DEA Imagerie Vision Robotique 2002 / 2003

Détection de collision et localisation de contact des polyèdres déformables en temps-réel dans un environnement virtuel

Romain Rodriguez

Rapport de DEA

Sous la direction de Kenneth Sundaraj et Christian Laugier

Projet réalisé au sein de l'équipe SHARP,
au Laboratoire GRAphique, VIsion, Robotique
à INRIA Rhône-Alpes
ZIRST, 655 av. de l'Europe
38330 Montbonnot St. Martin
FRANCE

Table des matières

Chapitre 1

Introduction

1.1 Définition du problème

La détection de collision entre deux objets dans un environnement virtuel dynamique est un problème essentiel en infographie, en robotique et en géométrie algorithmique. Il est important d'éviter l'interpénétration des objets et de simuler précisément différents phénomènes physiques afin d'améliorer le réalisme du monde virtuel. Ainsi, il est nécessaire de *détecter les collisions* entre les objets polyédriques. Il est également primordial de déterminer le plus précisément possible les éventuelles zones de l'objet en collision pour pouvoir fournir une simulation fidèle d'un phénomène physique. Malheureusement les algorithmes de détection de collision représentent toujours un goulot d'étranglement dans différents domaines de simulation dynamique.

Ce problème a été largement étudié dans la littérature. Cependant peu d'algorithmes robustes et rapides sont connus pour répondre à des requêtes d'interaction sur objets polyédriques quelconques.

Les scènes complexes restent problématiques. En effet, le coût de la détection de collision est fonction du nombre de paires de primitives élémentaires en collision. Ainsi l'on comprend bien que les objets possédant un nombre important de primitives (faces, arêtes, sommets) sont difficiles à gérer. Or, dans un simulateur chirurgical, les polyèdres représentant des organes possèdent un nombre considérable de primitives. De plus, une scène graphique peut contenir un nombre très important d'objets. Déterminer les interactions entre n objets est un problème dont la complexité est en $\mathcal{O}(n^2)$. Nous pouvons considérer le cas d'un studio virtuel pour illustrer ce problème des n objets en interactions dans lequel des personnages réels seront mêlés à des éléments virtuels.

Il est important d'être capable de détecter les collisions entre des objets se déformant au cours de leur mouvement. Par exemple, dans des simulations physiquement réalistes, les forces entre les objets en collision sont déterminées puis ils sont déformés suivant ces mêmes forces. De manière générale, un utilisateur

doit pouvoir déformer des objets dans un environnement virtuel. La plupart des algorithmes de détection de collision existant se bornent aux objets rigides et sont rarement adaptés aux objets déformables dont la convexité peut changer au cours du temps.

De nombreux algorithmes de détection de collision se bornent aux objets convexes. Or, il est clair que dans un soucis de réalisme, un environnement virtuel doit permettre à l'utilisateur de modéliser des objets concaves et de tester les interactions entre ces éléments. Une solution à ce problème est de représenter un objet concave comme un ensemble d'objets convexes. Le premier inconvénient de cette méthode est qu'elle insère de nouveaux objets à traiter dans la scène. Le second est qu'une telle méthode ne peut en aucun cas être appliquée à la détection de collision entre polyèdres déformables, les déformations subies par un objet modifient souvent sa concavité.

Afin d'illustrer l'intérêt de la détection de collision dans un application différente des simulateurs chirurgicaux et des studios virtuels précédemment cités, nous pouvons considérer l'exemple des planificateurs de trajectoire probabiliste. En effet, au sein d'une telle application, 90 % du temps d'exécution est consacré à la détection de collision.

Ainsi, nous nous intéressons à la détection de collision et la localisation de contact impliquant des objets déformables. Aucune supposition n'est faite sur les déformations au cours du temps. Certains objets peuvent être immobiles, d'autres peuvent être déplacés et/ou déformés. Dans notre étude, un objet devra être polyédrique, il s'agit là d'une faible restriction puisque la plupart des objets d'un environnement virtuel sont modélisés de cette façon.

1.2 Contributions

Dans ce projet, nous avons développé des algorithmes efficaces de détection de collision entre des polyèdres déformables convexes ou concaves. Notre méthode utilise des hiérarchies de volumes englobants que nous mettons à jour suite aux déformations subies par les objets. Elle permet également d'obtenir le contour de collision et le volume d'interpénétration si les objets sont en contact. De plus, nous avons étendu les algorithmes de calcul de distance entre des objets convexes au cas des polyèdres concaves en utilisant des informations tirées de la hiérarchie de volumes englobants. Ces algorithmes ont été testés et implémentés sous la forme d'une librairie intégrée dans le moteur physique aladynLight ainsi que dans le moteur physique développé au sein du projet Sharp en collaboration avec l'entreprise XL-Studio.

1.3 Structure du rapport

Nous allons étudier dans un premier temps les tests d'interférence statique, puis les tests d'interférence dynamique. Ensuite, nous allons expliquer les algorithmes développés nous permettant de résoudre le problème posé et d'aboutir à l'élaboration d'une librairie complète de détection de collision entre objets polyédriques déformables. Nos résultats seront comparés aux travaux existants. Enfin, nous présenterons nos conclusions ainsi que quelques travaux futurs.

Chapitre 2

Tests d'interférence statique

Au cours du processus de détection de collision, on effectue des tests d'intersection statique à différents instants. L'intervalle de temps entre deux tests est supposé suffisamment petit pour que l'on ne manque pas de collisions. Ces tests d'interférence statique (SIT) nous permettent de savoir si, à un instant t donné, deux polyèdres ou bien deux primitives élémentaires s'intersectent. Nous allons étudier les algorithmes de calcul de distances et les tests d'intersection entre deux polyèdres.

2.1 Calcul de distances

Les algorithmes de calcul de distances commencent généralement par déterminer les plus proches **éléments caractéristiques** (sommet, arête, facette) de deux polyèdres. On peut alors calculer la distance euclidienne qui les séparent. Si cette distance est négative, les objets s'interpénètrent. Trois méthodes efficaces ont été proposées pour parvenir à ce résultat. Deux d'entre elles procèdent par un calcul incrémental de la distance entre objets polygonaux convexes obtenue à partir de la distance à l'instant précédent, alors que la troisième méthode explore la frontière des polyèdres afin de trouver la distance minimale. Ces méthodes sont décrites dans les paragraphes suivants.

2.1.1 Dobkin et Kirkpatrick (DK)

Dans [Dobkin and Kirkpatrick, 1990], Dobkin et Kirkpatrick proposent d'utiliser la représentation hiérarchique de deux polyèdres ayant respectivement n et m sommets. La complexité de cette étape initiale est en $\mathcal{O}(n+m)$.

Soit P, un d-polytope. Soit $V(P)$, l'ensemble des sommets de P. Une séquence de polytopes $H(P) = P_1, \ldots, P_k$ est une représentation hiérarchique de P si :

- Le plus bas niveau P_1 représente le polyèdre original, *i.e.* $P_1 = P$ et le plus haut niveau P_k est un d-simplexe ;
- $P_{i+1} \in P_i$, pour $1 \leq i < k$;
- $V(P_{i+1}) \in V(P_i)$, pour $1 \leq i < k$, où $V(P_k)$ l'ensemble des sommets de P_k, et
- Les sommets $V(P_i) - V(P_{i+1})$ forment un ensemble indépendant (*i.e.* ils ne sont pas adjacents) dans P_i, pour $1 \leq i < k$.

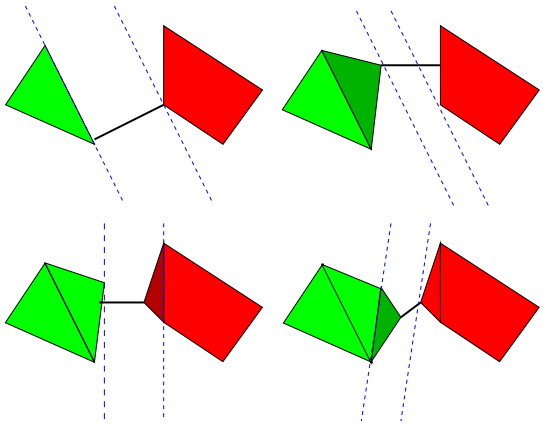

FIG. 2.1 – *Dans l'algorithme DK, les plus proches éléments caractéristiques sont trouvés en explorant seulement les parties de la hiérarchie qui sont en intersection avec le demi-espace défini par les plans dessinés en tirets.*

En utilisant la représentation hiérarchique polyédrique, le calcul de distance se fait dans le cas optimal en $\mathcal{O}(log\ n * log\ m)$. A chaque étape de la recherche de plus proches points correspond un niveau de la représentation hiérarchique. A l'étape initiale, les plus proches points des tétraèdres (le plus bas niveau de la hiérarchie) sont trivialement déterminés. En considérant le segment joignant les plus proches points trouvés à une étape donnée, deux plans perpendiculaires à sa direction et tels qu'ils soient en contact avec chaque polyèdre (au niveau considéré) peuvent être déterminés. Ces plans délimitent la zone où la prochaine paire de plus proches points doit être cherchée. L'intersection de cette portion de l'espace avec le polyèdre construit au prochain niveau peut être deux simplexes, un simplexe ou bien un ensemble vide. Si les plus proches points ne sont pas les mêmes qu'à l'étape précédente, alors au moins l'un d'entre eux appartient à l'un de ces simplexes. Ainsi, chaque étape de recherche est au plus limitée à deux simplexes. Le nombre d'étapes est borné par $log\ n * log\ m$. La figure 2.1 illustre cette procédure.

2.1.2 Gilbert, Johnson et Keerthi (GJK)

Cet algorithme [Gilbert et al., 1988] calcule la distance entre les enveloppes convexes de deux ensembles de points X et Y. Il donne la valeur de cette distance dans le cas où les deux enveloppes convexes sont séparées, et une approximation de la distance d'interpénétration dans le cas contraire.

Afin de comprendre cet algorithme, nous définissons les notations suivantes :

L'enveloppe convexe d'un ensemble de points X, (C_X) est donnée par :

$$C_X = \sum_{i=1}^{n} \lambda^i \boldsymbol{x}_i : \boldsymbol{x}_i \in X,\ \lambda^i \geq 0,\ \lambda^1 + \lambda^2 + \ldots + \lambda^n = 1 \tag{2.1}$$

La distance euclidienne entre les deux enveloppes convexes est définie par :

$$d(C_X, C_Y) = \min\{|\boldsymbol{x} - \boldsymbol{y}|,\ \boldsymbol{x} \in C_X,\ \boldsymbol{y} \in C_Y\} \tag{2.2}$$

Cette distance est égale à la distance entre la différence de Minkowski de X et Y et l'origine du repère $\mathbb{O}$. Ce résultat a été démontré dans [Gilbert et al., 1988].

La différence de Minkowski $C_X \ominus C_Y$ est définie par :

$$C_X \ominus C_Y = \{\boldsymbol{z} : \boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y},\ \boldsymbol{x} \in C_X,\ \boldsymbol{y} \in C_Y\} \tag{2.3}$$

Ainsi, nous avons :

$$d(C_X, C_Y) = \min\{|\boldsymbol{z}| : \boldsymbol{z} \in C_X \ominus C_Y\} \tag{2.4}$$

La différence de Minkowski de deux polyèdres est un polyèdre convexe. Afin de calculer la distance entre $C_X \ominus C_Y$ et $\mathbb{O}$, nous introduisons la fonction support $\mathcal{H}$ définie sur un ensemble quelconque Z par :

$$\mathcal{H}_Z(\boldsymbol{\eta}) = \max\{\boldsymbol{z} \cdot \boldsymbol{\eta} : \boldsymbol{z} \in Z\} \tag{2.5}$$

où $\boldsymbol{z} \cdot \boldsymbol{\eta}$ est la projection du point $\boldsymbol{z}$ sur l'axe $\boldsymbol{\eta}$. Ceci signifie que la fonction $\mathcal{H}$ donne la projection du point le plus loin possible dans la direction $\boldsymbol{\eta}$ (voir la figure 2.2).

Le point $\boldsymbol{M}_Z$ ($\boldsymbol{z}$ de Z) le plus proche de $\mathbb{O}$ est le plus proche dans une direction et le plus loin dans la direction opposée. Ainsi ce point vérifie la relation suivante :

$$|\boldsymbol{z}|^2 + \mathcal{H}_Z(-\boldsymbol{z}) = 0 \tag{2.6}$$

Dans l'algorithme GJK, pour trouver le point le plus proche de $\mathbb{O}$ pour un ensemble compact Z, on procède de la manière suivante :

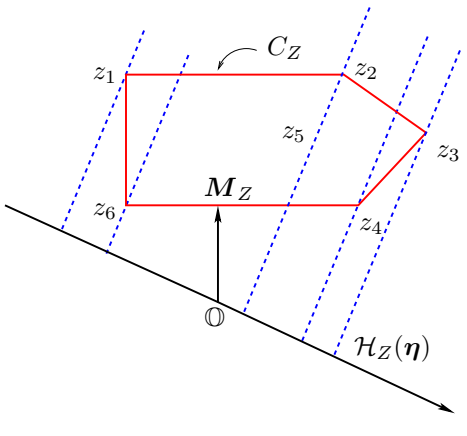

FIG. 2.2 – *La fonction support $\mathcal{H}_Z(\boldsymbol{\eta})$ donne la projection du point le plus loin possible dans la direction $\boldsymbol{\eta}$. Le point $\boldsymbol{M}_Z$ le plus proche de $\mathbb{O}$ est tel que $|\boldsymbol{M}_Z|^2 + \mathcal{H}_Z(-\boldsymbol{M}_Z) = 0$.*

- On commence par un ensemble initial de points $Z_k = z_1, z_2, \ldots, z_m$. Pour $k = 0$, Z_0 est composé de trois points non alignés en $2D$ ou de quatre points non coplanaires en $3D$.
- On détermine alors le point $\boldsymbol{\nu}_k$ de Z_k le plus proche de $\mathbb{O}$. Ceci peut-être fait de manière très efficace en utilisant l'algorithme de Johnson [Johnson, 1987].
- Si $|\boldsymbol{\nu}_k|^2 + \mathcal{H}_Z(-\boldsymbol{\nu}_k) = 0$ alors $\boldsymbol{\nu}_k$ est le point de C_Z le plus proche de $\mathbb{O}$.
- Sinon $Z_{k+1} = \hat{Z}_k \cup S_Z$ où S_Z est une solution de $h_Z(\boldsymbol{\nu})$ et $\hat{Z}_k \subset Z_k$ a moins de m éléments. S_Z satisfait aussi $S_Z \in C_{\hat{Z}_k}$. On recommence dès l'étape 2.

La figure 2.3 montre un exemple où $Z = z_1, z_2, z_3, z_4, z_5$, $Z_0 = z_1, z_2, z_3$, $\hat{Z}_0 = z_2, z_3$, $Z_1 = \hat{Z}_0 \cup z_4$, $\hat{Z}_1 = z_3, z_4$, $Z_2 = \hat{Z}_1 \cup z_5$ et $\boldsymbol{\nu}_2$ est le point le plus proche de Z à $\mathbb{O}$.

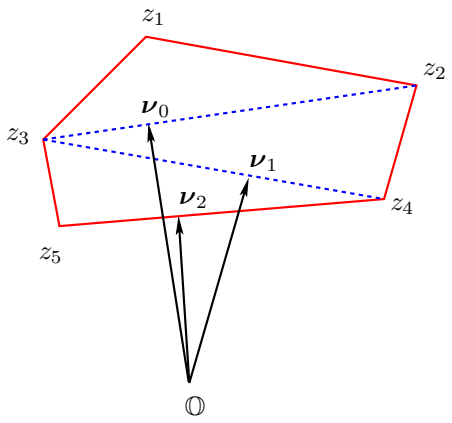

FIG. 2.3 – *Dans cet exemple, l'algorithme GJK converge en trois itérations*

Ainsi, dans l'algorithme GJK, pour calculer la distance entre les enveloppes convexes de deux ensembles de points X et Y, on calcule la distance entre $C_X \ominus C_Y$ et $\mathbb{O}$. L'étape la plus coûteuse correspond

à l'obtention de $C_X \ominus C_Y$. Cela se fait, dans le cas général, en $m * n$ opérations où m et n sont respectivement les nombres de points de X et de Y. Cette complexité peut être réduite grâce à la propriété de h suivante :

$$h_{C_X \ominus C_Y}(\eta) = h_{C_X}(\eta) - h_{C_Y}(\eta) \tag{2.7}$$

En utilisant cette propriété, le calcul de $h_{C_X \ominus C_Y}$ nécessite seulement $m + n$ opérations. Néanmoins, le nombre d'itérations effectuées dépend du choix initial de Z_0. Dans le pire des cas, $m + n$ itérations sont effectuées, ce qui donne une complexité en $\mathcal{O}(m + n)$.

Plusieurs extensions de l'algorithme GJK ont été proposées pour des applications spécifiques. Nous allons décrire les principales :

Enhanced GJK (EGJK) :

Cameron, dans [Cameron, 1997], a montré que dans le cas où les deux objets ont un mouvement continu, pour des rotations de faible amplitude, les nouveaux plus proches points seront dans le voisinage des plus proches points de l'instant précédent. Ainsi, si l'on acquiert au préalable des informations sur la topologie des voisinages, il est possible, en utilisant Z_k précédemment obtenu, de réduire la complexité à $\mathcal{O}(1)$. Cette technique est nommée *hill-climbing*. Cette extension se révèle très intéressante dans le cas où les objets bougent très lentement comme, par exemple, dans les simulateurs médicaux.

Joukhadar et Laugier (JL) :

Cet algorithme [Joukhadar et al., 1996] donne la distance négative [1] entre les enveloppes convexes des ensembles X and Y. Elle s'apparente à une mesure de l'interpénétration fictive quand X et Y sont en collision. Cet algorithme procède de la manière suivante :

- On sépare X et Y suivant la dernière direction de contact $\boldsymbol{N}$.
- On applique l'algorithme GJK pour obtenir une nouvelle valeur de la direction de contact $\boldsymbol{N'}$.
- Si $\boldsymbol{N} == \boldsymbol{N'}$ alors la distance négative est $|\boldsymbol{N}| - |\boldsymbol{N'}|$. Sinon $\boldsymbol{N} := \boldsymbol{N'}$ et on recommence à l'étape 1.

A chaque itération i, la valeur de $|\boldsymbol{N}_i|$ est plus proche de $|\boldsymbol{N}_{i-1}|$. Donc, cet algorithme converge après un nombre fixé d'itérations pour obtenir $\boldsymbol{N}$. Sa complexité est en $\mathcal{O}\{k(m+n)\}$ où k est le nombre d'itérations nécessaires à la convergence. Cette méthode est illustrée par la figure 2.4.

Cette méthode est intéressante pour le traitement de collision entre deux objets rigides. Si l'on utilise un modèle de *pénalité* pour le traitement de collision, la distance négative donnée par l'algorithme JL

[1] La distance entre deux polyèdres est dite positive quand les deux polyèdres ne s'intersectent pas. Dans le cas contraire, leur *distance négative* est la norme du plus petit vecteur de translation nécessaire pour les séparer.

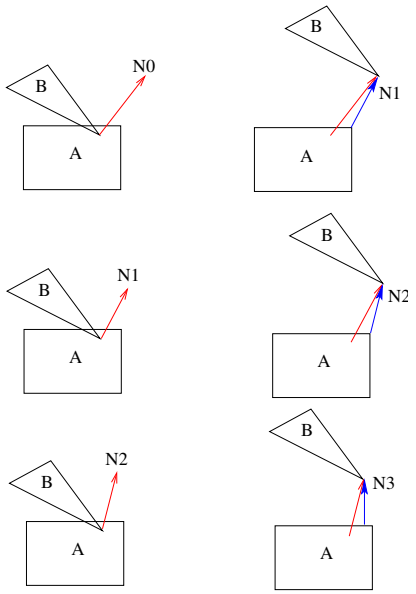

FIG. 2.4 – *Afin de trouver la distance négative par l'algorithme JL, on sépare A et B en utilisant la direction $\boldsymbol{N}_{i-1}$ telle que les enveloppes convexes ne soient pas en collision. Ensuite, l'algorithme GJK donne $\boldsymbol{N}_i$. Dans cet exemple $\boldsymbol{N}_4 = \boldsymbol{N}_3$, donc la distance négative est donnée par $\boldsymbol{N}_4 - \boldsymbol{N}_3$.*

peut être utilisée comme une mesure de l'interpénétration fictive pour calculer les forces de collision. Cette force est alors renvoyée à l'utilisateur par une interface haptique.

SOLID :

Cette librairie [Van der Bergen, 1999] implémente l'algorithme GJK avec certaines modifications afin de palier au problème suivant. GJK a tendance à générer des simplexes qui sont progressivement plus oblongs quand le nombre d'itérations k augmente. Donc l'erreur du vecteur de distance minimale $\boldsymbol{\nu}_k$ ne peut pas être facilement estimée.

Dans SOLID, on estime l'erreur de $\boldsymbol{\nu}_k$ en maintenant une borne inférieure de Z qui est la distance de l'origine au plan support $\mathcal{H}_Z\{-\boldsymbol{\nu}_k, (\boldsymbol{\nu}_k \cdot \boldsymbol{\eta})\}$, elle est donné par la formule suivante :

$$\delta_k = \frac{\boldsymbol{\nu}_k \cdot \boldsymbol{\eta}}{|\boldsymbol{\nu}_k|} \tag{2.8}$$

Il s'agit d'une borne inférieure appropriée puisque pour δ_k positif, l'origine est dans le demi-espace positif, alors que $C_X \ominus C_Y$ est dans le demi-espace négatif. Mais cette borne inférieure ne peut pas être monotone en k, *i.e.* on peut avoir $\delta_j < \delta_i$ pour $j > i$. On préférera donc utiliser la formule suivante

comme borne inférieure, elle est souvent plus compacte que δ_k :

$$\mu_k = \max\{0, \delta_0, \delta_1, \ldots, \delta_k\} \tag{2.9}$$

Étant donné ϵ, la tolérance de l'erreur absolue dans $|\boldsymbol{\nu}_k|$, l'algorithme s'arrête dès que $|\boldsymbol{\nu}_k| - \mu_k \leq \epsilon$.

Un autre modification importante de l'algorithme GJK est l'utilisation d'un test (appelé Separating Axis Test) afin d'accélérer les conditions d'arrêt de GJK. Si deux objets ne sont pas en collision, on n'a pas besoin de déterminer explicitement la distance entre eux pour connaître leurs situations. On a simplement besoin de savoir si la distance qui les séparent est nulle ou pas. Donc, la borne inférieure δ_k peut être utilisée comme critère d'arrêt pour savoir si les deux objets s'intersectent ou pas.

La borne inférieure est positive si et seulement si :

$$\boldsymbol{\nu}_k \cdot \boldsymbol{\eta} > 0 \tag{2.10}$$

i.e. $\boldsymbol{\nu}_k$ est un axe séparateur de X et Y. Dans le cas général, l'algorithme GJK a besoin de moins d'itérations pour trouver un axe séparateur pour un paire d'objets qui ne s'intersectent pas que pour calculer une approximation de $\boldsymbol{z}$:

$$\lim_{k \to \infty} \boldsymbol{\nu}_k = \boldsymbol{z} \tag{2.11}$$

En plus de nécessiter moins d'itérations dans le cas d'objets qui ne s'intersectent pas, cet algorithme offre de meilleurs résultats puisque l'on n'a pas besoin de calculer ou d'initialiser $|\boldsymbol{\nu}_k|$. Le calcul de $|\boldsymbol{\nu}_k|$ nécessite l'évaluation coûteuse d'une racine carrée. Ainsi, une itération de cet algorithme est sensiblement moins coûteuse que l'algorithme GJK original.

2.1.3 Lin et Canny (LC)

Cet algorithme [Lin and Canny, 1991] calcule également les deux plus proches éléments caractéristiques de deux soupes de polygones, X et Y, de respectivement p et q éléments caractéristiques (sommets, arêtes, facettes). A la différence de GJK, on doit obtenir au préalable des informations sur la topologie du voisinage. Cependant, cet algorithme donne deux éléments caractéristiques seulement dans le cas où il n'y a pas de contacts entre les deux enveloppes convexes des objets, sinon il entre dans une boucle infinie.

Ici, chaque élément caractéristique (sommet, arête, facette) d'un polyèdre est associé à une *région de Voronoi*, constituée de l'ensemble des points plus proches de cet élément que d'un autre. A chaque

itération, on vérifie grâce à une critère local que les deux éléments caractéristiques sont effectivement les plus proches. On procède de la façon suivante :

- Les points les plus proches des éléments caractéristiques dans les ensembles X et Y sont tout d'abord calculés.
- Si le point le plus proche de la caractéristique considérée dans X est dans la région de Voronoi de la caractéristique considérée dans Y, et que le point le plus proche de la caractéristique considérée dans Y est dans la région de Voronoi de la caractéristique considérée dans X, alors les points les plus proches des caractéristiques considérées sont aussi les points les plus proches de X et Y.
- Sinon, en utilisant la topologie du voisinage, de nouveaux éléments caractéristiques de X et Y sont trouvés. Les éléments caractéristiques mis à jour doivent être plus proches que les précédents. On reprend alors l'itération à l'étape précédente.

A chaque itération, on teste si le point le plus proche de l'élément caractéristique se trouve dans la région de Voronoi de l'autre élément caractéristique, on est donc ramené à trois cas :

- P est-il le point le plus proche du sommet S ?
- P est-il le point le plus proche de l'arête A ?
- P est-il le point le plus proche de la facette F ?

La région de Voronoi d'un point, d'une arête ou d'une facette d'un polyèdre est illustrée dans la figure 2.5.

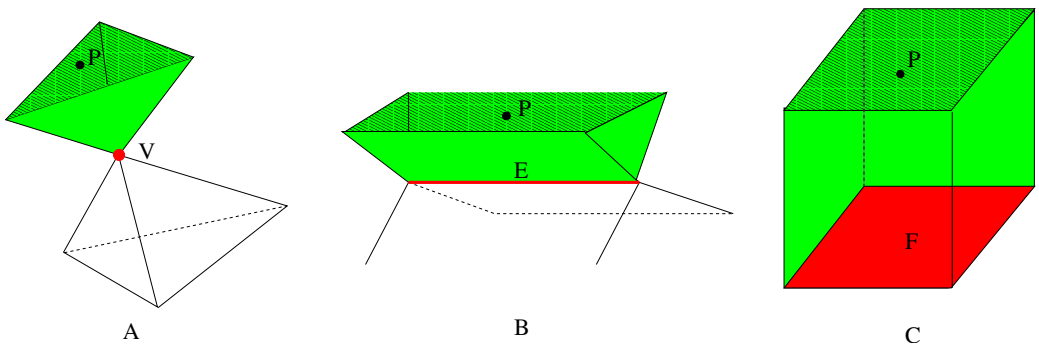

FIG. 2.5 – *Les régions de Voronoi d'un point, d'une arête et d'une facette d'un polyèdre convexe sont notées en vert. On teste si le point P, le plus proche de l'élément caractéristique d'un autre objet, est dans ces régions.*

Nous remarquons que si les objets sont en contact, l'algorithme entre dans une boucle infinie. Afin de remédier à ce problème, certains ont proposé l'utilisation de la notion de *pseudo région de Voronoi* afin de définir les régions de Voronoi intérieures à l'objet pour déterminer si les deux objets s'intersectent ou pas.

La complexité de ces algorithmes est linéaire en le nombre d'éléments caractéristiques, en d'autres termes $\mathcal{O}(p+q)$. De plus, comme dans GJK, la complexité dépend du choix initial des éléments caractéristiques. Dans le cas où les plus proches éléments caractéristiques de l'instant précédent sont réutilisés et que les objets n'ont pas trop bougé, l'algorithme est en $\mathcal{O}(1)$. Une extension de l'algorithme de LC fut proposée dans l'implémentation suivante :

V-Clip :

Cet algorithme [Mirtich, 1998] est une implémentation robuste de l'algorithme LC avec des changements mineurs. En effet, au lieu de calculer les points les plus proches de deux éléments caractéristiques à chaque itération, des opérations de clipping et des tests d'intersection entre les caractéristiques sont utilisés. Les configurations dégénérées ne sont pas problématiques dans V-Clip et l'implémentation est plus simple que celle de Lin-Canny. Les temps d'exécution de V-Clip sont sensiblement les mêmes que ceux de LC.

L'idée sur laquelle repose cet algorithme est appelée *Voronoi Clipping* et est décrite dans la figure 2.6. En effet dans cet exemple, dès que l'arête E est divisée suivant les points P et Q, on doit déterminer si le point de E le plus proche de F appartient à $K \supseteq \mathcal{VR}(X)$, où K est la région convexe délimitée par les plans définissant la région de Voronoi $\mathcal{VR}(X)$, et, dans le cas contraire, on doit mettre à jour F, les candidats potentiels sont M et N.

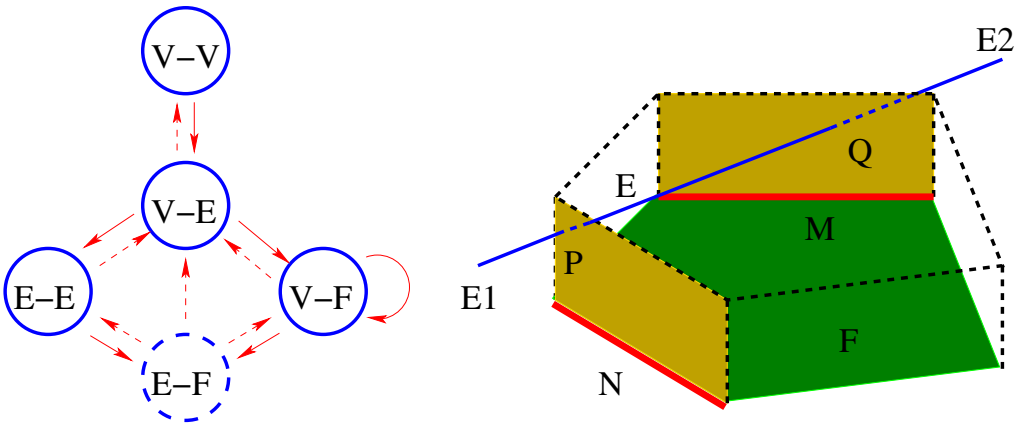

FIG. 2.6 – *Gauche : Cas possibles de transitions de l'algorithme V-Clip entre le sommet (V), l'arête (E) et la facette (F). Droite : L'arête (E) d'extrémité $E1$ et $E2$, opposée à la facette F. La région de Voronoi de F est notée $\mathcal{VR}(F)$. L'arête E intersecte $\mathcal{VR}(F)$ sur deux plans (en couleur dorée) en les points P et Q. Cité de [Mirtich, 1998].*

Définition 1. *Soient E une arête, et $e(\lambda)$ un point paramétré de E. Pour un élément caractéristique X, la* **fonction distance à l'arête** $D_{E,X}(\lambda) : \Re \rightarrow \Re$ *est définie de la façon suivante :*

$$D_{E,X}(\lambda) = \min_{x \in X} \|x - e(\lambda)\|$$

La mise à jour de ces caractéristiques se fait en vérifiant les signes des dérivées des fonctions de distance $D_{E,F}$ entre E et F (voir la définition 1) en P et Q. Si $N \neq \emptyset$ et $D_{E,F}(P) > 0$ alors F est mis à jour à N, sinon si $M \neq \emptyset$ et $D_{E,F}(Q) < 0$ alors F est mis à jour à M. Des fonctions de distance similaires existent pour les autres combinaisons possibles de caractéristiques. Pour les cas dégénérés où la fonction de distance n'est pas dérivable, l'algorithme retourne simplement la distance de pénétration en indiquant aussi l'intersection de l'arête E et l'autre caractéristique.

On remarque que tous les cas de combinaisons possibles de caractéristiques ne sont pas présentes dans l'algorithme. Au niveau géométrique, pour un polyèdre convexe, la distance minimale peut toujours être exprimée entre deux sommets, un sommet et une arête, un sommet et une facette et une arête et une arête. Le cas d'une arête et d'une facette peut toujours être ramené aux caractéristiques d'une dimension inférieure. Puisque la solution pour un sommet avec les plans de Voronoi d'un sommet, d'une arête ou d'une facette est évidente, la seule difficulté apparaît quand une arête est testée avec un sommet, une arête ou une facette. Le cas de la facette n'apparaît presque jamais (voir le figure 2.6-Gauche).

2.2 Tests d'intersection

Une autre forme d'algorithme SIT est le test d'intersection. L'interaction peut être détectée si au moins deux primitives s'intersectent. Nous allons considérer différents types de paires de primitives : sphère/sphère, boîte/boîte et triangle/triangle. Ce choix est justifié par le fait que ces cas apparaissent dans la *phase de vérification* finale de la plupart des algorithmes de détection de collision.

2.2.1 Sphère/Sphère

Le test d'intersection sphère/sphère est le plus simple et le plus rapide. Soient deux sphères A et B de rayon respectif r_A et r_B, l'intersection apparaît dès que :

$$|\boldsymbol{p}_A - \boldsymbol{p}_B| < r_A + r_B \tag{2.12}$$

où $\boldsymbol{p}$ désigne la position.

2.2.2 Triangle/Triangle

Un test d'intersection entre des primitives triangulaires dites *facettes* est essentiel car la plupart des maillages polyédriques sont en fait des maillages triangulaires.

Actuellement, le test le plus efficace est l'algorithme de Moller [Moller, 1997], la procédure permettant de déterminer l'intersection entre les facettes A et B est :

- On calcule l'équation de plan P_B de la facette B.
- Si tous les points de la facette A sont du même coté de P_B, on sort.
- Sinon, on calcule l'équation de plan P_A de la facette A.
- Si tous les points de la facette B sont du même coté de P_A, on sort.
- Sinon, il existe une droite d'intersection L_{AB} passant par A et B.
- On projette L_{AB} sur le plus grand axe et on calcule les intervalles I_A et I_B définissant $L_{AB} \cap (A \cup B)$.
- Si $I_A \cap I_B \neq \emptyset$, A et B s'intersectent, sinon A et B ne s'intersectent pas.

La figure 2.7 illustre l'intersection des intervalles sur L_{AB}. Le code implémenté par [Moller, 1997] a été optimisé et sa robustesse vérifiée.

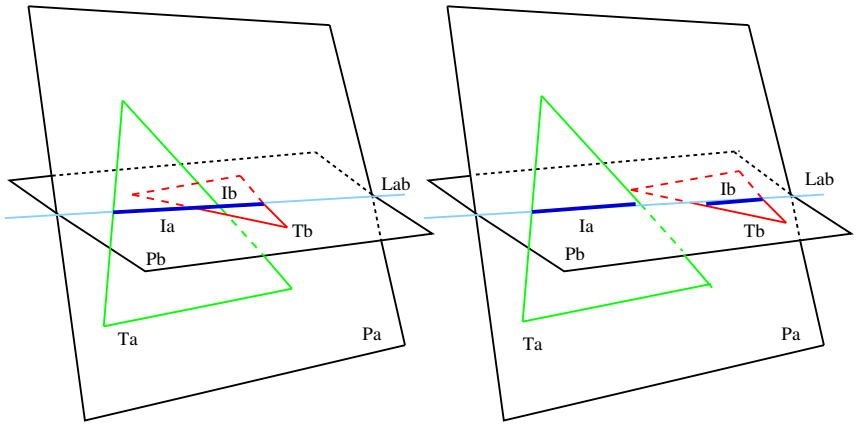

FIG. 2.7 – *Test d'intersection entre les triangles T_a et T_b appartenant respectivement aux plans P_a et P_b. Gauche : Les intervalles I_a et I_b (en bleu foncé) s'intersectent. Droite : Pas d'intersection sur la droite des intervalles sur L_{ab} (en bleu clair), donc il n'y a pas d'intersection entre les triangles.*

Des améliorations ont été apportées à cet algorithme. Il est maintenant aussi capable d'obtenir les deux points définissant L_{AB} quand l'intersection est détectée. Les facettes coplanaires peuvent s'intersecter. Si l'on suppose que les facettes coplanaires ne peuvent être qu'en *contact*, il est possible d'éviter un nombre important de tests dans le cas de facettes coplanaires, et donc d'accélérer le processus.

2.2.3 Boîte/Boîte

Le test d'intersection boîte/boîte est également très important. En effet, les représentation hiérarchiques utilisant un arbre de boîtes englobantes utilisent ce test pour descendre dans l'arbre. L'algorithme le plus rapide est le **Separating Axis Test** (SAT).

Soient les deux boîtes A et B, la position relative de B par rapport à A est définie par la rotation R et la translation T. Considérons les vecteurs de bases de A et B, $\boldsymbol{A}^i$ et $\boldsymbol{B}^i$ pour $i = 1, 2, 3$. Soient les dimensions aux centres de A et B a^i et b^i pour $i = 1, 2, 3$. La figure 2.8 illustre ces notations.

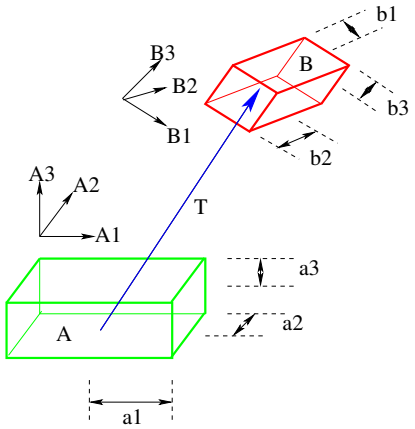

FIG. 2.8 – *Notations pour l'algorithme SAT*

Considérons le théorème suivant :

Théorème 1. Théorème de l'axe séparateur *: Si deux boîtes ne sont pas en contact alors il existe un axe séparateur* $\boldsymbol{L} = \boldsymbol{M} \times \boldsymbol{N}$ *où* $\boldsymbol{M}$ *et* $\boldsymbol{N}$ *sont des vecteurs distincts choisis parmi les six axes des boîtes.*

Il montre que 15 projections axiales sont suffisantes pour déterminer si deux boîtes orientées et positionnées arbitrairement sont en contact. Ainsi, on peut maintenant montrer que $\boldsymbol{L}$ est un axe séparateur s'il satisfait la condition suivante :

$$|\boldsymbol{T} \cdot \boldsymbol{L}| > \sum_i |(a_i \boldsymbol{A}^i) \cdot \boldsymbol{L}| + \sum_i |(b_i \boldsymbol{B}^i) \cdot \boldsymbol{L}| \tag{2.13}$$

L'équation 2.13 peut être simplifiée, ainsi dans le pire cas environs 200 opérations sont effectuées. Notons que durant l'exécution, les 15 axes ne sont pas tous testés. L'algorithme s'arrête dès la détection du premier axe.

Chapitre 3

Tests d'interférence dynamique

Nous allons décomposer le processus de détection de collision entre des polyèdres déformables en plusieurs phases. Dans un premier temps, on doit connaître les déformations subies par un objet afin de mettre à jour les éventuelles structures de données intermédiaires utilisées. Puis, nous détectons la collision potentielle entre les différents objets de la scène. En cas de collision, on doit connaître le contour de collision, c'est-à-dire les paires de primitives élémentaires en contact, puis le volume d'interpénétration des objets en collision. Enfin, nous devons savoir si les déformations subies par l'objet sont telles qu'il se trouve lui-même en auto-collision, *i.e.* deux arêtes non adjacentes de l'objet sont en intersection.

3.1 Détection de collision

La détection d'interaction entre deux objets virtuels est un processus connu sous le nom de *détection de collision*. Les différentes approches utilisées dépendent de la représentation de l'objet qui peut-être un modèle non polyédrique (*e.g.* surfaces implicites, surfaces paramétriques, surfaces de subdivision) ou polyédrique (*e.g.* soupe de polygones, objets structurés concaves ou convexes). Ici, nous traiterons seulement le cas polyédriques.

Dans le chapitre précédent, nous avons vu que pour les objets structurés convexes, le calcul de distance peut être utilisé pour tester l'interaction. Aucun pré-calcul n'est nécessaire pour les objets convexes. Mais, la plupart des objets réels sont concaves et déformables et leur concavité peut changer durant la déformation. Ainsi, des structures de données supplémentaires sont nécessaires à4

3.2 Contributions

une détection précise et efficace de l'interaction. Elles seront les principales données entrantes des procédures de détection de collision.

On peut décomposer la détection de collision entre objets déformables en deux étapes :

- **Phase de détection grossière** : Elle ne permet pas de déterminer exactement les entités en collision, mais elle donne une idée *grossière* de la région de l'objet sur laquelle une collision peut exister.
- **Phase de détection exacte** : Elle permet de connaître exactement, si elles existent, les entités en collision.

Dans le cas général, on applique en premier le phase de détection grossière sur les structures de données intermédiaires afin d'avoir une idée approximative de l'éventuelle zone de contact, on procède ensuite à le phase de détection exacte afin de déterminer exactement les entités en collision.

3.2.1 Détection grossière

Le but de cette étape dans l'algorithme de détection de collision est d'éliminer rapidement les parties de l'objet dont on a la certitude qu'elles n'intersectent aucun autre objet. Les algorithmes utilisés dans cette phase utilisent généralement une approche basée sur une décomposition spatiale et/ou sur des hiérarchies de volumes englobants.

3.2.1.1 Approche basée sur une décomposition spatiale

Parmi les principaux algorithmes de découpage de l'*espace*, on peut citer les grilles tridimensionnelles, les arbres octaux dits *octrees* ou encore les arbres binaires pour le partitionnement de l'espace (BSP).

Pour les grilles [Garcia-Alonso et al., 1994], l'espace est découpé en N sur chacune des dimensions soit en $N \times N \times N$ cellules dites *voxels*. Ces grilles peuvent être régulières ou adaptatives.

Pour les octrees [Hamada and Hori, 1996], l'espace est découpé en huit voxels contigus et de même taille ; chacun de ces voxels sera à son tour découpé en huit sous-voxels, et ainsi de suite, les voxels vides n'étant pas divisés. La structure de données est donc un arbre dont les noeuds sont les voxels contenant des surfaces, et les feuilles les voxels vides ou les voxels de précision suffisante.

L'octree permet un gain de mémoire par rapport aux grilles grâce à la moindre occupation des zones vides, qui sont en grande majorité. Cependant la durée d'exploration est plus importante pour les octrees que pour les grilles.

On teste la collision en identifiant les voxels communs entre une paire d'objets. Si un voxel commun est trouvé, alors chaque triangle obstacle associé à chacun de ces voxels sera étudié à l'étape suivante afin de déterminer s'il y a interaction. Une absence de voxel commun implique nécessairement une absence de collision.

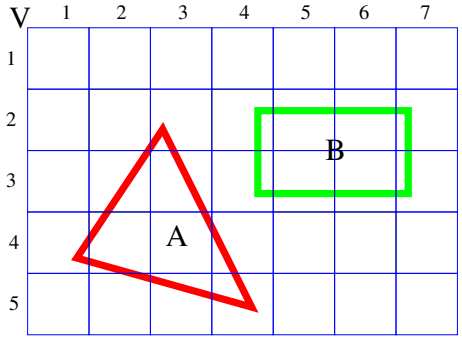

FIG. 3.1 – *Partitionnement de l'espace : Les voxels de la grille sont associés aux objets A et B. Si aucun voxel commun n'est trouvé, alors il ne peut y avoir d'interaction, sinon il peut y avoir collision et, dans ce cas, son existence sera vérifiée grâce aux primitives obstacles.*

Un arbre BSP [Naylor et al., 1990], [Thibault and Naylor, 1987] est une division récursive de l'espace qui considère chaque polygone comme un plan de coupe. Ce plan est utilisé pour classer tous les objets restant comme étant soit devant, soit derrière ce plan. Un exemple de division élémentaire est montré dans la figure 3.2. Autrement dit, quand on insère un polygone dans l'arbre, on le classe relativement à chaque noeud fils approprié.

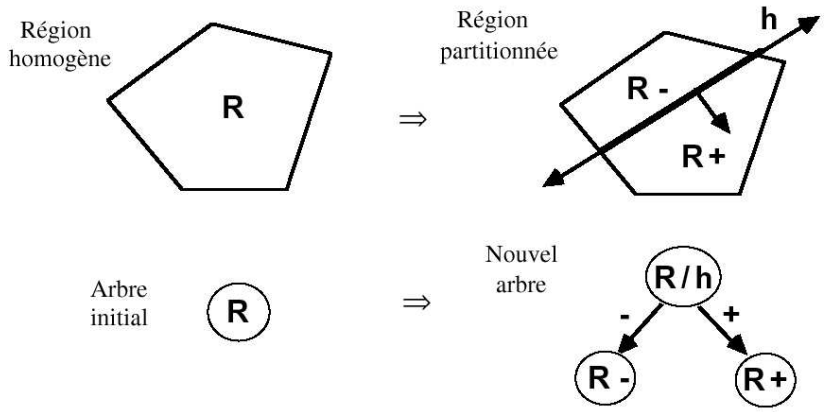

FIG. 3.2 – *Etape de division élémentaire de l'espace suivant un plan de coupe*

Un exemple de construction d'un arbre BSP est montré dans la figure 3.3.

Détecter si deux objets A et B s'intersectent nous ramène au problème de fusion des arbres.

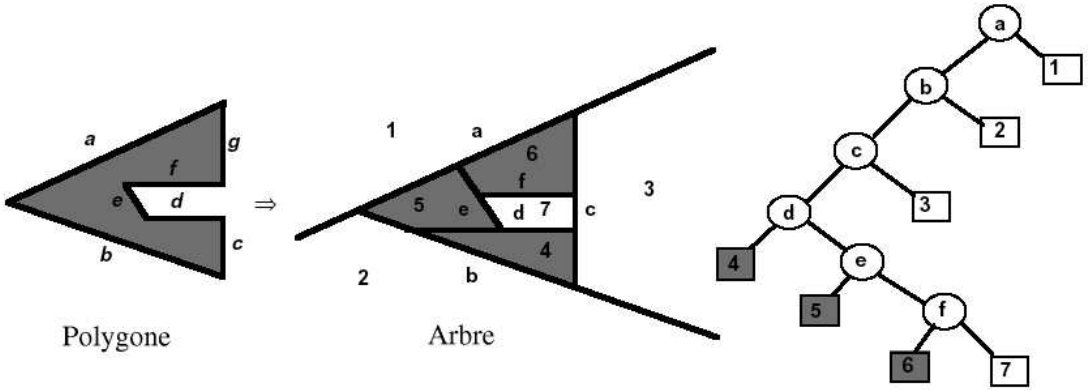

FIG. 3.3 – *Construction d'un arbre BSP à partir d'un polygone*

Les relations spatiales entre A et B, chacun étant représenté par un arbre BSP, peuvent être déterminées efficacement en fusionnant les deux arbres. On procède de la même façon que si l'on fusionnait des arbres de volumes englobants. En effet, dans ce cas, on effectue une descente récursive dans l'arbre, et l'on compare deux noeuds afin de déterminer si les volumes englobants qu'ils représentent s'intersectent, dans ce cas, on descend dans l'arbre afin de préciser la zone de contact. Si ce n'est pas le cas, on ne descend pas. Dans le cas des arbres BSP, pour fusionner les arbres $T1$ et $T2$, le plan de chaque noeud de $T1$ permet de partitionner $T2$ en deux moitiés. Puis, chaque moitié est fusionnée avec le sous-arbre de $T1$ placé du même coté par rapport au plan.

3.2.1.2 Approche basée sur des volumes englobants

Un volume englobant permet de délimiter une zone d'intérêt autour de l'objet virtuel. Par exemple, une sphère ou une boîte peuvent approximer une primitive polygonale, une soupe de polygones ou un objet complet. Le problème de détection de collision est réduit au test de superposition des deux volumes englobants, plutôt qu'aux objets directement. Quand la complexité d'un objet augmente, une hiérarchie de volumes englobants est utilisée. Il s'agit généralement d'un arbre de sphères ou bien de boîtes. Chaque volume délimite une ou plusieurs primitives géométriques. Un ensemble de ces volumes, dit *parent*, délimite l'espace de toutes les primitives géométriques de ses feuilles. Généralement, chaque volume englobant est optimisé en termes de volume, d'aire de surface, de diamètre, *etc.* dans le but d'avoir la meilleure compacité autour de la primitive encadrée. Selon les choix de conception, les feuilles d'un arbre peuvent contenir une unique primitive géométrique ou une collection de primitives géométriques.

La compacité d'un volume englobant influence fortement l'efficacité globale d'une hiérarchie. Plus le volume englobant est compact autour de l'objet, moins l'on doit effectuer de tests de superposition. La rapidité d'un test dépend du type de volume utilisé, à savoir, les Axis Aligned Bounding Boxes (AABB)

[Van der Bergen, 1997], les Oriented Bounding Boxes (OBB) [Gottschalk et al., 1996] ou les sphères [Hubbard, 1996].

Le test de contact entre deux sphères étant trivial, nous allons nous concentrer sur les boîtes.

Choisir des boîtes comme volume englobant implique aussi un choix sur l'orientation de la boîte afin d'obtenir la meilleure compacité et sur un critère de séparation des fils. Les OBB sont normalement orientées en utilisant une analyse en composantes principales sur les axes (les valeurs propres de la matrice de covariance de la distribution des sommets). Alors que les AABB sont orientées parallèlement aux axes du repère local de l'objet (voir la figure 3.4). La séparation des fils dépend du choix de construction, haut-bas ou bas-haut. En règle générale, la stratégie de construction de la hiérarchie bas-haut est considérée comme plus efficace, mais beaucoup de règles de fusion des frères et de partitionnement des fils existant rendent très difficile cette comparaison. Le coût d'une hiérarchie de volumes englobants peut être évalué en utilisant le critère suivant [Gottschalk et al., 1996] :

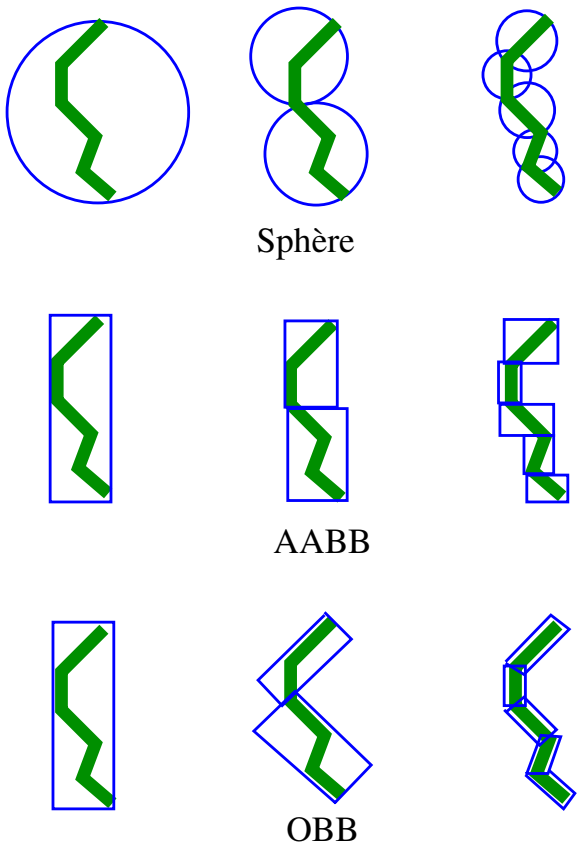

FIG. 3.4 – *Hiérarchies de volumes englobants. Haut : Les sphères englobantes ne sont pas très compactes mais permettent le test de superposition le plus rapide. Centre : Les AABB sont plus compactes mais les tests de superposition sont moins rapides. Bas : Les OBB sont les plus compactes et nécessitent le moins de tests de superposition. On note que pour les AABB et les OBB, l'algorithme de test de l'axe séparateur (SAT) permet de déterminer s'il y a contact de manière optimale.*

$$\mathcal{T} = (\mathcal{N} \times \mathcal{B}) + (\mathcal{P} \times \mathcal{C}) \tag{3.1}$$

où $\mathcal{T}$ est le fonction de coût total de la détection d'interférence, $\mathcal{N}$ est le nombre de tests de superposition des noeuds, $\mathcal{B}$ est le coût d'un test superposition entre deux noeuds, $\mathcal{P}$ est le nombre de paires de primitives testées pour l'interférence et $\mathcal{C}$ est le coût d'un test d'interférence pour une paire de primitives. Afin de diminuer $\mathcal{N}$ et $\mathcal{P}$, le volume englobant doit être le plus compact possible autour du modèle original. Afin de diminuer $\mathcal{B}$, l'algorithme utilisé pour le test de superposition boîte/boîte doit être optimal. $\mathcal{C}$ est minimisé pour le meilleur algorithme de test d'interférence facette/facette. Un résultat intéressant concernant le choix de l'algorithme pour le test boîte/boîte est donnée dans le tableau 3.1.

Test de l'axe Séparateur (SAT)	Plus proches caractéristiques (LC / GJK)	Programmation Linéaire
$5 - 7\ \mu s$	$45 - 105\ \mu s$	$180 - 230\ \mu s$

TAB. 3.1 – *Performance de algorithmes de tests de superposition des boîtes. Cité de [Gottschalk et al., 1996]*

3.2.2 Détection exacte

Cette étape permet de calculer de façon exacte les entités en collision une fois que la phase de *détection grossière* a donné une approximation de la zone de contact. Cette approximation est représentée par une liste de couples de feuilles des hiérarchies. Si cette approximation est un ensemble convexe (dans le cas d'une feuille de volume englobant contenant une collection de primitives formant une enveloppe convexe), alors les algorithmes présentés précédemment pour le calcul de distance peuvent être utilisés afin de tester l'intersection. Si les entités sont des simples triangles, ce qui est le cas pour des maillages triangulaires, l'algorithme présenté plus tôt dans la section 2.2.2 peut être utilisé pour tester l'intersection. A la fin de cette étape, la liste des paires de primitives s'intersectant est passée à l'étape de traitement de collision.

3.3 Mise à jour de hiérarchies

Dans la partie précédente, nous avons vu des techniques de détection de collision. Ces algorithmes sont le résultat de recherches sur la simulation de corps rigides pour les objets convexes ou concaves. Ils s'appuient lourdement sur des structures de données telles que la topologie du voisinage précédemment mentionnée ou la représentation hiérarchique de volumes englobants. Mais la déformation continue des objets complexes reste une limite dans beaucoup d'applications de simulation dynamique. Ceci implique

que les données issues du pré-traitement doivent être recalculées à la volée, d'où un ralentissement notable de la simulation empêchant de satisfaire les contraintes sur les temps d'interaction.

Peu de travaux ont été reportés afin de remédier à ce problème. Les plus récentes propositions sont celles de [Van der Bergen, 1997] et [Larsson and Moller, 2001]. Dans ces articles, une hiérarchie de AABB est utilisée pour représenter l'objet déformable. Or, nous avons montré que les boîtes OBB offraient de meilleurs résultats pour les corps rigides. Cependant les coûts de mises à jour de ces boîtes sont trois fois plus élevés que ceux des boîtes AABB. Donc, les AABB se révèlent être un choix judicieux dans le cas de mises à jour de hiérarchies. Les principales techniques utilisées pour cette mise à jour sont :

- **La reconstruction** : L'arbre est complètement reconstruit des parents vers les fils (haut-bas) ou des enfants vers les parents (bas-haut).
- **Le réajustement** : Seuls les noeuds associés aux volumes englobants affectés par la déformation sont modifiés et, on propage ces changements de noeud en noeud jusqu'à la racine ou bien jusqu'aux feuilles selon le type de construction choisi.

On peut distinguer deux cas principaux de changements sur des objets virtuels, les **déformations** et les **changements de topologie**.

3.3.1 Déformation

Van der Bergen a proposé dans [Van der Bergen, 1997] une méthode rapide de mise à jour des AABB utilisant un technique de réajustement. Cette méthode utilise une approche haut-bas pour construire la hiérarchie. Le test de superposition utilisé est le test de l'axe séparateur (SAT) [Gottschalk et al., 1996]. Afin de discuter du choix d'une hiérarchie AABB, rappelons que le nombre de tests de superposition des boîtes nécessaires dans une descente de hiérarchies en AABB est supérieur à celui des OBB, une hiérarchie OBB est cependant plus compacte sur l'objet que celle d'AABB. De plus, un test de superposition entre deux boîtes est plus rapide pour les AABB que pour les OBB. En effet, deux OBB ont normalement une orientation relative différente, les opérations matricielles ainsi que le calcul de norme de cette orientation sont répétés pour chaque paire testée. Le justification majeure du choix des AABB reste le fait que l'on puisse les mettre à jour trois fois plus rapidement que les OBB.

Le réajustement des arbres AABB peut dans certains cas poser des problèmes. En effet, suite aux changements de positions relatives des primitives après déformations, les boîtes d'un arbre réajusté peuvent avoir plus de superpositions possibles que celles d'un arbre reconstruit. Un exemple de reconstruction et de réajustement est montré dans la figure 3.5.

L'algorithme de réajustement peut être décrit de la façon suivante :

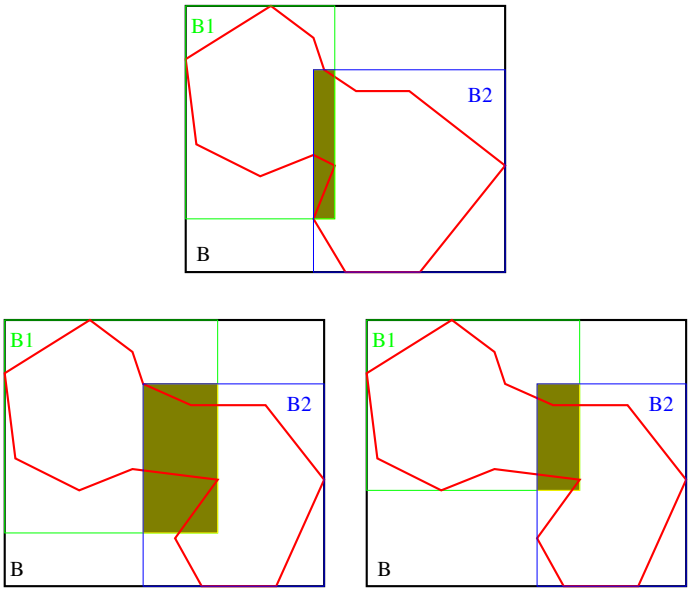

FIG. 3.5 – *Comparaison de la reconstruction et du réajustement d'un arbre de volume englobant. Haut : Modèle original. Bas (gauche) : Modèle réajusté après déformation. Bas (droite) : Modèle reconstruit après déformation. La région sombre indique les zones de superposition. Cité de [Van der Bergen, 1997].*

- Les boîtes englobantes des primitives soumises à la déformation sont recalculées.
- On reconstruit niveau par niveau les noeuds dont au moins un des fils a été reconstruit.

L'idée proposée dans [Van der Bergen, 1997] pour remonter dans la hiérarchie est d'utiliser un tableau de noeuds dans lequel on s'assure que l'indice d'un noeud est toujours plus grand que celui de son père. Ainsi le réajustement se fait en parcourant ce tableau dans l'ordre inverse. Une comparaison de performances montre que des arbres AABB sont réajustés et reconstruits en respectivement moins de 5% et 33% du temps nécessaire à la reconstruction d'un arbre OBB.

Dans [Larsson and Moller, 2001], la mise à jour d'un arbre AABB est légèrement différente. Le principe utilisé est similaire à celui de [Van der Bergen, 1997] mais au lieu d'utiliser une mise à jour des feuilles vers la racine, un niveau intermédiaire de la hiérarchie est choisi comme point de départ de la mise à jour. Ainsi, une combinaison d'un processus bas-haut *incrémental* et d'un processus haut-bas *sélectif* est utilisée par mettre à jour les volumes englobants qui en ont besoin. Cette méthode est dite *mise à jour hybride*.

Considérons deux cas de tests de collision, dans le premier, on ne doit vérifier que peu de niveaux de profondeur de l'arbre. L'approche haut-bas donne alors de très bons résultats pour la mise à jour des arbres AABB. Dans le second cas, un nombre important de niveaux de profondeur doivent être explorés, ici, l'approche bas-haut sera plus efficace. Ainsi, une combinaison des deux méthodes est implémentée afin d'exploiter leurs avantages.

Cet algorithme peut être résumé de la façon suivante :

- Pour un arbre de hauteur n, les $n/2$ premiers niveaux sont mis à jour en utilisant la stratégie bas-haut.
- Les niveaux restant sont examinés durant les tests de collision. Quand des noeuds non mis à jour sont atteints, ils peuvent être mis à jour par la méthode haut-bas ou un nombre fixé de niveaux des sous-arbres fils peuvent être mis à jour par une stratégie bas-haut.

Un exemple est montré dans la figure 3.6, où les niveaux (1 à 3) sont mis à jour par la stratégie bas-haut et les niveaux restant (4 à 5) sont mis à jour à la volée. Dans cet exemple, il y a 31 noeuds, mais seulement 11 d'entre eux sont mis à jour (les noeuds marqués en vert).

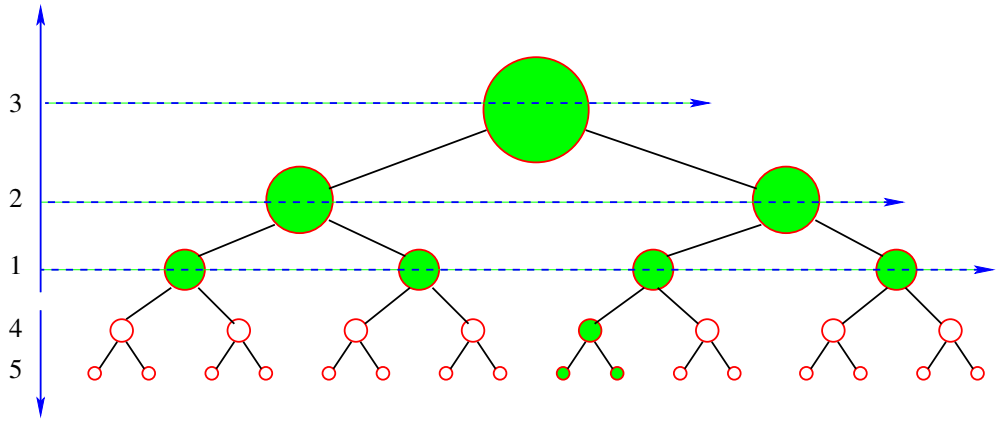

FIG. 3.6 – *Exemple de méthode de mise à jour hybride de l'arbre, mêlant les stratégies bas-haut et haut-bas. Cité de [Larsson and Moller, 2001].*

Des travaux venant des domaines de l'haptique et du calcul de distance se sont orientés vers les objets déformables ([Hubbard, 1996], [Quilan, 1994], [Davanne et al., 2002]), ils utilisent des sphères englobantes. Parmi ces publications, seul [Davanne et al., 2002] traite la mise à jour de hiérarchie. Des grilles de voxels et des sphères (voir la figure 3.7) sont utilisées, mais l'auteur déclare que la mise à jour de la hiérarchie après la déformation est beaucoup trop coûteuse.

3.3.2 Changement de topologie

Aucun travail n'a été reporté sur la mise à jour de hiérarchies due à des modifications de topologie. Lorsque l'on met à jour une hiérarchie due à des déformations on peut conserver la structure de l'arbre mais en cas de changement de topologie, la structure de l'arbre est également modifiée (des noeuds peuvent être insérés et/ou supprimés). Nous sommes donc face à un problème très coûteux et très difficile.

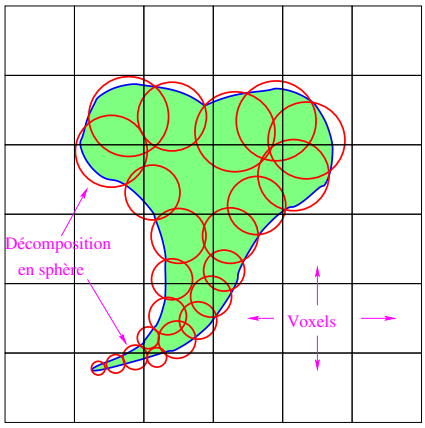

FIG. 3.7 – *Décomposition de volume par une collection de sphères. L'objet déformable est marqué en vert. Cité de [Davanne et al., 2002]*

3.4 Tests d'auto-collision

Les objets déformables posent un autre problème en termes d'interaction, l'auto-collision. On définit l'auto-collision par le fait qu'un objet est en collision avec lui-même, plus précisément, deux arêtes non adjacentes de l'objet sont en intersection. Lorsque l'on tente de détecter ce type d'interaction, les maillages polygonaux posent un problème lié à l'**adjacence** des éléments caractéristiques. Chaque polygone élémentaire de la surface du maillage est adjacent à ses polygones voisins. Ils sont par nature *déjà* en collision (plus précisément, en contact). Donc, ces adjacences qui sont dans ce contexte de *fausses* collisions, ne devraient pas perturber ou ralentir les algorithmes d'interaction. Les contributions majeures dans ce type d'algorithmes viennent des travaux sur la simulation de vêtements et de corps fortement déformables. On peut cependant voir une application des tests d'auto-collision dans le domaine de la simulation chirurgicale [Laks Raghupathi and Cani, 2003].

3.4.1 Volino et Thalmann (VT)

Cet algorithme [Volino and Thalmann, 1994], [Volino and Thalmann, 1995] détecte l'auto-collision d'une surface ouverte déformable. Il utilise la courbure de la surface. En effet, une surface déformable est en auto-collision si elle satisfait les deux conditions suivantes :

- La **surface** doit être suffisamment courbée pour que deux portions puissent être en contact.
- Le **contour** de la surface doit lui-même être en auto-collision.

Une formulation plus formelle de ces critères géométriques peut être donnée ainsi :

Définition 2. *Soient $\mathcal{S}$ une surface ouverte continue orientée et bornée par un contour $\mathcal{C}$, ses normales*

$\boldsymbol{N}$ dans l'espace euclidien . S'il existe un vecteur $\boldsymbol{V}$ tel que $\boldsymbol{N} \cdot \boldsymbol{V} > 0$ pour tout point de $\mathcal{S}$ et que la projection de $\mathcal{C}$ sur un plan orthogonal à $\boldsymbol{V}$ suivant la direction $\boldsymbol{V}$ ne soit pas en intersection, alors il n'y a pas d'auto-collision dans $\mathcal{S}$.

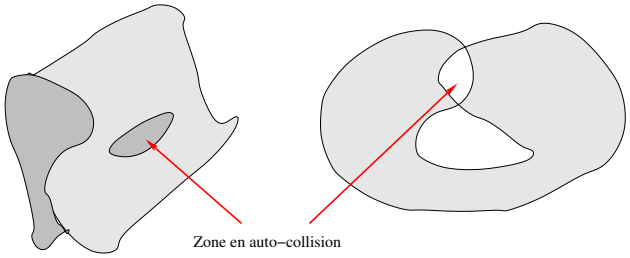

FIG. 3.8 – *Conditions pour l'auto-collision : La surface doit être suffisamment courbée (Gauche) et le contour de la surface doit être en intersection (Droite).*

Cette définition amène aux deux corollaires suivant :

Corollaire 1. Auto-collision dans une surface *: S'il existe un vecteur ayant un produit scalaire positif avec les normales de chaque triangle et la projection du contour suivant ce vecteur ne produit aucune intersection, alors il n'y a pas d'auto-collision dans cette surface.*

Corollaire 2. Collision entre deux surfaces *: Si dans deux surfaces connectées par au moins un sommet, il existe un vecteur ayant un produit scalaire positif avec les normales de chaque triangle des deux surfaces et les projections de leurs contours suivant ce vecteur ne s'intersectent pas, alors il n'y a pas de collision entre ces deux surfaces.*

Volino et Thalmann ont proposé un algorithme récursif pour trouver le vecteur $\boldsymbol{V}$. On construit une hiérarchie de sous-surfaces adjacentes sur la surface ouverte. On commence avec un ensemble fini initial de vecteurs normalisés représentant les directions de l'espace. Pour chaque test récursif, on élimine les vecteurs n'ayant pas un produit scalaire positif avec la normale d'une sous-surface. On répète ceci sur l'ensemble de la hiérarchie. La figure 3.9 illustre ce processus.

3.4.2 Méthode de subdivision des éléments (SEM)

Cet algorithme [Grinspun and Schröder, 2001], [Cirak et al., 2000] apparaît dans des articles sur la simulation de corps fortement déformables. Il s'agit d'une extension de l'algorithme VT où les données entrantes ne sont pas un maillage polygonal mais une surface non discrétisée de topologie arbitraire (par exemple des surfaces de subdivision). Nous allons décrire cette méthode car les idées présentées peuvent être étendues aux maillages polygonaux.

De la même façon que dans VT, une hiérarchie de patch décrite dans la figure 3.10 est utilisée dans

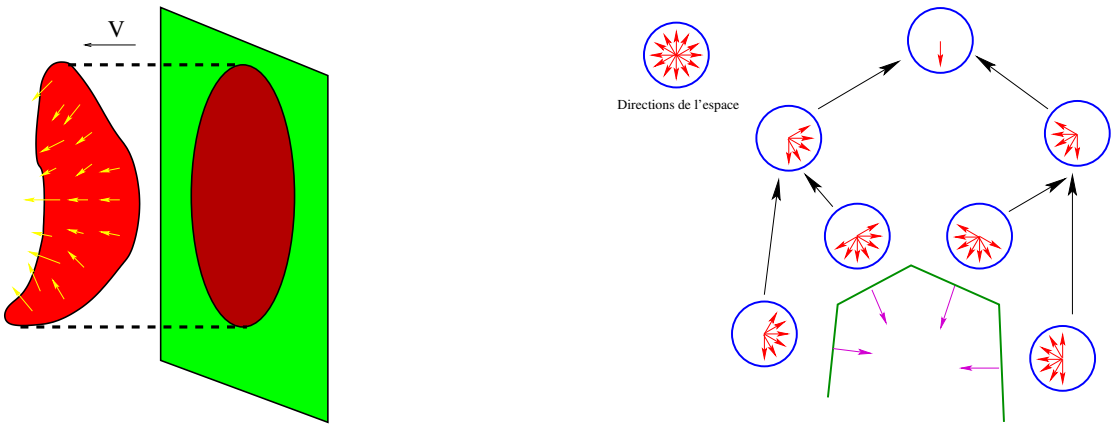

FIG. 3.9 – *Gauche : Une surface est sans auto-collision s'il existe un vecteur* $\boldsymbol{V}$ *ayant un produit scalaire positif avec toutes les normales* $\boldsymbol{N}$ *et la projection de son contour suivant* $\boldsymbol{V}$ *sera elle-même sans auto-collision. Droite : On peut trouver ce vecteur* $\boldsymbol{V}$ *récursivement en propageant les informations des normales de chaque facette de la surface.*

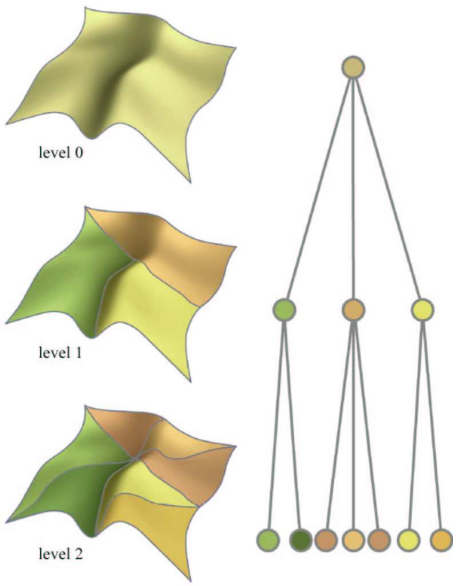

FIG. 3.10 – *La hiérarchie de patch utilisée dans la méthode SEM est telle que les feuilles sont des patch triangulaires. L'union de tous les patch,* i.e. *le plus haut niveau de la hiérarchie, représente la surface de l'objet. Cité de [Grinspun and Schröder, 2001].*

cet algorithme. Une idée intéressante dans ce travail est l'utilisation du **Gauss Map** d'un patch. Le Gauss Map est une fonction $(u, v) \rightarrow (f_u \times f_v)/|f_u \times f_v|$ qui à chaque point d'une surface 2-manifold orientable associe son vecteur normal unitaire et correspond donc à un point de la sphère unitaire. Ceci est illustré dans la figure 3.11.

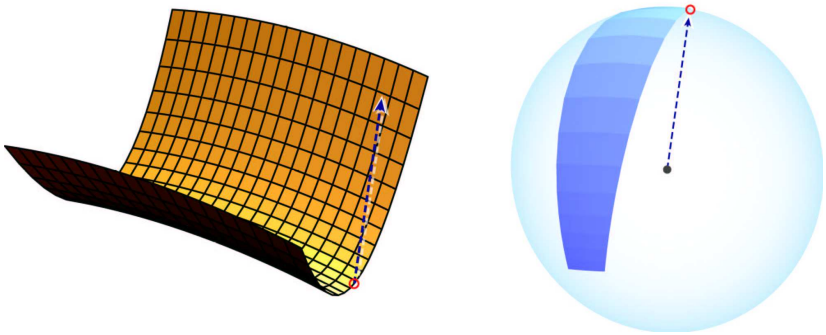

FIG. 3.11 – *Le* **Gauss Map** *associe un point de la surface à son vecteur normal unitaire. Cité de [Grinspun and Schröder, 2001]*

Deux conditions doivent être remplies pour éliminer l'auto-collision (voir figure 3.12). Premièrement, il doit exister un plan séparateur dans l'espace de la boule unitaire entre l'image du Gauss Map et le centre de la boule. Deuxièmement, la projection de la frontière du patch sur ce plan doit pas s'auto-intersecter. Des volumes englobants sont utilisés par éliminer l'interférence entre patch non adjacent, alors que le Gauss Map est utilisé pour éliminer le cas d'auto-collision sur un patch.

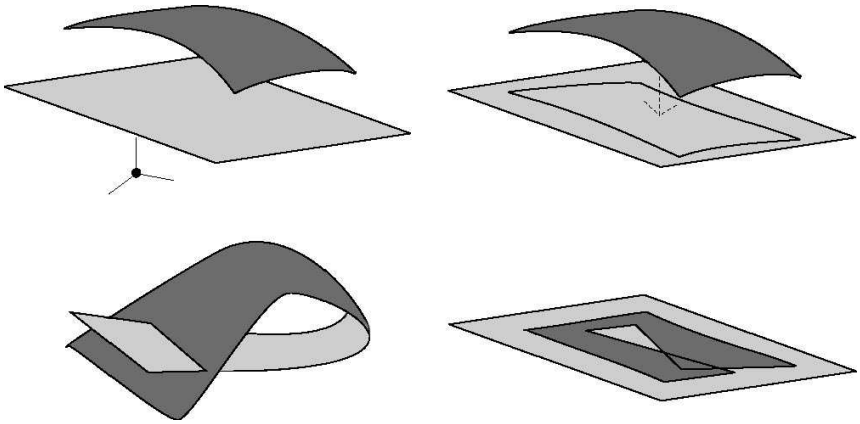

FIG. 3.12 – *Haut-Gauche : Si un plan séparateur est trouvé entre l'image du Gauss Map et l'origine, alors il n'y a pas d'auto-collision. Une absence de plan séparateur nécessite une descente dans la hiérarchie pour peut-être avoir une auto-collision. Haut-Droite : Un test supplémentaire pour l'auto-collision est la projection de la frontière de l'image du Gauss Map sur le plan séparateur, mais ce test identifie l'auto-collision due à un étirage et un cisaillement plan, Droite-Bas. Cité de [Grinspun and Schröder, 2001].*

Contrairement à l'algorithme VT qui peut probablement être exécuté dans des problèmes de taille

moyenne. La méthode SEM souffre de sa complexité. Le calcul de la frontière du Gauss Map sur des surfaces déformables est bien trop coûteux. Même pour des problèmes de petite taille, la détection d'auto-collision ce fait entre $100ms$ et $30s$, cela n'inclue pas les autres étapes de la simulation telles que le rendu ou le traitement de la collision.

Chapitre 4

Librairie de détection de collision entre polyèdres déformables

4.1 Définition du problème

Les algorithmes étudiés dans les parties précédentes s'attachent à résoudre efficacement un problème précis lié à une application donnée.

Ainsi, beaucoup de méthodes requièrent des propriétés géométriques (concavité, convexité) ou physiques (rigide, déformable) sur les objets. Certaines ont besoin de pré-calculs pour être efficaces et s'appuient donc sur des structures de données intermédiaires comme par exemple un diagramme de Voronoi, une hiérarchie de boîtes (AABB ou OBB) ou de sphères englobantes. D'autres méthodes nécessitent une discrétisation de l'espace en grilles de voxels, en arbre BSP ou en octree.

Dans de nombreuses applications, un même objet peut être soumis à plusieurs types de tests d'interférence. Par exemple, considérons un polyèdre représentant un personnage évoluant dans un environnement virtuel. Si l'on s'intéresse à son déplacement dans le décor statique, alors nous nous ramenons à un problème de planification de mouvement où seul des tests d'interférence statique comme des calculs de distance seront effectués. Mais, des éléments dynamiques peuvent apparaître dans l'environnement virtuel, notre personnage sera alors soumis à des tests d'intersection avec ces éléments. En cas de collision, on peut vouloir connaître toutes les informations de contact pour être à même de répondre à cette collision.

Ainsi, on a besoin de mettre en place un modèle générique de collision avec des structures de données intermédiaires adaptées aux différents tests d'interférence ainsi qu'aux propriétés géométriques et physiques des polyèdres.

4.2 Solution proposée

4.2.1 Structure de données

Les polyèdres traités peuvent être convexes ou concaves, mais également rigides ou déformables.

Dans le chapitre précédent, nous avons vu que les méthodes effectuant une discrétisation de l'espace se révèlent peu adaptées aux modèles très dynamiques ainsi qu'aux objets déformables. Les opérations de mises à jour de structures telles que des octrees, des grilles de voxels ou bien d'arbres BSP sont extrêmement coûteuses.

Ainsi, le choix d'une hiérarchie de volumes englobants semble naturel. Il reste à choisir le type de volume. Les sphères, par leur manque de compacité, nécessitent trop de tests pendant la descente dans l'arbre.

Concernant les boîtes, nous avons montré que les boîtes AABB sont un peu moins compactes que les OBB, mais leur mise à jour est trois fois plus rapide. De plus, d'après [Van der Bergen, 1997] et [Larsson and Moller, 2001], le choix des boîtes AABB se révèlent tout à fait judicieux dans le cas de mises à jour de hiérarchies pour des objets déformables.

Nous choisissons de représenter nos boîtes AABB par des parallélépipèdes décrits par un centre et un vecteur représentant une diagonale (on choisit le vecteur diagonal dont toutes les coordonnées sont positives).

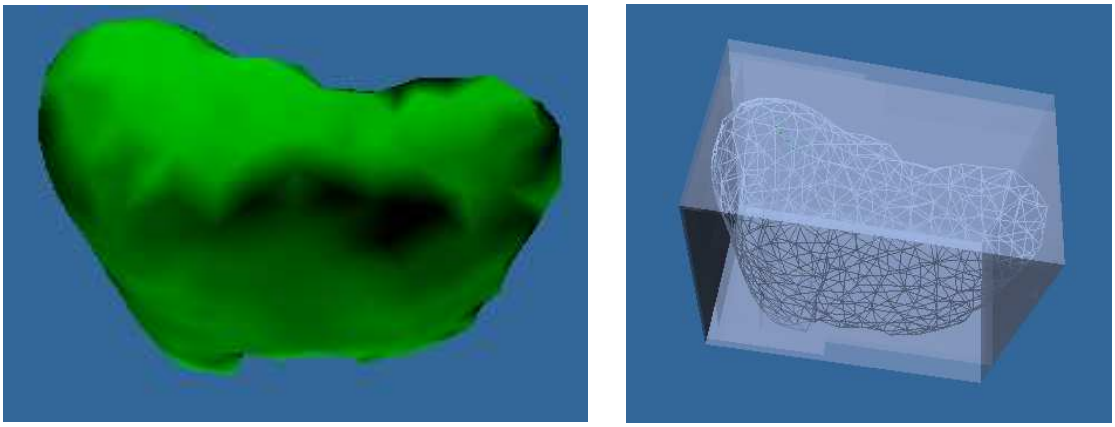

FIG. 4.1 – *Exemple de hiérarchie de boîtes AABB englobantes sur un maillage de 800 facettes et 402 sommets.*

4.2.2 Algorithmes utilisés

4.2.2.1 Calcul de distance entre polyèdres convexes

Afin de calculer la distance entre deux polyèdres convexes, nous avons choisi d'utiliser l'algorithme décrit dans [Sundaraj et al., 2000]. Cette implémentation repose sur l'algorithme GJK en apportant des améliorations afin d'éliminer le problème lié à la convergence dans les cas dégénérés ainsi que l'utilisation de la cohérence de frame améliorant la vitesse de la méthode.

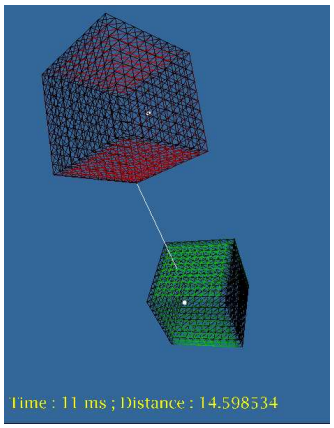

FIG. 4.2 – *Exemple de calcul de distance entre deux polyèdres convexes. Les maillages utilisés possèdent chacun 1200 facettes et 602 sommets. Le temps d'exécution de l'algorithme, implémenté dans ce cas en Java3D, est de* $11ms$.

4.2.2.2 Calcul de distance entre polyèdres non convexes

Définition du problème Les algorithmes efficaces de calcul de distance entre deux objets se bornent bien souvent aux objets convexes. Lumelsky [Lumelsky, 1985] a décrit un algorithme efficace pour les paires de segments. Gilbert [Gilbert et al., 1988], Bobrow [Bobrow, 1989], et Lin et Canny [Lin and Canny, 1991] ont présenté des algorithmes qui calculent la distance entre deux polyèdres convexes. Chacun de ces trois algorithmes itératifs trouve des paires de points, un sur chaque objet, tels que la distance entre ces points converge vers le minimum.

Ces algorithmes exploitent les propriétés convexes des objets et il semble difficile de les étendre directement au cas des objets non convexes.

Algorithme proposé L'algorithme utilisé est une adaptation de l'algorithme décrit dans [Quilan, 1994] à la structure de données décrite précédemment. De plus, cette méthode a été améliorée puisque les conditions de descente dans la hiérarchie sont différentes.

L'idée de l'algorithme exposé est d'utiliser les méthodes de calcul existant entre les polyèdres convexes. Pour ce faire, nous utilisons une description du polyèdre concave par un ensemble de polyèdres convexes. Nous construisons donc ici, une représentation de l'objet grâce à une hiérarchie de sphères englobantes.

Il convient de rappeler que dans la structure hiérarchique de boîtes (AABB) englobantes expliquée précédemment, les boîtes étaient décrites par un centre et un vecteur représentant une diagonale de l'objet. Nous pouvons donc parfaitement mettre à profit nos résultats et notre implémentation du problème de détection de collision dans ce cas. En effet, il nous suffit de décrire nos sphères par un centre et un rayon.

L'intérêt d'une hiérarchie de sphères et non de boîtes est de minimiser le nombre d'opérations et donc le temps de calcul de la distance entre deux éléments de la structure hiérarchique englobant chacun des polyèdres concaves.

Afin de calculer la distance d, nous initialisons d'abord d à l'infini.

Nous utilisons une routine de recherche qui nous permet de trouver quels objets sont au moins distants de d.

Supposons que la routine de recherche nous retourne deux faces distantes de d, où d est inférieur à la distance minimale connue.

Si d est nul alors nous savons que nos faces, donc nos objets, sont en intersection.

Sinon la distance minimale devient d, et la recherche continue.

La clé de l'algorithme est de montrer que la distance entre les objets est d sans explorer toutes les faces de chacun des objets. Comme chacune des faces des polyèdres est contenue dans une sphère englobante, nous devons seulement examiner les feuilles de l'arbre de sphères au moins distantes de d.

La routine de recherche trouve les paires de feuilles au moins distantes de d. Elle examine récursivement les noeuds de l'arbre de sphères depuis la racine. Si la distance entre ces noeuds est supérieure à d, alors nous n'explorons bien sûr pas leurs fils. Cette routine est décrite dans l'algorithme 1.

Le critère de descente dans l'arbre est prédominant pour diminuer le coût de l'algorithme. Nous avons choisi de descendre d'abord vers le fils le plus proche de la sphère représentant la racine de l'autre objet. Ce critère est une heuristique nous permettant d'avoir un algorithme plus efficace que celui de [Quilan, 1994].

Une fois que nous connaissons tous les couples de feuilles réalisant la distance minimale entre les objets, nous déterminons la distance minimale effective entre les deux polyèdres concaves, en appliquant

une version améliorée de l'algorithme GJK [Sundaraj et al., 2000] sur ces paires de faces incluses dans les paires de feuilles.

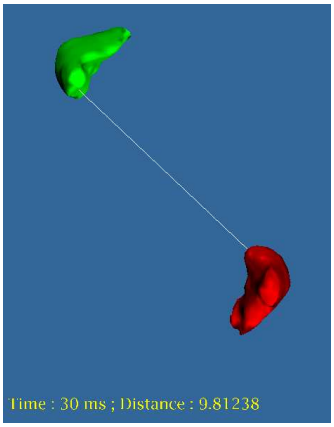

FIG. 4.3 – *Exemple de calcul de distance entre deux polyèdres non convexes. Les maillages utilisés possèdent chacun 800 facettes et 402 sommets. Le temps d'exécution de l'algorithme, implémenté dans ce cas en Java3D, est de* $30ms$.

4.2.2.3 Détection de collision entre polyèdres déformables

Nous avons choisi d'utiliser une hiérarchie de boîtes AABB englobantes. Sa construction se fait par une méthode bas-haut, dans laquelle, on regroupe dans une paire deux boîtes minimisant le volume de leur boîte mère. En effet, étant donné une boîte, nous recherchons parmi toutes les boîtes qui ne sont pas déjà regroupées dans une paire, celle dont le volume de la potentielle boîte mère résultant est minimal. Il en résulte une hiérarchie compacte autour de la surface de l'objet. Cette compacité n'est cependant pas idéale compte tenu de notre critère de construction.

A chaque pas de temps, on reconstruit les boîtes de toutes les feuilles de l'arbre dont au moins un des points de la facette associée subi une déformation. Puis, on propage ces changements vers les boîtes ascendantes en effectuant une remontée niveau par niveau dans la hiérarchie, on met à jour toute boîte dont au moins l'une des feuilles a été mise à jour. Notons qu'une boîte ne peut être reconstruite que si ses deux boîtes filles sont traitées. La remontée se fait en utilisant l'algorithme 2.

Une fois que les mises à jour éventuelles dans la hiérarchie sont effectuées pour les deux objets, le processus de détection de collision commence.

Lors de la descente dans la hiérarchie, on utilise le test de l'axe séparateur décrit en 2.2.3 afin de déterminer les boîtes en intersection. Si deux boîtes sont en collision, on vérifie si leurs boîtes filles respectives sont deux à deux en collision.

Si on est arrivé jusqu'aux feuilles de l'arbre, on teste la possible collision entre les faces du polyèdres

```
{bsA et bsB : noeuds de l'arbre de sphères englobantes}
{bsAMinVector et bsBMinVector : listes de feuilles minimisant la distance}
float d = distanceSphereSphere(bsA, bsB)
if d ¿ distanceMin then
   return
end if
if bsA.type == TREE then
   if bsB.type == TREE then
      {bsA et bsB sont des arbres}
      if bsA == bsARoot then
         if bsB == bsBRoot then
            {bsA et bsB sont des racines : on explore d'abord le plus gros volume}
            if bsA.volume ¡ bsB.volume then
               exploreB(bsA, bsB)
            else
               exploreA(bsA, bsB)
            end if
         else
            {bsA est une racine et bsB est un arbre : on explore B}
            exploreB(bsA, bsB)
         end if
      else
         {bsA est un arbre et bsB est une racine : on explore A}
         exploreA(bsA, bsB)
      end if
   else
      {bsA est un arbre et bsB est une feuille}
      exploreA(bsA, bsB)
   end if
else
   if bsB.type == TREE then
      {bsA est une feuille et bsB est un arbre}
      exploreB(bsA, bsB)
   else
      {bsA et bsB sont des feuilles}
      distanceMin = d
      bsAMinVector.addElement(bsA)
      bsBMinVector.addElement(bsB)
   end if
end if
```

Algorithme 1 – *Recherche de feuilles au moins distantes de d*

```
{filsDeformés : liste des noeuds d'un niveau devant être mis à jour}
{pèresDeformés : liste des noeuds du niveau suivant qui devront être mis à jour}
pèresDeformés.vide()
while ! filsDeformés.estVide() do
   for all noeud dans filsDéformé do
      if noeud.nécessiteMiseAJour then
         mettreAJourBoite(noeud)
         noeud.nécessiteMiseAJour = faux
         if noeud.pere != NUL then
            noeud.pere.nécessiteMiseAJour = true
            pèresDeformés.ajouteEnFin(noeud.pere)
         end if
      end if
   end for
   filsDéformés = pèresDeformés
   pèresDeformés.vide()
end while
```

Algorithme 2 – *Remontée niveau par niveau dans la hiérarchie de boîtes englobantes : Routine de mise à jour de l'arbre*

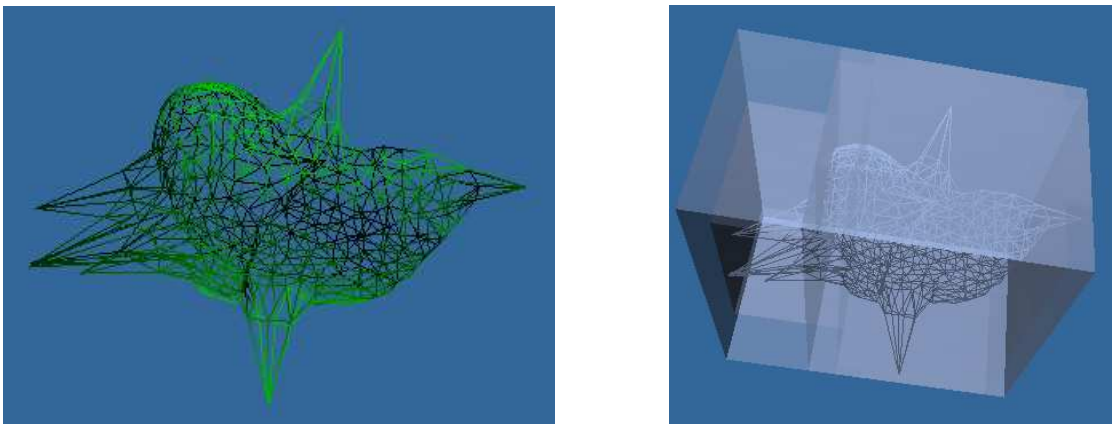

FIG. 4.4 – *Exemple mise à jour de la hiérarchie de boîtes AABB englobantes. Le maillage utilisé possède 800 facettes et 402 sommets.*

qu'elles englobent. On utilise pour cela l'algorithme [Moller, 1997] décrit en 2.2.2.

A l'issue de ce procédé, on obtient dans le cas d'intersection entre les objets, le contour de collision de chaque polyèdre, *i.e.* la liste des paires de facettes deux à deux en collision.

4.2.2.4 Détection du volume d'interpénétration

Il peut résulter de la collision des deux polyèdres, une interpénétration fictive des deux objets. Ainsi, on doit déterminer le volume de contact de chacun des objets afin de pouvoir calculer la force résultant de la collision, et en déduire les déformations locales des objets.

Nous construisons deux contours qui sont à l'extérieur et à l'intérieur du contour de collision. Enfin, nous cherchons tous les éléments en contact. Nous utilisons pour cela l'algorithme [Sundaraj and Laugier, 2000]. Il résulte de cette passe les deux surfaces délimitées par le contour de collision. L'union de ces deux surfaces peut alors être traitée pour obtenir les informations nécessaires au traitement de la collision.

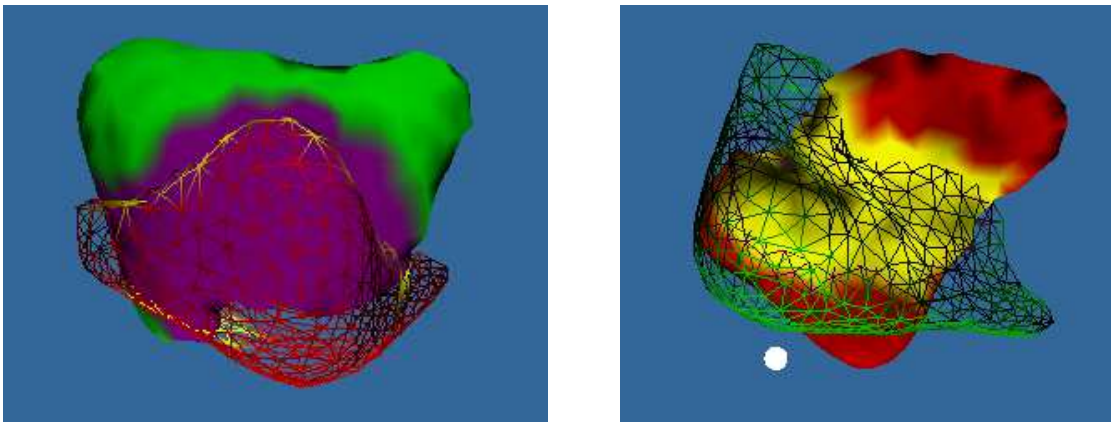

FIG. 4.5 – *Exemple de détection du volume d'interpénétration. Les maillages utilisés possèdent chacun 800 facettes et 402 sommets. Le temps d'exécution de l'algorithme, implémenté dans ce cas en Java3D, est de* $50ms$.

4.3 Résultats

Différentes librairies de détection de collision existent, elles sont pour la plupart destinées à des applications particulières et nécessitent souvent des propriétés géométriques ou physiques sur les objets spécifiques. De plus, leurs caractéristiques et les opérations fournies diffèrent également.

4.3.1 Librairies existantes

DEEP http://www.cs.unc.edu/~geom/DEEP

Cette librairie permet de calculer la distance d'interpénétration dans le cas de collision entre objets convexes et rigides. Elle s'appuie sur SWIFT++ pour effectuer la détection de collision.

Enhanced GJK http://users.comlab.ox.ac.uk/stephen.cameron/distances.html

Enhanced GJK est une librairie de calcul de distance entre deux objets convexes. Cet algorithme fut initialement conçu pou résoudre des problèmes de planification de déplacements et de simulation. Comme nous l'avons vu dans la partie 2.1.2, cette méthode est basée sur l'algorithme GJK.

Enhanced GJK a été comparée à I-Collide et V-Clip. Ces algorithmes semblent avoir la même vitesse.

H-Collide http://www.cs.unc.edu/~geom/H_COLLIDE

H-Collide est un framework pour la détection de collision dans le cas d'interactions haptiques. Cette librairie utilise plusieurs algorithmes. Elle est spécialisée dans le calcul de contact entre une sonde et un objet virtuel.

Elle utilise une grille uniforme de voxels pour décomposer l'espace ainsi que des arbres OBB. Elle n'est distribuée que dans la librairie haptique GHOST.

I-Collide http://www.cs.unc.edu/~geom/I_COLLIDE

La librairie I-Collide permet de détecter les collisions exactes et de calculer les distances pour des modèles composés de polyèdres convexes. Elle ne supporte pas les objets déformables.

Cette librairie souffre d'un manque de vitesse et de robustesse. Les auteurs d'I-Collide conseillent l'utilisation de SWIFT afin de régler ces problèmes de performances et de fiabilité.

PIVOT http://www.cs.unc.edu/~geom/PIVOT

PIVOT est une librairie permettant de répondre à des requêtes de proximité en 2D. Elle ne permet pas de détecter de collision mais il est possible de calculer des distances, des intersections, des distances de séparation avec des objets rigides ou déformables en 2D.

PQP http://www.cs.unc.edu/~geom/SSV

Cette librairie permet de répondre à des requêtes de proximité sur des polyèdres triangulaires. Elle permet de détecter des collisions, de calculer des distances et de vérifier un facteur de tolérances.

Il s'agit d'une adaptation de RAPID, dans laquelle les arbres OBB ont été remplacés par des arbres RSS [1] (Rectangle Swept Sphere) pour répondre aux requêtes de distance et de tolérance.

Comme dans RAPID, les polyèdres déformables ne sont pas traités.

Q-Collide http://www.csis.hku.hk/GraphicsGroup/collision.htm

Cette librairie permet la détection de collision entre des polytopes convexes. Elle permet de trouver un plan séparateur entre les deux objets s'ils ne sont pas en contact et retourne les deux plus proches points entre les objets dans le cas contraire.

Il s'agit d'une version améliorée de la librairie I-Collide.

Elle est limitée aux polyèdres rigides. Elle ne permet pas d'obtenir le volume ou la profondeur d'interpénétration.

QuickCD http://www.ams.sunysb.edu/~jklosow/quickcd

Cette librairie permet la détection de collision entre des soupes de polygones.

Elle utilise un hiérarchie de k-dops [2] englobants afin d'approximer les objets.

Les polyèdres déformables ne sont pas traités.

RAPID http://www.cs.unc.edu/~geom/OBB/OBBT.html

Cette librairie permet la détection d'interférence entre des soupes de triangles.

Elle ne permet pas de calculer les distances, mais elle retourne les paires de triangles en contact.

De nombreuses optimisations sont possibles notamment en réduisant les coûts en mémoire (dans l'implémentation des arbres OBB utilisés) et la vitesse de l'algorithme. Pierre Terdiman a proposé plusieurs modifications.

Les polyèdres déformables ne sont pas traités.

SOLID http://www.win.tue.nl/~gino/solid

[1] Un RSS est un volume recouvert par une sphère dont le centre balaye la surface d'un rectangle 3D.
[2] Les k-dops sont des polytopes convexes dont les facettes ont des normales appartenant à un ensemble discret de k vecteurs

Il s'agit d'une librairie de détection de collision d'objets rigides et déformables, pouvant être représentés par des formes primitives (cubes, sphères, cônes, cylindres) ou des polytopes (segments de lignes, polygones convexes, polyèdres convexes).

Une version améliorée de l'algorithme GJK est utilisée et des arbres AABB permettent d'approximer les objets.

Dans le cas de modèles déformables, si un objet déformable est convexe ou formé de polygones convexes, l'utilisateur doit garantir que les déformations subies par les sommets ne modifient pas la convexité ni la topologie de l'objet.

En cas de collision, la librairie ne retourne pas la volume d'interpénétration ni la profondeur d'interpénétration.

SWIFT http://www.cs.unc.edu/~geom/SWIFT

Il s'agit d'une librairie de détection de collision, de calcul de distance et de localisation de contact. Elle est limitée au objets convexes.

Les auteurs de SWIFT conseillent l'utilisation de SWIFT++ dans le cas d'objets concaves.

En cas de collision, cette librairie ne permet pas d'obtenir le volume d'interpénétration. Les objets déformables ne sont pas traités.

SWIFT a été comparée à I-Collide, V-Clip et Enhanced GJK. Elle est supposée être plus rapide et plus robuste. Cependant, il arrive que SWIFT entre dans des boucles infinies.

SWIFT++ http://www.cs.unc.edu/~geom/SWIFT++

SWIFT++ est une librairie de détection de collision, de calcul de distance et de localisation de contact dans le cas d'objets polyédriques rigides. Il s'agit d'une extension de SWIFT aux cas d'objets non convexes.

Tout comme dans SWIFT, en cas de collision, cette librairie ne permet pas d'obtenir le volume d'interpénétration. Les objets déformables ne sont pas traités.

SWIFT++ a été comparée à Rapid, PQP et QuickCD. Les résultats se montrent extrêmement variables selon la nature de la scène. De plus, il arrive que SWIFT++ entre dans des boucles infinies.

V-Clip http://www.merl.com/projects/vclip

Voronoi Clip est une librairie de détection de collision d'objets polyédriques. Elle fonctionne avec des polyèdres convexes ou concaves. Seuls les objets rigides sont gérés.

En cas de collision, elle renvoie les plus proches points des objets ainsi que la profondeur de pénétration.

L'algorithme utilisé a été décrit en 2.1.3. Les configurations dégénérées ne sont pas problématiques et l'implémentation est plus simple que celle de Lin-Canny. Les temps d'exécution de V-Clip sont sensiblement les mêmes que ceux de LC.

V-Clip se révèle être moins efficace que Enhanced GJK dans le cas d'objets à la géométrie complexe.

V-Collide `http://www.cs.unc.edu/~geom/V_COLLIDE`

Il s'agit d'une librairie de détection de collision pour les environnements virtuels vastes composés de soupes de polygones.

Elle permet de gérer des objets convexes et concaves.

Elle ne retourne la distance entre deux objets que s'ils sont en collision. Elle ne fournit pas le volume d'interpénétration ni même le contour de collision. En effet, les paires de triangles en collision ne sont pas fournies.

4.3.2 Comparaison qualitative

Seuls notre algorithme et celui de SOLID permettent de gérer la détection de collision entre deux polyèdres déformables convexes ou concaves.

De plus, nous détectons l'intersection entre les deux objets et nous calculons l'éventuel volume d'intersection. Contrairement à SOLID, le changement de convexité d'un objet à la suite d'une déformation n'est pas problématique.

De plus, au sein de la même librairie, nous pouvons répondre à un grand nombre de requêtes (calcul de distance entre objets convexes et/ou concaves, détection de collision entre objets rigides et/ou déformables et convexes et/ou concaves, calcul du contour de collision en cas de collision, calcul du volume d'intersection en cas de collision) sur des objets polyédriques qu'ils soient convexes ou concaves, rigides ou déformables.

4.3.3 Comparaison quantitative

Les vitesses des algorithmes V-Clip, Lin-Canny et Enhanced GJK ont été comparées dans [Mirtich, 1998]. L'efficacité des différentes hiérarchies de volumes englobants fut évaluée dans [Larsen et al., 2000]

Une comparaison des principales librairies de détection de collision (V-Clip, RAPID, SOLID, PQP et V-Collide) a été faite dans [Reggiani et al., 2002].

Le manque des résultats expérimentaux est du à la difficulté de concevoir des tests capables de quantifier précisément la robustesse et l'efficacité des algorithmes de détection de collision. Trop de facteurs affectent leurs performances : ils sont sensibles à la complexité de la représentation, à la position des objets et à la zone du déplacement. De plus, les tests doivent aussi tenir compte des problèmes d'implémentations, tels que le langage, le compilateur, les problèmes de représentations des flottants et le niveau d'optimisation.

Ainsi, nous allons nous concentrer sur une comparaison des différentes routines élémentaires utilisées dans nos algorithmes.

L'efficacité de l'algorithme de calcul de distance entre objets convexes a été démontrée et comparée aux autres algorithmes dans [Sundaraj et al., 2000].

Dans le cas de la détection de collision, la construction des boîtes AABB par le procédé bas-haut en utilisant un critère de minimisation du volume permet d'arriver à une hiérarchie très compacte autour de l'objet et ainsi on minimise les tests de contact des boîtes pendant la descente dans l'arbre. Le test d'intersection boîte/boîte utilisé (l'algorithme de l'axe séparateur) est à ce jour le plus rapide dans le cas de boîtes AABB. Le test de contact trianle/triangle choisi est l'algorithme de Moller [Moller, 1997], il est à ce jour lui aussi optimal. L'efficacité de la localisation du volume d'interpénétration a été démontrée dans [Sundaraj and Laugier, 2000] et se révèle très efficace.

De plus, le procédé de mise à jour de la hiérarchie de volumes englobants utilisé diffère de celui de Gino Van der Bergen utilisé dans SOLID. En effet, il parcourt tous les noeuds de l'arbre dans un ordre prédéterminé afin de mettre à jour les boîtes déformées (voir 3.3.1) tandis que nous ne parcourons que les boîtes devant être modifiées. Les autres noeuds de l'arbre ne sont pas étudiés tandis que dans SOLID l'intégralité de l'arbre est explorée. En cela, notre algorithme se révèle être plus rapide que celui de SOLID.

4.3.4 Évaluation expérimentale

Dans la partie suivante, nous allons décrire les choix et la méthodologie utilisés afin de quantifier les temps d'exécution de nos algorithmes.

4.3.4.1 Conditions expérimentales

Les algorithmes présentés précédemment (4.2.2) ont été implémentés dans une librairie liée dynamiquement avec un moteur de rendu 3D OpenGL. La plate-forme d'exécution est un processeur Pentium IV 1,7 Ghz avec 256 mB de mémoire. Le moteur de rendu et la librairie ont été compilés avec le compilateur C++ GNU avec l'option d'optimisation '-O3'.

4.3.4.2 Résultats expérimentaux

Le tableau 4.1 représente les temps d'exécution en microsecondes de la mise à jour de la hiérarchie de boîtes AABB de différents objets. Les polyèdres utilisés ont un nombre de faces, de sommets et des propriétés géométriques différentes. Afin de procéder à ce test, nous avons, à chaque pas de temps, choisi aléatoirement un nombre de points du maillage. Puis nous avons affecté à ces points des coordonnées choisies elles aussi aléatoirement. Enfin, nous avons lancé la mise à jour de la hiérarchie de boîtes AABB. Nous allons indiquer une moyenne des temps obtenus pour les différents maillages. La figure 4.6 représente les différents objets.

Les résultats du tableau 4.1 nous montre que le temps d'exécution augmente avec le nombre de sommets et de faces. Ceci vient du fait que les objets ayant un nombre plus important de primitives ont plus de niveaux dans la hiérarchie de volumes englobants utilisée. Ainsi, la remontée dans la hiérarchie devient plus complexe.

Maillage	Nombre de sommets	Nombre de faces	Temps d'exécution (μs)
Théière	530	1024	7867
Foie (simplifié)	402	800	9917
Foie (original)	1691	3366	27858
Fémur	1998	3992	28762

TAB. 4.1 – *Temps d'exécution de la mise à jour de hiérarchie de boîtes AABB pour différents maillages.*

Le tableau 4.2 représente les temps d'exécution moyen en microsecondes du calcul de distances entre deux polyèdres concaves. Les objets utilisés ont des propriétés géométriques différentes. Les maillages employés pour ce test sont les mêmes que ceux décrits dans la figure 4.6.

Notre algorithme de calcul de distance entre des objets concaves tire des informations de la hiérarchie

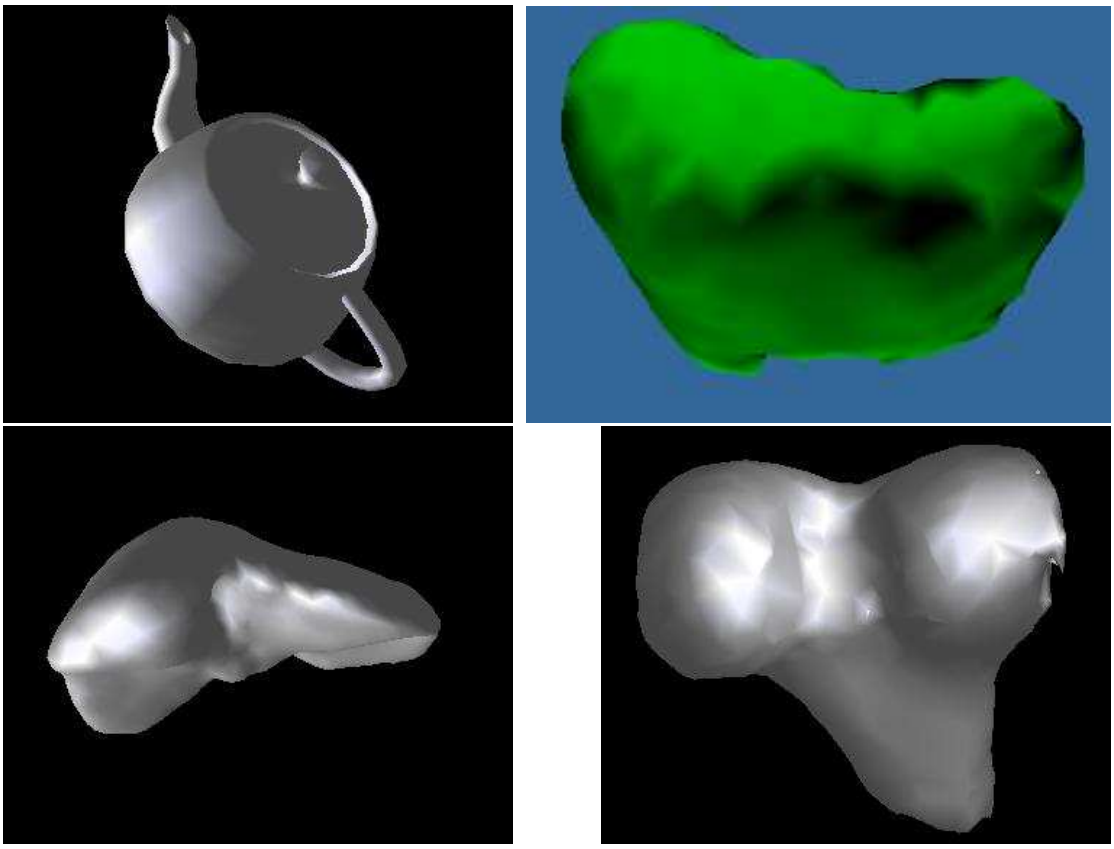

FIG. 4.6 – *Maillages utilisés pour mesurer le temps d'exécution de la mise à jour de hiérarchies de boîtes AABB. Haut-Gauche : Théière. Haut-Droite : Foie simplifié. Bas-Gauche : Foie original. Bas-Droite : Fémur.*

de volumes englobants dont le nombre de noeuds dépend du nombre de faces de l'objet. Ceci explique le fait que les temps d'exécution augmentent avec le nombre de faces.

Maillage	Nombre de sommets	Nombre de faces	Temps d'exécution (μs)
Théière	530	1024	6430
Foie (simplifié)	402	800	6012
Foie (original)	1691	3366	11625
Fémur	1998	3992	20426

TAB. 4.2 – *Temps d'exécution moyen du calcul de distances entre deux maillages concaves identiques.*

Les graphes représentés dans la figure 4.7 représentent les temps d'exécution de la détection de collision et du calcul du contour de collision en fonction du nombre de paires de faces en collision. Les maillages employés pour ce test sont les mêmes que ceux décrits dans la figure 4.6. Le test a été fait en utilisant deux objets ayant le même maillage.

Les graphes montrent que le temps d'exécution est une fonction linéaire du nombre de paires de faces en collision. Les différences entre les temps d'exécution pour différents maillages et pour un même nombre de paires s'expliquent par les différences existant dans les hiérarchies. En effet, notre construction de l'arbre des volumes englobants est basée sur un critère de minimisation du volume des boîtes. Ainsi, l'arbre obtenue n'est pas nécessairement équilibré, mais, est cependant plus compact autour de l'objet. Les différences entre les chemins dans les arbres expliquent les temps obtenus pour un même nombre de paires de faces en collision et des maillages différents.

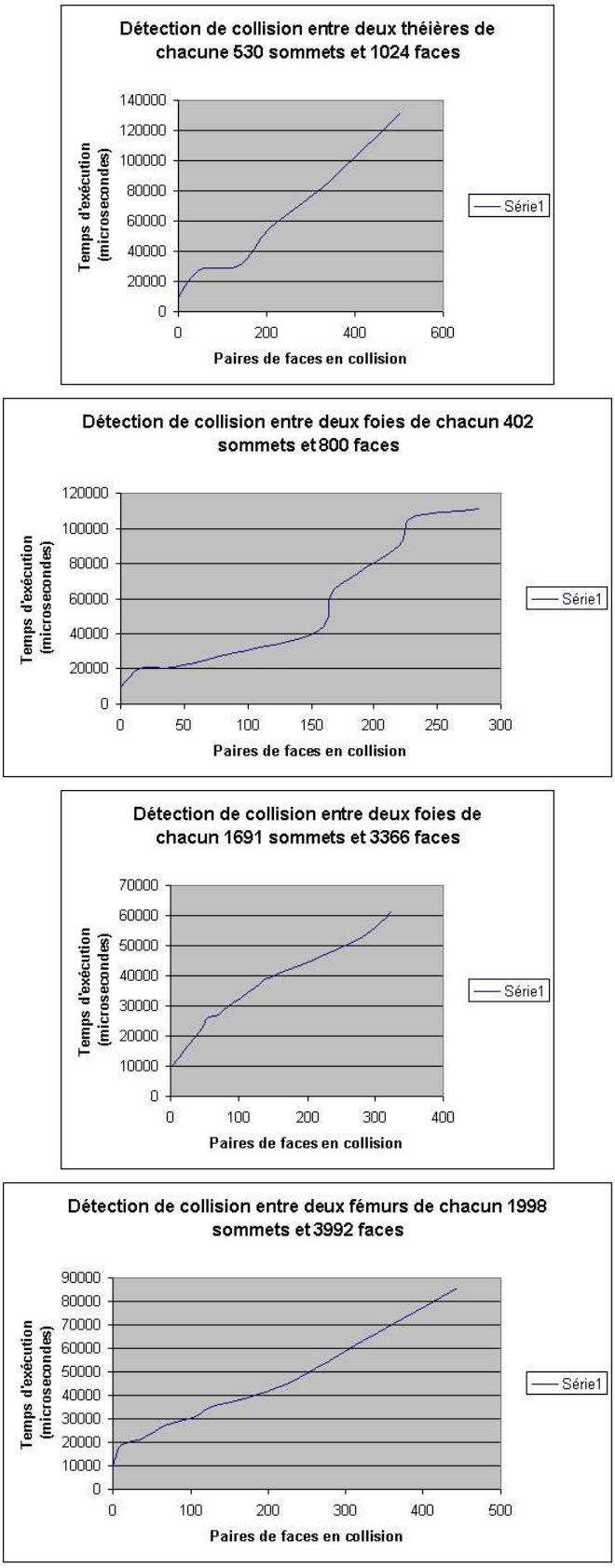

FIG. 4.7 – *Temps d'exécution en microsecondes de la détection de collision et du calcul du contour de collision de différents objets.*

Chapitre 5

Conclusion et perspectives

5.1 Contributions

Nous avons présenté dans ce rapport des aspects importants de la modélisation d'interactions.

Nous avons proposé un modèle de représentation hiérarchique de l'objet permettant de répondre rapidement et précisément aux différentes requêtes d'interaction.

Nous avons proposé un algorithme de calcul de distance entre objets concaves ainsi qu'un algorithme de détection de collision et de localisation de contact dans le cas d'objets déformables.

Ces deux méthodes sont à la fois rapide et numériquement stable.

Ces algorithmes ont été implémentés dans une librairie de détection de collision permettant :

- de calculer les distances entre des objets convexes ;
- de calculer les distances entre des objets concaves ;
- de détecter les collisions entre des objets rigides (convexes ou concaves) ;
- de détecter les collisions entre des objets déformables (convexes ou concaves) ;
- de fournir les éventuels contours de collision ;
- de fournir les éventuels volumes d'interpénétration.

Cette librairie est décrite et téléchargeable à l'url suivante :

http://www.inrialpes.fr/sharp/coldetection

Elle est, à ce jour, également intégrée au moteur physique aladynLight ainsi qu'au moteur physique développée au sein du projet Sharp en collaboration avec l'entreprise XL-Studio.

Notre approche permet donc une détection de collision plus rapide et plus stable. Elle s'applique à des objets polyédriques sans nécessité de propriétés géométriques ou physiques particulières puisqu'ils

peuvent être convexes, concaves, rigides ou déformables.

5.2 Perspectives

Une extension à l'algorithme de détection de collision entre polyèdres déformables peut être apportée afin de gérer l'auto-collision. En effet, il est possible d'étendre notre hiérarchie de boîtes AABB englobantes à une hiérarchie de patch adjacents. Ainsi, à chaque noeud correspondrait un patch et une boîte. On serait alors à même d'utiliser les algorithmes de test d'auto-collision décrits dans la partie 3.4. Une autre amélioration serait la gestion de la mise à jour de la hiérarchie dans le cas de changement de topologie de l'objet.

Bibliographie

[Bobrow, 1989] Bobrow, J. E. (1989). A direct minimization approach for obtainig the distance between convex polyhedra. *The International Journal of Robotics Research*. 33

[Cameron, 1997] Cameron, S. (1997). Enhancing GJK : Computing minimum penetration distances between convex polyhedra. In *Proceedings of IEEE International Conference on Robotics and Automation*. 9

[Cirak et al., 2000] Cirak, F., Ortiz, M., and Schröder, P. (2000). A new paradigm for thin-shell finite element analysis. *Journal of Numerical Methods Engineering*. 27

[Davanne et al., 2002] Davanne, J., Messure, P., and Chaillou, C. (2002). Stable haptic interaction in a dynamic virtual environment. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. 25, 26

[Dobkin and Kirkpatrick, 1990] Dobkin, D. and Kirkpatrick, D. (1990). Determining the separation of preprocessed polyhedra - A unified approach. In *Lecture Notes in Computer Science*, volume 443. Springer-Verlag. 5

[Garcia-Alonso et al., 1994] Garcia-Alonso, A., Serrano, N., and Flaquer, J. (1994). Solving the collision detection problem. In *IEEE Transactions on Computer Graphics and Applications*. 18

[Gilbert et al., 1988] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between objects in three-dimentional space. In *Proceedings of IEEE International Conference on Robotics and Automation*. 6, 7, 33

[Gottschalk et al., 1996] Gottschalk, S., Lin, M. C., and Manocha, D. (1996). OBB-Tree : A hierarchical structure for rapid interference detection. In *Proceedings of ACM SIGGRAPH*. 21, 22, 23

[Grinspun and Schröder, 2001] Grinspun, E. and Schröder, P. (2001). Normal bounds for subdivison-surface inteference detection. In *IEEE Transactions on Scientific Visualization*. 27, 28, 29

[Hamada and Hori, 1996] Hamada, K. and Hori, Y. (1996). Octree based approach to real-time collision-free path planning for robot manipulator. In *Proceedings of IEEE 4th International Workshop on Advanced Motion Control (AMC)*. 18

[Hubbard, 1996] Hubbard, P. M. (1996). Approximating polyhedra with spheres for time-critical collision detection. In *ACM Transactions on Graphics*. 21, 25

[Johnson, 1987] Johnson, D. W. (1987). *The optimisation of robot motion in the presence of obstacles*. PhD thesis, University of Michigan, USA. 8

[Joukhadar et al., 1996] Joukhadar, A., Wabbi, A., and Laugier, C. (1996). Fast contact localisation between deformable polyhedra in motion. In *Proceedings of CGS Conference on Computer Animation*. 9

[Laks Raghupathi and Cani, 2003] Laks Raghupathi, Vincent Cantin, F. F. and Cani, M.-P. (2003). Real-time simulation of self-collisions for virtual intestinal surgery. In *International Symposium on Surgery Simulation and Soft Tissue Modeling*. 26

[Larsen et al., 2000] Larsen, E., Gottschalk, S., Lin, M. C., and Manocha, D. (2000). Fast distance queries with rectangular swept sphere volumes. 43

[Larsson and Moller, 2001] Larsson, T. and Moller, T. (2001). Collision detection for continuously deforming bodies. In *Proceedings of EACG Conference on Eurographics*. 23, 24, 25, 32

[Lin and Canny, 1991] Lin, M. C. and Canny, J. F. (1991). A fast algorithm for incremental distance calculation. In *Proceedings of IEEE International Conference on Robotics and Automation*. 11, 33

[Lumelsky, 1985] Lumelsky, V. J. (1985). On fast computation of distance between line segments. *Information Processing Letters*. 33

[Mirtich, 1998] Mirtich, B. (1998). V-clip : fast and robust polyhedral collision detection. *ACM Transactions on Graphics (TOG)*, 17(3) :177–208. 13, 43

[Moller, 1997] Moller, T. (1997). A fast triangle-triangle intersection test. *Journal of Graphics Tools*. 15, 38, 43

[Naylor et al., 1990] Naylor, B. F., Amantides, J. A., and Thibault, W. C. (1990). Merging BSP trees yields polyhedral set operations. In *Proceedings of ACM SIGGRAPH*. 19

[Quilan, 1994] Quilan, S. (1994). Efficient distance computation between non-convex objects. In *Proceedings of IEEE International Conference on Robotics and Automation*. 25, 33, 34

[Reggiani et al., 2002] Reggiani, M., Mazzoli, M., and Caselli, S. (2002). An Experimental Evaluation of Collision Detection Packages for Robot Motion Planning. In *IEEE International Conference on Intelligent Robots and Systems, IROS'02*, Lausanne, Switzerland. 43

[Sundaraj et al., 2000] Sundaraj, K., Aulignac, D., and Mazer, E. (2000). A new algorithm for computing minimum distance. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. 33, 35, 43

[Sundaraj and Laugier, 2000] Sundaraj, K. and Laugier, C. (2000). Fast contact localisation of moving deformable polyhedras. In *Proceedings of IEEE International Conference on Automation, Robotics, Control and Vision*. 38, 43

[Thibault and Naylor, 1987] Thibault, W. C. and Naylor, B. F. (1987). Set operations on polyhedra using Binary Space Partioning trees. In *Proceedings of ACM Computer Graphics*. 19

[Van der Bergen, 1997] Van der Bergen, G. (1997). Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphic Tools*. 21, 23, 24, 32

[Van der Bergen, 1999] Van der Bergen, G. (1999). A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphic Tools*. 10

[Volino and Thalmann, 1994] Volino, P. and Thalmann, N. M. (1994). Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. In *Proceedings of the EACG Conference on Eurographics*. 26

[Volino and Thalmann, 1995] Volino, P. and Thalmann, N. M. (1995). Collision and self-collision detection : Efficient and robust solutions for highly deformable surfaces. In Terzopoulos, D. and Thalmann, D., editors, *Computer Graphics and Animation*. Springer-Verlag. 26