM. [29](#)
- [Beckmann et al., 1990] Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. (1990). *The R*-tree : an efficient and robust access method for points and rectangles*, volume 19. ACM. [15](#)
- [Bender et al., 2013] Bender, J., Müller, M., Otaduy, M. A., and Teschner, M. (2013). Position-based methods for the simulation of solid objects in computer graphics. Eurographics. [79](#)

- [Benitez et al., 2005] Benitez, A., Ramirez, M. d. C., and Vallejo, D. (2005). Collision detection using sphere-tree construction. In *Electronics, Communications and Computers, 2005. CONIELECOMP 2005. Proceedings. 15th International Conference on*, pages 286–291. IEEE. [21](#)
- [Bentley and Friedman, 1979] Bentley, J. L. and Friedman, J. H. (1979). Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4) :397–409. [15](#)
- [Bergen, 1997] Bergen, G. v. d. (1997). Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, 2(4) :1–13. [21](#), [22](#)
- [Bergen, 1999] Bergen, G. v. d. (1999). A fast and robust gjk implementation for collision detection of convex objects. *Journal of graphics tools*, 4(2) :7–25. [19](#)
- [Boldt and Meyer, 2005] Boldt, N. and Meyer, J. (2005). Self-intersections with cullide. *DIKU project*, pages 04–02. [27](#)
- [Cameron, 1997] Cameron, S. (1997). Enhancing gjk : Computing minimum and penetration distances between convex polyhedra. In *ICRA*, volume 97, pages 20–25. [19](#)
- [Cameron and Culley, 1986] Cameron, S. A. and Culley, R. (1986). Determining the minimum translational distance between two convex polyhedra. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 591–596. IEEE. [10](#), [67](#)
- [Coming and Staadt, 2006] Coming, D. S. and Staadt, O. G. (2006). Kinetic sweep and prune for multi-body continuous motion. *Computers & Graphics*, 30(3) :439–449. [16](#)
- [Coming and Staadt, 2008] Coming, D. S. and Staadt, O. G. (2008). Velocity-aligned discrete oriented polytopes for dynamic collision detection. *Visualization and Computer Graphics, IEEE Transactions on*, 14(1) :1–12. [21](#)
- [Curtis et al., 2008] Curtis, S., Tamstorf, R., and Manocha, D. (2008). Fast collision detection for deformable models using representative-triangles. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 61–69. ACM. [29](#)
- [Devadoss and O’Rourke, 2011] Devadoss, S. L. and O’Rourke, J. (2011). *Discrete and computational geometry*. Princeton University Press. [25](#)
- [Ehmann and Lin, 2000] Ehmann, S. A. and Lin, M. C. (2000). Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2101–2106. IEEE. [17](#)
- [Ehmann and Lin, 2001] Ehmann, S. A. and Lin, M. C. (2001). Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Computer Graphics Forum*, volume 20, pages 500–511. Wiley Online Library. [20](#), [21](#)
- [Eitz and Lixu, 2007] Eitz, M. and Lixu, G. (2007). Hierarchical spatial hashing for real-time collision detection. In *Shape Modeling and Applications, 2007. SMI’07. IEEE International Conference on*, pages 61–70. IEEE. [15](#)

- [Faure et al., 2008] Faure, F., Barbier, S., Allard, J., and Falipou, F. (2008). Image-based collision detection and response between arbitrary volume objects. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–162. Eurographics Association. [25](#), [39](#)
- [Fernando, 2004] Fernando, R. (2004). *GPU Gems : Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education. [49](#)
- [Finkel and Bentley, 1974] Finkel, R. A. and Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1) :1–9. [15](#)
- [Fisher and Lin, 2001] Fisher, S. and Lin, M. C. (2001). *Deformed distance fields for simulation of non-penetrating flexible bodies*. Springer. [27](#)
- [Foley and Sugerman, 2005] Foley, T. and Sugerman, J. (2005). Kd-tree acceleration structures for a gpu raytracer. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22. ACM. [38](#)
- [Fuchs, 1996] Fuchs, P. (1996). *Les interfaces de la réalité virtuelle*. éditeur AJIMD. [1](#)
- [Fuhrmann et al., 2003] Fuhrmann, A., Sobotka, G., and Groß, C. (2003). Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon 2003*, pages 58–65. [28](#)
- [Fünfzig and Fellner, 2003] Fünfzig, C. and Fellner, D. W. (2003). Easy realignment of k-dop bounding volumes. In *Graphics Interface*, volume 3, pages 257–264. [15](#), [21](#)
- [Gilbert and Foo, 1990] Gilbert, E. G. and Foo, C.-P. (1990). Computing the distance between general convex objects in three-dimensional space. *Robotics and Automation, IEEE Transactions on*, 6(1) :53–61. [18](#)
- [Gilbert et al., 1988] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2) :193–203. [18](#)
- [Glondu et al., 2012] Glondu, L., Schwartzman, S. C., Marchal, M., Dumont, G., and Otaduy, M. A. (2012). Efficient collision detection for brittle fracture. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 285–294. Eurographics Association. [12](#), [28](#)
- [Glondu et al., 2014] Glondu, L., Schwartzman, S. C., Marchal, M., Dumont, G., and Otaduy, M. A. (2014). Fast collision detection for fracturing rigid bodies. *Visualization and Computer Graphics, IEEE Transactions on*, 20(1) :30–41. [28](#)
- [Goldsmith and Salmon, 1987] Goldsmith, J. and Salmon, J. (1987). Automatic creation of object hierarchies for ray tracing. *Computer Graphics and Applications, IEEE*, 7(5) :14–20. [22](#)
- [Gottschalk, 1996] Gottschalk, S. (1996). Separating axis theorem. Technical report, Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill. [18](#)
- [Gottschalk, 1997] Gottschalk, S. (1997). Collision detection techniques for 3d models. [5](#)
- [Gottschalk, 2000] Gottschalk, S. (2000). *Collision queries using oriented bounding boxes*. PhD thesis, The University of North Carolina at Chapel Hill. [21](#)

- [Gottschalk et al., 1996] Gottschalk, S., Lin, M. C., and Manocha, D. (1996). Obbtree : A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM. 21
- [Govindaraju et al., 2005] Govindaraju, N. K., Knott, D., Jain, N., Kabul, I., Tamstorf, R., Gayle, R., Lin, M. C., and Manocha, D. (2005). Interactive collision detection between deformable models using chromatic decomposition. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 991–999. ACM. 25
- [Govindaraju et al., 2003] Govindaraju, N. K., Redon, S., Lin, M. C., and Manocha, D. (2003). Cullide : Interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 25–32. Eurographics Association. 27
- [Grisoni, 2005] Grisoni, L. (2005). *Vers une simulation physique temps réel multi-modèle (HDR)*. PhD thesis, PhD thesis, Université des sciences et des technologies de Lille (LIFL). 7
- [Guttman, 1984] Guttman, A. (1984). *R-trees : a dynamic index structure for spatial searching*, volume 14. ACM. 15
- [Hapala et al., 2011] Hapala, M., Davidović, T., Wald, I., Havran, V., and Slusallek, P. (2011). Efficient stack-less bvh traversal for ray tracing. In *Proceedings of the 27th Spring Conference on Computer Graphics*, pages 7–12. ACM. 39
- [He, 1999] He, T. (1999). Fast collision detection using quospo trees. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 55–62. ACM. 21
- [Heidelberger et al., 2004] Heidelberger, B., Teschner, M., and Gross, M. (2004). Detection of collisions and self-collisions using image-space techniques. 24
- [Heidelberger et al., 2003] Heidelberger, B., Teschner, M., and Gross, M. H. (2003). Real-time volumetric intersections of deforming objects. In *VMV*, volume 3, pages 461–468. 24
- [Held et al., 1995] Held, M., Klosowski, J. T., and Mitchell, J. S. (1995). Evaluation of collision detection methods for virtual reality fly-throughs. In *Canadian Conference on Computational Geometry*, pages 205–210. Citeseer. 30
- [Hermann et al., 2008] Hermann, E., Faure, F., and Raffin, B. (2008). Ray-traced collision detection for deformable bodies. In *GRAPP 2008-3rd International Conference on Computer Graphics Theory and Applications*, pages 293–299. INSTICC. 26, 35, 37, 39, 66, 68, 70, 72, 113
- [Hubbard, 1993] Hubbard, P. M. (1993). Interactive collision detection. In *Virtual Reality, 1993. Proceedings., IEEE 1993 Symposium on Research Frontiers in*, pages 24–31. IEEE. 13
- [Hubbard, 1996] Hubbard, P. M. (1996). Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3) :179–210. 21
- [Hutter and Fuhrmann, 2007] Hutter, M. and Fuhrmann, A. (2007). Optimized continuous collision detection for deformable triangle meshes. 29

- [Ize et al., 2006] Ize, T., Wald, I., Robertson, C., and Parker, S. G. (2006). An evaluation of parallel grid construction for ray tracing dynamic scenes. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 47–55. IEEE. 38
- [James and Pai, 2004] James, D. L. and Pai, D. K. (2004). Bd-tree : output-sensitive collision detection for reduced deformable models. *ACM Transactions on Graphics (TOG)*, 23(3) :393–398. 22
- [Kang and Cho, 2002] Kang, Y.-M. and Cho, H.-G. (2002). Bilayered approximate integration for rapid and plausible animation of virtual cloth with realistic wrinkles. In *Computer Animation, 2002. Proceedings of*, pages 203–211. IEEE. 30
- [Kim et al., 2013] Kim, D., Koh, W., Narain, R., Fatahalian, K., Treuille, A., and O’Brien, J. F. (2013). Near-exhaustive precomputation of secondary cloth effects. *ACM Transactions on Graphics (TOG)*, 32(4) :87. 116
- [Kim et al., 2010] Kim, T.-J., Moon, B., Kim, D., and Yoon, S.-E. (2010). Racbvhs : Random-accessible compressed bounding volume hierarchies. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2) :273–286. 22
- [Kim et al., 2002a] Kim, Y. J., Hoff, K., Lin, M. G., and Manocha, D. (2002a). Closest point query among the union of convex polytopes using rasterization hardware. *Journal of Graphics Tools*, 7(4) :43–51. 27
- [Kim et al., 2002b] Kim, Y. J., Otaduy, M. A., Lin, M. C., and Manocha, D. (2002b). Fast penetration depth computation for physically-based animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 23–31. ACM. 27
- [Klosowski et al., 1998] Klosowski, J. T., Held, M., Mitchell, J. S., Sowizral, H., and Zikan, K. (1998). Efficient collision detection using bounding volume hierarchies of k-dops. *Visualization and Computer Graphics, IEEE Transactions on*, 4(1) :21–36. 15, 21
- [Knott, 2003] Knott, D. (2003). Cinder : collision and interference detection in real time using graphics hardware. 25
- [Kockara et al., 2007] Kockara, S., Halic, T., Iqbal, K., Bayrak, C., and Rowe, R. (2007). Collision detection : A survey. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 4046–4051. IEEE. 14, 17
- [Krishnan et al., 1998] Krishnan, S., Gopi, M., Lin, M., Manocha, D., and Pattekar, A. (1998). Rapid and accurate contact determination between spline models using shelltrees. In *Computer Graphics Forum*, volume 17, pages 315–326. Wiley Online Library. 21
- [Krishnan et al., 1997] Krishnan, S., Pattekar, A., and Lin, M. C. (1997). Spherical shell : A higher order bounding volume for fast proximity queries. 21
- [Larsen et al., 2000] Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. (2000). Fast distance queries with rectangular swept sphere volumes. In *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*, volume 4, pages 3719–3726. IEEE. 21
- [Larsson and Akenine-Möller, 2001] Larsson, T. and Akenine-Möller, T. (2001). Collision detection for continuously deforming bodies. *Eurographics*. 16, 21, 22, 103

- [Larsson and Akenine-Möller, 2006] Larsson, T. and Akenine-Möller, T. (2006). A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics*, 30(3) :450–459. [23](#)
- [Lauterbach et al., 2009] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. (2009). Fast bvh construction on gpus. In *Computer Graphics Forum*, volume 28, pages 375–384. Wiley Online Library. [33](#)
- [Lauterbach et al., 2010] Lauterbach, C., Mo, Q., and Manocha, D. (2010). gproximity : Hierarchical gpu-based operations for collision and distance queries. In *Computer Graphics Forum*, volume 29, pages 419–428. Wiley Online Library. [33](#)
- [Lauterbach et al., 2006] Lauterbach, C., Yoon, S.-E., Tuft, D., and Manocha, D. (2006). Rt-deform : Interactive ray tracing of dynamic scenes using bvhs. In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 39–46. IEEE. [39](#)
- [Le Grand, 2007] Le Grand, S. (2007). Broad-phase collision detection with cuda. *GPU gems*, 3 :697–721. [33](#)
- [Li and Chen, 1998] Li, T.-Y. and Chen, J.-S. (1998). Incremental 3d collision detection with hierarchical data structures. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 139–144. ACM. [23](#)
- [Lin and Gottschalk, 1998] Lin, M. and Gottschalk, S. (1998). Collision detection between geometric models : A survey. In *Proc. of IMA conference on mathematics of surfaces*, volume 1, pages 602–608. [5](#), [11](#), [13](#)
- [Lin and Canny, 1991] Lin, M. C. and Canny, J. F. (1991). A fast algorithm for incremental distance calculation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1008–1014. IEEE. [17](#)
- [Liu et al., 2010] Liu, F., Harada, T., Lee, Y., and Kim, Y. J. (2010). Real-time collision culling of a million bodies on graphics processing units. In *ACM Transactions on Graphics (TOG)*, volume 29, page 154. ACM. [33](#)
- [Luque et al., 2005] Luque, R. G., Comba, J. L., and Freitas, C. M. (2005). Broad-phase collision detection using semi-adjusting bsp-trees. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 179–186. ACM. [15](#)
- [Macklin et al., 2014] Macklin, M., Müller, M., Chentanez, N., and Kim, T.-Y. (2014). Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4) :104. [6](#)
- [Mezger et al., 2003] Mezger, J., Kimmerle, S., and Etzmuß, O. (2003). Hierarchical techniques in collision detection for cloth animation. [20](#), [22](#), [23](#), [29](#)
- [Mirtich, 1998] Mirtich, B. (1998). V-clip : Fast and robust polyhedral collision detection. *ACM Transactions On Graphics (TOG)*, 17(3) :177–208. [17](#)
- [Möller and Trumbore, 2005] Möller, T. and Trumbore, B. (2005). Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*, page 7. ACM. [64](#), [87](#)
- [Moore and Wilhelms, 1988] Moore, M. and Wilhelms, J. (1988). Collision detection and response for computer animation. In *ACM Siggraph Computer Graphics*, volume 22, pages 289–298. ACM. [17](#)

- [Morvan et al., 2008] Morvan, T., Reimers, M., and Samset, E. (2008). High performance gpu-based proximity queries using distance fields. In *Computer graphics forum*, volume 27, pages 2040–2052. Wiley Online Library. [28](#)
- [Müller and Chentanez, 2010] Müller, M. and Chentanez, N. (2010). Wrinkle meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 85–92. Eurographics Association. [30](#), [82](#)
- [Müller et al., 2015] Müller, M., Chentanez, N., Kim, T.-Y., and Macklin, M. (2015). Air meshes for robust collision handling. *ACM Transactions on Graphics (TOG)*, 34(4) :133. [30](#), [77](#)
- [Myszkowski et al., 1995] Myszkowski, K., Okunev, O. G., and Kunii, T. L. (1995). Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9) :497–511. [24](#)
- [Navarro and Hitschfeld, 2013] Navarro, C. A. and Hitschfeld, N. (2013). Improving the gpu space of computation under triangular domain problems. *arXiv preprint arXiv :1308.1419*. [33](#)
- [Naylor et al., 1990] Naylor, B., Amanatides, J., and Thibault, W. (1990). Merging bsp trees yields polyhedral set operations. In *ACM Siggraph Computer Graphics*, volume 24, pages 115–124. ACM. [15](#)
- [Naylor, 1992] Naylor, B. F. (1992). Interactive solid geometry via partitioning trees. In *Proc. Graphics Interface*, volume 92, pages 11–18. Citeseer. [15](#)
- [Nguyen, 2007] Nguyen, H. (2007). *Gpu gems 3*. Addison-Wesley Professional. [49](#)
- [O’SULLIVAN and Dingliana, 1999] O’SULLIVAN, C. and Dingliana, J. (1999). Real-time collision detection and response using sphere-trees. [21](#)
- [Overmars, 1992] Overmars, M. H. (1992). Point location in fat subdivisions. *Information Processing Letters*, 44(5) :261–265. [15](#)
- [Pabst et al., 2010] Pabst, S., Koch, A., and Straßer, W. (2010). Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. In *Computer Graphics Forum*, volume 29, pages 1605–1612. Wiley Online Library. [12](#), [34](#)
- [Palmer and Grimsdale, 1995] Palmer, I. J. and Grimsdale, R. L. (1995). Collision detection for animation using sphere-trees. In *Computer Graphics Forum*, volume 14, pages 105–116. Wiley Online Library. [21](#)
- [Pharr and Fernando, 2005] Pharr, M. and Fernando, R. (2005). *Gpu gems 2 : programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional. [49](#)
- [Ponce and Faugeras, 1987] Ponce, J. and Faugeras, O. (1987). An object centered hierarchical representation for 3d objects : The prism tree. *Computer Vision, Graphics, and Image Processing*, 38(1) :1–28. [21](#)
- [Popov et al., 2007] Popov, S., Günther, J., Seidel, H.-P., and Slusallek, P. (2007). Stackless kd-tree traversal for high performance gpu ray tracing. In *Computer Graphics Forum*, volume 26, pages 415–424. Wiley Online Library. [39](#), [51](#)
- [Redon et al., 2002] Redon, S., Kheddar, A., and Coquillart, S. (2002). Fast continuous collision detection between rigid bodies. In *Computer graphics forum*, volume 21, pages 279–287. Wiley Online Library. [21](#)

- [Redon et al., 2005] Redon, S., Lin, M. C., Manocha, D., and Kim, Y. J. (2005). Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering*, 5(2) :126–137. [21](#)
- [Roussopoulos and Leifker, 1985] Roussopoulos, N. and Leifker, D. (1985). Direct spatial search on pictorial databases using packed r-trees. In *ACM SIGMOD Record*, volume 14, pages 17–31. ACM. [22](#)
- [Sellis et al., 1987] Sellis, T., Roussopoulos, N., and Faloutsos, C. (1987). The r+-tree : A dynamic index for multi-dimensional objects. [15](#)
- [Shade et al., 1998] Shade, J., Gortler, S., He, L.-w., and Szeliski, R. (1998). Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 231–242. ACM. [23](#)
- [Shinya and Forgue, 1991] Shinya, M. and Forgue, M.-C. (1991). Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, 2(4) :132–134. [24](#)
- [Smith, 2012] Smith, W. A. (2012). Representing, storing and visualizing 3d data. In *3D Imaging, Analysis and Applications*, pages 139–182. Springer. [5](#)
- [Sud et al., 2004] Sud, A., Otaduy, M. A., and Manocha, D. (2004). Difi : Fast 3d distance field computation using graphics hardware. In *Computer Graphics Forum*, volume 23, pages 557–566. Wiley Online Library. [28](#)
- [Sundaraj et al., 2000] Sundaraj, K., d’Aulignac, D., and Mazer, E. (2000). A new algorithm for computing minimum distance. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2115–2120. IEEE. [20](#)
- [Tang et al., 2011a] Tang, M., Manocha, D., Lin, J., and Tong, R. (2011a). Collision-streams : fast gpu-based collision detection for deformable models. In *Symposium on interactive 3D graphics and games*, pages 63–70. ACM. [34](#)
- [Tang et al., 2011b] Tang, M., Manocha, D., Yoon, S.-E., Du, P., Heo, J.-P., and Tong, R.-F. (2011b). Volccd : Fast continuous collision culling between deforming volume meshes. *ACM Transactions on Graphics (TOG)*, 30(5) :111. [20](#)
- [Teschner et al., 2005] Teschner, M., Kimmerle, S., Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M.-P., Faure, F., Magnenat-Thalmann, N., Strasser, W., et al. (2005). Collision detection for deformable objects. In *Computer graphics forum*, volume 24, pages 61–81. Wiley Online Library. [11, 17, 28](#)
- [Tompson and Schlachter, 2012] Tompson, J. and Schlachter, K. (2012). An introduction to the opencl programming model. *Person Education*. [49](#)
- [Tracy et al., 2009] Tracy, D. J., Buss, S. R., and Woods, B. M. (2009). Efficient large-scale sweep and prune methods with aabb insertion and removal. In *Virtual Reality Conference, 2009. VR 2009. IEEE*, pages 191–198. IEEE. [16](#)
- [Turk, 1989] Turk, G. (1989). *Interactive collision detection for molecular graphics*. PhD thesis, The University of North Carolina at Chapel Hill. [15](#)
- [Vaněkček, 1994] Vaněkček, G. (1994). Back-face culling applied to collision detection of polyhedra. *The Journal of Visualization and Computer Animation*, 5(1) :55–63. [29](#)

- [Vassilev et al., 2001] Vassilev, T., Spanlang, B., and Chrysanthou, Y. (2001). Fast cloth animation on walking avatars. In *Computer Graphics Forum*, volume 20, pages 260–267. Wiley Online Library. [25](#)
- [Vemuri et al., 1998] Vemuri, B. C., Cao, Y., and Chen, L. (1998). Fast collision detection algorithms with applications to particle flow. In *Computer Graphics Forum*, volume 17, pages 121–134. Wiley Online Library. [15](#)
- [Volino and Magnenat-Thalmann, 2006] Volino, P. and Magnenat-Thalmann, N. (2006). *Resolving surface collisions through intersection contour minimization*, volume 25. ACM. [81](#)
- [Volino and Thalmann, 1995] Volino, P. and Thalmann, N. M. (1995). *Collision and self-collision detection : Efficient and robust solutions for highly deformable surfaces*. Springer. [22](#)
- [Wald et al., 2007] Wald, I., Boulos, S., and Shirley, P. (2007). Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics (TOG)*, 26(1) :6. [38](#), [51](#)
- [Wald et al., 2009] Wald, I., Mark, W. R., Günther, J., Boulos, S., Ize, T., Hunt, W., Parker, S. G., and Shirley, P. (2009). State of the art in ray tracing animated scenes. In *Computer Graphics Forum*, volume 28, pages 1691–1722. Wiley Online Library. [38](#)
- [Wald and Slusallek, 2001] Wald, I. and Slusallek, P. (2001). State of the art in interactive ray tracing. *State of the Art Reports, EUROGRAPHICS*, 2001 :21–42. [38](#)
- [Walter et al., 1999] Walter, B., Drettakis, G., and Parker, S. (1999). Interactive rendering using the render cache. In *Rendering techniques' 99*, pages 19–30. Springer. [39](#)
- [Wang et al., 2012] Wang, B., Faure, F., and Pai, D. K. (2012). Adaptive image-based intersection volume. *ACM Transactions on Graphics (TOG)*, 31(4) :97. [25](#)
- [Wong et al., 2013] Wong, S.-K., Lin, W.-C., Hung, C.-H., Huang, Y.-J., and Lii, S.-Y. (2013). Radial view based culling for continuous self-collision detection of skeletal models. *ACM Transactions on Graphics (TOG)*, 32(4) :114. [26](#), [101](#)
- [Zachmann, 1998] Zachmann, G. (1998). Rapid collision detection by dynamically aligned dop-trees. In *Virtual Reality Annual International Symposium, 1998. Proceedings., IEEE 1998*, pages 90–97. IEEE. [21](#)
- [Zachmann and Felger, 1995] Zachmann, G. and Felger, W. (1995). The boxtree : Enabling real-time and exact collision detection of arbitrary polyhedra. In *Informal Proc. First Workshop on Simulation and Interaction in Virtual Environments, SIVE*, volume 95, pages 104–112. [21](#)
- [Zheng and James, 2012] Zheng, C. and James, D. L. (2012). Energy-based self-collision culling for arbitrary mesh deformations. *ACM Transactions on Graphics (TOG)*, 31(4) :98. [29](#)

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:

Détection de collision par lancer de rayon : la quête de la performance

Nom Prénom de l'auteur : LEHERICEY FRANCOIS

Membres du jury :

- Monsieur RAFFIN Bruno
- Monsieur ARNALDI BRUNO
- Monsieur PAZAT Jean-Louis
- Madame GOURANTON Valérie
- Monsieur GRISONI Laurent
- Monsieur FAURE François

Président du jury : Jean-Louis PAZAT

Date de la soutenance : 20 Septembre 2016

Reproduction de la these soutenue

Thèse pouvant être reproduite en l'état

~~Thèse pouvant être reproduite après corrections suggérées~~

Fait à Rennes, le 20 Septembre 2016

Signature du président de jury

Le Directeur,

M'hamed DRISSI



Résumé

La détection de collision est une tâche essentielle pour la simulation physique d'environnements virtuels. De nos jours, la détection de collision est l'un des goulets d'étranglement calculatoire dans les applications de réalité virtuelle dû à la complexité des environnements que l'on souhaite simuler et par la contrainte d'interaction en temps-réel. Nous avons concentré nos travaux sur la seconde étape de la détection de collision (narrow-phase) dans laquelle les tests de collisions sont effectués sur des paires d'objets. Contrairement à la première étape, les tests de collisions sont effectués sur des versions détaillées des modèles géométriques et sont donc très sensibles au niveau calculatoire à la complexité géométrique de ceux-ci.

Cette thèse vise à améliorer les performances de la détection de collision lors de l'utilisation d'objets géométriques complexes (formes représentées par un maillage, éventuellement non-convexe et/ou déformable). Nos méthodes permettent d'accélérer le calcul de la narrow-phase. Les algorithmes proposés peuvent être implémentés sur GPU pour profiter de leur puissance de calcul et ainsi améliorer les performances.

Pour réaliser la narrow-phase, nous proposons un pipeline adapté à une implémentation GPU. Celui-ci permet de réaliser la narrow-phase à l'aide d'algorithme basés lancer de rayon. Notre méthode repose sur un principe commun où tous les tests de collision sont effectués par lancer de rayon à partir des sommets des objets. Cette généralité nous permet de réaliser les tests sur des maillages ayant n'importe quelles propriétés (rigide ou déformable, convexe ou non-convexe). Les algorithmes de lancer de rayon utilisés étant choisis en fonction des propriétés des objets pour optimiser les performances.

Nous avons généralisé la méthode de détection de collision utilisée pour supporter, en plus des objets volumiques, des objets surfaciques (tels que des tissus). Cette méthode est compatible avec le pipeline proposé et permet de réaliser des tests de collisions entre n'importe quelle combinaison d'objets.

De plus, nous proposons d'exploiter la cohérence temporelle pour améliorer les performances. Le résultat de la détection de collision est généralement très semblable entre deux pas de temps successifs. Suite à cette observation, nous proposons un algorithme de lancer de rayon itératif qui intègre le résultat du pas précédent pour exploiter cette cohérence temporelle. Cet algorithme peut être utilisé conjointement avec des algorithmes de lancer de rayon standard. Il permet de remplacer certains tests unitaires pour mettre à jour de manière incrémentale le résultat de la détection de collision. L'algorithme de lancer de rayon itératif est ajouté au pipeline en tant qu'alternative aux autres algorithmes de lancer de rayon et est utilisé prioritairement dû à son faible coût calculatoire.

Abstract

Collision detection is an essential task for physical simulation of virtual environments. Nowadays, collision detection is one of the main bottleneck of virtual reality applications. This is due to the complexity of the environments we want to simulate and the real-time interaction constraint. We have concentrated our work on the second phase of collision detection, the narrow-phase, in which collision queries are performed on pairs of objects. Contrary to the first phase of collision detection, collision queries are performed on the full representation of the objects (with all details) and are thus sensible to the geometric complexity of the objects in term of computation time.

This thesis is aimed to improve the performances of collision detection when using geometrically complex objects (represented with triangle meshes, potentially non-convex and deformable). Our methods are able to reduce computation times. Our proposed algorithms can be implemented on GPU to take advantage of their computational power and thus further improve the performances of our methods.

To implements our narrow-phase, we propose a pipeline which is adapted for GPU execution. This pipeline perform collision detection with ray-tracing algorithms. Our methods rely on a shared principle where all collision queries are performed by casting rays from the vertices of the objects. This genericity allow us to perform collision detection on triangle meshes with any properties (rigid or deformable, convex or non-convex). The ray-tracing algorithms used are dynamically selected depending on the properties of the objects to improve the performances.

We have generalized the collision detection method we use in our pipeline to handle, in addition to volumetric objects (represented by their surface), surface objects (such as cloth). This method is compatible with our pipeline and allow us to perform collision detection with any combination of volumetric and surface objects.

Furthermore, we propose to exploit temporal coherency to improve performances. The result of collision queries (contact points) are generally similar between successive time-step. Starting from this observation, we propose a ray-racing algorithm which use the result of the previous time-step to exploit this temporal coherency. This ray-tracing algorithm can be used in conjunction to standard ray-tracing algorithms. It is used to replace standard ray-tracing algorithms in specific condition to update the result of the previous time-step instead of computing it from scratch. The iterative ray-tracing algorithm is added in our pipeline as an alternative to other ray-tracing algorithms and is used in priority due to his lower computational cost compared to other algorithms.