Détection de collisions entre objets rigides convexes autonomes

Jérémie Dequidt — Laurent Grisoni — Philippe Meseure — Christophe Chaillou

Laboratoire LIFL (CNRS - USTL)
Cité Scientifique - Bât M3
F-59655 Villeneuve d'Ascq Cedex
{dequidt,grisoni,meseure,chaillou}@lifl.fr

RÉSUMÉ. *Dans cet article, nous proposons une méthode complète de détection de collisions, c'est à dire une architecture qui permet d'obtenir les informations nécessaires à une simulation physique de polyèdres rigides convexes. Elle est constituée de différentes étapes détectant de plus en plus finement les collisions entre ces polyèdres. La dernière étape fournit des informations nécessaires à la génération d'une force de pénalités. De plus, cette architecture est compatible avec des objets autonomes (objets capables de gérer leur propre comportement).*

ABSTRACT. *This paper introduces a complete collision detection framework, which gives all necessary informations for the physical simulation of rigid convex polyhedra. This architecture is divided into different stages detecting collisions between polyedra more and more precisely. The last stage of this framework gives useful information to create penalty forces. Moreover, this framework may deal with autonomous bodies (bodies which can manage their own behaviour).*

MOTS-CLÉS : *Polyèdres rigides convexes, détection de collision, profondeur d'interpénétration, forces de pénalités.*

KEYWORDS: *Rigid convex polyedra, collision detection, penetration depth, penalty forces*

1. Introduction

Une application de réalité virtuelle régie par une simulation physique implique généralement un traitement systématique, chaque itération comprenant classiquement une étape de détection de collisions, une étape faisant un bilan des forces appliquées sur chaque objet, puis une intégration des équations du mouvement et/ou de déformation qui régissent le comportement de chaque objet. La détermination des collisions est donc une étape importante de ce traitement, et d'autant plus délicate qu'une version naïve impliquerait de tester la collision potentielle de toutes les paires d'objets de la scène (et donc un nombre $\mathcal{O}(n^2)$ de couples, n désignant ici le nombre d'objets de la scène), le traitement de chaque paire impliquant lui aussi un traitement de complexité quadratique en fonction du nombre de primitives géométriques utilisées pour définir les objets.

Pour les interactions entre objets, il existe deux types d'algorithmes : les algorithmes de contact ou les algorithmes d'interpénétrations. La première catégorie d'algorithmes garantit de ne pas violer la contrainte de contact mais implique de nombreux calculs et s'utilise principalement sur des objets rigides (ou sur une classe restreinte d'objets déformables [BAR 92]). L'autre catégorie d'algorithmes [BER 01, KIM 02] nécessite d'évaluer la *force d'interpénétration* qui permet, dans le bilan des forces, d'introduire une force de répulsion. C'est à ce deuxième cadre d'étude que nous nous intéressons.

La détection de collisions est un sujet qui a été largement étudié car il concerne un grand nombre de disciplines, comme par exemple la robotique (planification de trajectoires), l'IHM et la synthèse d'images. La majorité des algorithmes de détection exacte [LIN 91, GIL 88, DOB 90] se basent sur des polyèdres rigides convexes (la détection de collisions entre objets concaves peut être ramenée à une détection entre objets convexes par une décomposition de l'objet en parties convexes [CHA 95]). Cependant ces méthodes reposent sur une architecture centralisée. Le projet Alcove de l'équipe Graphix du LIFL a pour objectif de réaliser une plate-forme de simulation physique collaborative. La principale caractéristique de cette plate-forme est d'avoir une architecture totalement distribuée (absence de serveur). La simulation des objets doit donc être répartie sur un ensemble de machines. Pour cela, on se base sur la notion d'objets autonomes, capables de déterminer leur propre comportement dans l'environnement. Il est donc nécessaire de s'assurer que chaque étape de la simulation physique puisse être réalisée par les objets. Nous proposons donc une méthode de traitements de collisions qui s'affranchit d'une architecture centralisée et qui, pour le traitement des collisions, rend chaque objet autonome.

Cet article s'articule de la manière suivante : la section 2 présente les principales méthodes de détection connues et utilisées ainsi que les algorithmes donnant une distance permettant de séparer deux objets. La section 3 expose notre modèle de détection et détaille son originalité par rapport aux méthodes existantes.

2. Etat de l'art

De nombreux travaux ont été réalisés pour déterminer les collisions entre objets convexes. Un grand nombre se sont intéressés à la détection de collisions exactes entre deux polyèdres et d'autres aux moyens d'accélérer cette détection. Cependant peu de travaux se sont penchés sur l'élaboration d'une architecture complète de détection de collisions *i.e.* comment combiner des principes d'accélérations et une détection exacte afin d'obtenir une méthode complète et performante. Zachmann [ZAC 01] explique que cette architecture (qu'il désigne par le terme *pipeline*) doit être composé de filtres de plus en plus précis (et donc de plus en plus lourds en calcul) qui permettent de réduire le nombre de tests de collisions exactes à effectuer. Dans cette section, nous présentons les méthodes les plus utilisées pour la détection, puis celles permettant de calculer la distance d'interpénétration de deux objets en collision. Enfin nous évoquons les architectures existantes.

2.1. *Détection de collisions*

Détecter si deux objets sont en collision peut se faire à deux niveaux : celui de l'objet (on parle de détection exacte), ou celui d'une approximation plus ou moins fine (détection entre volumes englobants). Nous présentons tout d'abord les algorithmes de détection exacte puis nous donnons un aperçu des concepts permettant d'accélérer ce traitement.

2.1.1. *Détections exactes*

Pour déterminer si deux polyèdres convexes A et B sont en collision, il existe plusieurs familles d'approches :

• déterminer s'il existe un plan partitionnant l'espace en deux demi-espaces, l'un contenant A, l'autre B. C'est la solution proposée par Chung [CHU 96] et van den Bergen [BER 98] (méthodes itératives).

• calculer la distance entre ces deux objets. Si elle est inférieure ou égale à zéro, il y a collision [LIN 91, GIL 88, DOB 90]

L'algorithme proposé par Lin et Canny [LIN 91] consiste à déterminer les éléments (*i.e.* sommet, arête, facette) de A et de B permettant d'obtenir la plus petite distance. A partir d'un couple (f_A^0,f_B^0) d'éléments de A et B, on cherche par utilisation des régions de Voronoï entourant A et B un nouveau couple (f_A^1,f_B^1) plus proche. Cet algorithme est répété itérativement jusqu'à ce qu'un minimum local soit trouvé (qui est un minimum global grâce à la convexité de l'objet). Cet algorithme peut être optimisé en mémorisant le dernier couple trouvé. La cohérence temporelle [1] permet alors d'espérer une convergence en temps constant $\mathcal{O}(1)$. Une fois que ce couple est

1. les objets ont des déplacements faibles entre chaque pas de temps

trouvé, la distance euclidienne entre les deux objets est la plus petite distance entre le couple d'éléments trouvés.

Cependant, il est possible de calculer la distance de manière moins directe en passant par la différence de Minkowski (notée $\mathcal{M}$) qui est définie de la manière suivante :

$$\mathcal{M} = A - B = \{x - y \setminus x \in A,\ y \in B\}$$

Cette différence est convexe lorsque les polyèdres A et B sont convexes. L'intérêt de cette différence est que l'on ramène le calcul de la distance entre deux objets à la distance entre $\mathcal{M}$ et l'origine. De plus si l'origine se trouve à l'intérieur de $\mathcal{M}$ les objets sont en collision.

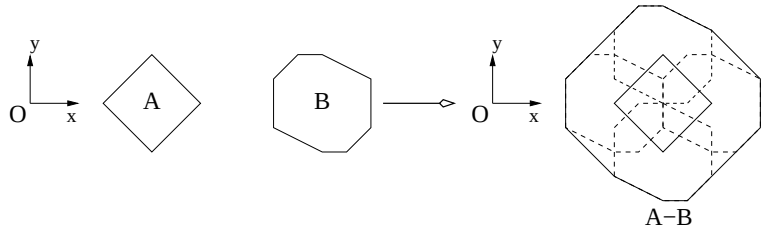

Figure 1. *Différence de Minkowski de deux polyèdres convexes.*

Cependant la construction de cette différence est en $\mathcal{O}(nm)$ (où n et m sont les nombres de sommets de A et de B). Pour éviter une construction explicite de $\mathcal{M}$, on peut utiliser les points de support. Le point de support d'un polyèdre A par rapport à un vecteur $\vec{v}$ donné est le point $s_A(\vec{v})$ appartenant à A tel que le produit scalaire $\vec{v} \cdot s_A(v)$ soit maximal. Une propriété sur les points de support énonce qu'un point de support de la différence de Minkowski peut facilement être obtenu à partir des points de support de A et de B suivant la formule suivante : $s_{\mathcal{M}}(\vec{v}) = s_A(\vec{v}) - s_B(-\vec{v})$. Cette propriété est un des points de départ de l'algorithme GJK [GIL 88]. En effet, cet algorithme itératif construit à chaque étape un simplexe [2] en se basant sur cette propriété, ce nouveau simplexe étant plus proche de l'origine que le simplexe précédent. Cet algorithme converge en $\mathcal{O}(n)$ (où n est le nombre de sommets de $\mathcal{M}$).

2.1.2. *Accélération de la détection*

Lorsque l'application contient un nombre d'objets relativement élevé, il devient impossible d'utiliser ces algorithmes sur la totalité des couples. L'idée est donc d'éliminer rapidement par des critères simples les couples d'objets ne pouvant entrer en collision. Ces techniques sont nombreuses [LIN 98, JIM 01, MES 02] et peuvent être regroupées en 2 catégories : les techniques de *recherche de proximité* et les techniques de *détection approximative* :

2. enveloppe convexe d'au plus $n + 1$ points pour un espace de dimension n

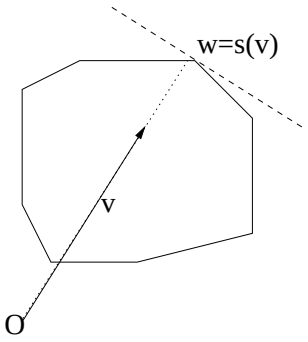

Figure 2. *Point de support d'un objet convexe selon une direction v.*

• Les techniques de *recherche de proximité* vont considérer l'ensemble des objets de la scène et vont déterminer les collisions potentielles. Dans cette catégorie, on trouve des méthodes de partitionnement spatial, que ce soit en grilles régulières, en BSP-Trees ou Octrees. Il existe aussi des méthodes qui utilisent la position des objets dans l'espace ou leur déplacement. La plus performante est le *Sweep And Prune* [COH 95] qui consiste à projeter les objets (ou une approximation des objets) sur les 3 axes (x,y,z). On obtient ainsi des intervalles et si les intervalles appartenant à deux objets sont disjoints alors les objets correspondant sont disjoints eux aussi.

• Les accélérations de type *détection approximative* travaillent uniquement sur des couples d'objets. Il est courant d'utiliser des volumes simples (ou des hiérarchies des volumes simples) approximant les objets dont on veut résoudre les collisions. Ces volumes peuvent être englobants et dans ce cas si les volumes englobants sont disjoints, les objets le sont aussi. Les volumes les plus utilisés sont les sphères [HUB 95], les boîtes alignées ou non sur les axes [BER 97, GOT 96], les k-DOP [KLO 98, ZAC 98], les enveloppes sphériques (partie du volume obtenu à partir de deux sphères concentriques) [KRI 97] et les enveloppes convexe [BAR 96]. Généralement, la *détection approximative* s'achève par une phase de détection exacte décrite dans 2.1.1.

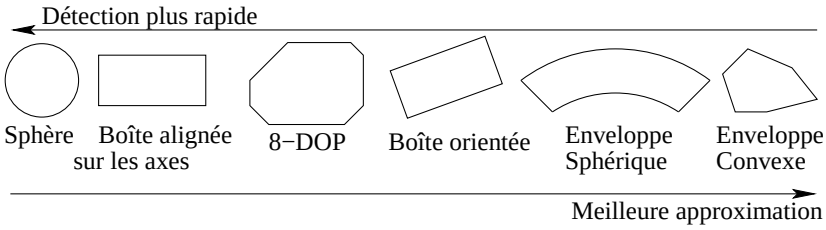

Figure 3. *Quelques exemples de volumes englobants. Plus le volume englobant est complexe, mieux il approxime l'objet mais plus longue est la détection de collisions entre deux volumes de ce type (et réciproquement).*

Cette sous-section a mis en évidence les principaux algorithmes permettant de déterminer si deux objets étaient en intersection et les techniques permettant d'accélérer cette détection. Cependant, nous n'avons pas assez d'informations pour séparer de manière correcte ces objets. Ce point est traité dans la section suivante.

2.2. *Profondeur d'interpénétration*

Lorsque deux objets sont en collision, il est nécessaire d'estimer le degré d'interpénétration de ces deux objets. Pour cela, on définit la profondeur d'interpénétration : c'est la norme de la plus petite translation qui permet d'avoir les objets en contact (et seulement en contact). Cette distance peut être facilement obtenue avec la différence de Minkowski : c'est la plus petite distance entre $\mathcal{M}$ et l'origine (avec cette fois l'origine se trouvant à l'intérieur de $\mathcal{M}$).

Cameron propose dans [CAM 97] de borner cette distance à partir du simplexe fourni à la terminaison de l'algorithme *GJK*. Cette méthode est directe, cependant la borne obtenue n'est pas assez précise dans la majorité des cas. Joukhadar [JOU 99] utilise un algorithme incrémental qui applique l'algorithme *GJK* après avoir translaté un des deux objets selon un certain vecteur. Cela lui permet de déterminer la distance d'interpénétration mais aussi la direction de contact. Les inconvénients de cette méthode sont sa lenteur de convergence (même si elle peut être réduite par utilisation de la cohérence temporelle) et le fait que la résolution soit une méthode de recherche locale (le minimum trouvé ne sera pas forcément le minimum global). L'algorithme DEEP [KIM 02] est aussi un algorithme de recherche locale. Son auteur restreint le calcul de la différence de Minkowski grâce à la carte des normales de Gauss. En parcourant la surface de la différence de Minkowski, il obtient le couple d'éléments donnant la distance d'interpénétration. Cet algorithme même s'il possède de bonnes performances, connaît des problèmes de convergence (liés à la recherche locale). La méthode que nous avons implémentée est celle de van den Bergen [BER 01]. Elle utilise en entrée le simplexe fourni par la dernière itération de l'algorithme *GJK*. Par la suite, elle va raffiner cette approximation de $\mathcal{M}$ jusqu'à obtenir une bonne approximation de la distance d'interpénétration. L'algorithme assure une convergence rapide (moins rapide que DEEP [KIM 02]) mais possède des problèmes de précision dans certains cas, notamment les cas dégénérés où les objets en collision ont certaines de leurs faces parallèles.

2.3. *Architectures existantes*

Comme nous l'avons signalé en introduction, rares sont les travaux concernant une architecture de détection de collision (voir figure 4). Zachmann [ZAC 01] propose d'utiliser une hiérarchie de volumes englobants de type k-DOP associée à une méthode de détection probabiliste.

Ramaeker [3] propose une architecture dont l'implémentation est parallèle. Elle est constituée de deux étages. La première détecte les collisions entre volumes englobants de type boîte (orientée ou non). La deuxième utilise l'algorithme *V-Clip* [MIR 98] pour la détection exacte (*V-Clip* est une extension de l'algorithme de Lin-Canny [LIN 91]).
Lin [LIN 96] construit une architecture beaucoup plus étoffée qui implique de calculer un certain nombre de structures : chaque objet possède un volume englobant (boîte alignée sur les axes du repère global), une hiérarchie de volumes englobants (boîtes orientées) et une enveloppe convexe. La *recherche de proximité* est réduite à l'application de l'algorithme de *Sweep And Prune* sur les boîtes alignées sur les axes. La *détection approximative* détermine tout d'abord si les enveloppes convexes des objets sont en collision (à l'aide des régions de Voronoï). Dans le cas d'une collision entre enveloppes convexes, on détecte si les hiérarchies de boîtes orientées sont en intersection. Enfin la détection exacte se fait entre triangles dont les boîtes orientées s'interpénètrent.
La méthode de Chung [CHU 96] propose, pour la *recherche de proximité* d'utiliser une décomposition spatiale de type grille régulière suivie d'un *Sweep And Prune* sur des boîtes alignées par rapport aux axes. Les deux étapes suivantes de l'architecture sont incluses dans la *détection approximative*. On recherche tout d'abord de manière itérative un axe séparateur (pour déterminer s'il y a collision ou non). S'il y a collision, l'algorithme *GJK* est appliqué sur l'instant précédant la collision afin d'estimer la profondeur d'interpénétration.

Les architectures de [ZAC 01, LIN 96] ne se limitent pas aux objets convexes. Cependant celle proposée dans [ZAC 01] est assez rudimentaire puisqu'elle est la juxtaposition d'une détection de type hiérarchie de volumes englobants et d'une détection exacte. [LIN 96] est beaucoup plus intéressante d'une part car la *recherche de proximité* est très efficace et d'autre part car la *détection approximative* est très complète mais elle requiert beaucoup de calculs (notamment pour la détection entre hiérarchies de boîtes orientées). L'inconvénient principal de [ZAC 01, LIN 96] est qu'aucune estimation de la distance d'interpénétration n'est fournie contrairement à la méthode de Chung [CHU 96]. Cette dernière méthode est très efficace mais elle peut être optimisée car on trouve dans les deux dernières étapes une redondance de certains calculs.

3. Proposition d'une architecture

Par rapport aux travaux existants, l'architecture que nous présentons doit être :

– plus complète : une réponse booléenne (vrai si collision, faux sinon) est insuffisante. Une simulation mécanique nécessite des informations plus complexes, comme une direction et une profondeur de l'interpénétration lorsqu'il y a collision.

– non centralisée : la majorité des calculs à effectuer sont fait par les objets eux-mêmes.

3. http ://parallel.vub.ac.be/documentation/pvm/Example/Marc_Ramaekers/parallel.html

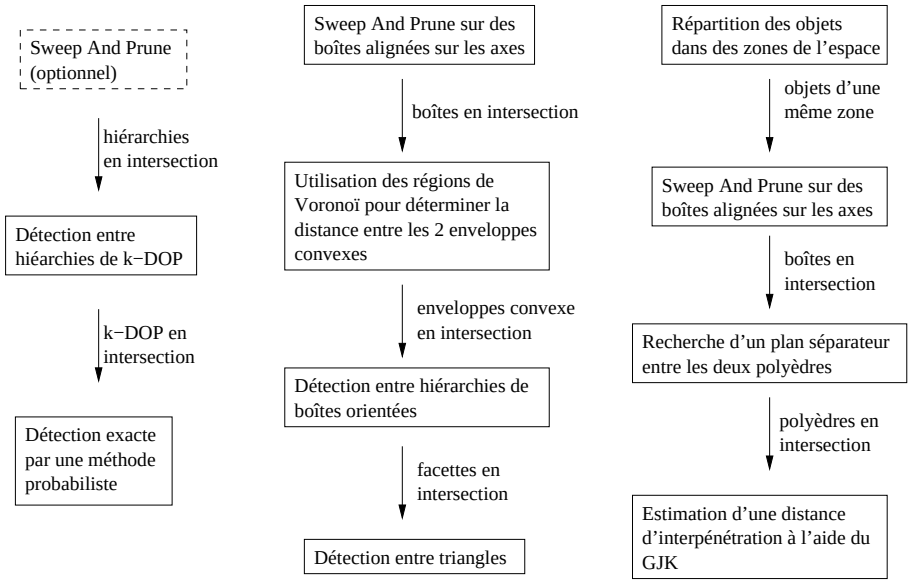

Figure 4. *Comparaison de différentes architectures (de gauche à droite [ZAC 01], [LIN 96], [CHU 96]).*

– et enfin rapide et plus cohérente que celles existantes.

Nous détaillons tout d'abord cette architecture puis nous présentons quelques résultats.

3.1. *L'architecture en détail*

Comme les architectures présentées précédemment, notre environnement est constitué de deux phases distinctes : une phase d'accélération de la détection, puis une phase de détection exacte. Nous allons détailler ces deux phases (voir figure 5).

3.1.1. *Accélération de la détection*

Pour éliminer rapidement les couples d'objets qui ne sont pas en collision, nous subdivisions la totalité de l'espace en un certain nombre de cellules régulières. Dans chacune de ces zones se trouve un agent. Ces agents ont pour objectif de permettre un traitement plus rapide de la *recherche de proximité* dans le cas d'environnements répartis : soit par un traitement parallèle (pour un environnement synchrone) soit de manière complètement autonome (pour un environnement asynchrone). La méthode la plus efficace de *recherche de proximité* lorsque le nombre d'objets est élevé (plus d'une centaine d'objets) est le *Sweep And Prune* [COH 95]. Cependant, il est couramment utilisé avec des boîtes alignées sur les axes ce qui génère un nombre trop

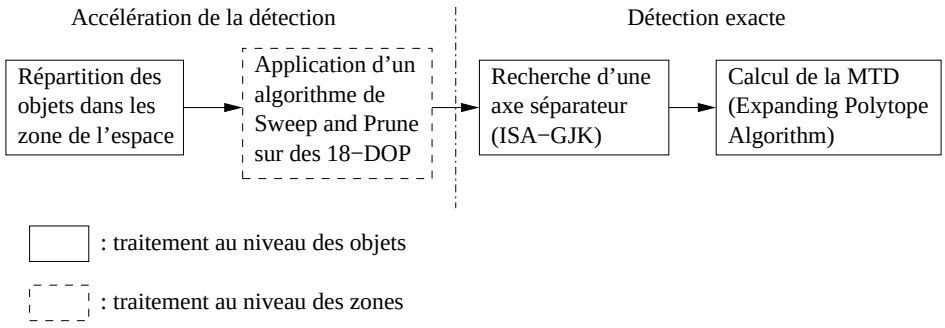

Figure 5. *Architecture de traitement des collisions.*

important de collisions potentielles (i.e. le rapport collisions potentielles sur collisions exactes est très élevé). L'idée est donc d'utiliser un volume englobant qui approxime mieux les objets, dont la détection et la construction soit rapide et qui soit compatible avec l'utilisation du *Sweep And Prune*. Nous avons donc choisi les k-DOP de [KLO 98, ZAC 98]. Ces volumes sont des extensions des boîtes alignées sur les axes à $k/2$ axes (on peut considérer les boîtes alignées sur les axes comme des 6-DOP). Plus k est grand et meilleure est l'approximation des volumes, mais plus coûteuse est la détection de ces volumes. C'est pourquoi nous nous sommes tournés vers les 18-DOP [4] qui allie une assez bonne approximation de l'objet et dont la détection est assez rapide [KLO 98]. La construction de ce genre de volume et leur mise à jour sont faites de manière rapide en utilisant les points de support définis en 2.1.1 et en utilisant la cohérence temporelle associée à un algorithme de type escalade de collines : on calcule les points de supports des 9 axes et de leurs opposés ce qui nous permet d'obtenir les 18 paramètres du volume englobant. Ainsi par l'utilisation, dans chacune des zones, d'un algorithme de *Sweep And Prune* sur des 18-DOP, nous avons une accélération de la détection rapide et qui génère beaucoup moins de collisions potentielles que l'algorithme classique basé sur des boîtes alignées sur les axes. En sortie de cet étage de l'architecture, les agents envoient à chaque objet l'identifiant des objets avec lesquels il est entré en collision. Chaque objet reçoit donc une liste d'identifiants et indique sa position courante à tous les objets désignés dans la liste d'identifiants (voir figure 6).

3.1.2. *Détection exacte*

A partir des indications de position obtenues dans la phase précédente, chaque corps va résoudre ses collisions. Pour cela, il applique une variante de l'algorithme *GJK* : l'*ISA-GJK* [BER 98]. *ISA-GJK* (*ISA* pour *Incremental Separating Axis*) comme son nom l'indique ne détermine pas la distance entre deux objets mais cherche l'existence d'un axe séparateur. Cet algorithme est très robuste, possède une convergence très rapide en temps constant (avec l'utilisation de la cohérence temporelle) et de

4. construits à l'aide de neuf axes : $\vec{i}$, $\vec{j}$, $\vec{k}$ qui constituent le repère orthonormé mais aussi $\vec{i}+\vec{j}$, $\vec{i}+\vec{k}$, $\vec{j}+\vec{k}$, $\vec{i}-\vec{j}$, $\vec{i}-\vec{k}$ et $\vec{j}-\vec{k}$

meilleures performances que celui de Lin-Canny [LIN 91].
Si aucun axe séparateur n'a été trouvé, la dernière étape de notre architecture est activée. La distance d'interpénétration est calculée en utilisant l'algorithme de van den Bergen [BER 01] qui par raffinement progressif d'une approximation de la différence de Minkowski peut déterminer une estimation de la distance d'interpénétration. A partir de cette distance d'interpénétration, il est possible de calculer une composante vectorielle permettant de déterminer la force de réaction (i.e. force de pénalité) en assimilant ce vecteur à l'allongement d'un ressort possédant une certaine raideur k (soit une expression de la force $\vec{F} = -k \cdot \vec{x}$ où $\vec{x}$ est une estimation de l'interpénétration).

Comme on peut le constater, si on considère deux objets en collision, la détection et le traitement associé sont calculés séparément par les deux objets concernés (i.e. A va traiter sa collision avec B et B sa collision avec A). Si la simulation est synchrone les forces de pénalités sont identiques (mais de sens opposé). Par contre, pour une simulation asynchrone où chaque corps fonctionne à une fréquence qui lui est propre, il est possible que la position de A reçue par B ne soit pas la dernière calculée et donc les forces de pénalités générées ne sont pas nécessairement symétriques. Par conséquent, le principe d'action / réaction n'est pas nécessairement garanti.

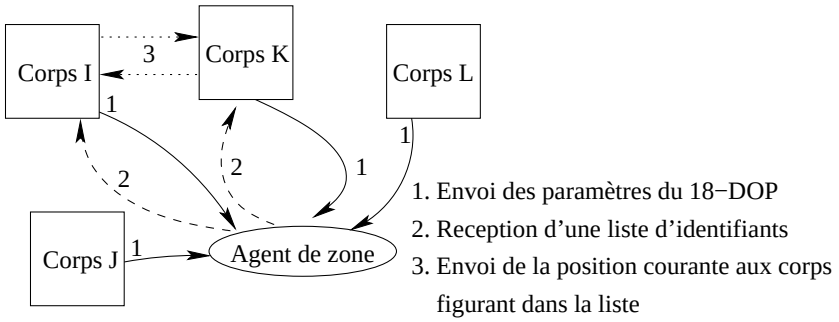

Figure 6. *Informations échangées lors de la détection de collisions : les différents corps envoient les paramètres de leur 18-DOP (1). Si l'agent de zone détecte que les volumes englobants de I et K sont en collision, il prévient ces deux corps (2). Ensuite les corps se transmettent leur position respective (3).*

3.2. *Résultats*

La méthode proposée est naturelle et cohérente dans l'enchaînement des différentes étapes. Notre *détection large* permet de réduire sensiblement les couples d'objets à tester par l'utilisation de volumes englobants approximant de manière assez fine les objets. De même l'utilisation des points de support n'ajoute qu'un faible surcoût à la construction de k-DOP par rapport à des volumes englobants plus simples. Les deux étapes de la phase de détection exacte s'imbriquent de manière logique et per-

mettent de réutiliser certains calculs effectués par l'étage précédent (ex : l'obtention de la première approximation de $\mathcal{M}$ se fait lors de la recherche d'un axe séparateur).

Pour illustrer notre architecture, nous avons choisi de simuler de manière très simple des corps rigides. A chaque pas de temps, les collisions sont résolues (*i.e.* calcul des forces de pénalités), un bilan de forces est effectué et les équations de mouvements sont intégrées (à l'aide de la méthode numérique d'Euler), afin de déterminer la position et la vitesse des objets. Cette résolution mécanique est faite elle aussi de manière autonome par chaque objet. La figure 7 montre des sphères tombant en chute libre sur quelques briques fixes. Les tests montrent que le temps de calcul moyen pour la résolution mécanique d'un système de 200 objets est de 16 ms sur un pentium IV 2Ghz (41 ms pour 600 objets).

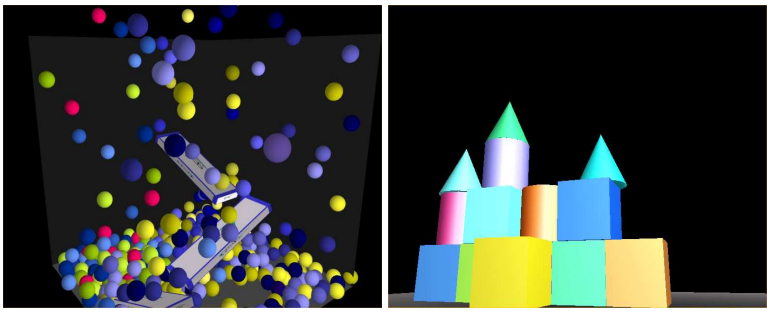

Figure 7. *Simulation de corps rigides convexes. A gauche, quelques centaines de sphères en chute libre sur des briques fixes inclinées. A droite, empilement de plusieurs types d'objets (cubes, cylindres, cônes)*

Une autre particularité de notre architecture est que nous n'avons pas opté pour une approche centralisée. Le fait que chaque objet détecte les collisions exactes avec d'autres objets sans passer par un serveur permet un portage dans un environnement virtuel distribué. Dans le cas où une telle implémentation était réalisée, les choix effectués pour notre architecture ne sont pas en contradiction avec les contraintes d'un environnement réparti. Notamment, la stratégie d'avoir des volumes englobants assez complexes permet de réduire les nombres de messages échangés entre objets et aussi de diminuer le nombre de tests de détection exacte. De plus la résolution des collisions se faisant au niveau de chaque objet composant la scène, il est possible de rendre chaque objet autonome (l'objet résout lui même ses collisions, ses équations de mouvement ...) et donc d'engendrer un simulateur complètement distribué.

4. Conclusion

Dans cet article, nous avons présenté un modèle complet de traitements de collision. Ce modèle détermine les collisions et permet de générer des forces de pénalités en réponse à une collision. Il se limite à l'heure actuelle aux objets convexes et rigides

mais est extensible aux objets déformables. Pour cela, il est nécessaire de complexifier le pipeline soit en utilisant la méthode de Fisher [FIS 01] qui permet de déterminer la distance d'interpénétration d'objets déformables en se basant sur les champs de distance et sur l'approche Level-Set [OSH 88]. Une autre possibilité est de sélectionner les facettes potentiellement en collision (voir [JOU 99]).

Cette architecture est le premier pas vers une simulation physique répartie. De nombreux problèmes existent encore pour avoir une simulation répartie effective : par exemple la gestion des contraintes entre objets (*i.e.* objets articulés). De même la simulation devant se faire à une fréquence élevée (proche du kHz), de nombreux choix doivent être effectués pour garantir une simulation réaliste avec les limitations matérielles actuelles (temps de latence et bande-passante des réseaux).

5. Bibliographie

[BAR 92] BARAFF D., WITKIN A., « Dynamic Simulation of Non-penetrating Flexible Bodies », *SIGGRAPH'92 Conference Proceedings*, 1992, p. 303-308.

[BAR 96] BARBER C. B., DOBKIN D. P., HUHDANPAA H., « The Quickhull Algorithm for Convex Hulls », *ACM Transactions on Mathematical Software*, vol. 22(4), 1996, p. 469–483.

[BER 97] VAN DEN BERGEN G., « Efficient Collision detection of Complex Derformable Models using AABB Trees », *Journal of Graphics Tools*, vol. 4, 1997, p. 1–13.

[BER 98] VAN DEN BERGEN G., « A Fast and Robust GJK Implementation for Collision Detection of Convex Objects », *Journal of Graphic Tools*, vol. 4(2), 1998, p. 7-25.

[BER 01] VAN DEN BERGEN G., « Proximity Queries and Penetration Depth Computation on 3D Game Object », *Game Developers Conference*, 2001.

[CAM 97] CAMERON S., « Enhancing GJK : Computing Minimum and Pentration Distances between Convex Polyhedra », *International Conference on Robotics and Automation*, 1997, p. 3112-3117.

[CHA 95] CHAZELLE B., DOBKIN D., SHOURABOURA N., TAL A., « Strategies for Polyhedral Surface Decomposition : An Experimental Study », *11th Symposium on Computational Geometry*, 1995, p. 297-305.

[CHU 96] CHUNG K., « An Efficient Collision Detection Algorithm for Polytopes in Virtual Environment », PhD thesis, Department of Computer Science, University of Hong Kong, 1996.

[COH 95] COHEN J., LIN M., MANOCHA D., PONAMGI M., « I-Collide : An Interactive and Exact Collision Detection System for Large-Scale Environments », *ACM interactive 3D Graphics Conference*, 1995, p. 189-196.

[DOB 90] DOBKIN D., KIRKPATRICK D., « Determining the Separation of Preprocessed Polyhedra : A Unified Approach », *ICALP*, 1990, p. 400-413.

[FIS 01] FISHER S., LIN M., « Fast Penetration Depth Estimation for Elastic Bodies Using Deformed Distance Field », *Intelligent Robots and Systems*, 2001.

[GIL 88] GILBERT E., JOHNSON D., KEERTHI S., « A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space », *IEEE Journal of Robotics*

and Automation, vol. RA-4, 1988, p. 193-203.

[GOT 96] GOTTSCHALK S., LIN M. C., MANOCHA D., « OBBTree : A Hierarchical Structure for Rapid Interference Detection », *Computer Graphics*, vol. 30, 1996, p. 171–180.

[HUB 95] HUBBARD P., « Real-time Collision Detection and Time-critical Computing », *1st Workshop on Simulation and Interaction in Virtual Environments*, 1995.

[JIM 01] JIMÉNEZ P., THOMAS F., TORRAS C., « 3D Collision Detection : A Survey », *Computer Graphics*, vol. 25, 2001, p. 269-285.

[JOU 99] JOUKHADAR A., SCHEUER A., LAUGIER C., « Fast Contact Detection Between Moving Deformable Polyhedra », *IEEE-RSJ Intelligent Robots and Systems*, 1999.

[KIM 02] KIM Y., LIN M., MANOCHA D., « DEEP : Dual-Space Expansion for Estimating Penetration Depth Between Convex Polytopes », *IEEE International Conference on Robotics and Automation*, Mai 2002.

[KLO 98] KLOSOWSKI J., HELD M., MITCHELL J., SOWIZRAL H., ZIKAN K., « Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs », *T-VCG(4)*, 1998, p. 21-36.

[KRI 97] KRISHNAN S., PATTEKAR A., LIN M., MANOCHA D., « Spherical shells : A higher-order bounding volume for fast proximity queries », rapport, 1997, Department Department of Computer Science, University of North Carolina at Chapel Hill.

[LIN 91] LIN M., CANNY J., « A Fast Algorithm for Incremental Distance Calculation », *IEEE International Conference on Robotics and Automation*, 1991.

[LIN 96] LIN M., MANOCHA D., COHEN J., GOTTSCHALK S., « Collision Detection : Algorithms and Applications », *Algorithms for robotics motion and manipulation*, 1996, p. 129-142.

[LIN 98] LIN M., GOTTSCHALK S., « Collision Detection Between Geometric Models : A Survey », *IMA Conference on Mathematics of Surfaces*, 1998.

[MES 02] MESEURE P., *Animation basée sur la physique pour les environnements intéractifs temps réel : habilitation à diriger des recherches*, Université des Sciences et Technologies de Lille, 2002.

[MIR 98] MIRTICH B., « V-Clip : Fast and robust polyhedral collision detection », *ACM Transactions on Graphics*, vol. 17, 1998, p. 177–208.

[OSH 88] OSHER S., SETHIAN J., « Fronts Propagating with Curvature-Dependent Speed : Algorithms Based on Hamilton-Jacobi Formulations », *Journal of Computational Physics*, vol. 79, 1988, p. 12–49.

[ZAC 98] ZACHMANN G., « Rapid Collision Detection by dynamically aligned DOP-trees », *Proceedings of IEEE, VRAIS*, 1998.

[ZAC 01] ZACHMANN G., « Optimizing the collision detection pipeline », *First International Game Technology Conference (GTEC)*, 2001.