# Détection de collisions entre objets rigides convexes autonomes

**J. Dequidt, L. Grisoni, P. Meseure, C. Chaillou**

LIFL - Université de Lille 1
Bât M3, 59655 Villeneuve d'Ascq cedex
{dequidt,grisoni,meseure,chaillou}@lifl.fr

**Résumé :** *Dans cet article, nous proposons une méthode complète de traitement de collisions. Elle est constituée de différentes étapes qui permettent de détecter de plus en plus finement les collisions entre objets rigides convexes. La dernière étape fournit des informations nécessaires à la génération d'une force de pénalités. Ce framework est compatible avec des objets autonomes (objets capables de gèrer leur propre comportement).*

**Mots-clés :** Objets rigides convexes, détection de collision, distance d'interpénétration, forces de pénalités.

## 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 se limite aux objets rigides [RKC02]. L'autre catégorie d'algorithmes 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 travaux se sont penchés sur une détection de collisions entre objets 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). 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 de manière autonome par les objets. Nous proposons donc une méthode de traitements de collisions qui s'affanchit de l'hypothèse d'une architecture centralisée et qui permet, pour le traitement des collisions, de rendre 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 très 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'un framework complet 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 [Zac01] explique que ce framework (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 frameworks existants.

## 2.1 Détection de collisions

Détecter si deux objets sont en intersection peut se faire à différents "grains". Au niveau le plus fin de l'objet lui-même, on parle de détection exacte. Un autre choix est d'utiliser une approximation plus ou moins fine de l'objet et dans ce cas, la détection est faite 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 (avec par exemple une détection entre volumes englobants).

### 2.1.1 Détections exactes

Pour déterminer si deux objets convexes $A$ et $B$ sont interpénétrés, 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 [Chu96] et van den Bergen [vdB98] (méthodes itératives).
- calculer la distance entre ces deux objets. Si elle est inférieure ou égale à zéro, il y a collision [LC91, GJK88]

L'algorithme proposé par Lin et Canny [LC91] 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 un nouveau couple $(f_A^1, f_B^1)$ plus proche. Cet algorithme est répété itérativement jusqu'à ce qu'un 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 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 objets $A$ 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.

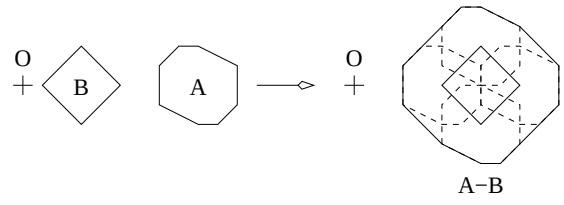


FIG. 1 – Différence de Minkowski de deux objets convexes.

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 supports. 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 $GJK$ [GJK88]. En effet,

[1] les objets ont des d´eplacements faibles entre chaque pas de temps

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 $\mathcal{M}$).

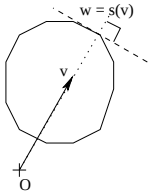


FIG. 2 – Point de support d'un objet convexe selon une direction $v$.

### 2.1.2 Accélérer 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 [LG98, JTT01, Mes02] et peuvent être regroupées en 2 catégories : les techniques de *broad phase* et les techniques de *narrow phase* :

- Les techniques de *broad phase* 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 de voxels (grille régulière), 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* [CLMP95] qui consiste à projeter les 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 accélerations de type *narrow phase* 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 englobant et dans ce cas si les volumes englobants sont disjoints, les objets le sont aussi. Mais on peut aussi construire des volumes simples englobés par les objets, qui s'ils sont en collision implique que les objets sont aussi en collision. Généralement, la *narrow phase* s'achève par une phase de détection exacte décrite dans 2.1.1.

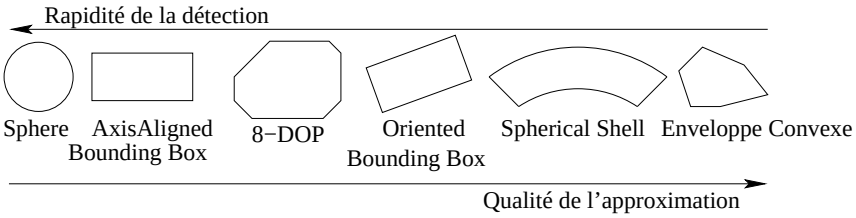


FIG. 3 – Quelques exemples de volumes englobants.

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 Distance 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 distance d'interpénétration [3] : c'est la norme du plus petit vecteur (*i.e.* de plus petite norme) qui permet par translation d'avoir les objets en contact (et seulement en contact). Cette distance d'interpénétration 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 $\mathcal{M}$).

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

[3] aussi not´ee MTD : Minimum Translational Distance

Cameron propose dans [Cam97] de borner cette distance à partir du dernier simplexe fourni par l'algorithme *GJK*. Cette méthode est directe, cependant la borne obtenue n'est pas assez précise dans la majorité des cas. Joukhadar [JSL99] utilise un algorithme incrémental qui applique l'algrithme *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 [KLM02] est aussi un algorithme de recherche locale. Kim calcule de manière implicite la différence de Minkowski en utilisant l'espace dual de l'espace objet (*i.e.* l'espace des normales). 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 [vdB01]. 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 [KLM02]) mais possède des problèmes de précision dans certains cas.

### 2.3 Pipelines existants

Comme nous l'avons signalé en introduction, rares sont les travaux concernant un framework complet de détection de collision. Zachmann [Zac01] propose d'utiliser une hiérarchie de volumes englobants de type $k$-DOP associée à une méthode de détection probabiliste.
Lin [LMCG96] construit un pipeline beaucoup plus étoffé qui implique de calculer un certain nombre de structures : chaque objet possède un volume englobant de type AABB [4], une hiérarchie de volumes englobants de type OBB [5] et une enveloppe convexe. La *broad phase* est réduite à l'application de l'algorithme de *Sweep And Prune* sur les AABB. La *narrow phase* détermine tout d'abord si les enveloppes convexes des objets sont en intersection (à l'aide des régions de Voronoï). Dans le cas d'une collision entre enveloppes convexes, on détecte si les hiérarchie d'OBB sont en intersection. Enfin la détection exacte se fait entre triangles dont les OBB s'interpénétrent.
La méthode de Chung [Chu96] propose, pour la *broad phase* d'utiliser une décomposition spatiale régulière en grille de voxels suivie d'un *Sweep And Prune* sur des boîtes alignées par rapport aux axes. Les deux étapes suivantes dans le pipieline sont incluses dans la *narrow phase*. 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 la frame précédant la collision afin d'estimer la distance d'interpénétration.

Les pipelines [Zac01, LMCG96] ne se limitent pas aux objets convexes. Cependant le framework proposé dans [Zac01] est assez rudimentaire puisqu'il estla juxtaposition d'une détection de type hiérarchie de volumes englobants et d'une détection exacte. [LMCG96] est beaucoup plus intéressant d'une part car la *broad phase* est très efficace et d'autre part car la *narrow phase* est très complète mais elle requiert beaucoup de calculs (notamment pour la détection entre hiérarchies de OBB). L'inconvénient principal de [Zac01, LMCG96] est qu'aucune estimation de la distance d'interpénétration n'est fournie contrairement à la méthode de Chung [Chu96]. Cette méthode est très efficace mais elle peut être optimisée car on trouve dans les deux dernières étapes une redondance de certains calculs.

## 3 Traitement de collisions

Le framework que nous exposons doit être plus complet et plus cohérent que ceux existants en garantissant une détection rapide, exacte et qui donne la distance d'interpénétration dans le cas de collisions. Nous détaillons tout d'abord notre pipeline puis nous présentons quelques résultats.

### 3.1 Le framework en détail

Comme les pipelines présentés précédemment, notre framework est constitué de deux phases distinctes : une *broad phase* (qui s'applique sur tous les objets de la scène) et une *narrow phase* (qui ne fonctionne que sur des couples

[4] Axis Aligned Bounding Box
[5] Oriented Bounding Box

d'objets). Nous allons détailler ces deux phases constitutives de notre framework (voir figure 4).

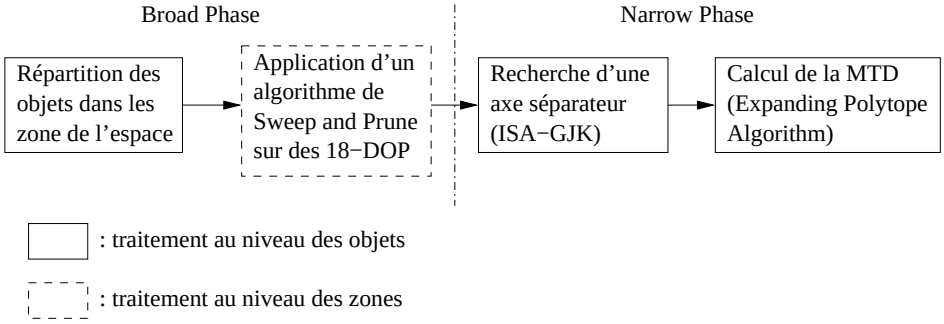


FIG. 4 – Pipeline de traitement de collisions.

### 3.1.1 Broad Phase

Cette première partie consiste à éliminer rapidement les couples d'objets qui ne sont pas en collision. Pour cela, 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 *broad phase* 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 *broad phase* lorsque le nombre d'objets est elevé (plus d'une centaine d'objets) est le *Sweep And Prune* [CLMP95]. Cependant, il est couramment utilisé avec des AABB ce qui génère un nombre trop important de collisions potentielles (i.e. le rapport collisions potentielles sur collisions exactes est très elevé). 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 [KHM+98, Zac98]. Ces volumes sont des extensions de *AABB* à $k/2$ axes (on peut considérer les *AABB* 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 qui allie une assez bonne approximation de l'objet et dont la détection est assez rapide. 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 : 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 *broad phase* rapide et qui génère beaucoup moins de collisions potentielles que l'algorithme classique basé sur des AABB. En sortie de cet étage du pipeline, 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 5).

### 3.1.2 Narrow Phase

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* [vdB98]. *ISA-GJK* (*ISA* pour *Incremantal 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 meilleures performances que celui de Lin-Canny [LC91].
Si aucun axe séparateur n'a été trouvé, la dernière étape du framework est activée. La distance d'interpénétration est calculée en utilisant l'algorithme de van den Bergen [vdB01] qui par raffinement progressif d'une approximation de la différence de Minkowski peut déterminer une estimation de la *MTD*. 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.

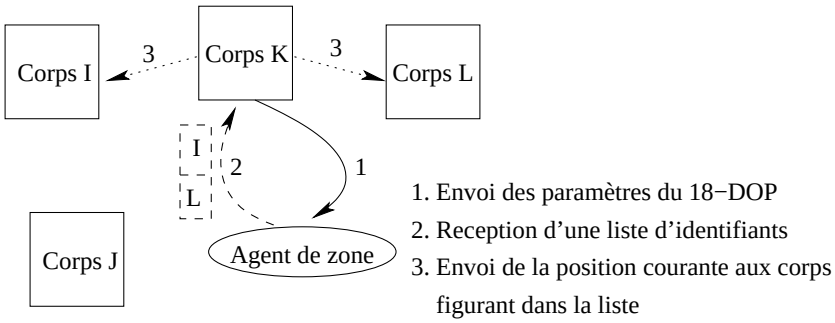


FIG. 5 – Informations échangées lors de la détection de collisions.

## 3.2 Résultats

La méthode proposée est naturelle et cohérente dans l'enchaînement des différentes étapes. Notre *Broad Phase* 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 permettent 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 framework, 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 6 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).

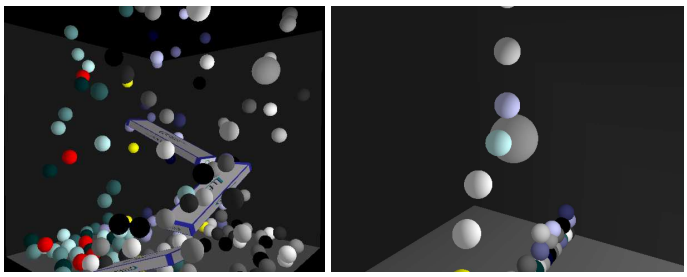

FIG. 6 – Simulation de corps rigides convexes.

Une autre particularité de notre framework 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 framework 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 résoud lui même ses collisions, ses équations de mouvement . . . ) et donc d'engendrer un simulateur complétement 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 [FL01] 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 [OS88]. Une autre possibilité est de sélectionner les facettes potentiellement en collision (voir [JSL99]).

Ce framework 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 interactions, la gestion de 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).

# Références


[Cam97] S. Cameron. Enhancing GJK : Computing minimum and pentration distances between convex polyhedra. In *International Conference on Robotics and Automation*, pages 3112–3117, 1997.

[Chu96] K. Chung. *An Efficient Collision Detection Algorithm for Polytopes in Virtual Environment*. PhD thesis, Department of Computer Science, University of Hong Kong, 1996.

[CLMP95] J. Cohen, M.C. Lin, D. Manocha, and M.K. Ponamgi. I-collide : An interactive and exact collision detection system for large-scale environments. In *ACM interactive 3D Graphics Conference*, pages 189–196, 1995.

[FL01] S. Fisher and M.C. Lin. Fast penetration depth estimation for elastic bodies using deformed distance field. In *Intelligent Robots and Systems*, 2001.

[GJK88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. In *IEEE Journal of Robotics and Automation*, volume RA-4, pages 193–203, 1988.

[JSL99] A. Joukhadar, A. Scheuer, and C. Laugier. Fast contact detection between moving deformable polyhedra. In *IEEE-RSJ Intelligent Robots and Systems*, 1999.

[JTT01] P. Jiménez, F. Thomas, and C. Torras. 3D collision detection : A survey. In *Computer Graphics*, volume 25, pages 269–285, 2001.

[KHM+98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. In *T-VCG(4)*, pages 21–36, 1998.

[KLM02] Y.J. Kim, M.C. Lin, and D. Manocha. Deep : Dual-space expansion for estimating penetration depth between convex polytopes. In *IEEE International Conference on Robotics and Automation*, Mai 2002.

[LC91] M.C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation*, 1991.

[LG98] M.C. Lin and S. Gottschalk. Collision detection between geometric models : A survey. In *IMA Conference on Mathematics of Surfaces*, 1998.

[LMCG96] M.C. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision Detection : Algorithms and Applications. In *Algorithms for robotics motion and manipulation*, pages 129–142, 1996.

[Mes02] P. Meseure. *Animation basée sur la physique pour les environnements intéractifs temps réel : habilitation à diriger des recherches*. Université des Sciences et Technologies de Lille, 2002.

[OS88] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed : Algorithms based on Hamilton-Jacobi formulations. In *Journal of Computational Physics*, volume 79, pages 12–49, 1988.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Fast continuous collision detection between rigid bodies. In *Eurographics*, 2002.

[vdB98] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. In *Journal of Graphic Tools*, volume 4(2), pages 7–25, 1998.

[vdB01] G. van den Bergen. Proximity queries and penetration depth computation on 3D game object. In *Game Developers Conference*, 2001.

[Zac98] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proceedings of IEEE, VRAIS*, 1998.

[Zac01] G. Zachmann. Optimizing the collision detection pipeline. In *First International Game Technology Conference (GTEC)*, 2001.