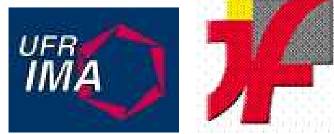
UFR IMA - Université Joseph Fourier



Année Universitaire 2002 / 2003

MÉMOIRE DE MAGISTÈRE 2

DÉTECTION EN TEMPS RÉEL D'AUTO-COLLISIONS SUR DES OBJETS HAUTEMENT DÉFORMABLES

Projet EVASION - Laboratoire GRAVIR

par Matthieu Nesme
tuteur : François Faure

Table des matières

1 Introduction **3**

1.1 Le contexte du stage 3

1.1.1 Le problème abordé 3

2 Existant sur la détection de collision **5**

2.1 Complexité 5

2.2 Volumes englobants 5

2.2.1 Les volumes englobants « classiques » 6

2.2.2 Les k-DOP orientés et gonflés 6

2.3 Hiérarchies 8

2.3.1 Partionnement des objets 9

2.3.2 Partitionnement de l'espace 10

2.3.3 Limitations 10

2.4 Sweep and prune 11

2.5 Auto-collisions de surfaces 11

2.6 Analyse globale de l'intersection 12

2.7 Particules sensitives 12

2.7.1 Présentation 12

2.7.2 Limitations 13

2.8 Utilisation de la cohérence temporelle et de la recherche stochastique 13

2.8.1 Cohérence temporelle 13

2.8.2 Recherche stochastique 14

2.8.3 Algorithme 14

3 Ma contribution **17**

3.1 Utilisation de modèles X3D 17

3.1.1 Le format X3D 17

3.1.2 Intérêt 17

3.2 Détection d'auto-collisions 21

3.2.1 La méthode choisie 21

3.2.2 Sa réalisation 21

3.3 Réaction aux collisions 25

4 Résultats et perspectives **26**

4.1 Résultats 26

4.1.1 Premier essai 26

4.1.2 Second essai 31

4.2 Travaux restants 31

4.2.1 Réglages des paramètres 31

4.2.2 Intégration de la méthode stochastique dans une hierarchie de volumes englobants 31

4.2.3 Vers un simulateur 32

5 Acquisitions personnelles **33**

A Système masses-ressorts **34**
A.1 Domaines d'application des systèmes masses-ressorts 34
A.2 Schéma d'intégration 35

B Réaction aux collisions par correction de position et de vitesse **36**
B.1 Collision arête / arête 36
B.1.1 Mise à jour des vitesses 36
B.1.2 Mise à jour des positions 37
B.2 Collision point / triangle 37
B.2.1 Mise à jour des vitesses 38
B.2.2 Mise à jour des positions 38

Chapitre 1

Introduction

1.1 Le contexte du stage

Mon stage de magistère 2 se déroule dans le laboratoire GRAVIR, au sein de l'équipe EVASION dans les locaux de l'INRIA à Monbonnot, sous la tutelle de François Faure.

Le laboratoire GRAVIR (GRAphisme Vision et Robotique) travaille sur l'infographie, la vision et la robotique. Il est commun au CNRS, à l'INPG, à l'INRIA et l'UJF et fait partie de l'IMAG.

Le projet EVASION (Environnements Virtuels pour l'Animation et la Synthèse d'Images d'Objets Naturels - anciennement iMAGIS) s'intéresse plus particulièrement à l'informatique graphique et la synthèse d'images. Ses principaux thèmes de recherche portent sur la simulation graphique de scènes naturelles végétales, animales et minérales. Les domaines d'application vont des simulateurs médicaux aux jeux vidéo, en passant par la réalité virtuelle et la visualisation scientifique.

1.1.1 Le problème abordé

1.1.1.1 Les collisions

Le domaine de l'animation par ordinateur utilise de plus en plus des techniques de simulation physique pour augmenter le réalisme des scènes rendues. Nous n'avons qu'à observer le domaine du divertissement visuel pour s'apercevoir de la qualité des effets devenus possibles. En effet, les interactions entre plusieurs objets peuvent se faire d'une manière très convaincante, et apparaissent plausibles. Ces interactions doivent être produites par une gestion rigoureuse des collisions entre l'objet et l'environnement qui l'entoure.

Grâce à l'algorithme 1 qui représente la boucle de base d'un simulateur, on peut voir où se situe le traitement des collisions :

Algorithme 1 Algorithme de base d'une simulation

```
- initialisation
- répéter :
    - affichage
    - gestion des entrées du sytème
    - gestion des déplacements, des déformations
    - traitement des collisions
```

1.1.1.2 Les auto-collisions

Dans le cas particulier d'un objet déformable, il peut y avoir en plus des collisions de l'objet sur lui-même (cf figure 1.1).

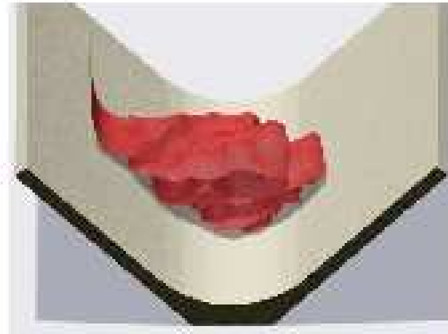


FIG. 1.1 – Exemple d'auto-collisions d'un ruban de tissu replié sur lui-même

Le sujet du stage consiste à détecter les collisions d'un objet très déformable sur lui-même, nous les appelons auto-collisions. Le véritable enjeu est de le faire en temps réel, c'est à dire que l'utilisateur puisse influencer la simulation et observer les changements en même temps, rendant la simulation interactive. Les algorithmes de détection de collisions actuels peuvent permettre de résoudre les auto-collisions, mais dans une durée très longue ; afin de respecter la contrainte du temps réel, les calculs doivent donc être fortement minimisés.

Chapitre 2

Existant sur la détection de collision

En matière de détection de collisions, il existe beaucoup d'algorithmes [6, 7, 10]. On ne verra ici que les grands principes qu'utilisent ces algorithmes, en précisant s'ils sont applicables efficacement dans le cas d'objets déformables.

Les objets auxquels nous nous intéressons sont modélisés par des polygones, nous les considérons composés d' « éléments » (ces éléments pouvant être les arêtes, les faces, des formes géométriques simples). Détecter les collisions revient à trouver les intersections entre ces éléments.

2.1 Complexité

Afin de bien montrer la difficulté de la détection des collisions entre n éléments dans une scène, on peut regarder l'algorithme naïf de détection.

Dans le cas général, chaque élément est susceptible de rentrer en collision avec chaque autre élément. Donc si l'on teste explicitement la collision entre toutes les paires d'éléments de la scène, on est sûr de trouver toutes les collisions effectives entre les éléments. Le nombre de paires à tester est exactement $\frac{n\times(n-1)}{2}$ et donc on doit tester $O(n^2)$ collisions potentielles.

Un objet déformable est généralement représenté par un grand nombre de polygones, et donc en utilisant cette méthode, le nombre de paires d'éléments à tester à chaque pas d'animation est beaucoup trop grand pour que la simulation soit en temps réel. Des algorithmes destinés à accélérer ces détections ont été crées. Ils utilisent pour la plupart des hiérarchies leur permettant dans le cas moyen de ne tester que $O(n \times \log(n))$ collisions.

2.2 Volumes englobants

Il est possible de réduire la complexité des tests de collision. L'idée est de placer chaque objet dans une forme géométrique simple, dont la détection de collision est quasi immédiate (par exemple une sphère où il suffit de tester si la distance entre les deux sphères est inférieure à la somme de leur rayon - cf figure 2.1). Les éléments dont leur boites ne sont pas en collision ne peuvent être en collision. On cherche à éliminer les calculs inutiles, la complexité reste tout de même en $O(n^2)$.

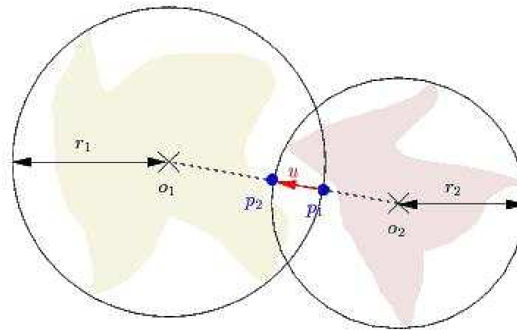
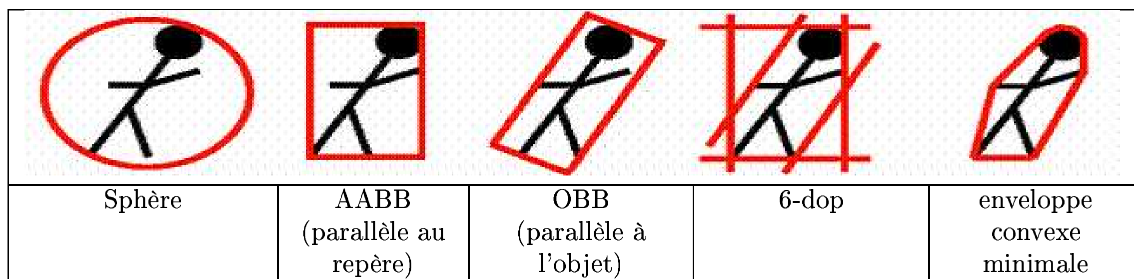


FIG. 2.1 – Collision sphère-sphère

2.2.1 Les volumes englobants « classiques »

Pour réduire au maximum les temps de calcul, il faut le meilleur compromis entre la compacité du volume englobant, pour éliminer au mieux les collisions impossibles, et la simplicite du test d'intersection entre deux volumes (figure 2.2).

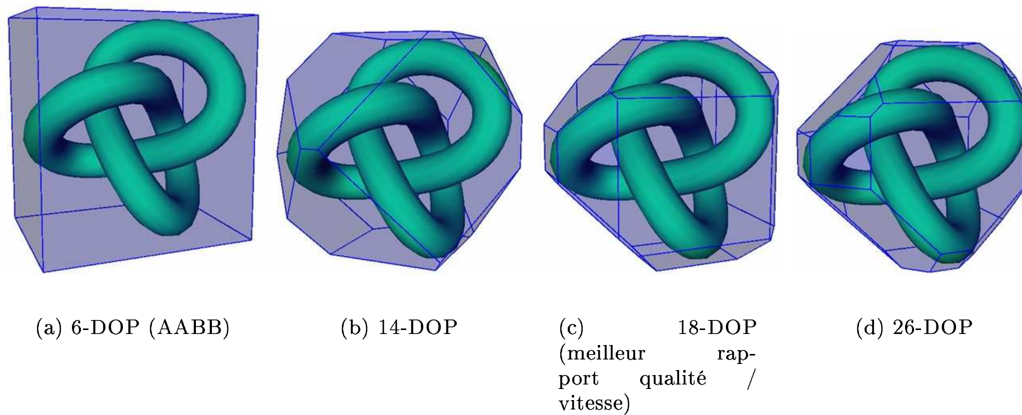


Sphère	AABB (parallèle au repère)	OBB (parallèle à l'objet)	6-dop	enveloppe convexe minimale

FIG. 2.2 – Différents types de volumes englobants

2.2.2 Les k-DOP orientés et gonflés

Un type de volume englobant qui nous intéresse particulièrement est le k-DOP (cf figure 2.3), car il a déjà été utilisé pour faire de la détection de collision sur des vêtements (objets très déformables) dans [11].



(a) 6-DOP (AABB) (b) 14-DOP (c) 18-DOP (meilleur rapport qualité / vitesse) (d) 26-DOP

FIG. 2.3 – Différents k-DOP ou polyèdres convexes

Dans cette méthode, le volume englobant est un k-DOP gonflé par rapport au k-DOP minimal couvrant l'objet, ensuite il est orienté dans la direction du mouvement de l'objet qu'il englobe (cf figure 2.4). Le but de ces deux manipulations étant de laisser une « marge de détection » et ne pas oublier une collision qui se déroulerait entre deux pas de temps. En effet, la discrétisation du temps pose un problème qui est que entre deux pas de temps, on ne peut rien analyser ; il faut donc des pas de temps les plus petits possibles, mais plus ils sont petits, plus il faut faire de calcul, il faut donc trouver le bon compromis entre bonne discrétisation du temps et temps réel.

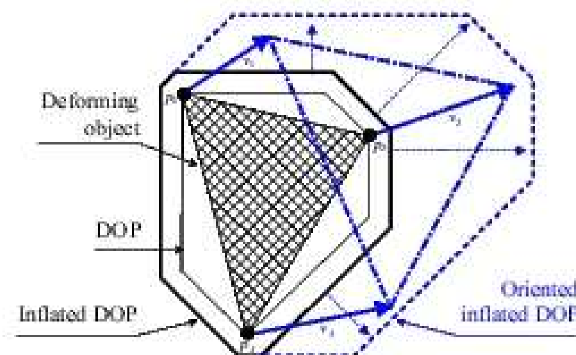
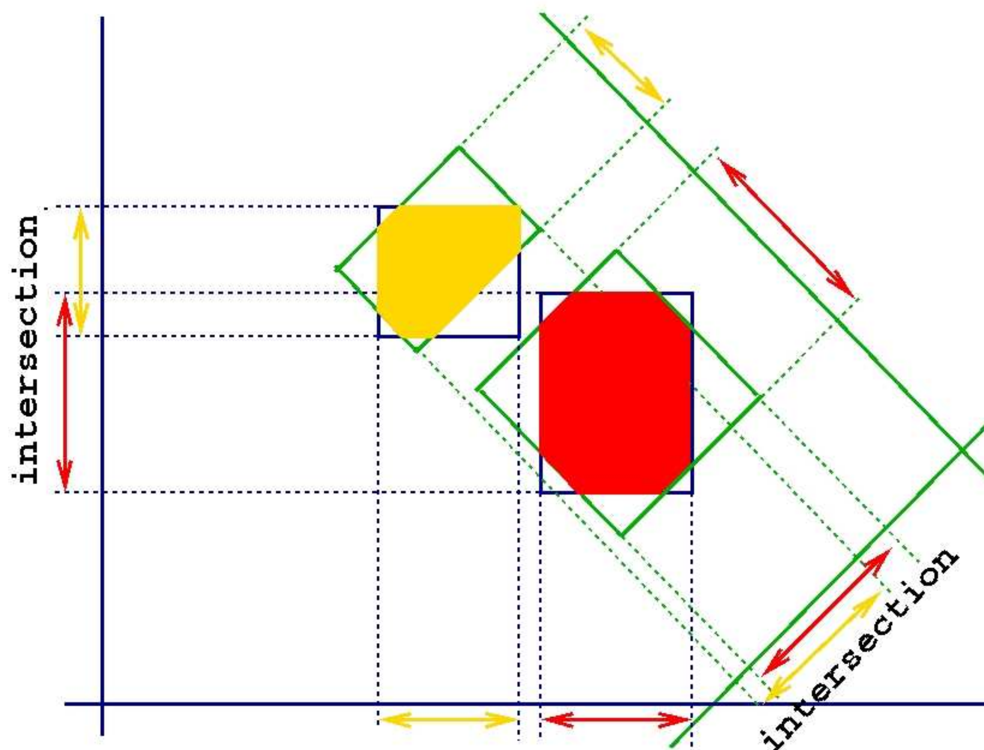


FIG. 2.4 – Estimation du mouvement et inflation orientée d'un 8-DOP

Grâce à sa « marge de détection », cette méthode obtient de bons résultats dans la détection des collisions, avec une certaine efficacité, car le calcul d'intersection entre kDOP est treès rapide (cf figure 2.5), mais son temps de traitement reste encore trop long pour être utilisable en temps réel.



il suffit de tester si les projections orthogonales des kDOP dans tous les repères s'intersectent

FIG. 2.5 – Le test d'intersection entre kDOP

Il serait intéressant de savoir si la qualité des détections de cette méthode sont concervables en accélérant son temps de calcul.

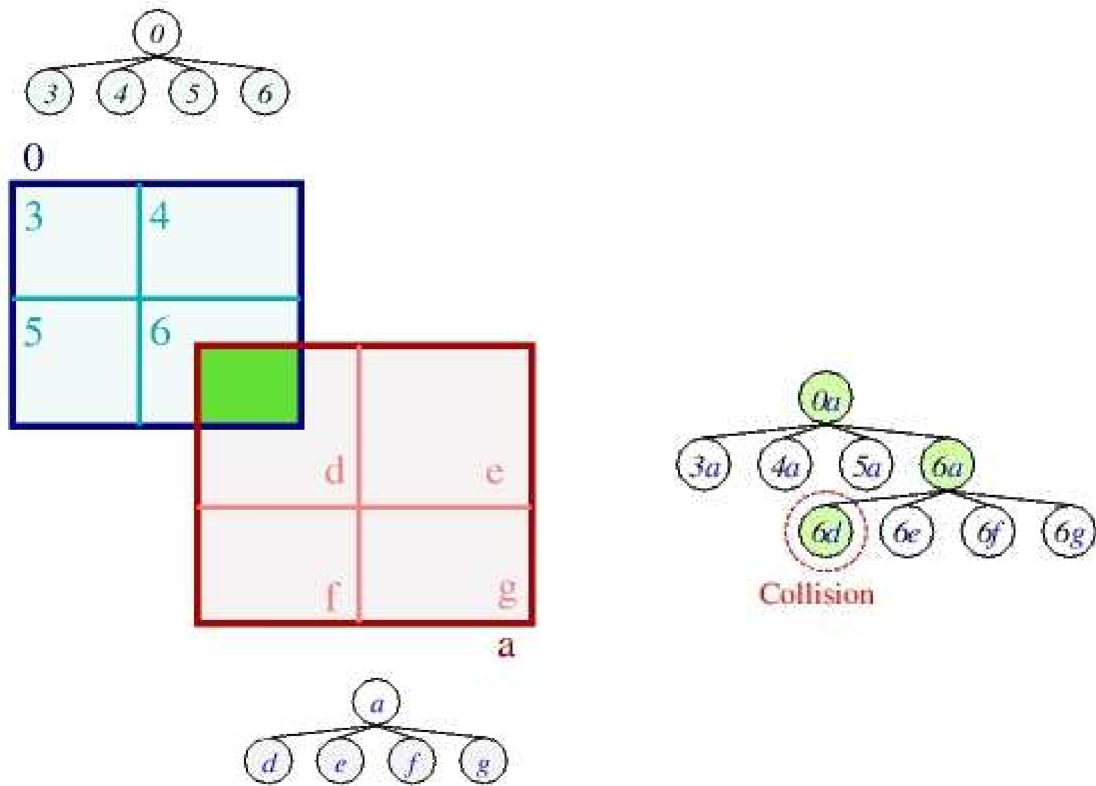
2.3 Hiérarchies

Une hiérarchie permet de regrouper dans un même ensemble les éléments géographiquement proches. Le but étant de faire des tests de collision sur des volumes englobants pour éliminer par bloc des ensembles de paire d'éléments qui ne peuvent pas être en collision.

On commence par tester entre les plus gros volumes, s'il y a collisions, on descend d'un niveau et on itère.

Un élément important dans ce type d'algorithme est le choix de la construction de l'arbre représentant la hiérarchie, et l'algorithme de parcours de celui-ci.

La construction d'une hiérarchie, en précalcul, est $O(n)$, ensuite son parcours afin de détecter les collisions est $O(n \times \log(n))$.



(a) Deux hiérarchies représentées par des arbres quaternaires

(b) Les tests effectués entre les volumes englobants pour détecter la collision

FIG. 2.6 – Exemple de représentation hiérarchique avec un arbre quaternaire et de parcours en cas de collision

2.3.1 Partionnement des objets

On peut diviser les objets en plusieurs volumes englobants en créant une hiérarchie où la boite la plus grande englobe l'objet entièrement, et on descend jusqu'à l'encapsulation des éléments formant la scène (figure 2.7).

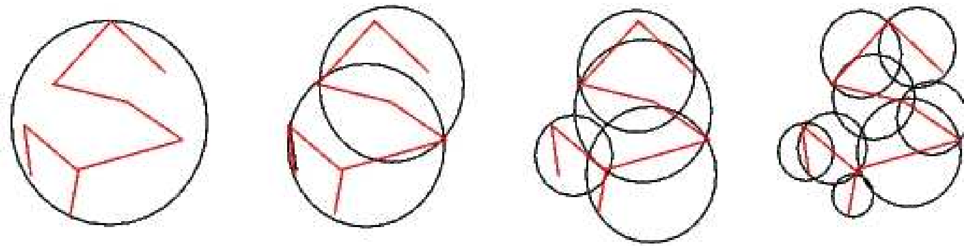


FIG. 2.7 – Exemple d'une hiérarchie de sphères

On commence donc par tester sur la globalité de l'objet, et on raffine de plus en plus seulement là où il y a des collisions.

2.3.2 Partitionnement de l'espace

En partitionnant l'espace, les volumes englobants ont une position et une dimension fixe dans l'espace. On obtient la hiérarchie en divisant l'espace par une grille de plus en plus fine (figure 2.8). Les collisions entre des éléments inclus dans des volumes non adjacents ne peuvent pas avoir lieu, on évite alors de tester la collision entre de nombreux blocs de paires d'éléments.

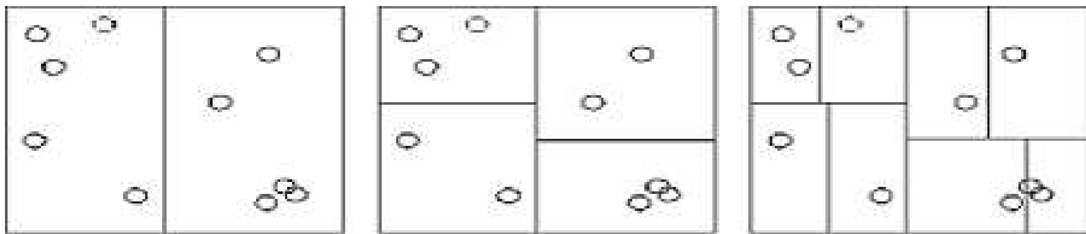


FIG. 2.8 – Exemple de partitionnement en 2d-tree

Là encore, il existe plusieurs forme de découpage (figure 2.9) ; chacune de ces décomposition a des avantages et des inconvénients : plus de précision pour certaines mais plus de difficulté à générer.

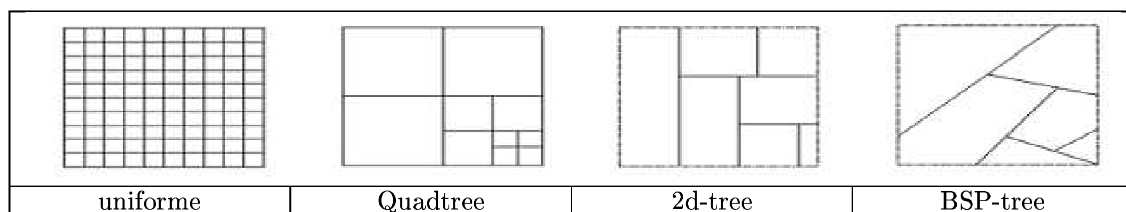


FIG. 2.9 – Les différents types de partitionnement de l'espace

2.3.3 Limitations

Bien que cette méthode réduise la complexité moyenne, elle ne peut pas être appliquée efficacement dans notre cas, car la mise à jour de cette hiérarchie prendrait trop de temps, il faudrait énormément de volume englobants pour recouvrir toute la surface. De plus, les performances de cet algorithme se dégradent lorsqu'il y a des collisions, puisque le nombre de feuilles à parcourir

augmente avec le rapprochement de deux objets. Dans le cas d'objets déformables, son traitement serait trop coûteux à cause de leurs formes repliées potentielles, car il peut y avoir énormément d'éléments dans un même endroit spatial.

2.4 Sweep and prune

Une alternative aux hiérarchies est la méthode décrite dans [16], particulièrement efficace pour un grand nombre d'objets solides. Son intérêt est que son efficacité ne dépend pas des la tailles des objets, qui peut être très hétéroclites.

Son principe est de projeter des volumes englobants orientés sur les axes, sur tous les axes (cf figure 2.10). Les intervalles ainsi trouvés sont triés par permutation, qui tend vers du $O(n)$, grâce à la cohérence temporelle qui assure qu'il y aura peu de changements d'ordre.

L'idée très intéressante se situe justement quand il y a une permutation, car c'est seulement dans ce cas, qu'il y a risque de collision, donc que l'on va tester explicitement les boites englobantes.

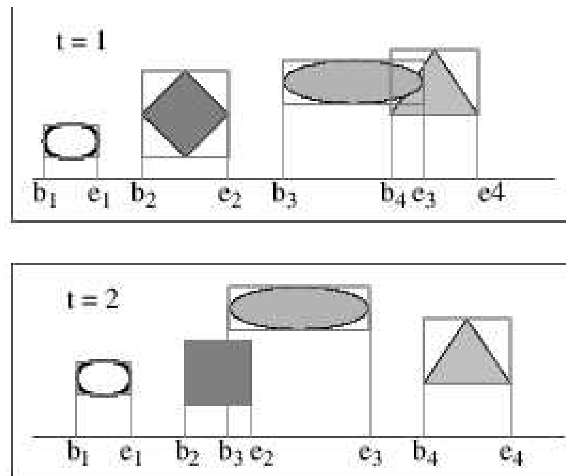


FIG. 2.10 – Projection des volumes englobants sur un axe

2.5 Auto-collisions de surfaces

Des algorithmes utilisant les propriétés d'adjacence des éléments d'une surface ont été développés par Pascal Volino et Nadia Magnenat Thalmann dans [8, 9]. Ils exploitent le fait que sur une surface, il existe beaucoup de zones où la surface est localement à peu près aplatie. C'est à dire que l'on peut trouver un plan d'une certaine orientation sur lequel les éléments de la zone peuvent se projeter sans que leur projection ne se recouvre. Au sein de ces zones, il est inutile de chercher à faire une recherche de collisions entre les éléments de la surface. Leur algorithme permet de trouver de manière rapide de telles zones dans la surface, et construisent une hiérarchie de zones regroupant les différentes orientations des plans sur lesquels les zones peuvent se projeter sans se recouvrir.

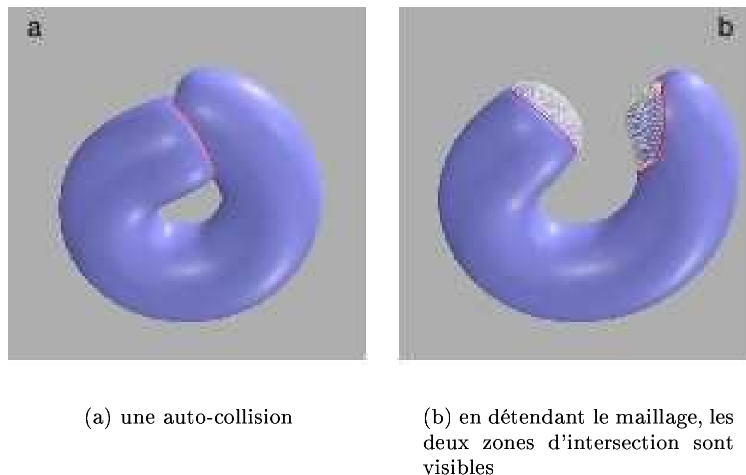
Cet algorithme est intéressant car il apparaît être rapide, mais il se peut qu'il soit inefficace dans un grand nombre de cas où les objets possèdent beaucoup d'angulations ; l'algorithme est plus efficace sur des surfaces lisses, pas trop chiffonnées. Toutefois, il serait bon de l'essayer pour mieux l'estimer.

2.6 Analyse globale de l'intersection

La méthode GIA (Global Intersection Analysis) est décrite dans [12], où elle est appliquée pour des simulations de vêtements, donc particulièrement adaptée aux auto-collisions. C'est un méthode très lente mais qui assure une détection très complète permettant ensuite un très bon traitement de la réponse aux collisions. De plus, si des collisions s'effectuent entre deux pas de temps, cette méthode permet de les prendre en compte et de pouvoir les traiter efficacement après coup.

La première étape consiste à détecter la ligne d'intersection entre deux maillages. Pour cela, il faut chercher une paire d'éléments qui s'intersectent avec une méthode « classique » de hierarchie de volumes englobants. A partir de cette premiere paire en collision choisie arbitrairement, on en découvre une autre parmi ses voisines, et ainsi de suite jusqu'à avoir une courbe complète d'intersections.

Le seconde étape consite à différencier les éléments à l'intérieur et ceux à l'extérieur du contour. La plus petite partie est considérée comme celle étant à l'intérieur, parce que si les vitesses de déplacements des objets et la durée du pas de temps sont bien choisies, cela sera toujours le cas. C'est cette seconde étape qui prend du temps.



(a) une auto-collision

(b) en détendant le maillage, les deux zones d'intersection sont visibles

FIG. 2.11 – Analyse globale de l'intersection

Il subsite tout de même un problème lors d'une auto-collision particulière lorsque il n'y a pas deux zones en intersection, mais seulement une zone (figure 2.12). Pour l'instant, ce cas n'est pas traité ; n'étant pas courant, les seules forces exercées sur les autres éléments suffisent à donner une impression réelle.

Ce type de détection de collision est très bon pour assurer un rendu le plus réaliste possible, mais il prend trop de temps pour être utilisé dans une simulation temps réel.

2.7 Particules sensitives

2.7.1 Présentation

Cette méthode décrite dans [15] très récente est particulièrement intéressante, car l'idée est nouvelle et surtout, elle a des similitudes avec notre méthode décrite plus tard.

Le principe est de faire evoluer des particules sur les objets. Ces particules agissant comme des charges électriques, un objet est chargé négativement et l'autre positivement. La position de ces particules évoluent sur les objets, les mêmes charges se repoussant, pour recouvrir un maximum

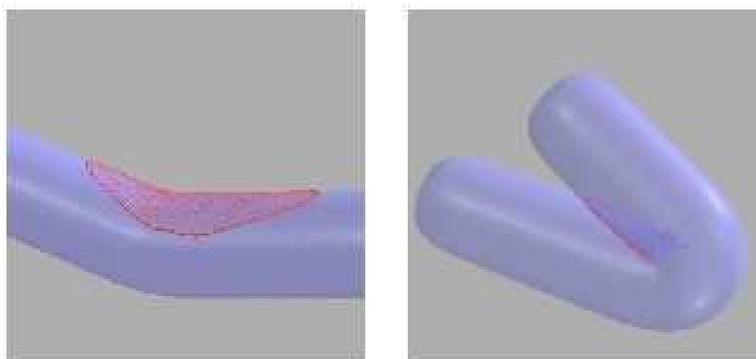
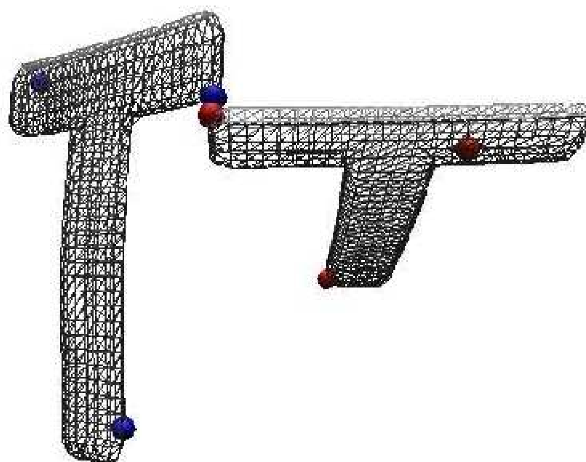


FIG. 2.12 – Cas défavorable au GIA (une seule zone d'intersection)

de l'objet, et les charges opposées s'attirant pour detecter une collision quand elles se touchent (cd figure 2.13).



Un objet est « protégé » par des particules bleues, et l'autre par des rouges.

FIG. 2.13 – Une collision est détectée

2.7.2 Limitations

Cette méthode en est vraiment à son premier stade, et pour l'instant elle fonctionne à la même vitesse que les méthodes classiques en hiérarchie. Son gros défaut pour nous, est qu'elle n'est pas applicable pour détecter des auto-collisions puisque un objet ne comporte que des particules de même charge.

Il sera intéressant de voir l'évolution de cette méthode.

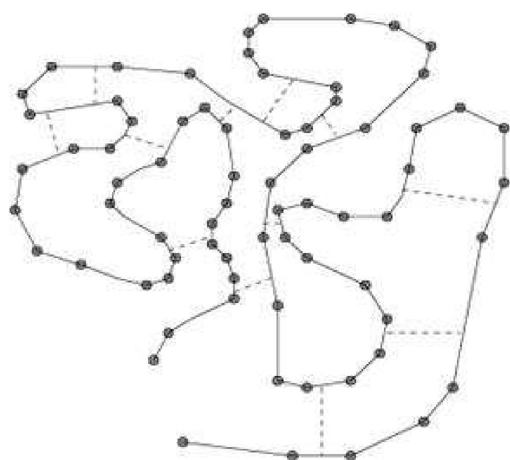
2.8 Utilisation de la cohérence temporelle et de la recherche stochastique

2.8.1 Cohérence temporelle

Le travail auquel nous nous sommes le plus intéressés est celui de Gilles Debunne [2], car il utilise la cohérence temporelle pour accélérer la détection des collisions. L'idée est que si deux

éléments de la scène sont proches à un instant t, ils le resteront plus ou moins à l'instant t+dt. Cette supposition s'appuie sur le fait que les déplacements des éléments dans une simulation réaliste se font de manière continue, et que si l'on fait la supposition que la vitesse de ces éléments est bornée, alors leur déplacement pendant un pas de temps l'est aussi. La notion d'élément proche est donc quantifiable.

Détecter les collisions revient à trouver les minima de distance dans l'ensemble des paires d'éléments. Grâce à la cohérence temporelle, on peut dire que les emplacements de ces minima ne varient pas beaucoup au cours du temps, et donc que d'un pas de temps sur l'autre on peut les retrouver dans le voisinage des anciens minima. C'est pourquoi Gilles Debunne a proposé de faire une recherche des collisions par parcours dans le voisinage des paires d'éléments des anciens minima d'un pas de temps à un autre, en se dirigeant vers les distances les plus courtes pour converger vers les paires d'éléments en collision. Ainsi les seules paires en collision possibles sont des voisines aux minima de distance du pas de temps précédent.



S'il doit y avoir des collisions, les voisines aux paires en pointillés (les minima locaux de distance) seront les premières en collision (figure 2.14).

FIG. 2.14 – Représentation des minima locaux sur un objet déformable

2.8.2 Recherche stochastique

Un autre aspect intéressant du travail de Gilles Debunne dans [2, 3] est la recherche stochastique des collisions. Son approche tire au hasard des paires d'éléments dans la scène à chaque pas d'animation, et, au cours du temps, les fait converger vers les minima de distance en suivant un parcours parmi les paires voisines.

Cette approche ne garantit pas de pouvoir trouver toutes les collisions entre les éléments de la scène, mais assure que la majeure partie des collisions sera trouvée si l'on tire un nombre suffisant de paires à chaque instant.

Dans le cas de notre simulation, nous pensons que cette méthode est assez bonne, car elle ne souffrira pas comme les autres de la complexité possible de la forme des objets déformables tels des vêtements ou des tissus organiques. Cette méthode gardera une trace des éléments les plus proches, en un temps linéaire en fonction du nombre de paires en collision potentielle qu'il entretient à un instant.

2.8.3 Algorithme

Cette méthode a été utilisée dans [1] pour faire une simulation d'organes digestifs humains. Les tissus humains étant des objets très déformables, où les nombre d'auto-collisions est très important, cette méthode est celle à developper pour notre cas.

2.8.3.1 Présentation de la méthode

Ellle utilise à la fois la cohérence temporelle et la recherche stochastique. On recherche donc au hasard les minima de distance susceptible d'entrer en collisions, tout en conservant leur évolution au cours du temps :

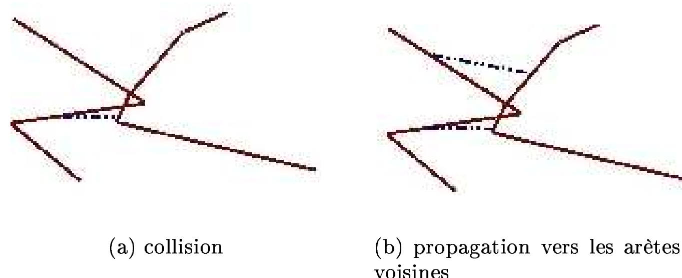
Algorithme 2 Méthode stochastique avec utilisation de la cohérence temporelle

```
pour chaque pas de temps
    on tire k paires d'arêtes
    pour chaque paires
        converger vers minimum local
        si collision
            propager
        sinon si trop eloigné ou existe déjà
            effacer
```

On entretient une liste de paires d'arêtes assez proches pour être en collisions dans un futur proche. On place dans cette liste les paires d'arêtes représentant un minimum local assez proche, on retire celles qui sont devenues trop éloignées et on complète par un tirage au hasard.

Il suffit alors de faire converger vers des minima locaux les couples ainsi sauvegardés et de mettre à jour la liste.

Dans le cas où il y a collisions, il y a de fortes chances pour que les arêtes voisines au contact soient elles-mêmes en collisions, c'est pourquoi il y a une propagation vers les arêtes voisines à la collision (cf figure 2.15).



(a) collision (b) propagation vers les arêtes voisines

FIG. 2.15 – Propagation de la collision vers les voisins

Cet algorithme est quasi linéaire, puisqu'on ne fait que mettre à jour la liste de paires actives ($O(n)$) puis une « petite » convergence due à la déformation. Il est donc particulièrement adapté au temps réel.

2.8.3.2 Limitations

Cette méthode ne garantie pas de trouver 100% des collisions, mais un maximum possible ; elle n'est donc pas la plus appropriée pour un réalisme poussé.

Les éléments de base utilisés dans cette détection de collision sont les arêtes, hors il existe des collisions indetectables avec la seule utilisation d'arêtes (cf figure 2.16). Il faut donc étendre la détection aux surfaces.

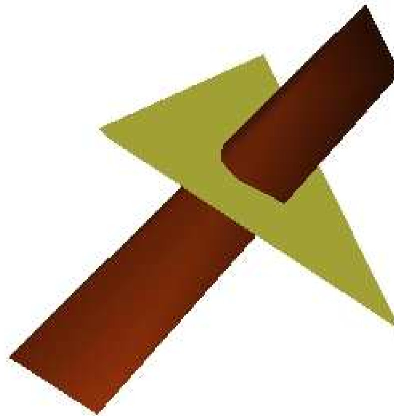


FIG. 2.16 – Collision non détectée si on ne regarde que les intersections arête-arête.

L'enjeu du stage est d'améliorer et d'étendre cette méthode pour des maillages quelconques, et ainsi faire une méthode de détection d'auto-collisions temps réel très générale et s'appliquant dans tous les cas.

Chapitre 3

Ma contribution

La détection des auto-collisions que j'ai fait devait s'intégrer dans la bibliothèque AnimAL (Animation Algorithms Library) développée par François Faure.

Cette librairie a pour but de mutualiser les travaux menés dans l'équipe et de les rendre aisément réutilisables par différentes personnes dans différents contextes. Elle consiste en une collection de classes et algorithmes dédiés à la simulation dynamique et la visualisation graphique de phénomènes complexes. Elle rassemble des classes de base (vecteurs, quaternions, ...), des méthodes d'intégration numérique (euler, Runge-Kutta, euler implicite...), des outils divers, des méthodes expérimentales pour l'animation par modèles physiques, et de nombreux programmes d'exemple dont un système masse-ressorts. Ecrite en C++, elle fait largement appel aux types génériques (template) et à la programmation objet. Elle se veut une boite à outils où chacun puise les éléments nécessaires plutot qu'une plate-forme monolithique.

Dans un premier temps, pour prendre en main la bibliothèque, je lui ai ajouté quelques fonctionnalités. En même temps j'étudiais les méthodes de détection de collisions.

Ensuite, il a fallu ajouter des méthodes de détection de collisions, facilement manipulables par quelqu'un qui utilise la bibliothèque AnimAL, tout en ecrivant du code réutilisable et en laissant la possibilité d'ajouter facilement d'autres méthodes.

3.1 Utilisation de modèles X3D

La première étape de mon développement a consisté à ajouter la manipulation de fichiers X3D dans AnimAL.

3.1.1 Le format X3D

X3D est le format 3D standard de prochaine génération pour le web. Il n'existe pas encore de spécification définitive. C'est un format extensible qui est destiné aussi bien a l'importation qu'à l'exportation de scènes 3D interactives. X3D est le fruit du travail conjoint du "Web 3D Consortium's X3D Task Group" et du "Browser Working Group" et répond aux besoins suivants. Il est complètement compatible avec VRML, qui est le standard actuel du web. Il est extensible, c'est à dire qu'on peut facilement lui ajouter de nouvelles caractéristiques. Il a été défini un noyau X3D, lui permettant ainsi d'être adopté facilement par le maximum de logiciels 3D en import et export.

X3D est en réalité VRML200x c'est a dire VRML97 en XML, on comprend donc pourquoi il est totalement compatible.

3.1.2 Intérêt

La lecture de fichiers X3D permet de pouvoir importer n'importe quel modèle, fait avec n'importe quel logiciel de modelage, et le convertir en système masses-ressorts dans la bibliothèque.

Alors que jusqu'à maintenant, les fichiers représentant un état du système masses-ressorts étaient fait à la main.

3.1.2.1 Extensibilité

L'avantage du format X3D est sa capacité a pouvoir ajouter des nouvelles caractéristiques à tous les éléments composants le fichier. Ainsi pour le système masses-ressorts, il était pratique d'ajouter une masse, une position, et une vitesse aux particules, informations nécessaires pour représenter un état. Par la suite, chaque application utilisant la bibliothèque pourra ajouter ses propres informations (par exemple des informations sur la matière d'un objet).

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D>
  <Scene>
   <StaticGroup>
        <Shape>
             <MassSpring ccw="FALSE" convex="false" solid="false" >
                    <Vertex  point="0 0 0, 1 0 0 , 2 0 0 , 3 0 0 , 4 0 0 , 5 0 0 , 6 0 0 ,7 0 0 ,8 0 0 , 9 0 0,
                                0 1 0, 1 1 0 , 2 1 0 , 3 1 0 , 4 1 0 , 5 1 0 , 6 1 0 ,7 1 0 ,8 1 0 , 9 1 0"
                        mass="0 1000 0 1000 2000 3000 2000 1000 0 0 0 0 100000.5 0 0 0 0 0 0 0"
                        velocity="0.1 0.1 0.1, -0.1 0 0.2"
                    />
                    <Edge vertex="0 1 , 1 2 , 2 3 , 3 4 , 4 5 , 5 6  , 6 7 , 7 8 ,8 9 ,
                                 10 11 , 11 12 , 12 13 , 13 14 , 14 15 , 15 16  , 16 17 , 17 18 , 18 19 ,
                                 0 10 , 1 11 , 2 12 , 3 13 , 4 14 , 5 15 , 6 16 , 7 17 , 8 18 , 9 19"
                              stiffness="5000 20000 10000 30000 7500 5000"
                              dampingRatio="0.1 0.2 0.005 0.15 "
                              length="1 2 3"
                     />
             </MassSpring>
         </Shape>
     </StaticGroup>
  </Scene>
</X3D>
```

FIG. 3.1 – Exemple de fichier X3D pour le système masses-ressorts

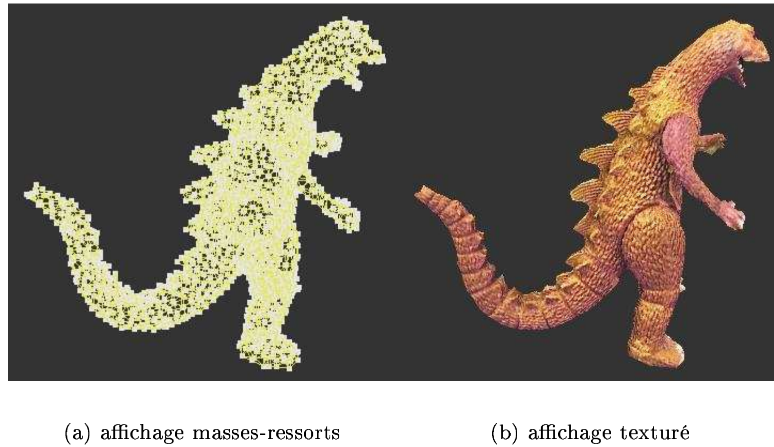
Pour ce faire, j'ai utilisé la bibliothèque X3DToolKit développée par Yann Le Goc ([14]).

Maintenant, l'application est très facilement utilisable avec n'importe quel modèle permettant une multitude de simulations beaucoup plus réalistes.

Dans une même logique, on peut sauvegarder un état des objets simulés dans un fichier X3D.

3.1.2.2 Rendu texturé

Un aspect intéressant de la bibliothèque X3DToolKit est son rendu openGL très complet et optimisé. Elle a donc été utilisée pour l'affichage, car même si la bibliothèque AnimAL dispose de ses propres fonctions d'affichage, elles ne prennent pas en compte, les textures par exemple. Ce système apporte plus d'esthétisme et de réalisme aux simulations (cf figure 3.2).

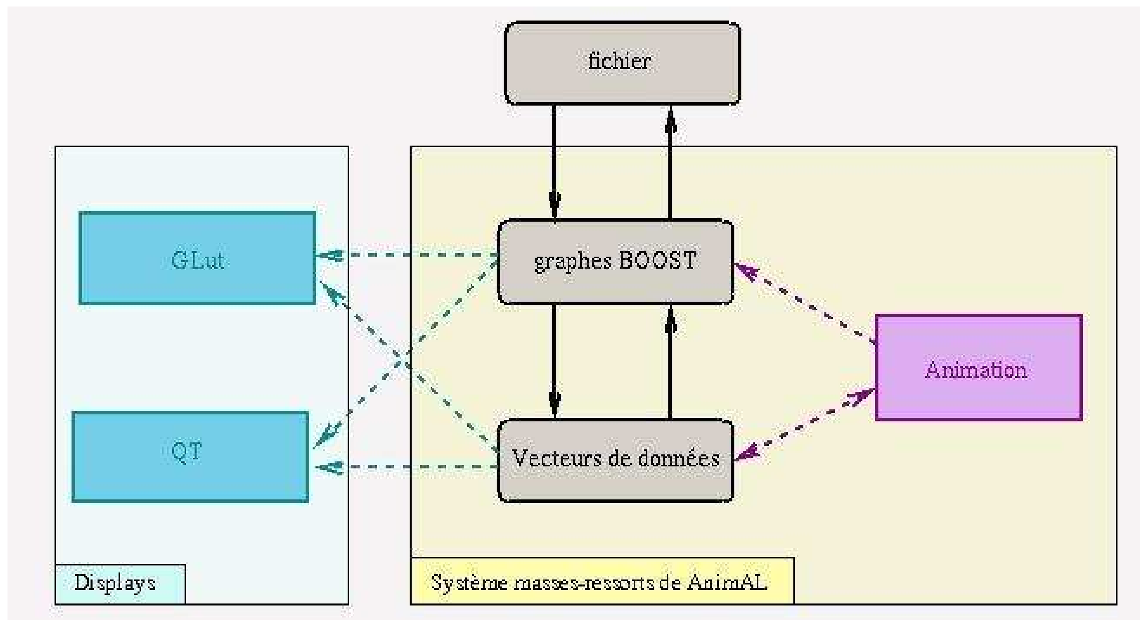


(a) affichage masses-ressorts (b) affichage texturé

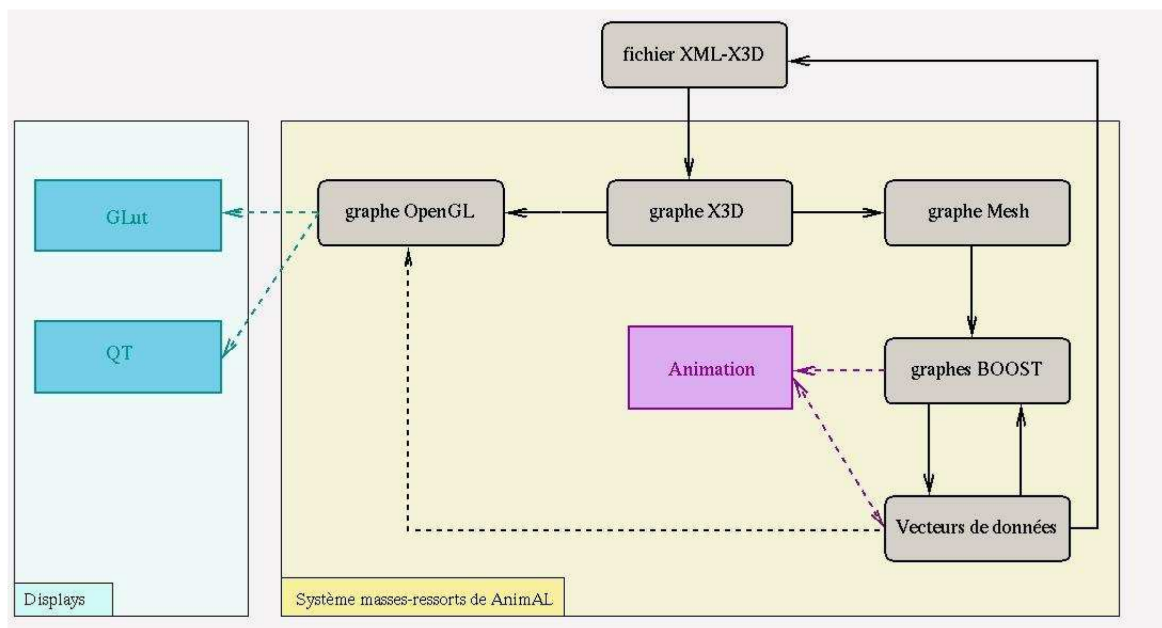
FIG. 3.2 – Visualisation d'un modèle de dinosaure animé par AnimAL

3.1.2.3 Schéma d'architecture logicielle

Dans la figure 3.3, on peut voir l'évolution des données du fichiers X3D, qui sont représentées dans un arbre grâce à la bibliothèque X3DToolKit, puis transformer d'une part dans un arbre OpenGL optimisé pour l'affichage, et dans un arbre représentant les maillages. C'est avec ce derniers qu'on extrait les données nécessaires à l'animation (déplacement, détection de collisions).



(a) architecture initiale



(b) nouvelle architecture

FIG. 3.3 – Architecture des structures de données

3.2 Détection d'auto-collisions

3.2.1 La méthode choisie

L'enjeu du stage est d'améliorer et d'étendre la méthode stochastique (2.8) pour des maillages quelconques, et ainsi faire une méthode de détection d'auto-collisions temps réel très générale et s'appliquant dans tous les cas.

La méthode stochastique s'applique particulièrement bien au temps réel, qui est notre véritable enjeu, car au travers des paramètres, on a un contrôle sur le temps de calcul. Alors que dans une méthode « classique » (hierarchique), il y a de grosses variations de temps de calcul entre le cas où il y a très peu de collisions, et le cas où leur quantité est énorme ; dans la méthode stochastique, le temps de calcul change très peu suivant les cas. Il n'y a donc pas de ralentissements au cours d'une simulation.

De plus, notre intérêt ne réside pas dans le fait de faire une méthode qui détecte la totalité des collisions de façon parfaite, mais de faire une méthode qui en détecte un maximum pour assurer le rendu le plus esthétique possible dans un temps fixé, pour respecter la contrainte du temps réel. Là encore, la méthode stochastique est bien adaptée, car on peut choisir la quantité maximum de collisions que l'on va détecter, pour assurer un temps de calcul maximum ; alors que dans les méthodes classiques, on n'a pas le choix de détecter autre chose que tout.

3.2.2 Sa réalisation

3.2.2.1 L'interface de la bibliothèque

L'algorithme doit s'intégrer dans la bibliothèque AnimAL, en respectant une interface réutilisable pour n'importe quelle méthode de détection de collision, avec un code réutilisable, améliorable et optimisé.

La première étape a donc été de concevoir l'interface de la bibliothèque de détection de collision sur des maillages, en relation avec Stefan Kimmerle de l'université de Tübingen. Celui-ci va implémenter sa méthode en hierarchie k-DOP utilisé pour simuler des vêtements (2.2.2, [11]) dans AnimAL. Nous pourrons ainsi faire des comparaisons de vitesse, de qualité des détections.

3.2.2.2 Les types d'intersection à détecter

Concernant le choix des éléments formant l'objet, on doit choisir une méthode permettant de détecter toutes les collisions de surface.

La première idée est de prendre les triangles formant le maillage, le problème est que le calcul de distance (qui est l'opération de base de l'algorithme) entre deux triangles est beaucoup trop long, même si la detection serait assurément plus efficace.

Une autre manière plus rapide est de détecter les intersections arêtes-triangles, mais là encore la calcul est trop long.

La méthode choisie est d'analyser à la fois les détection arête-arête et les détections sommet-triangle. L'algorithme est plus compliqué puisqu'on a deux détections à gerer en parallèle, mais les coût de calcul sont fortement allégés. De plus, l'implémentation de la réponse aux collisions est facilité, car entre deux arêtes ou un sommet et un triangle, de nombreuses méthodes existent.

3.2.2.3 Algorithme final

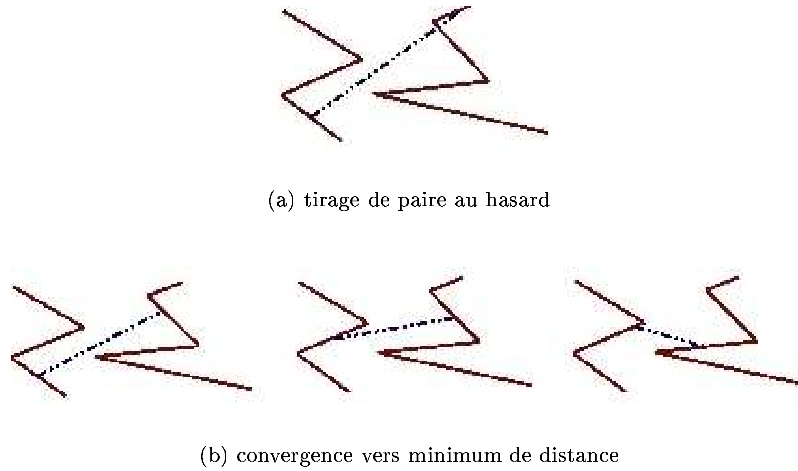
Algorithme 3 Algorithme de détection de collisions utilisant la méthode stochastique

```
Variables :
une liste de paires d'arêtes proches
Une liste de paires d'arêtes en collision
une liste de paires sommet-triangle proches
Une liste de paires sommet-triangle en collision


Pré-calculs :
construction des graphes de voisinage arête-arête et sommet-triangle
(utilisés dans la convergence et la propagation)


A chaque pas de temps :

- Pour chaque paire d'arêtes proches de la liste :
    - La faire converger
        - si trop éloignée ou déjà existante
              la supprimer
          sinon si collision
              l'ajouter dans la liste des paires en collision
              propager la collision aux paires voisine
- Tant qu'on ne dépasse pas le nombre maximum de paires a tirer :
    - tirer au hasard une nouvelle paire d'arêtes
    - la faire converger
    - si trop éloignée ou déjà existante
        la supprimer

- Idem pour des paires sommet-triangle


- Pour toutes les paires d'arêtes en collision
   propager la collision

- Idem pour les paires sommet-triangle en collision
```

3.2.2.4 Les points importants de l'algorithme

1. *le calcul de distance entre éléments* : c'est le calcul de base de notre algorithme, il est donc très important qu'il soit efficace et rapide. Par défaut, il nous font donc le calcul de distance entre deux segments et entre un sommet et un triangle en 3D. Les algorithmes utilisés sont ceux décrits dans [13], car ils sont reconnus comme très performants; de plus leur code est facilement réimplémentable.

2. *la convergence* : c'est la fonction la plus appelée, c'est en quelque sorte elle qui fait « vivre » notre algorithme en entrenant toujours les minimum locaux de distances.
Pour minimiser les appels récursifs, la convergence s'effectue de la manière suivante : on prend la voisine de l'arête 1 la plus proche de l'arête 2, puis on prend la voisine de l'arête 2 la plus proche de l'arête 1, jusqu'à trouver les arêtes les plus proches (cf figure 3.4), on est donc en $O(n)$.



(a) tirage de paire au hasard

(b) convergence vers minimum de distance

FIG. 3.4 – Convergence vers un minima local de distance

3. *la propagation* : lors d'une collision d'une paire, il y aura forcement des paires voisines aussi en collision, l'idée est donc de visiter les paires voisines à la collision. Si on fait un test simple, qui est de tester chaque voisine du premier élement de la paire avec chaque voisins du second élement, celà nous donne du $O(n^2)$ localement qui ralenti énormement l'algorithme.
   Si on teste seulement les voisins du premier élément, puis ceux du second élément avec la paire en collision, on reste en $O(n)$, c'est cette méthode qui est utilisée dans ([1]). Elle a tout de même un gros inconvénient car elle oublie des collisions très fréquentes (cf figure 3.5).

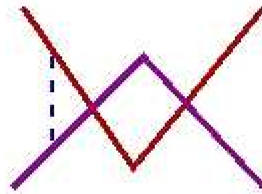


FIG. 3.5 – Collision non détectée par propagation linéaire

3.2.2.5 Optimisation de l'algorithme

Méthode simple :

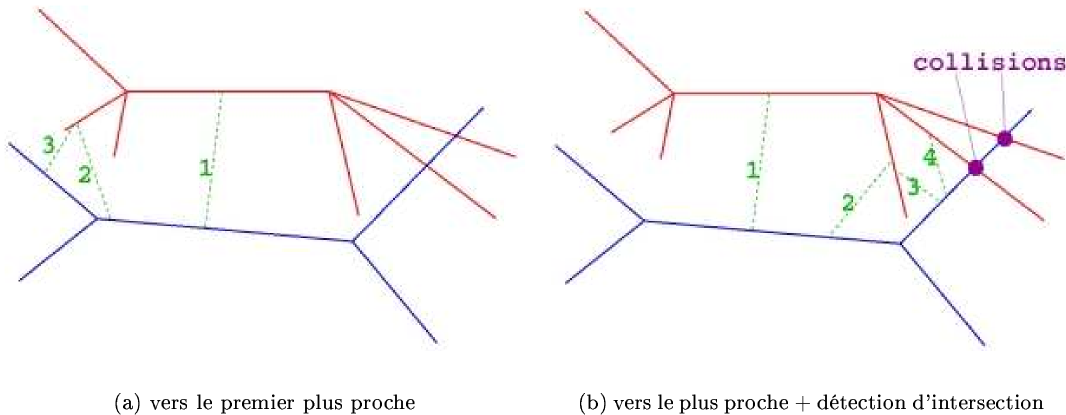
- *convergence* : on converge vers la première paire voisine plus proche non déjà visitée (cf figure 3.6). Pour savoir si une paire à déjà été visitée, toutes les paires parcourues sont stockées dans un hash-set ; une paire déjà visitée est inutile car elle redonnera la même convergence.
- *propagation* : La première solution trouvée est de rester en $O(n^2)$ mais en accélerant le processus, en faisant en sorte de ne jamais retester deux fois les mêmes paires, de la même façon que pour la convergence, en stockant les paires visitées dans un hash-set.
- *analyse* : une légère accélération est notable, mais l'algorithme reste le même, ce n'est pas suffisant.

Convergence durant propagation :

- *convergence* : même convergence que précedemment.

– *propagation* : pour rester en $O(n)$, une bonne idée a été de réutiliser la fonction de convergence lors de la propagation. Ainsi lorsqu'on veut chercher des collisions autour d'une paire en collision, on lance une convergence sur toutes les paires formées par les voisines du premier élément de la collision et le deuxième élément de la collision.
– *analyse* : cette propagation est plus rapide, mais cause un problème, car si un élément est en collision avec deux élements, la convergence ne passera que par une seule de ces deux collisions, oubliant donc une collision très fréquente.

Convergence directe :
– *convergence* : on ne cherche plus la première paire voisine plus proche, mais on recherche la paire non visitée la plus proche parmis les voisines. Ceci implique de tester toutes les voisines, donc on a ajouter, sans perte, un effet de bord, qui est de tester pour chacune de ces voisines, si elle est en collision. On garde une convergence en $O(n)$, mais qui en plus ajoute un grand nombre de collisions (cf figure 3.6).
Comme précedemment, on accelère le traitement en stockant les paires déjà visitées dans un hash-set. Mais en plus, on garde les valeurs de distances déjà calculées au pas de temps courant, car maintenant, on calcule des distances entre les éléments d'une paire sans forcement passer par cette paire.
– *propagation* : même propagation que précedemment.
– *analyse* : ce qui nous assure qu'on détecte un maximum de collisions maintenant, c'est l'effet de bord de la convergence, qui permet de detecter toutes les collisions dans lesquelles intervient un élément.



(a) vers le premier plus proche (b) vers le plus proche + détection d'intersection

la convergence débute avec la paire 1, on commence à chercher un voisin sur l'objet rouge

FIG. 3.6 – Les deux types de convergence (sur les arête en 2D)

Convergence sur les sommets :

Si on regarde les limitations l'algorithme actuel, c'est qu'il effectue trop de calculs de distance, qui reste un calcul pas complétement anodin.

Le calcul de distance entre arêtes est plutôt rapide, le problème est que le nombre de paires arête-arête est très grand, donc qu'il est appelé très souvent.

Dans le cas des tests sommet / triangle le nombre de paires est réduit, mais le calcul de distance est long.

Je pense que le problème vient du fait qu'on a trop assimilé les éléments de convergence avec les éléments dont on veux détecter la collision.

Il faudrait donc converger avec le calcul de distance le plus rapide possible, tout en ayant un minimum de calculs à effectuer, juste pour trouver des zones, pas précises, où il peut y avoir

des collisions. Ensuite, on peut effectuer des tests dans les alentours sur les éléments qui nous intéressent.

Ce qui vient tout de suite à l'esprit est de converger sur les sommets, dont le test de distance est quasi instantané ; et ensuite, tester des collisions entre les éléments désirés. Lors de la convergence, une grande précision de la distance n'étant pas nécessaire, puisque ce qui nous intéresse n'est que de connaître les zones proches, on peut utiliser le calcul de distance de norme 1 (distance de Manatthan) entre deux points : $d(p_1, p_2) = |p_1.x - p_2.x| + |p_1.y - p_2.y| + |p_1.z - p_2.z|$ encore plus rapide à calculer.

De plus, les tests de collision entre les élements peuvent être réduits grâce à des volumes englobants. Par exemple, l'intersection entre triangles reste coûteuse, et pour éviter des calculs inutiles, la non collision peut être détectée par un pré-test sur des kDOP englobants les triangles, car leur construction et leur test d'intersection sont très simples (cf figure 2.5).

> *Remarque* : utiliser les kDOP durant la convergence ne semble pas être une bonne solution, car si leur test d'intersection est très rapide, leur calcul de distance est trop coûteux.

Cette solution a été très rapide à mettre en place grâce aux méthodes qui ont été programmées de façon générique, il a suffit de construire l'objet *sommet* et son calcul de distance.

Il faut tout de même prendre garde à la convergence sur les sommets, car elle impose que les éléments aient des tailles dans le même ordre de grandeur. Sinon, dans le cas de deux triangles de taille totalement différente, chaque sommet des triangles peuvent être éloignés alors que les triangles s'intersectent. Mais cette contrainte n'est pas absurde ; en pratique, il suffirait de limiter l'élongation et la compression des ressorts utilisés pour l'animation.

3.2.2.6 Programmation générique

Dans l'algorithme, la détection sur des paires arête-arête ou sommet-triangle sont basées sur la même logique. L'algorithme a donc été écrit de façon générique, à l'aide de template en c++, où l'on peut choisir les primitives de base entre sommet, arête et triangle. Seules les méthodes de calcul de distance entre ces primitives sont explicitement surgargées.

Une détection de collision arête-triangle a ainsi été aisement ajoutée, il en sera de même si on veut ajouter une détection entre triangles pour effectuer des tests ou encore utiliser des kDOP.

3.3 Réaction aux collisions

Bien que ce n'était pas le but du stage, il était regrétable d'avoir des méthodes qui détectent les collisions, sans avoir véritablement de différence au niveau du rendu. C'est pour cela qu'avec Stefan Kimmerle on a ajouté des méthodes très simples de réaction aux collisions. Le principe de base est de « sortir » les éléments collisionnés, en mettant à jour leur vitesse et leur position sans ajouter, ni retirer d'énergie dans le simulateur (cf anexe B). On peut ainsi créer des simulations physiquement probables sans états impossibles, plus jolies et compréhensibles par l'oeil, mais aussi qui permettent de tester la détection en situation réelle.

Chapitre 4

Résultats et perspectives

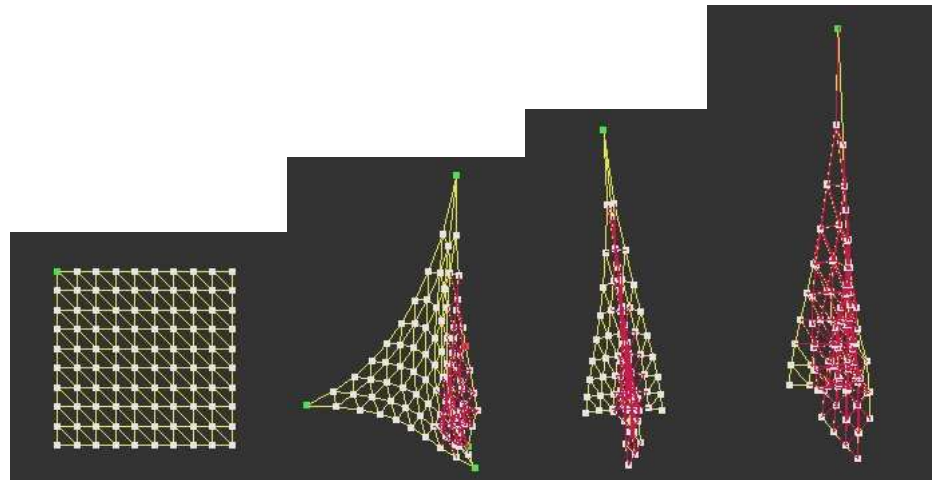
4.1 Résultats

4.1.1 Premier essai

Les différentes implémentations de l'algorithme ont été confrontées (cf 3.2.2.5), ainsi que définies avec des paramètres différentes (cf 4.2.1). Il faut noter que ces tests sont effectués sans aucune réponses aux collisions, permettant des états physiquement impossibles avec des cas très défavorables, et que les paramètres de l'algorithme ne sont pas encore réglés de façon optimale.

Stefan Kimmerle ayant ajouté sa méthode en hiérarchie de kDOP ([11]) dans AnimAL, nous pouvons ainsi comparer nos résultats de façon très précise, car nos détections fonctionnent maintenant dans la même bibliothèque, utilisant les mêmes algorithmes d'animation, les mêmes schémas d'intégrations, les mêmes objets, etc...

Le test présenté ici a été effectué sur un pentium 4 2.4GHz, avec GeForce4 MX et 512 Mo de RAM. L'objet sur lequel détecter les collisions est comme un bout de tissus de taille suspendu par un coin (cf image 4.1).



(a) état inital, sans collision

(b) environ 30 pourcents en collision

(c) environ 50 pourcents en collision

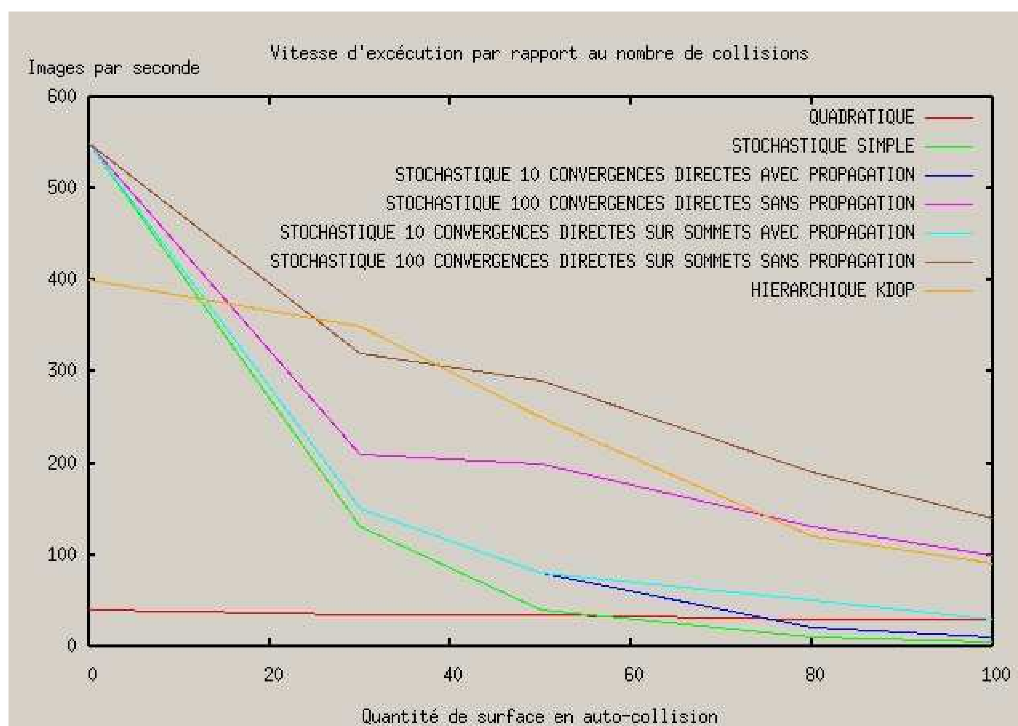
(d) plus de 80 pourcents en collision

Seules les masses et les ressorts sont représentés. Les arêtes en rouges sont celles en collision.

FIG. 4.1 – Animation d'un treillis 10×10 de triangles suspendu par un coin

4.1.1.1 Vitesse

La figure 4.2 nous donne le nombre d'images par secondes affichées pour chaque méthodes en fonctions du pourcentage de l'objet en collision.



quantité de surface en auto-collision	**0%**	**30%**	**50%**	**80%**
algorithme naïf	40	35	35	30
méthode stochastique simple	550	130	40	10
méthode stochastique avec convergence directe (réglage 1)	550	150	80	20
méthode stochastique avec convergence directe (réglage 2)	550	210	200	130
méthode stochastique avec convergence sur les sommets (réglage 1)	550	150	80	50
méthode stochastique avec convergence sur les sommets (réglage 2)	550	320	290	190
hiérarchie de kDOP gonflés et orientés	400	350	250	120

réglage 1 : nombre de paires actives = 10, avec propagation
réglage 2 : nombre de paires actives = 100, sans propagation

Dans ce test, seules les collisions arêtes / arêtes sont détectées car dans la méthodes stochastique, la détection sommet / triangle n'est pas dans la même méthode, ce qui aurait faussé la comparaison avec la méthode hiérarchique.

FIG. 4.2 – Nombre d'images par seconde affichées

Analyse de la première implémentation La détection fonctionne de façon très rapide lorsqu'il y a peu de collisions, mais se dégrade très vite lorsque leur nombre devient trop important. Quand plus d'environ 80% de la totalité de l'objet est en collision, même l'algorithme naïf devient plus rapide. Mais ce cas le plus défavorable est rarissime, voire inexistant en cas réel.

Analyse de la troisième implémentation On voit qu'il y a peu d'améliorations lorsqu'on garde les même réglages que la 1ère implémentation. Par contre, son gros avantage est de pouvoir avoir les deuxièmes réglages, qui assurent un bien meilleure vitesse lorsque le nombre de collisions est important.

Analyse de la convergence sur les points On garde la même logique que la 3ème implémentation avec encore une accéleration notable. Lorsque qu'il n' y pas du tout de collisions, la méthode est lente, car aucune paires actives n'est sauvegardée, et il faut re-effectuer tout le travaille de convergence à la fois sur les points, puis sur les arêtes.

Analyse globale Si on compare les méthodes avec ou sans propagation, on se rend compte que la propagation fait chuter les performances, il y a vraiment de quoi améliorer de ce côté là je pense.

Si on compare nos méthodes avec une méthode plus classique et pourtant très performante en hiérarchie, on se rend compte, qu'on est dans la même rapidité à tous les niveaux, si la méthode est bien réglée. Il est à noter que pour toutes nos implémentations, quand il n'y a pas de collision, et qu'aucunes zones ne sont proches, il faut retirer toutes les paires a chaque pas de temps, et re-effectuer toutes les convergences, dans ce cas là, on tombe dans les 170 images par secondes pour 0%, mais bien sur, dans ce cas là, on peut imaginer ne refaire la totalité des convergences une fois de temps en temps, et tirer très peu de paires le reste du temps.

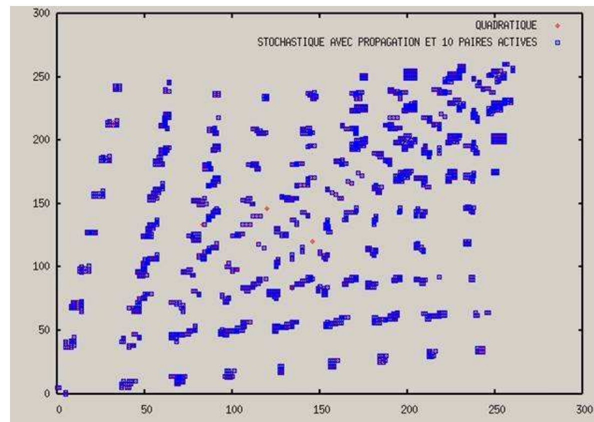
4.1.1.2 Qualité

Alors que la méthode hiérarchique assure de trouver toutes les collisions, la méthode stochastique ne peut le garantir puisque tout est dû au hasard. En utilisation pratique, on se rend compte que quasiment tout est détecté, à quelques paires près.

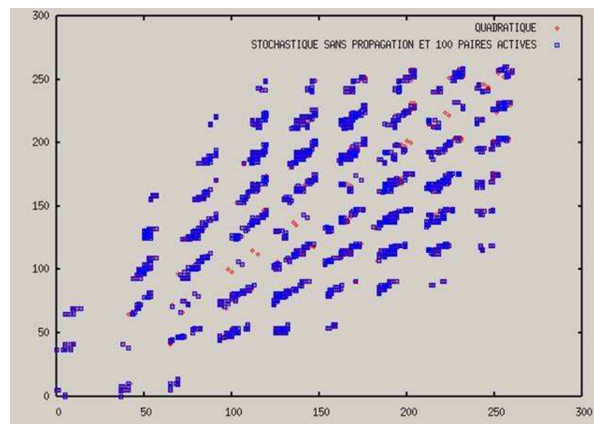
Sur la figure 4.3, on a schématisé les résultats en représentant toutes les arêtes en collisions détectées par la méthode quadratique en rouge, puis superposées en bleu, celles détectées par la méthode stochastique avec puis sans propagation. Les paires rouges ainsi apparentes sont donc celles non détectées. L'analyse et tirée de deux états différents d'un simulation, lors d'une quantité très importantes de collisions (>80%).

Dans le cas avec propagation (a), on remarque qu'une seule collision n'est pas détectée (attention, le schéma est symétrique), donc la qualité des résultats est très bonne. Dans la cas sans propagation, on remarque un peu plus de paires non détectées, mais la qaulité reste très correcte par rapport au gain gagné en terme de performances (cf 4.2).

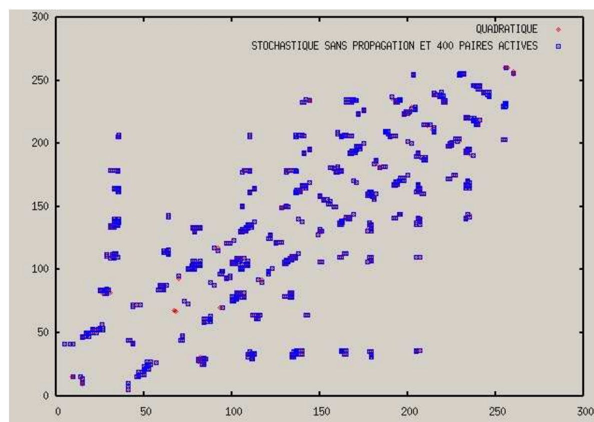
Pour mieux tester la méthode sans propagation, j'ai effectué un troisième test avec 400 paires actives, la vitesse descend alors à 70 images par secondes, mais on rate seulement une collision. Entre 100 et 400 paires, on distingue peu de changements, dans tous les cas, on rate moins de 10 collisions quelque soient leur quantité.



(a) stochastique : 10 paires actives + propagation (convergence sur sommets)



(b) stochastique : 100 paires actives sans propagation



(c) stochastique : 400 paires actives sans propagation

FIG. 4.3 – Paires (représentées par deux indices d'arête) en collisions détectées

4.1.2 Second essai

Pour analyser comment réagit notre méthode si le nombre de particules devient grand, je l'ai testé avec un treillis 100×10. Avec 300 paires actives sans propagation, on reste a 40 images par secondes, notre méthode résiste plutôt bien à la charge. Il faudra quand même faire des tests plus poussés sur des gros modèles et étudier la qualité de la détection, mais le temps manque avant le rédaction du rapport...

4.2 Travaux restants

4.2.1 Réglages des paramètres

Dans notre algorithme, il existe de nombreux paramètres tels que le nombre de paires à tirer, la distance maximum pour laquelle on considère une paire assez proche pour être active, la distance pour laquelle on considère qu'il y a collision.
Ces paramètres permettent de régler le rapport vitesse d'excecution / réalisme produit.
Il serait intéressant de bien savoir gérer ses paramètres pour mieux les adapter à un simulateur afin de réduire les coûts au maximum.
Le mieux serait de savoir calculer directement ces paramètres par rapport à la topologie des objets, le pas de temps de la simulation, les vitesses de déplacements. Ces paramètres pourraient évoluer au cours du temps.
On pourrait ainsi à chaque instant être au meilleur du réalisme permis par l'algorithme tout en restant en temps réel.

Dans les premiers essais, on peut remarquer de grandes différences suivant les paramètres et déduire des premières estimations. Ainsi, les première remarques concernent le nombre de paires tirées. Lorsque la quantité de collision est inférieure à 30%, il semble que tirer 10% du nombre de particules comme paires actives est efficace, dans ce cas là, la propagation est obligatoire pour tout détecter. Il y a besoin de peu de paires actives, puisqu'il y a peu de zones en collision.

Lorsque les collisions dépassent 50%, il semble bon de tirer une paire par particules. Dans ce cas là, on tire beaucoup de paires afin de bien trouver toutes les zones. On remarque aussi que dans ce cas là, la propagation n'est plus nécessaire, et qu'il est préférable de tirer plus de paires pour tout trouver. Sachant que la partie la plus lente de l'algorithme est la propagation, notre méthode s'accélère fortement.

Il est donc très important de savoir bien ajuster ces paramètres au cours de la simulation, et c'est là que réside tout l'intérêt de la méthode stochastique, c'est qu'elle peut s'adapter à tous les cas.

Si on considère la méthode sans propagation, on pourrait même penser avoir un contrôle précis du temps de cacul car si on fait évoluer au cours du temps la moyenne du temps de convergence d'une paire, le temps de calcul semble revenir à :

$$temps\ de\ calcul = moyenne\ temps\ de\ convergence \times nombre\ de\ paires\ actives$$

4.2.2 Intégration de la méthode stochastique dans une hierarchie de volumes englobants

En cas général, dans un maillage, quand on tire une paire d'éléments au hasard, et qu'on la fait converger, il y a de fortes chances qu'on obtienne une paire formée de deux éléments voisins. Puisqu'on ne garde pas les voisins, on tire beaucoup trop de paires par rapport au nombre paires intéressantes qu'il nous reste à la fin. De plus, le tirage aléatoire offre toujours un risque de passer à côté de toutes les collisions en « oubliant » totalement une zone. Il serait donc bon de tirer les paires de façon plus intelligente que le hasard.
Une solution est de diviser les objets ou l'espace en une hiérarchie de volumes englobants, comme décrit dans 2.3. Ces méthodes étant trop lentes pour le temps réel, il faut les accélérer, en utilisant notre méthode entre ces volumes. La méthode finale revient donc à une hiérarchie de volumes

grossiers, où la détection d'intersections entre ces volumes se fera de manière stochastique. Cette méthode nous assure qu'on va regarder s'il y a collision sur toute la surface de l'objet de façon sure, et non, seulement par un tirage aléatoire uniforme. De plus, la valeur des paramètres vu dans 4.2.1 peuvent être différents pour chaque paires de volumes, assurant le meilleur compromis possible. Il est même possible d'adopter une stratégie différente entre chaque volumes suivant la configuration, et ne pas utiliser les mêmes méthodes de détection.

La hierarchie qui semble la plus efficace et la plus adaptée pour notre problème est la hierarchie en k-DOP orientés et gonflés définies en 2.2.2. La suite de mon stage aurait était de poursuivre mon travail avec Stefan Kimmerle de l'université de Tübingen qui a déjà réalisé une telle méthode pour simuler des vêtements ([11]). D'après nos tests, cette méthode est déjà très efficace, et on pense qu'elle pourrait encore être accélérée à l'aide de la méthode stochastique. Chaque méthode peut pallier aux problèmes de l'autre.

4.2.3 Vers un simulateur

Avant de pouvoir utiliser ces méthodes de détection de collisions dans un simulateur, il faut ajouter des objets solides simples tels que des sphères, des parallélépipèdes... On pourrait ainsi faire réagir des objets déformables et des objets solides entre eux, ce qui permetterait de concevoir des simulations vraiment complètes.

De plus, un autre stagiaire, Sumit Jain, a développé une application permettant de créer des maillages de vêtements, avec l'idée de les simuler grâce à AnimAL. Une étape intéressante serait donc d'appliquer la détection de collisions à ces vêtements, portés par des personnages composés de solides de base.

Chapitre 5

Acquisitions personnelles

Durant cette année, j'ai appris énormément de choses. Tant au point de vu informatique théorique et technique, qu'au point de vu environnement de travail.

J'ai découvert un sujet passionnant qu'est l'animation, domaine dans lequel j'étais totalement néophyte. J'ai progressé en programmation avec la généricité et l'optimisation.

J'ai surtout découvert un univers très chaleureux qu'est le monde de la recherche, et fait la rencontre d'un grand nombre de personnes très intéressantes et dotées de grandes connaissances à partager.

Annexe A

Système masses-ressorts

Un système masses-ressorts est un ensemble de masses reliées entre elles par des ressorts (cf figure A.1).

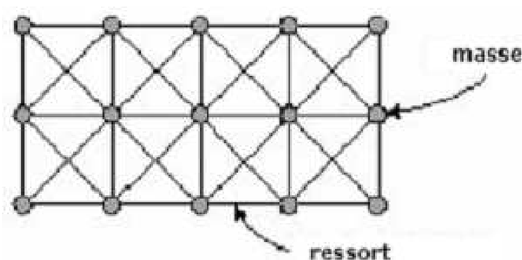


FIG. A.1 – Schéma d'un système masses-ressorts

L'étude de ce genre de structures est englobée dans un domaine plus large, les corps déformables.

A.1 Domaines d'application des systèmes masses-ressorts

On les utilise notamment pour modéliser des tissus, dans ce cas, ces systèmes sont représentés par des surfaces dans l'espace, mais on s'en sert aussi pour modéliser des volumes comme des organes dans les applications médicales.

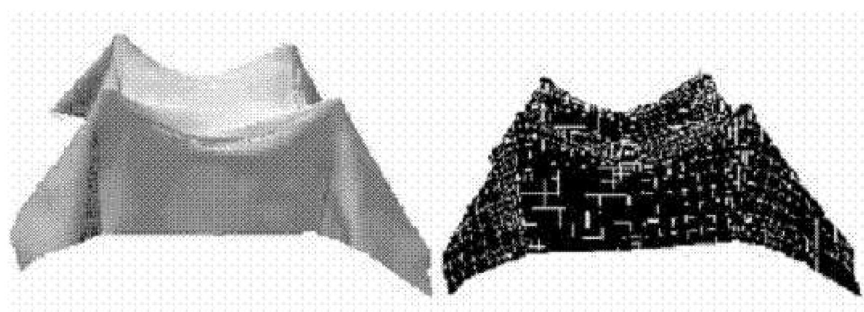


FIG. A.2 – Exemple d'utilisation d'un système masses-ressorts pour modéliser un tissu

Les interactions entres ces éléments sous l'effet de la gravité et d'autres forces donnent un

résultat visuel similaire à une surface élastique, bien qu'il n'y ait pas de correspondance rigoureuse avec l'état réel des choses. La simulation résultante n'est pas conforme à la réalité physique sous-jacente mais reste visuellement cohérente, ce qui satisfait à la plupart des applications.

A.2 Schéma d'intégration

Le schéma d'intégration est l'algorithme qui permet de calculer l'état à l'instant suivant à partir de l'instant courant en effectuant un intégration du temps, comme exprimé par la formule :

$$q(t+h) = q(t) + \int_{t}^{t+h} \dot{q}(q,t)qt$$

où h représente la longueur du pas de temps effectué.

Il existe quantité de schémas d'intégration, qui se distinguent par leur ordre de précision, leur stabilité et leur coût en terme de temps de calcul. Tous sont approximatifs car on ne sait pas calculer l'intégrale de la formule dans le cas général.

Par exemple, le schéma d'Euler fait l'approximation que la dérivée tout au long du pas de temps est égale à sa valeur au début du pas :

$$q(t+h) = q(t) + h\dot{q}(t) + O(h^2)$$

Annexe B

Réaction aux collisions par correction de position et de vitesse

notations :
- p_i : point i
- x_i : position du point i
- v_i : vitesse du point i
- a_i : arête i

B.1 Collision arête / arête

Les arêtes $a_1 : (p_1, p_2)$ et $a_2 : (p'_1, p'_2)$ sont en collision au point p de coordonnées (x, v) sur $a_1$et (x', v') sur a_2. s et t sont ses coordonnées barycentriques sur les deux arêtes. u est la direction dans laquelle effectuer la réaction (cf figure B.1).

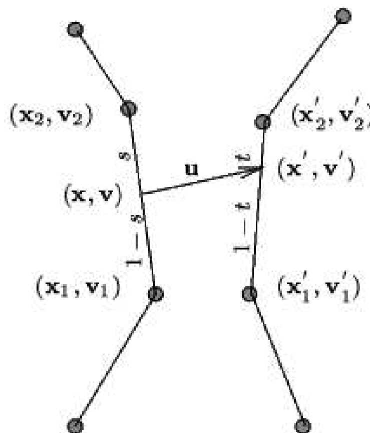


FIG. B.1 – Deux arêtes en collision *(dessinées éloignées pour une meilleure lisibilité)*

B.1.1 Mise à jour des vitesses

La vitesse se répartie suivant les coordonnées barycentriques :

$$v = (1-s)v_1 + sv_2$$

$$v' = (1-t)v'_1 + tv'_2$$

On va appliquer une force f sur $a_1$et une force f' sur a_2 dans le direction de la collision pour modifier les vitesses telles que la vitesse relative entre les deux arêtes soit nulle. Les nouvelles vitesses au point de collision sont v_{new} et v'_{new}.

$$(v_{new} - v'_{new}).u = 0 \tag{B.1}$$

La force f peut être appliquée aux extrémités des arêtes en la répartissant suivant les coordonnées barycentriques, d'où :

$$v_{new1} = v_1 + (1-s)fu \qquad v_{new2} = v_2 + sfu$$
$$v'_{new1} = v'_1 + (1-t)fu \qquad v'_{new2} = v'_2 + tfu$$

La vitesse se répartissant aussi suivant les coordonnées barycentriques :

$$\begin{aligned} v_{new} &= (1-s)v_{new1} + sv_{new2} \\ &= v + ((1-s)^2 + s^2)fu \end{aligned} \tag{B.2}$$

$$v'_{new} = v' + ((1-t)^2 + t^2)fu \tag{B.3}$$

En remplaçant B.2 et B.3 dans B.1, on obtient f :

$$f = \frac{(v' - v).u}{(1-s)^2 + s^2 + (1-t)^2 + t^2}$$

B.1.2 Mise à jour des positions

En procédant de la même façon que les vitesses, on obtient la force g qui va changer instantanément les positions :

$$g = \frac{(x' - x).u}{(1-s)^2 + s^2 + (1-t)^2 + t^2}$$

B.2 Collision point / triangle

On utilisa le même raisonnement qu'entre les arêtes. Le triangle (P_1, P_2, P_3) est en collision avec le point P_4.

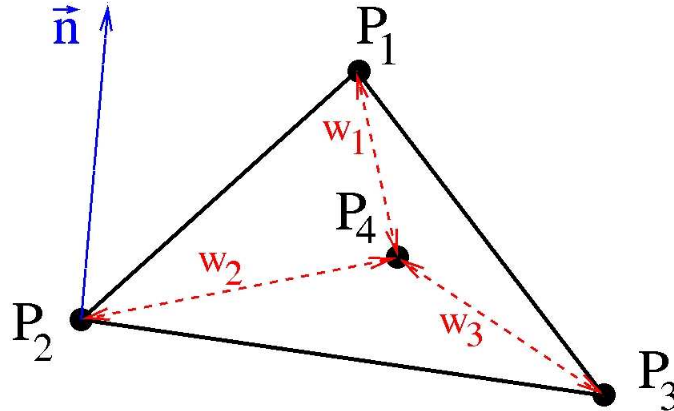


FIG. B.2 – Un point et un triangle en collision

B.2.1 Mise à jour des vitesses

Le point P_4, est aussi le point de collision sur le triangle, donc définissable par :

$$P_4 = w_1 P_1 + w_2 P_2 + w_3 P_3$$

avec w_1, w_2, w_3 les coordonnées barycentriques de P_4 par rapport aux points P_1, P_2, P_3.
La vitesse relative entre le triangle et le point au point P_4 est :

$$\overrightarrow{v_{rel}} = \overrightarrow{v_4} - w_1 \overrightarrow{v_1} - w_2 \overrightarrow{v_2} - w_3 \overrightarrow{v_3}$$

On veut $\overrightarrow{v_{rel}}.\overrightarrow{n} = 0$, pour cela on applique une force $f\overrightarrow{n}$ en P_4 et $-f\overrightarrow{n}$ sur le triangle au point P_4, avec la répartition barycentrique, on applique$-w_1 f\overrightarrow{n}$ en P_1,$-w_2 f\overrightarrow{n}$ en P_2,$-w_3 f\overrightarrow{n}$ en P_3.

Soient Δv_1,Δv_2,Δv_3,Δv_4 les variations de vitesses le long de $\overrightarrow{n}$, telle que $\overrightarrow{v_i} + = \Delta v_i \overrightarrow{n}$, d'où :

$$\Delta v_1 = -\frac{w_1}{m_1} f$$

$$\Delta v_2 = -\frac{w_2}{m_2} f$$

$$\Delta v_3 = -\frac{w_3}{m_3} f$$

$$\Delta v_4 = \frac{1}{m_4} f$$

donc :

$$\overrightarrow{v_{rel}}.\overrightarrow{n} = (\frac{1}{m_4} + \frac{w_1^2}{m_1} + \frac{w_2^2}{m_2} + \frac{w_3^2}{m_3}) f$$

d'où :

$$f = \frac{\overrightarrow{v_{rel}}.\overrightarrow{n}}{\frac{1}{m_4} + \frac{w_1^2}{m_1} + \frac{w_2^2}{m_2} + \frac{w_3^2}{m_3}}$$

.

B.2.2 Mise à jour des positions

En procédant de la même façon que les vitesses, on obtient la force g qui va changer instantanément les positions :

$$g = \frac{distance\, de\, penetration}{\frac{1}{m_4} + \frac{w_1^2}{m_1} + \frac{w_2^2}{m_2} + \frac{w_3^2}{m_3}}$$

Bibliographie

[1] L. Raghupathi, V. Cantin, F. Faure et M.-P. Cani, real-time simulation of self-collisions for virtual intestinal surgery, IS4TM conference (http://www-imagis.imag.fr/Membres/Francois.Faure/papers/intestine/intestine.pdf)

[2] G. Debunne, M. Desbrun, M.-P. Cani et A. H. Barr, Dynamic real-time deformations using space and time adaptive sampling. In Proc, SIGGRAPH '01, août 2001

[3] G. Debunne et S. Guy, Layered Shells for Fast Collision Detection, 2002

[4] P. Meseure, L. Hilde et C. Chaillou, Accélération de la détection de collisions entre corps rigides et déformables Actes des 6émes journées du Groupe de Travail Réalité Virtuelle, mars 1998 (http://www.lifl.fr/GRAPHIX/rechfonda/calcul-coll/publications/postcript/GTRV98-Meseure.ps.gz)

[5] V. Cantin, Auto-collisions de surfaces déformables pour le temps réel, 2002

[6] BISMI Mustapha, Collisions : Oriented Bounding Box et SphereTree (http://mustapha.bismi.free.fr/articles/obb.pdf)

[7] S. Gottschalk, M. Lin, et D. Manocha, OBB-Tree : A Hierarchical Structure for Rapid Interference Detection, ACM SIGGRAPH, 1996 (ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/COLLISION/sig96.pdf)

[8] N. M. Thalmann et P. Volino, Efficient self collision detection on smoothly discretized surface animations using geometrical shape regularity, 1994

[9] N. M. Thalmann et P. Volino, Collision and self-collision detection : Efficient and robust solutions for highly deformable surfaces, 1998

[10] R. Bridson, R. Fedkiw et J. Anderson, Robust treatment of collisions, contact and friction for cloth animation

[11] J. Mezgern S. Kimmerle et O. Etzmuß, Improved collision detection and response techniques for cloth animation, WSI-2002-5, août 2002 (http://www.gris.uni-tuebingen.de/publics/paper/Mezger-2002-Improved.pdf)

[12] D. Baraff, A. Witkin et M. Kass (Pixar Animation Studios), Untangling Cloth, SIGGRAPH 2003, juillet 2003 (http://www.pixar.com/companyinfo/research/deb/untangling.pdf)

[13] P. Schneider et David H. Eberly, Geometric Tools for Computer Graphics (Morgan Kaufmann)

[14] Y. Le Goc, X3DToolKit (http://www-imagis.imag.fr/Membres/Yannick.Legoc/X3D/index.html)

[15] M. Senin, N. Kojekine, V. Savchenko et I. Hagiwara, Particle-based Collision Detection, EUROGRAPHICS 2003 (http://www.mech.titech.ac.jp/~karlson/papers/coll_det.pdf)

[16] J. Cohen, M. C. Lin, D. Manocha et M.Ponamgi, I-COLLIDE : An Interactive and Exact Collision Detection System for Large-Scale Environnements, Symposium on Interactive 3D Graphics 1995 (http://citeseer.nj.nec.com/cache/papers/cs/798/ftp:zSzzSzftp.cs.unc.eduzSzpubzSzuserszSzmanochazSzPAPERSzSzCOLLISIONzSzpaper3dint.pdf/cohen95icollide.pdf)