

Simulation en temps réel d'objets déformables et découpables

THÈSE

présentée et soutenue publiquement le 12 octobre 2006

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Damien Marchal

Composition du jury

Président :	Sophie Tison, Professeur	Université de Lille I
Rapporteurs :	Marie-Paule Cani, Professeur	INP Grenoble
	Hervé Delingette, DR-Inria	INRIA Sophie Antipolis
Examinateurs :	Stéphane Redon, CR-Inria	INRIA Rhônes-Alpes
	Laurent Grisoni, Maître de Conférence	Université de Lille I
	Fabrice Aubert, Maître de Conférence	Université de Lille I
Directeur :	Christophe Chaillou, Professeur	Université de Lille I

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE
Laboratoire d'Informatique Fondamentale de Lille — UPRESA 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : direction@lifl.fr

Mis en page et composé avec LaTeX et la classe thlifl.

Remerciements

En premier lieu, je remercie Marie-Paule Cani et Hervé Delingette qui m'ont fait l'honneur de rapporter cette thèse ; leurs remarques et commentaires contribuant d'autant à son amélioration. Je remercie aussi les autres membres du jury, Sophie Tison et Stéphane Redon pour l'intérêt porté à ce travail.

Cette thèse a été rendue possible grâce au soutien et à la confiance accordés par les différents membres de l'équipe ; je pense en particulier à Christophe Chaillou, Fabrice Aubert et Laurent Grisoni. Leur soutien, ainsi que les nombreuses discussions techniques ou philosophiques, calmes ou enflammées, ont jalonné le chemin de cette thèse.

Je suis convaincu que cette thèse aurait aussi été bien différente sans les très nombreuses collaborations avec les membres de l'équipe Graphix/Alcove. Je veux remercier plus spécifiquement Jérémie et Julien pour leur soutien en math et en physique ; et d'une manière plus générale, toute l'équipe *simulation* : Pierre-Jean, Stéphane, Christian, SB, FredB, Sylvère et Philippe. Une pensée pour mes compagnons de thèse : François, Sylvain, Cédric (qui m'a enseigné le self-contrôle !). Merci aussi à Nico, toujours prêt à venir débugger du C++ trop templaté. Enfin, je veux remercier les nombreux membres (anciens et présents) qui ont animé le bureau 212 : FredT, Amhed, LoL, Lucianna et SamD.

Je veux encore remercier ma famille et tout particulièrement ma maman, pour la relecture patiente de ce manuscrit. Un coucou aussi à mes frères et soeurs Emmanuel, Olivier, Ann'Lise et Claire.

Merci enfin à Iovka, ma compagne, avec qui j'ai partagé les hauts et les bas de la vie de thésard, et avec laquelle j'espère partager ma vie tout court...

Damien

Le 12 octobre 2006

Résumé

L'animation de scènes virtuelles en synthèse d'images est une technique maîtrisée, largement utilisée pour les effets spéciaux cinématographiques et les jeux vidéos. Récemment, la simulation physique a été introduite pour produire ces animations ; utiliser la simulation permet de produire automatiquement des animations réalistes dans les environnements virtuels interactifs. Dans ce contexte, cette thèse présente de nouvelles techniques de simulation, en temps réel, d'objets déformables qui peuvent être découpés. On s'intéresse aussi à la détection des collisions, aux méthodes d'interpolation de champs du type Moindres Carrées Mobiles et aux méthodes sans maillage. Les trois contributions principales de la thèse sont :

- un modèle d'objet déformable temps réel basé sur la théorie de l'élasticité et la méthode des éléments finis dont la particularité est d'adapter la complexité des calculs en fonction de la puissance disponible ; s'appuyant sur plusieurs représentations géométriques découplées, il permet aussi la simulation d'objets dont la géométrie n'est pas manifold.
- un algorithme rapide de calcul de carte de distance. Cet algorithme est utilisé pour traiter les collisions entre des objets déformables et découpables ; il est aussi à la base d'une méthode de segmentation d'objets volumiques.
- un modèle d'objet déformable basé sur la méthode du Shape-Matching qui supporte les opérations de découpe. L'intérêt du modèle repose sur la combinaison d'une méthode sans maillage efficace et de l'algorithme de calcul de carte de distance précédemment présenté. Ce modèle semble bien adapté pour l'animation interactive.

Abstract

Animating virtual environment is now a well known methods that is strongly used to produce special effect in movies or games. Recently, physics simulation has been introduced as a way to produce such animation. Using simulation allows realistic animation in virtual environment where the users can interact. In this context, the thesis present new methods for realtime simulation of deformable object that can be cut, we also work on collision detection, radial based interpolation methods (Moving Least Square) and meshless simulation. This thesis work is composed of three majors contributions : an adaptive deformable model based on Finite Element method where the accuracy of both physics simulation and collision detections can be adapted to match a given computation time constraint. an algorithm to compute distance field on tetrahedral mesh, this algorithm is based on a Diskjstra's shortest-path method and is also related to the Fast-Marching method. We employed this algorithm to update, in realtime, the distance field used for collision detection for deformable object. Using the tetrahedral mesh (Finite Element Method) to compute the collision make the method well designed to handle object cutting. finally we introduce a model of deformable object that can be cut in real-time. The deformation system is based on the Shape-Matching method. This implies that the object is correctly segmented in several parts. To update the segmentation we introduce a new segmentation algorithm, based on the fast distance field computation.

Introduction

Le développement de l'animation basé sur la physique est un domaine de recherche particulièrement actif en informatique graphique et en réalité virtuelle, en raison de la variété des applications possibles (comme le développement de simulateurs pédagogiques, la synthèse du mouvement pour le cinéma ou encore les jeux vidéo). Du point de vue de la recherche, l'usage de simulation suppose à la fois de développer des moteurs de simulation efficaces, des modèles d'objets déformables, des techniques de gestion des collisions et des algorithmes d'affichage performants. En plus, il faut aussi identifier et modéliser les différents comportements d'objets réels afin de pouvoir les utiliser dans une simulation.

L'objectif central de cette thèse est de développer de nouvelles techniques pour simuler, en temps réel, des objets déformables ainsi que des objets qui peuvent être découpés. Dans la suite de la thèse, on parle d'objets *découpables*. Pour cela, il faut développer des solutions dynamiques et robustes permettant la modification de la géométrie des objets. En plus, la découpe d'un objet induit des modifications topologiques, ce qui se répercute sur la simulation mécanique et de la détection de collision (fig. 1).

L'efficacité de la plupart des techniques de simulation et de détection de collision repose sur l'hypothèse que la topologie des objets ne change pas. Le rôle central joué par la géométrie et la topologie nous conduit à nous intéresser aux différentes représentations géométriques ainsi qu'aux concepts de multimodèle et de multirésolution et leurs différents usages pour la simulation.

Ce document est décomposé en cinq chapitres. Le premier est un état de l'art des techniques de simulation d'objets déformables, de détection de collision et de modélisation d'objets découpables. Dans le second chapitre, nous faisons une présentation plus personnelle de deux aspects importants des simulateurs, à savoir la difficulté à produire des logiciels pour la simulation temps-réel d'un ensemble important d'objets déformables et l'importance de la géométrie dans la représentation visuelle ainsi que pour les calculs mécaniques et la gestion des interactions. Ce chapitre sert également à introduire nos trois contributions plus techniques présentées dans les trois chapitres suivants : une technique de simulation adaptative d'objets déformables (ch. 3), un algorithme de type Fast Marching qui permet de traiter la détection de collision entre objets déformables (ch. 4) et enfin un modèle d'objets découpables (ch. 5).

Nous avons aussi choisi de présenter en annexe des éléments de réflexion qui nous semblent intéressants mais qui ne s'intègrent pas directement dans la thèse. Dans la première de ces annexes, on présente nos expérimentations autour de l'approximation aux moindres carrés mobiles et introduisons comment construire une méthode qui réalise exactement une interpolation au lieu d'une approximation. Dans la seconde annexe, nous présentons une architecture pour la parallélisation d'un moteur de simulation que nous avons développé dans le cadre du projet Sofa [sof].

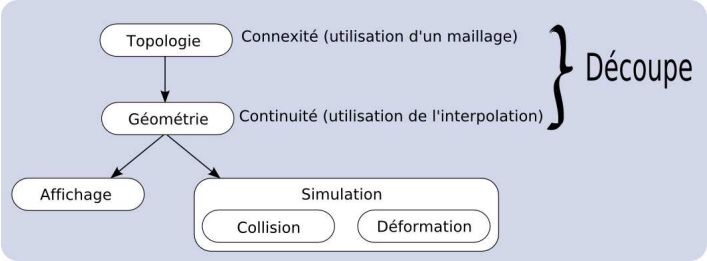

Fig. 1 : *La découpe est une opération géométrique. Les conséquences d'une découpe se propagent au niveau de la simulation et de la gestion des collisions.*

Nomenclature

x : une grandeur scalaire.
$\mathbf{x}$: un élément d'un espace vectoriel de $\mathbb{R}^n$.
x_n : la n-ième composante d'un vecteur.
$\overrightarrow{v}$: un vecteur de l'espace euclidien.
$v_{x,y,z}$: les composantes x, y, z d'un vecteur de l'espace euclidien.

$\dot{u}(x,t)$: la dérivée de la fonction $u(x,t)$ suivant le temps.
$\ddot{u}(x,t)$: la dérivée seconde suivant le temps.

Ω : un domaine (matériel, géométrique).

$f^n(x)$: la dérivée n'ième de la fonction $f(x)$.
$f'(x)$: la dérivée première de la fonction $f(x)$.
$f''(x)$: la dérivée première de la fonction $f(x)$.

$\frac{\delta f(x)}{\delta x_n}$: la dérivée partielle de la fonction $f(x)$ suivant la variable x_n.

$\tilde{u}(x)$: le tilde indique que la fonction est une fonction d'approximation (spline, moindres carrés).

Table des matières

Chapitre 1

Etat de l'art pour la simulation d'objets déformables

Sommaire

Introduction

La simulation d'objets virtuels permet de modéliser des comportements dynamiques complexes comme le mouvement réaliste d'un objet, ses déformations ou encore les chocs entre différents objets. Pour cela, on doit se tourner vers le domaine de la physique qui propose des modèles mécaniques du comportement. C'est d'ailleurs dans les domaines de la mécanique et de l'ingénieurie que furent développés les premiers moteurs de simulation. La finalité de ces moteurs de simulation était bien différente, visant à évaluer de la résistance des pièces manufacturées et des constructions. Ces moteurs sont maintenant devenus une étape essentielle du processus de fabrication des immeubles, voitures, avions, ponts .... La simulation pour l'ingénieurie suppose que les modèles mécaniques utilisés soient conformes à la réalité, *prédictifs*, dans la mesure où le résultat de la simulation doit être très proche du comportement de l'objet matériel réel. La simulation nous renseigne alors, a priori, sur les propriétés finales d'un produit.

Les moteurs de simulation pour l'informatique graphique s'inspirent largement de moteurs de simulation pour la mécanique. Pourtant ils s'en différencient nettement, tant au niveau de leur usage que des moyens techniques pour leur mise en œuvre. En l'état actuel, les moteurs de simulation pour l'informatique graphique ne cherchent pas à produire des résultats pouvant être comparés au réel. Ils sont plutôt utilisés pour calculer un comportement paraissant plausible aux personnes visualisant l'animation[1] sans qu'il y ait comparaison fine avec le réel. Cette différence d'objectif oriente le choix de techniques qui sont jugées acceptables. Ainsi, dans les moteurs de simulation pour l'informatique graphique, on cherche à mélanger, au sein d'une même simulation, des objets de natures différentes : certains sont issus de la modélisation mécanique, et sont donc très réalistes, d'autre objets pouvant provenir d'autres modélisations simplifiées. En mélangeant les différentes modélisations, on cherche à trouver un équilibre entre les temps de calcul nécessaires à la simulation et un réalisme suffisant.

Depuis quelques années, les moteurs de simulation tendent à se généraliser. On trouve essentiellement deux catégories d'utilisation différentes des moteurs de simulation : la production d'animation de synthèse et les environnements virtuels interactifs. Dans la première catégorie, la simulation est utilisée comme un outil permettant de calculer facilement le mouvement des objets dans une animation de synthèse. Depuis le développement de simulateurs réalistes et facilement contrôlables par des artistes, les simulateurs sont de plus en plus utilisés et intégrés dans des suites de modélisation 3D comme : Maya, 3DS Max ou Blender (figure 1.3). Avant l'apparition de la simulation dans ces suites de modélisation 3D, les seules techniques d'animation disponibles étaient basées sur la description explicite des différentes positions des objets à l'aide d'équations paramétriques. Cette approche est satisfaisante lorsque le mouvement des objets est simple. Ainsi, le mouvement d'un objet rigide est considéré comme simple puisque sa configuration, à un instant de l'animation, se résume à six paramètres appelés *degrés de liberté* : trois des ces degrés décrivent la position et trois autres l'orientation. Par contre, les objets complexes, déformables, sont composés

[1] On peut remarquer que cette notion de "physiquement plausible" est fortement liée aux habitudes des spectateurs. Un expert du domaine, qui a l'œil du métier, verra probablement des comportements non plausibles là où le néophite ne distinguera rien de particulier. On peut donc se demander si, avec la généralisation de l'utilisation de la physique, les comportements qualifiés de plausibles actuellement ne se verraient pas déclassés au rang de comportements non plausibles à mesure que les spectateurs s'habituerairent à visualiser des simulations.

d'un grand nombre de ces degrés de liberté ; par exemple, pour l'animation manuelle de corps mous comme les tissus (fig. 1.2), les cheveux (fig. 1.1), les fluides (fig. 1.3), où plusieurs milliers de degrés de liberté doivent être définis afin de traduire, de manière réaliste, les différentes déformations. Cela est difficilement faisable manuellement et la simulation apporte une solution à ce problème. C'est le moteur de simulation qui calcule la position et la déformation des objets pendant l'animation, à partir d'une configuration initiale et de lois physiques. Le travail de l'animateur s'en trouve considérablement simplifié.

Fig. 1.1 : *L'utilisation de la simulation dispense les animateurs de la spécification de positions intermédiaires pour les objets. La coiffure est directement obtenue par la simulation, l'animateur n'ayant à spécifier que les propriétés mécaniques du cheveu (extraits de [BMC05]).*

Fig. 1.2 : *Extrait de [BFA02]. Les mouvements du tissu sont calculés par un moteur de simulation alors que les mouvements du personnage sont réalisés manuellement par des animateurs.*

Fig. 1.3 : *La simulation de fluide est directement intégrée dans le modeleur Blender.*

Le second usage des moteurs de simulation en informatique graphique est probablement celui qui nous est le plus familier : les environnements virtuels réalistes – jeux vidéo, simulateurs pédagogiques .... Dans ce type d'environnements virtuels, l'utilisateur interagit avec les objets qui composent la scène (fig. 1.4). La simulation se prête alors particulièrement bien à la modélisation réaliste de tel environnement et permet d'intégrer l'utilisateur dans la simulation. Les environnements virtuels à but pédagogique sont probablement les premiers environnements virtuels à avoir fait usage de la simulation pour décrire le comportement des objets. L'objectif de ces environnements est de permettre l'acquisition de nouvelles compétences grâce à l'ordinateur plutôt que par une expérimentation réelle ; cela supposant un certain niveau de réalisme. Parmi les environnements virtuels à but pédagogique, ce sont probablement les simulateurs de vols qui ont été les premiers utilisés (fig. 1.5). Cependant, d'autres domaines d'application des environnements pédagogiques sont envisageables. En effet, on peut supposer que le développement de simulateurs pédagogiques est intéressant pour toute activité dont l'apprentissage est risqué, complexe et/ou coûteux. On trouve actuellement un certain nombre d'entreprises et de laboratoires de recherche qui travaille au développement de simulateurs pédagogiques pour la formation médicale (fig. 1.6). La difficulté, liée au développement de ces simulations, tient d'une part au réalisme[2] pour une *bonne* formation des médecins, et d'autre part au besoin de simplification des méthodes de calcul, afin de permettre une simulation en un temps raisonnable. De manière générale, la simulation est de plus en plus présente dans toutes les formes d'environnements virtuels. De plus en plus, les jeux vidéo incorporent des moteurs de simulation pour animer certains éléments de l'environnement, comme la modélisation de voitures dans un jeu de course ou de systèmes de particule pour la production de fumées, de nuages et d'explosions ... (figure 1.7). Un certain nombre de sociétés comme Havok ou Aegia ont pour principales activités le développement et la vente de moteurs de simulation pour les jeux vidéo.

[2]Le réalisme suppose une bonne connaissance des comportements biomécaniques des organes. On peut remarquer que les comportements d'organes biologiques sont nettement plus complexes que les comportements mécaniques d'outils manufacturés.

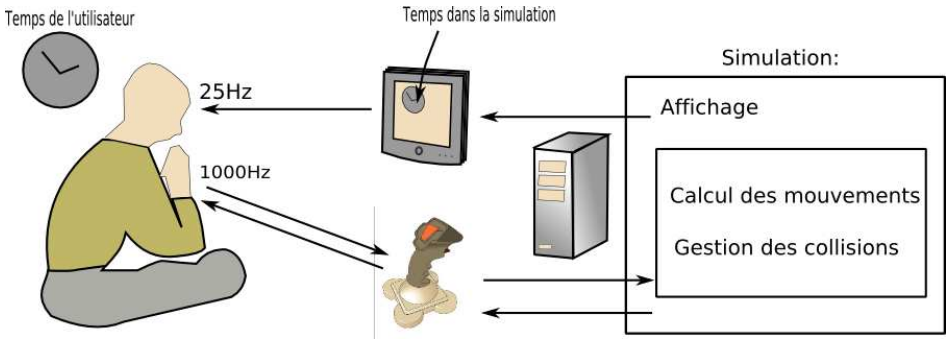

Fig. 1.4 : *Environnement virtuel basé sur un moteur de simulation. L'utilisateur est un acteur de la simulation.*

Fig. 1.5 : *Les simulateurs de vol sont maintenant utilisés couramment dans le processus de formation des pilotes.*

Fig. 1.6 : *L'utilisation de la simulation permet le développement d'environnements virtuels à but pédagogique. Ci-dessus, des prototypes de simulateurs médicaux (extrait de [CDL+05] et [med]).*

Fig. 1.7 : *Les jeux vidéo font de plus en plus l'usage de la simulation physique pour l'animation des explosions, de la fumée, des débris (extrait de [Hav]).*

Dans ce chapitre, nous décrivons les différents composants d'un environnement virtuel basé sur la simulation. Nous commençons par décrire de manière générale l'architecture logicielle d'environnements virtuels pour ensuite décrire plus en détail comment simuler des objets déformables aux travers des aspects suivants : calcul des déformations, calcul des collisions et intégration temporelle. La dernière section du chapitre présente les techniques de découpe d'objets déformables.

1.1 Architecture d'environnements virtuels simulés

Les moteurs de simulation utilisés dans les environnements virtuels simulés s'appuient en général sur une architecture découpée en trois unités fonctionnelles (fig. 1.8) :

- *l'unité d'animation :* elle a pour tâche de calculer les comportements dynamiques, liés au temps, des objets composant la scène ; ces comportements peuvent être calculés par simulation (dynamique ou statique) ou bien par des procédés cinématiques.
- *l'unité d'affichage :* pour calculer les images à partir de la représentation géométrique (textures incluses) des objets ainsi que des conditions d'éclairage.
- *l'unité de contrôle :* donne à l'utilisateur la possibilité d'influer sur la simulation par le biais de périphériques adaptés.

1.1.1 L'unité d'affichage

C'est l'unité d'affichage qui permet la visualisation de l'environnement virtuel. Elle transforme la description géométrique de la scène en une suite d'images qui, une fois affichée, produit l'animation. Les calculs nécessaires à cette transformation, souvent appellée "rendu", sont en général pris en charge par une carte spécialisée. Cette carte est dotée d'un processeur spécifique contrôlé au travers d'une API dédiée (OpenGL ou Direct3D). De telles cartes sont aujourd'hui largement disponibles et utilisées notamment par les jeux vidéo. La puissance de ces cartes rend possible un rendu réaliste en temps réel de scènes simulées.

La tendance actuelle de ce type de carte est l'utilisation d'architectures de plus en plus génériques et programmables à l'aide de petits programmes appelés *shaders*. Les *shaders* permettent :

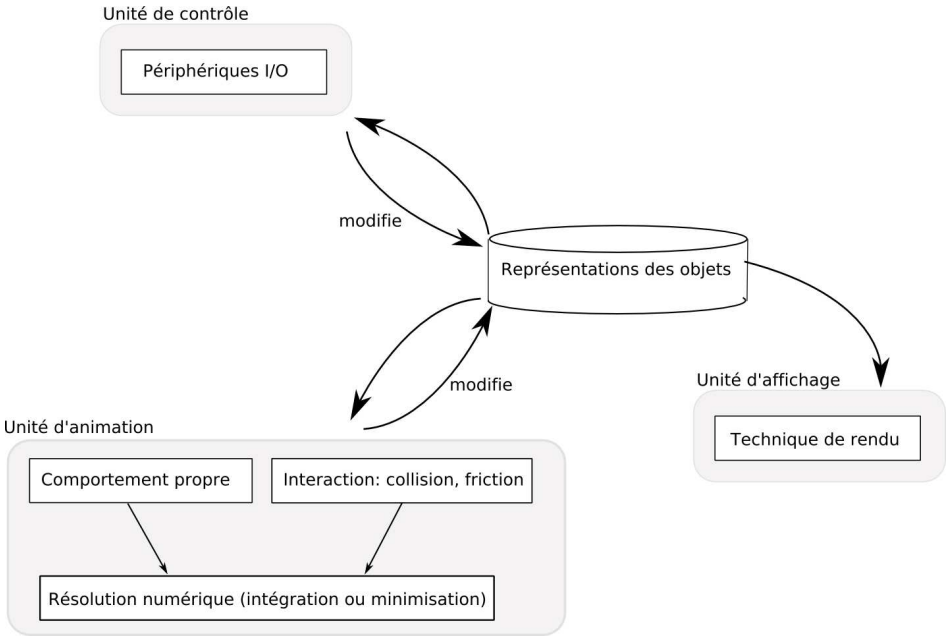

Fig. 1.8 : *Architecture d'un environnement virtuel « simple », celui-ci est composé de trois unités. Le point central d'un environnement est la représentation du monde – la scène – manipulée par les différentes unités qui y sont connectées.*

- l'expérimentation de nouvelles méthodes de rendu, comme les techniques à base de points (*point based rendering*), les éléments de surface (*surfels*), ou encore l'usage de surfaces à subdivision directement accélérées par la carte. L'usage de telles méthodes reste assez marginal dans le domaine de l'affichage d'objets simulés, mais on peut penser que cela pourrait se répandre, si les méthodes de simulation continuent à s'appuyer sur des approches sans maillage.
- le développement d'algorithmes non graphiques (calcul générique sur carte graphique). On trouve en particulier un nombre de plus en plus important de travaux qui concernent l'exécution des calculs physiques et permettent la simulation de fluides, de tissus, ou encore la détection de collision, directement sur la carte 3d.

On peut noter que l'unité d'affichage n'est pas, ou plutôt n'est plus, un facteur pénalisant l'interactivité de l'environnement virtuel. Très souvent, c'est la complexité des calculs de l'unité d'animation qui empêche le développement d'environnements plus complexes. Dans la suite de la thèse, nous concentrerons nos travaux sur ces calculs d'animation.

1.1.2 L'unité de contrôle

L'unité de contrôle est connectée aux périphériques d'entrée-sortie permettant à l'utilisateur d'interagir avec la simulation. On identifie deux types de périphériques : ceux qui permettent à l'utilisateur d'agir sur l'environnement et ceux qui, en plus, lui offrent un retour. Lorsque ce retour est envoyé à l'usager sous forme kinesthésique, on parle de périphériques haptiques [Wika], et lorsque le retour est plus précis et se manifeste par le sens du toucher, il s'agit de périphériques tactiles [Wikd].

1.1.3 L'unité d'animation

C'est l'unité d'animation qui prend en charge le calcul du comportement des objets et leur évolution au cours du temps. On peut modéliser une animation par une fonction mathématique $f(t)$. Cette fonction décrit l'état courant, la configuration d'un objet en un instant t d'une simulation. On peut identifier plusieurs catégories de comportement :

- *le comportement inerte :* un objet inerte est animé au travers d'une fonction constante $f(t) = const$. C'est un objet qui ne bouge pas et ne se déforme pas. Identifier des objets au comportement inerte est important, car cela favorise la mise en place d'optimisations et de précalculs. En particulier, il n'est pas nécessaire de gérer la collision entre des paires d'objets inertes ; ce qui diminue considérablement la quantité de calculs à effectuer.
- *le comportement cinématique :* les objets cinématiques sont animés au travers d'une équation explicite dépendant du temps t. Deux techniques coexistent : la première,assez rare , définit le mouvement à l'aide de fonction analytique. Par exemple la fonction : $x = cos(t), y = sin(t)$ décrit le mouvement circulaire d'une particule dans l'espace 2d. La seconde, dite de keyframing, s'appuie sur l'utilisation de courbes d'interpolation. Ces courbes sont obtenues à partir de configurations-clefs interpolées afin de former un mouvement intermédiaire continu. Bien que mobiles, les objets cinématiques ne sont pas encore des objets simulés. En effet, les clefs de position sont définies explicitement et manuellement. Les mouvements imprévus ne sont pas possibles.
- *le comportement dynamique :* les objets dynamiques sont des objets simulés dont le comportement évolue au cours du temps. Ils sont décris à partir d'équations physiques ; ces équations décrivent le comportement de la matière en fonction des efforts subis ainsi que d'un processus d'intégration temporelle qui permet de calculer, à partir d'une configuration donnée en un instant t, une nouvelle configuration en un instant suivant $t + \delta t$.
- *comportement statique :* les objets statiques sont des objets simulés dont le comportement n'évolue pas au cours du temps. Si l'utilisateur agit sur les objets, une succession d'états d'équilibre est calculée formant une animation : on parle alors d'approche "quasi-statique".
- *objet contrôlé :* les objets contrôlés forment une catégorie un peu particulière d'objets. Ils correspondent aux objets sous le contrôle de l'utilisateur via un périphérique comme la souris, un joystick ou un périphérique haptique. On distingue deux catégories d'objets contrôlés suivant la manière dont ils s'insèrent dans la simulation. La première catégorie peut être vue comme un objet cinématique dont la fonction $f(t)$ ne serait pas connue à l'avance. Ce sont des objets qui peuvent être visualisés et qui peuvent entrer en collision. Cependant, comme ils ne sont pas simulés, ils ne réagissent pas de manière physique lors de collisions avec l'environnement. L'autre catégorie correspond aux objets qui sont vus comme des entités simulables au niveau de la simulation. Ce sont des objets qui fournissent une configuration, mais qui peuvent aussi réagir à l'application d'une force (ex : périphériques haptiques). Cette opération est possible par l'utilisation d'un système de couplage entre l'objet simulé et le périphérique manipulé par l'utilisateur ; ce couplage est appelé *god-objects* [MLFC04] ou *par objet proxy*.

Dans le cadre de cette thèse, nous ne considérons que l'unité d'animation comme strictement équivalente à un moteur de simulation, car nous ne manipulons que des objets dont le comportement est *simulé* ou *contrôlé*. Afin de simuler ces objets, on emploie un moteur de

simulation décomposable en trois parties fonctionnelles différentes : le calcul du comportement propre des objets, dans lequel on modélise et décrit les propriétés de la matière qui compose l'objet (sec. 1.2) ; le calcul du comportement des objets en interaction, c'est la gestion des collisions (sec. 1.3) ; et enfin le calcul effectif du mouvement des objets (sec. 1.4) qui prend soit la forme d'une intégration temporelle, soit celle d'une minimisation énergétique.

1.2 Calcul du mouvement d'objets déformables

Les objets sont modélisés à l'aide des lois de la mécanique. Ces lois sont exprimées à l'aide de systèmes d'équations différentielles qui permettent le calcul du mouvement des objets et de leurs déformations. Une modélisation assez courante, retrouvée de plus en plus souvent dans les jeux vidéo, est la mécanique des corps rigides. Dans le cadre de la simulation de corps rigides, la configuration d'un objet est décrite à l'aide de six degrés de liberté. Ces six paramètres sont mis à jour rapidement donnant lieu à des simulations comportant un grand nombre d'objets. L'utilisation de la mécanique des corps rigides suffit pour modéliser une grande partie des objets qui nous entourent. Cependant, dans certains contextes particuliers comme la simulation médicale, on doit pouvoir modéliser des objets déformables. La difficulté de la simulation de corps déformables réside dans le fait que les temps de calcul sont nettement plus importants que pour les corps rigides. Ces temps de calcul sont corrélés au nombre très important de degrés de liberté nécessaire à la description précise et réaliste de l'état d'un objet déformable. En outre, les objets déformables les plus complexes nécessitent des lois de comportement non-linéaires, ainsi que des comportements d'anisotropie ou d'hystérésie. Ces lois de comportements évoluées, nécessaires à certains domaines comme la simulation biomédicale, se traduisent mathématiquement par des systèmes d'équations non-linéaires qui sont complexes à résoudre. C'est pourquoi, la plupart des modèles d'objets déformables temps-réel actuels s'appuient sur des lois de comportements linéaires ou sur des simplifications ad-hoc (masses-ressorts). Nous considérons que ces modèles d'objets déformables peuvent être regroupés en deux grandes catégories suivant qu'ils s'appuient ou non sur la mécanique des milieux continus.

1.2.1 Description de quelques comportements d'objets simulés

Nous allons commencer par établir une petite classification du vocabulaire utilisé pour décrire les comportements mécaniques d'un objet. Pour cela, nous nous inspirons des taxonomies existantes dans le milieu de la modélisation mécanique [CG02] et plus particulièrement de la théorie de l'élasticité. Cette théorie énonce que tout objet est a priori déformable s'il subit des forces suffisamment importantes. Comme la modélisation systématique avec des objets déformables est bien trop coûteuse pour la simulation temps-réel, on doit recourir à des modèles simplifiés. Néanmoins, ces autres modèles sont définis à partir de cette notion d'objet déformable.

Objet déformable : on parle d'objet déformable lorsque les distances et les angles entre les particules de matière qui le composent varient en fonction de la configuration. Dans le cadre de la théorie de l'élasticité, un objet est associé à une forme de référence, dite de *repos* ; s'il est

soumis à des efforts externes (des forces), l'objet se déforme. Lorsque les efforts disparaissent, l'objet reprend sa forme de référence. La capacité générale d'un objet à se déformer est désignée par le terme de *rigidité*. Suivant le modèle choisi pour la simulation, les paramètres pour décrire la rigidité varient : module de Young et coefficient Poisson en élasticité linéaire, raideur des ressorts sur un maillage masses-ressorts, etc.., .

Objet rigide : si par hypothèse, on suppose que la rigidité d'un objet est infinie, on peut alors parler d'objet rigide. L'utilisation d'une modélisation sous la forme d'objet rigide est très avantageuse et largement présente en informatique graphique. En effet, la configuration d'un objet peut alors être décrite de manière très concise sous la forme d'une position (un vecteur) et d'une orientation (un quaternion). En outre, cette simplicité de représentation est liée à des équations dynamiques nettement simplifiées. Une bonne introduction à la simulation d'objet rigide est présente dans [Ebe03].

Objet fluide : à l'opposé des objets rigides, de raideur infinie et indéformables, on trouve les objets infiniment déformables : type fluide. Dans le domaine de la simulation, les fluides occupent une place assez particulière. Ils ont été initialement simulés à l'aide de méthodes lagrangiennes,[3] comme les forces de Lennard-Johnes ou à l'aide de méthodes SPH [MCG03]. Les fluides peuvent aussi être simulés sous la forme d'un environnement, d'un milieu, dans lequel évoluent les particules du fluide et les différents objets de la scène (approche eulérienne). Le lecteur intéressé par les approches eulériennes pour simulation de fluide pour l'informatique graphique peut consulter [Sta99][LGF04][LIG06].

Objet ductile/plastique : les objets, lorsqu'ils sont soumis à des efforts trop importants, ne reprennent plus leur position de repos. Ils conservent alors une partie de la déformation. On parle alors de déformation plastique et d'objet ductile [OH99].

Objet découpable, objet fragile : au sens de la théorie de l'élasticité, dans un objet déformable, deux particules de matière infinitésimalement voisine le resteront, même si le chemin qui les lie s'allonge du fait des déformations. Un objet déformable est considéré découpable, ou fragile, si une (ou plusieurs) liaison élastique, entre deux particules du matériau, peut être rompue pendant la simulation. En pratique, les objets découpables peuvent servir à modéliser un organe subissant les découpes définies par un médecin via un scalpel ou encore, de façon plus générale, le comportement d'un objet qui se fracture, se casse, sous l'action de sollicitations (trop) importantes.

Lois de comportement linéaire/non-linéaire : on appelle loi de comportement la description, sous forme mathématique, traduisant comment le matériau réagit aux sollications internes et externes. Ainsi, un matériau qui respecte la loi de Hooke offre une "résistance" linéaire aux sollicitations externes. En informatique graphique, on privilégie souvent la loi de Hooke,qui aboutit à un système numérique linaire et donc plus simple et plus rapide à résoudre. Cependant, de plus en plus d'auteurs cherchent à modéliser des objets ayant des lois de

[3] Les modélisations par "objets" s'appuient sur une formulation lagrangienne de la matière et des équations du mouvement, tandis que les fluides adoptent en général une formulation dite eulérienne où le milieu est fixe et les particules se déplacent. Dans l'ensemble de la thèse, on manipule des objets sous une formulation lagrangienne.

comportement "non-linéaires" ou incluant des propriétés comme l'anisotropie ou l'hystérésie (les phases d'élongation et de retour à la position initiale n'ont pas la même loi de comportement).

Anisotropie : certains objets ont des propriétés mécaniques qui diffèrent en fonction des axes suivant lesquels s'applique la sollicitation. Les muscles, par exemple, présentent une anistropie importante en raison de leur structure fibrée. Un effort perpendiculaire à la structure fibrée entraînera une réaction différente de celle obtenue à partir d'un effort exercé suivant l'axe des fibres.

1.2.2 Simulation basée sur la mécanique des milieux continus

La mécanique des milieux continus forme un cadre théorique basé sur l'hypothèse suivante : la matière, au lieu d'être un assemblage de particules discrètes comme le décrit la physique, est vue à une échelle suffisamment grande pour apparaître tel un milieu continu [Guaa]. Une fois posée cette hypothèse, les contraintes mécaniques (comme la préservation des distances ou des volumes) peuvent s'exprimer, de manière "infinitésimale", sous la forme d'équations aux dérivées partielles du champ matériel. Dans le cadre de la mécanique des milieux continus, on trouve un certain nombre de théories permettant la modélisation d'objets déformables. La théorie des poutres [Guac] s'intéresse à la simulation d'objets filiformes[4]. La théorie des coques [Guab] s'affaire à modéliser les déformations de surface, et la théorie de l'élasticité [Guad] s'occupe, quant à elle, à modéliser des objets tridimensionnels déformables. Les équations qui décrivent le comportement des objets sont fournies par les mécaniciens. La discrétisation des équations physiques sur le domaine géométrique se traduit par la production de systèmes d'équations qu'il faut résoudre numériquement. La complexité du système d'équations à résoudre dépend : de la représentation géométrique adoptée ; des lois de comportement choisies, et de l'utilisation d'une formulation forte ou non de ces lois (1.2.2.0). L'objectif à atteindre, quand on veut faire de la simulation pour les environnements virtuels, est de trouver un modèle simulant le plus grand éventail possible de comportements tout en limitant la quantité de calculs.

La théorie de l'élasticité

Dans cette thèse, on s'intéresse à la simulation d'objets déformables volumiques. On privilégie donc la théorie de l'élasticité et on laisse de côté les théories des poutres (voir [LCDN06] pour des utilisations concrètes) et des coques. Dans ce cadre, un objet déformable est défini par sa forme au repos, sa loi de comportement et les paramètres mécaniques. Lorsque l'objet est soumis à des contraintes externes (appuyer dessus, par exemple), il se déforme. Une fois la contrainte supprimée, l'objet retourne à sa configuration initiale. La manière dont l'objet se déforme est décrit par une loi de comportement. La théorie de l'élasticité forme un cadre théorique très complet et complexe pour la simulation mécanique. Nous n'en abordons que la partie strictement nécessaire à la mise en place de méthodes de simulation dans les environnements virtuels. Soit M l'ensemble continu de matière dans $\mathbb{R}^3$ définissant un objet. Chacun des points matériels p de M possède une position au repos $\mathbf{x}^r = [x^r_x, x^r_y, x^r_z]^T$. Lorsque

[4] dont l'une des dimensions est nettement plus grande que les autres

l'objet se déforme sous l'action de forces externes, les points matériels se déplacent et occupent une nouvelle position $\mathbf{x}^d = [x_x^d, x_y^d, x_z^d]^T$ (fig. 1.9).

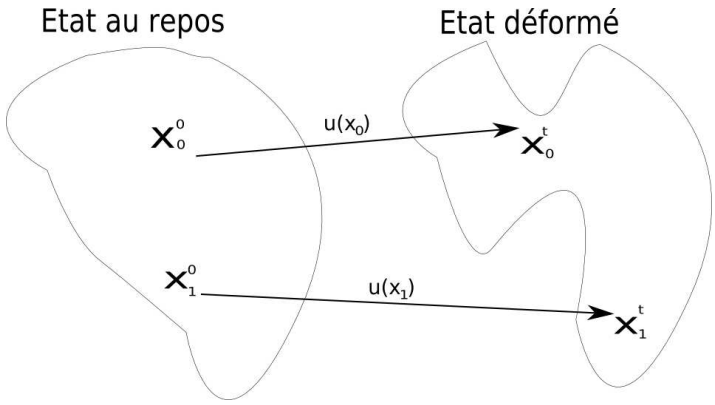

Fig. 1.9 : *Le champ vectoriel $\boldsymbol{u}(x)$ associe à tout point matriel de position au repos $\boldsymbol{x}^r$, le vecteur le liant à sa position déformée $\boldsymbol{x}^d$.*

On note $\mathbf{u}(p)$ la fonction qui associe la position d'un point matériel p dans la configuration au repos, à la position d'un point dans la configuration déformée. Cette fonction $\mathbf{u}(p)$, nommée champ de déplacement, associe à un point p le vecteur déplacement défini comme suit :

$$\mathbf{u}(p) = \mathbf{x}^d - \mathbf{x}^r = [u_x = x_x^d - x_x^r, u_y = x_y^d - x_y^r, u_z = x_z^d - x_z^r]^T \tag{1.1}$$

La fonction $\mathbf{u}(p)$ est constante (pour tout $p \in M$) si l'objet effectue une translation sans déformation. Lorsque l'objet se déforme, l'étude des dérivées partielles du champ de déplacement $u = \mathbf{u}(x)$ nous renseigne sur les variations de position entre des particules *infinitésimales* proches. Les variations du champ de déplacement sont en général représentées sous la forme d'une matrice dont les lignes stockent les dérivées partielles, suivant x, y, z, des différentes composantes u_x, u_y, u_z du champ vectoriel.

$$\nabla u = \begin{pmatrix} \frac{\delta u_x}{\delta x} & \frac{\delta u_x}{\delta y} & \frac{\delta u_x}{\delta z} \\ \frac{\delta u_y}{\delta x} & \frac{\delta u_y}{\delta y} & \frac{\delta u_y}{\delta z} \\ \frac{\delta u_z}{\delta x} & \frac{\delta u_z}{\delta y} & \frac{\delta u_z}{\delta z} \end{pmatrix} \tag{1.2}$$

Le tenseur de déformation : le tenseur de déformation mesure l'élongation subie entre deux points M, N initialement voisins [5]. Imaginons un petit cube de matière, ce cube sera transformé en parallélépipède après déformation. Il y a, d'une part une élongation (ou contraction) des différentes arêtes de ce cube selon les trois axes, mais aussi une variation de l'angle liant les axes de ce cube. Ces nombres sont organisés dans un « tableau », et donc représentent la déformation sous la forme d'une matrice (ou tenseur). Le tenseur ϵ_g est appelé tenseur de Green/Lagrange et traduit l'élongation de MN entre la position au repos et la

[5] On utilise le terme voisin au sens infinitésimal.

position déformée par la formule :

$$\epsilon_g = \frac{1}{2}(\nabla u.\nabla u^T + \nabla u + \nabla u^T) \tag{1.3}$$

$$\tag{1.4}$$

En pratique, dans le cadre des petits déplacements, on remplace souvent le tenseur de Green/Lagrange par sa linéarisation, le tenseur de Cauchy, plus simple à calculer :

$$\epsilon_c = \frac{1}{2}(\nabla u + \nabla u^T) \tag{1.5}$$

Ces tenseurs ϵ_g et ϵ_c décomposent la mesure de la déformation suivant les différentes composantes du champ de déplacement.

Le tenseur de contrainte : le tenseur de contrainte est obtenu à partir du tenseur de déformation qui mesure la quantité de déformation et de la loi de comportement qui lie cette mesure de la déformation à une force qui tend à ramener la matière à son état de repos. Dans le domaine de l'informatique graphique, on utilise essentiellement une loi de Hooke qui traduit une variation linéaire de la force de rappel en fonction de la mesure de la déformation (le tenseur de déformation ϵ) :

$$\omega = C.\epsilon \tag{1.6}$$

Avec C, représentant les différentes composantes de la lois de comportement (C varie suivant le comportement matériel à approcher : linéaire, non-linéaire, isotrope...). A partir du tenseur de contrainte ω, on calcule l'énergie potentielle accumulée par l'objet qui se déforme. En dérivant cette énergie potentielle, on calcule les forces élastiques qui ramenènent l'objet déformé à sa position initiale.

Les formulations faibles et fortes

Le calcul variationnel, par analogie au calcul différentiel, étudie, non plus les variations d'un paramètre, mais les variations d'une fonction [Weia]. Une application particulière du calcul variationnel est de fournir une série d'équivalences entre différentes formulations d'équation aux dérivées partielles. L'intérêt de ces formulations, du point de vue de la simulation et de l'informatique graphique, réside dans le fait que certaines d'entre-elles sont moins coûteuses. Dans la littérature, on parle souvent de formulations faibles et fortes des équations physiques.

Les formes fortes : supposons qu'on veuille simuler un objet à l'aide de l'équation de Navier :

$$D''(x) = \rho.\mathbf{a} = \mu.\mathbf{div}(\mathbf{grad}(\mathbf{u}(x))) + (\lambda + \mu).\mathbf{grad}(\mathrm{div}(\mathbf{u}(x))) \tag{1.7}$$

Cette équation décrit l'accélération subie par une particule de matière en fonction du déplacement de ces voisines et tend à homogénéiser le champ de déplacement u, en tout point de l'objet. Les termes μ et λ étant des coefficients décrivant le comportement matériel (rigide ou non) de l'objet. Une fois l'accélération calculée en un ensemble d'échantillons,

cette accélération doit être intégrée sur le volume de l'objet et reportée sur les *degrés de liberté*[Duf04] :

$$\int_{\Omega} g(p).D''(x) \tag{1.8}$$

La fonction g lie les degrés de liberté aux échantillons qui ont servi à évaluer l'équation de Navier $D''(x)$. D'un point de vue pragmatique, l'équation de Navier suppose que le champ de déformation soit deux fois dérivables. Pour offrir un tel champ matériel, deux méthodes prédominent : les techniques à base d'interpolation/approximation aux moindres carrés comme dans [MKN+04], et les méthodes à base de différences finies comme dans [DDCB01].

Les formes faibles : il est possible de remplacer les équations intégrales de 1.8 par des équations équivalentes ([Guaa, page 46]) ; l'intérêt d'un tel remplacement étant : le degré de dérivation de la fonction $u(x)$ est abaissé, tandis que celui de $g(x)$ est augmenté. L'abaissement du degré de dérivation se traduit par l'utilisation d'une interpolation du champ de déplacement à l'ordre C^0. Une approximation du champ de déplacement linéaire par morceaux est donc utilisable. Une bonne illustration de l'utilisation des formes faibles pour la simulation est la méthode des Éléments Finis qui se contente bien souvent d'une interpolation linéaire.

Pour présenter et cataloguer les différentes méthodes de simulation d'objets déformables nous utilisons deux critères de classification. Le premier critère indique la présence ou non d'un maillage comme représentation géométrique ; un aspect particulièrement intéressant en lien avec les différents travaux sur la découpe d'objet des chapitres (2) et (sec. 1.5). Le second critère de classification est établi en fonction de l'emploi d'une formulation forte ou faible des équations à résoudre. L'intérêt de celui-ci est de relier la simulation mécanique avec le degré de dérivation de la représentation géométrique qui la sous-tend ; lien étudié plus en détail dans le chapitre (2).

Maillage et formulation forte : les différences finies

La méthode des différences finies est probablement la méthode de résolution d'équations différentielles la plus facile d'accès. Elle s'appuie sur une discrétisation particulière, en forme de grille, des opérateurs différentiels spatiaux des équations. Le développement de Taylor à l'ordre n d'une fonction $f(x)$ de continuité C^n au voisinage h d'un point x est donné par :

$$f(x+h) = f(x) + \sum_{i=1}^{n} \frac{h'i}{i^i} f^i(x) + \omega(x) \tag{1.9}$$

Si on se contente de l'ordre 1 du développement de Taylor, on obtient la formule : $f(x+h) = f(x) + h.f^i(x) + \omega(x)$, ce qui permet d'obtenir l'approximation suivante de la dérivée de f au point x :

$$f^1(x) \approx f^1_f(x) = \frac{f(x+h) - f(x)}{h} \tag{1.10}$$

Cette approximation est appelée *first forward difference*. L'approximation duale est appelée *first backward difference*, définie par :

$$f^1(x) \approx f^1_b(x) = \frac{f(x) - f(x-h)}{h} \tag{1.11}$$

En suivant le même principe, les dérivées d'ordre supérieur sont approchées comme suit :

$$f^2(x) \approx \frac{f^1(x) - f^1(x-h)}{h} = \frac{f(x-2h) - 2f(x-h) + f(x)}{h^2} \tag{1.12}$$

L'utilisation des différences finies pour résoudre les équations de la mécanique remonte au travaux précurseurs de Terzopoulos dans [TPBF87] et [TF88] pour discrétiser les opérateurs différentiels des équations du mouvement. Dans sa thèse, Gilles Debunne [Deb00] utilise le développement de Taylor pour approcher les opérateurs différentiels **div** et **grad** de l'équation de Navier. Opérateurs différentiels discrets qu'il cherche à étendre pour traiter les cas où la discrétisation n'est pas répartie équitablement sur une grille.

En résumé, l'utilisation des différences finies est simple à comprendre, à mettre en oeuvre en plus d'être efficace du point de vue des performances. Elle offre la possibilité d'approximer les opérateurs différentiels, quel que soit leur ordre de différienciation ; cela permet l'utilisation des équations mécaniques dans leur formulation forte. Par contre, son usage suppose une discrétisation assez contraignante sous la forme d'une grille aux points équirépartis[6].

Maillage et formulation faible : les Élément Finis

La méthode des Éléments Finis est une technique très générale de résolution numérique d'équations aux dérivées partielles [Guae]. En pratique, le domaine (la géométrie) sur lequel on tente de résoudre l'équation est pavé sous la forme d'un maillage composé d'éléments simples (des triangles en 2d ou des tétraèdres en 3d). Le pavage est associé à un processus d'interpolation liant les échantillons et qui permet l'intégration des équations mécaniques sur l'ensemble du volume de l'objet. Les équations à résoudre sont alors mises sous formes faibles, permettant l'utilisation d'une interpolation linéaire du champ de déplacement ; cette interpolation étant calculée à partir des coordonnées barycentriques (2.2.1.0) des éléments composant le maillage. Une fois la méthodes des Éléments Finis appliquée, il reste à résoudre le système d'équations différentielles suivant (dans le cas dynamique) :

$$\mathrm{M}\ddot{\mathbf{u}} + \mathrm{D}\dot{\mathbf{u}} + \mathrm{K}(\mathbf{u}).\mathbf{u} = \mathbf{f} \tag{1.13}$$

Ou encore, la forme, dite statique, qui néglige les termes dynamiques (dépendant de la vitesse et de l'accélération) :

$$\mathrm{K}(\mathbf{u}).\mathbf{u} = \mathbf{f} \tag{1.14}$$

Avec $\mathbf{u}$ le vecteur déplacement, $\mathrm{K}(\mathbf{u})$ la matrice de rigidité et $\mathbf{f}$ les forces externes (comme la pression).

On a vu dans le cadre de la théorie de l'élasticité que suivant la quantité de déplacement subie par le matériau à simuler, on utilisera soit les tenseurs de Green/Lagrange, soit le tenseur de Cauchy pour mesurer les déformations. Comme le tenseur de Cauchy est nettement plus simple à calculer, il sera souvent choisi en simulation temps-réel. Le choix entre le tenseur de

[6] Dans sa thèse, Gilles Debunne souhaite utiliser un maillage différent de la grille régulière et propose une extension au développement de Taylor des opérateurs différentiels. Il remarque cependant que cette approximation donne des résultats invalides. Dans ce contexte, un lien intéressant est à faire avec les méthodes sans maillage qui permettraient cette généralisation.

Cauchy ou celui de Green/Lagrange est lié à la "complexité géométrique" du mouvement. De manière analogue, suivant la complexité mécanique de l'objet à modéliser, on pourra utiliser une loi de comportement linéaire comme la loi de Hooke ou des lois non-linéaires plus réalistes mais aussi complexes.

Lorsque le tenseur de Cauchy est utilisé avec la loi de Hooke, le terme $\mathrm{K}(\mathbf{u})$ de l'équation (1.13) devient indépendant de $\mathbf{u}$ et l'inversion de la matrice K peut être précalculée. Ainsi dans le cadre d'une simulation statique, l'unique calcul à faire dans la boucle de simulation est un produit matrice/vecteur $\mathbf{u} = \mathrm{K}^1.\mathbf{f}$ [DCA99] (on connaît K et $\mathbf{f}$) ; la position déformée étant calculée par $\mathbf{x}_d = \mathbf{x}_r + \mathbf{u}$. Dans le cadre de la simulation dynamique, on ajoute à la discrétisation spatiale, une discrétisation temporelle. On veut calculer les forces à un instant donné à partir du déplacement. Suivant le schéma d'intégration temporel choisi, l'équation (1.14) aboutit à différents systèmes d'équations. En général, on privilégie les schémas explicites qui sont simples à résoudre et peu coûteux en comparaison aux schémas implicites. L'utilisation d'un schéma d'intégration explicite revient à calculer les forces $\mathbf{f}$ à l'instant $t + \Delta t$, à partir des positions connues à l'instant précédent t de la simulation :

$$\mathbf{f}(t + \Delta t) = \mathrm{K}.\mathbf{u}(t) \tag{1.15}$$

Dans la communauté de l'informatique graphique, les Éléments Finis explicites sont à la base du modèle masses-tenseurs [DCA99]. Ce modèle masses-tenseurs a la particularité de ne pas utiliser une matrice de rigidité K assemblée (globale), mais des matrices associées aux différents éléments (on parle alors de matrices élémentaires K^{Elem}). L'usage de matrices élémentaires, locales, est intéressant car il rend possible des modifications ponctuelles de l'objet (découpe, fracture).

Les premières utilisations de la méthode des Éléments Finis étaient réduites au domaine des petits déplacements (tenseur de déformation linéaire) et des petites déformations (tenseur de contrainte linéaire). A cause du tenseur de déformation linéaire (de Cauchy), une partie de la rotation des objets est interprétée comme une déformation et entraîne la production de forces non désirées (fig. 1.10). Ces forces, au lieu de ramener l'objet à son état initial, vont au contraire l'en éloigner. Pour contourner ce problème, deux approches sont possibles : la première, [PDA02] étend la méthode des masses-tenseurs à l'utilisation du tenseur de Green/Lagrange, et la seconde méthode, plus récente, a pour objectif de supprimer la composante rigide de rotation du mouvement de l'objet (on parle de méthodes *corotationnelles*). L'intérêt des méthodes corotationnelles est de conserver le tenseur de déformation linéaire alors que l'objet subit des grandes rotations. On trouve des formulations corotationnelles dites globales, qui cherchent à associer une seule rotation par objet déformable [Fel00][Dur04], ainsi que des approches locales, qui associent une rotation soit aux éléments du maillage [NPF05], soit aux noeuds [MMD+02]. Dans [HS04], les auteurs recommandent les méthodes locales par éléments, car les approches par noeuds introduisent des forces fantômes qui violent le principe physique d'action-réaction. Les méthodes corotationnelles sont assez proches de la proposition de [ITF06] de gestion d'Éléments Finis inversibles. Gérer l'inversion des éléments est une avancée importante car dans la méthode Éléments Finis initiale, l'inversion d'un élément entraîne l'arrêt de la simulation.

Quand on utilise la méthode des Éléments Finis, une attention particulière doit être apportée à la génération du maillage [MT][ACY+05]. Il faut que celui-ci soit suffisamment fin pour représenter correctement les déformations des objets ; mais, s'il est trop fin, les

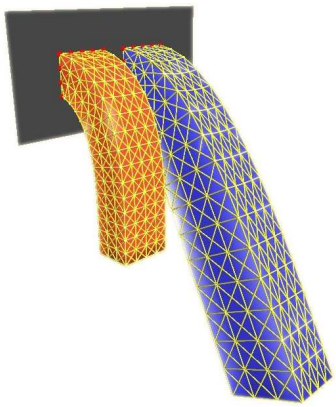

Fig. 1.10 : *A droite, utilisation du tenseur de Cauchy : des déformations non désirées apparaissent. A gauche, l'utilisation d'une méthode à base de corotationnel.*

éléments risquent de s'inverser en cas de grandes déformations. Il faut aussi éviter les éléments non réguliers ou qui ont leurs quatre sommets à la surface de l'objet [Mol04]. Suivant les algorithmes, l'obtention d'un maillage prend de plusieurs minutes à plusieurs heures de calcul. La qualité du maillage influe directement sur la difficulté qu'il y a à découper des objets animés par les Éléments Finis (sec. 1.5).

Les méthodes sans maillage avec formulation faible

Dans le cadre de la simulation d'objets déformables basée sur la mécanique des milieux continus, certaines méthodes se différencient des approches présentées précédemment par le fait qu'elles ne s'appuient pas sur un maillage pour discrétiser les équations différentielles à résoudre. On parle donc de méthodes sans maillage (meshless ou meshfree). De telles méthodes semblent plus adaptées à la simulation d'objets déformables dont la topologie change fortement, et s'appuient sur une formulation continue du champ matériel obtenue par une technique d'interpolation (elle-même sans maillage). Dans la plupart des cas, la méthode d'interpolation dérive de l'approximation aux moindres carrés mobiles (MLS), présentée dans 2.2. Néanmoins, l'usage d'autres techniques d'interpolation, telle que l'approximation par voisinage naturel [SMB98], est possible mais moins courante.

Une présentation très complète des méthodes sans maillage, centrée sur la mécanique de la rupture, est disponible dans [FM03]. On y trouve l'évaluation d'un grand nombre de techniques d'interpolation permettant, à l'aide d'un ensemble donné d'échantillons, de reconstruire une approximation continue du champ de déplacement. Pour les lecteurs francophiles, une thèse récente traite de ce sujet avec, pour objectif, la modélisation de fracture [Duf04]. Pour l'informatique graphique, on consultera [MKN+04], puis [PKA+05] dans lesquels les auteurs proposent d'utiliser la technique d'interpolation aux moindres carrés mobiles pour animer des objets déformables, plastiques, ductiles et fragiles.

Une difficulté supplémentaire apparaît lorsqu'on simule des objets à l'aide d'une méthode sans maillage : c'est la représentation de la surface de l'objet. On observe qu'en pratique, les techniques de simulation sans maillage sont associées à l'utilisation de technique d'affichage

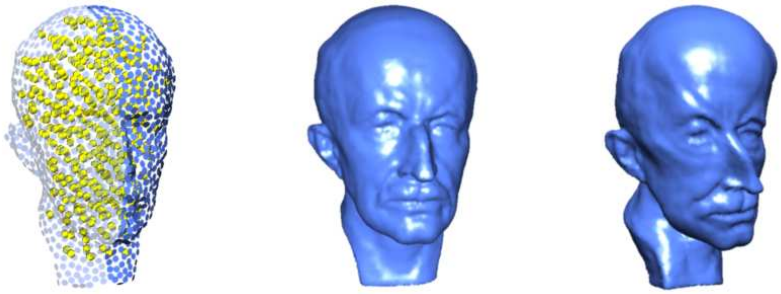

Fig. 1.11 : *Technique d'animation à base de points : à gauche, en jaune, les particules simulées et en bleu, les éléments de surface utilisés par le rendu sans maillage (extrait de [MKN+ 04]).*

sans maillage comme les surfaces à base de points [Pau03], ou encore les surfaces implicites [DC95].

Les méthodes sans maillage avec formulation forte

Les processus d'interpolations qui sont décrit dans la section précédente, sont suffisamment dérivables pour être utilisable comme base d'une simulation avec une formulation forte des équations physiques. L'usage de formulations fortes permet la simulation d'objets déformables sans passer par une phase où les équations physiques sont intégrées sur le domaine matériel ; ceci diminuant la quantité de calculs. De telles techniques sans maillage sont apparues relativement tôt dans le domaine de l'informatique graphique : ainsi la méthode des SPH (Smoothed Particle Hydrodynamics) est employée par Matthieu Desbrun, dans[DC96], pour modéliser des objets mous (fig. 1.12) puis, plus récemment, pour simuler des fluides en temps réel [MCG03].

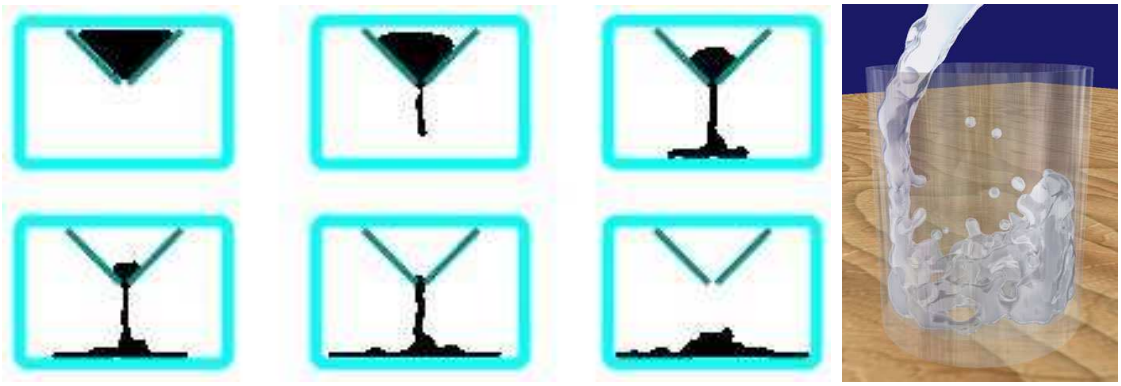

Fig. 1.12 : *Quelques résultats obtenus par les méthodes SPH. A gauche, extrait de [DC96] et, à droite extrait de [MCG03].*

1.2.3 Méthodes de simulation basées sur une géométrie discrète (ou particulaire)

Les méthodes basées sur la mécanique des milieux continus présentent l'avantage d'un cadre théorique robuste et général à la simulation d'objets déformables ; ce cadre favorise la modélisation d'objets déformables aux lois de comportement bien précises et dont les paramètres peuvent être identifiés (par mesures mécaniques [CG02], par vidéo [BTH+03]). Cependant, la mécanique des milieux continus n'est pas la seule modélisation possible pour simuler des objets déformables. En effet, un certain nombre de modèles ne se basent pas sur l'hypothèse d'un milieu matériel continu. Au lieu d'exprimer les lois de comportement de la matière à l'aide d'équations aux dérivées partielles, des invariants géométriques sont définis de manière à préserver certaines quantités comme les distances, les angles, les volumes, ... En pratique, les objets sont considérés comme un ensemble de n particules liées par des forces d'interaction [CLF93]. Ces particules sont animées à l'aide du formalisme physique de Newton et du principe fondamental de la dynamique associant l'accélération d'une particule à la somme des forces s'y appliquant :

$$\sum_{i=0}^{n} F_i = m_i.\mathbf{a} \tag{1.16}$$

On distingue trois catégories de méthode selon que les forces, s'appliquant sur une particule, soient définies à l'aide d'un maillage, sans maillage mais basées sur une mesure de distance et enfin, sans maillage ni mesure de distance. Dans la mécanique des milieux continus, les lois de comportement sont indépendantes de la géométrie ; il suffit qu'un processus d'interpolation discrétise cette géométrie pour pouvoir résoudre le problème. Ce n'est pas le cas des méthodes géométriques, nettement plus dépendantes de la description géométrique de l'objet pour calculer les forces assurant la cohésion entre les particules. Après discrétisation géométrique, toute les méthodes sont analogues aux méthodes particulaires ; c'est pour cela que nous préférons le terme "méthode géométrique" à celui de méthode particulaire, celui-ci étant ambigu quant à la place des méthodes masses-tenseurs [Mes03].

Les méthodes géométriques avec maillage : les masses-ressorts

Dans le cadre de la technique des masses-ressorts, la cohésion entre les différentes particules de matière est réalisée par des ressorts qui relient lesdites particules [Mil88][Jou97]. Le ressort le plus simple tend à préserver la distance séparant deux particules. Il rapproche ainsi deux particules qui s'éloignent et éloigne deux particules qui se rapprochent. Ce ressort est appelé ressort d'élongation et se définit à partir de deux particules A et B. La longueur du ressort au repos est noté l_0. Son élasticité (la rigidité) est modélisée par un coefficient scalaire noté k. Les forces liant les particules A et B sont calculées :

$$F_{A\to B} = k.(||AB|| - l_0).\frac{AB}{||AB||} \tag{1.17}$$

$$F_{B\to A} = -F_{A\to B} \tag{1.18}$$

Le comportement du ressort d'élongation est enrichi par des termes de dissipations internes. Les forces sont calculées sur l'axe liant les particules ; on parle alors de ressort 1d. A partir de

ce ressort 1d, on construit des objets complexes et de dimension supérieure par assemblage de ressorts. Cela permet de simuler des domaines 2d (surface) et 3d (volume). La figure (1.13) montre quelques exemples d'assemblage de ressorts modélisant des contraintes surfaciques et volumiques. En plus des ressorts d'élongation, il existe, dans la littérature, d'autres types de ressorts : angulaire, de courbure, de torsion ; de tels ressorts sont employés pour la simulation réaliste de tissu dans, par exemple, [NNB04].

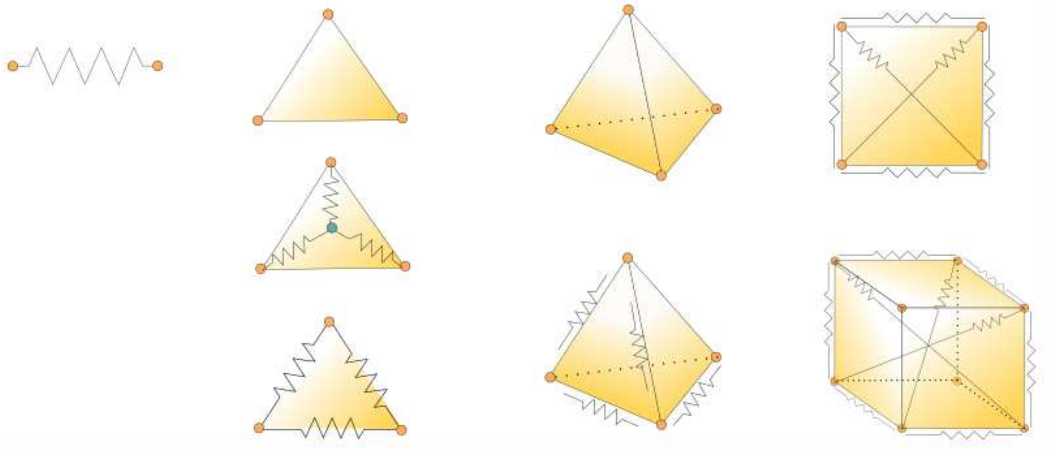

Fig. 1.13 : *Les différents types de ressort. De 1D à 3D*

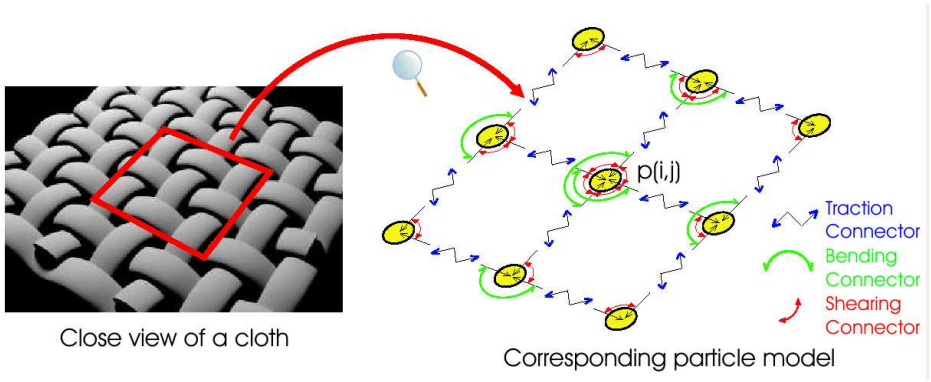

Fig. 1.14 : *Modélisation précise de tissu à l'aide de modèle masses-ressorts (extrait de [NNB04])*

Plutôt que de procéder à des assemblages de ressorts 1d, les auteurs de [THMG04] proposent une généralisation du principe des maillages masses-ressorts en introduisant des ressorts de surface (2d) ou de volume (3d). Ces "ressorts" traduisent alors correctement les déviations des invariants mécaniques, car basés sur une mesure de même dimension que la contrainte modélisée : volume en 3d, aire en 2d, et bien sûr, longueur en 1d. L'accélération des particules est calculée à partir des forces générées à l'aide du maillage masses-ressorts. A partir ce cette accélération les vitesses et positions sont déduites par intégration numérique. En

pratique, la simulation d'objets réels suppose d'utiliser des ressorts de raideur élevée. Dans ce contexte, l'intégration numérique des équations différentielles produites par une modélisation à base de maillage masses-ressorts est difficile à résoudre numériquement (on parle alors de problème raide [Hil02]). La résolution de ces problèmes oblige à garder un pas de temps suffisamment petit pour éviter la divergence de la méthode numérique. Une autre solution au problème de stabilité de l'intégration numérique, assez coûteuse du point de vue des temps de calcul, consiste à utiliser des intégrateurs qui soient inconditionnellement stables comme *l'Euler implicite* [Bar89].

Les méthodes géométriques sans maillage

La technique des masses-ressorts suppose de représenter la géométrie de l'objet à l'aide d'un maillage. De manière analogue aux méthodes continues avec ou sans maillage, on peut simuler des objets faiblement structurés (ex : fluides) à l'aide de méthodes géométriques sans maillage. Les ressorts sont remplacés par d'autres formulations des forces de liaison. Par exemple : les forces de Lennard-Jones, définies à l'aide des distances interparticules, sont employées pour modéliser des fluides [MP89] ou des objets déformables non structurés comme dans [DC95]. D'une manière plus générale, l'environnement Cordis-Anima[CLF93] propose une technique générique de simulation de modèles particulaires. Les objets sont représentés à l'aide de particules reliées par des composants d'interactions génériques (fig. 1.15) ; ces composants d'interaction étant aussi bien des composants de type ressorts que des composants plus complexes. Cet environnement permet ainsi la modélisation de lois de comportements très complexes tels : l'anisotropie, l'hystérésie ou la non-linéarité.

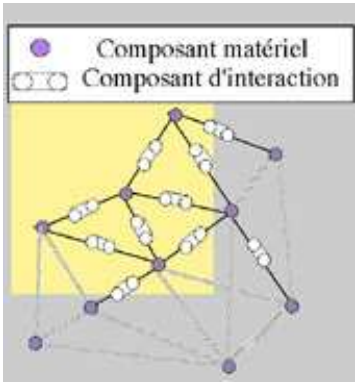

Fig. 1.15 : *Dans Cordis-Anima, tous les modèles sont basés sur la même représentation à base de particules. Ces particules sont ensuite liées entre elles par des liens d'interaction.*

Méthode géométrique réellement sans maillage

Dans [MHTG05], les auteurs présentent une méthode sans maillage tout à fait originale et donc difficile à classer. C'est une méthode qui ne dérive pas de la mécanique des milieux continus. Elle est sans maillage mais, contrairement aux autres approches, elle ne s'appuie

pas sur un calcul de distance (comme les forces de Lennard-Jones, les SPH, ou les ressorts). Pendant l'animation d'un objet, les particules sont animées comme des particules libres que rien ne relie. A partir du nuage, résultat du mouvement libre, la position sans déformation est obtenue par extraction de la composante rigide du mouvement par une méthode de Shape-Matching. La méthode du Shape-Matching, décrite plus en détail dans le chapitre (5), permet de trouver l'objet non déformé le plus proche (au sens des moindres carrés) de la position du nuage de particules ; la différence entre la position résultant du mouvement libre et celle obtenue par Shape-Matching permet de calculer les accélérations de particules afin de ramener celles-ci à leur position non déformée. Cette méthode est intéressante pour la simulation d'objet déformable de rigidité élevée. En plus, elle garantit la stabilité de la phase d'intégration numérique. Cette méthode est employée, en raison de sa stabilité, pour le modèle d'objets déformables et découpables présenté dans le chapitre 5.

1.3 Modélisation des interactions

Les méthodes exposées depuis le début de ce chapitre traitent uniquement du calcul du comportement d'un seul objet. Lorsque plusieurs objets sont simulés dans le même environnement, il faut aussi traiter les interactions qui peuvent se produire. Sans prise en compte de ces interactions, les objets "s'ignorent" mutuellement et peuvent se traverser les uns les autres. Pour modéliser les interactions, on emploie la notion de contrainte (comme pour la simulation mécanique des objets (sec. 1.2.2)). Une contrainte qui sert à modéliser une loi de comportement propre aux objets est qualifiée de "contrainte interne" ou "d'intrinsèque". Lorsque plusieurs objets entrent en collision, les contraintes sont qualifiées "d'externes" ou "d'extrinsèques". Comme pour les contraintes intrinsèques, la modélisation des interactions suppose que la contrainte soit explicitée sur une géométrie. Pour la gestion des collisions, cette géométrie n'est pas connue a priori et doit être établie pendant la simulation. C'est la phase de *détection des intersections* (ou *de détection de collision* par abus de langage, car sous-entendant une finalité physique). Cette phase de détection des intersections est coûteuse en temps de calcul et demande, dès lors, l'usage d'algorithmes d'accélération (voir 1.3.5).

Dans la littérature, la modélisation des collisions entre objets est bien souvent présentée dans l'ordre : détection des intersections, puis modélisation de la réponse. Dans ce mémoire, nous avons privilégiée l'ordre inverse, car il met plus en valeur le lien entre la propriété physique modélisée (non-intersection) explicitée sur une géométrie donnée. En fonction de la modélisation physique choisie, les informations retournées par phase de détection des intersections diffèrent. Suivant qu'il faut un booléen, ou les paires de triangles en intersection, ou encore une mesure de l'intersection, les algorithmes de détection employés sont différents.

1.3.1 Gestion des contraintes dans les systèmes dynamiques

Les contraintes permettent, de manière générale, d'enrichir un modèle en lui imposant des configurations à respecter. Ces configurations peuvent être de nature différente : géométrique ou physique. Un exemple de contrainte géométrique classique consiste à dire qu'un point de l'objet est fixé dans l'espace. Au final, les contraintes sont exprimées sous forme d'équations/inéquations mathématiques. Si la contrainte s'exprime par des

équations, il s'agit de contrainte *bilatérale* ; lorsque l'expression s'appuie sur une inéquation, le terme de contrainte *unilatérale* est retenu [Mes03]. Outre cette séparation entre contraintes *unilatérale/bilatérale*, on distingue souvent celles qui font intervenir les degrés de liberté de l'objet (contraintes *holonomes*) de celles faisant aussi intervenir les dérivées suivant le temps des degrés de liberté (contraintes *cinématiques*). De manière plus générale, un système dynamique contraint peut s'écrire sous la forme [Len04] :

$$\begin{cases} \text{minimiser } f(\mathbf{q}(t), \dot{\mathbf{q}}(t), t) \\ \quad \text{sujette à } \begin{cases} g_i(\mathbf{q}\ (t), \dot{\mathbf{q}}(t), t) \leq 0 & \forall_i \in \{0..m\} \\ h_j(\mathbf{q}\ (t), \dot{\mathbf{q}}(t) = 0 & \forall_j \in \{0..j\} \end{cases} \end{cases} \tag{1.19}$$

Avec $\mathbf{q}(t)$, les degrés de liberté du système dynamique, $\dot{\mathbf{q}}(t)$ leur dérivée première par rapport au temps et t le temps. On cherche à minimiser une fonction f, décrivant le mouvement des objets, tout en vérifiant les j inéquations h_j et les m équations g_i. Dans le contexte de la gestion des interactions entre objets déformables, on observe que les degrés de liberté $\mathbf{q}(t)$ peuvent appartenir à des objets différents. Une contrainte – *holonomme* ou *cinématique*, *unilatérale* ou *bilatérale* – qui ne concerne que les degrés de liberté d'un objet donné, est nommée *contrainte non-couplante*, par opposition à une contrainte qui mélange des degrés de liberté associés à deux objets différents.

Contraintes couplantes et non couplantes

Le terme de contraintes non couplantes désigne toute forme de contrainte qui ne met en jeu qu'un seul objet ; par exemple, une contrainte qui réduit le mouvement global d'un objet : coulisser le long d'un axe, se déplacer sur un plan, etc ... . L'imposition de telles contraintes combinées au mouvement dynamique, est étudié dans [Bar89] et [BW97]. Ces travaux concernent essentiellement la modélisation de contraintes appliquées à des objets rigides. De telles contraintes sont aussi expérimentées sur les objets déformables notamment dans la thèse de Julien Lenoir [Len04]. Pour cela, son auteur s'appuie sur une formulation Lagrangienne de la mécanique qui conduit naturellement à gérer les contraintes à l'aide des multiplicateurs de Lagrange [GW].

Lorsque deux objets, disposant de degrés de liberté propres, sont reliés par des contraintes, l'ensemble du système formé doit être résolu d'une manière globale. On parle alors de contraintes couplantes. L'usage le plus courant de telles contraintes est la modélisation des interactions entre plusieurs objets lors des collisions ou des contacts. Il est important de différencier la contrainte dite couplante d'une contrainte dont la méthode de résolution permet le découplage. Par exemple, dans sa thèse, Jérémie Dequidt [Deq05] propose d'intégrer les contraintes de manière explicite dans le bilan des forces des objets. Cela permet de simuler les objets de façon indépendante.

Techniques de résolution de contraintes

Différentes stratégies sont développées pour résoudre un problème sous contrainte. En effet, la méthode de résolution doit être choisie suivant la complexité de la contrainte : linéaire ou non. Exemple : une contrainte de collision qui est modélisée sous la forme d'une contrainte

d'interpénétration (avec g mesure de l'intersection) : $g(\mathbf{q}(t), \dot{\mathbf{q}}(t), t) \leq 0$ n'est pas linéaire. C'est pourquoi on cherche à simplifier la contrainte pour ramener le problème à une forme linéaire (LCP), qu'il est possible de résoudre efficacement avec un algorithme adapté [CP]. Lorsque les contraintes ne peuvent pas être linéarisées, on a recours à des méthodes adaptées à la résolution de contraintes complexes comme :

- *la pénalité :* le principe de la résolution des contraintes par pénalité consiste à assouplir la contrainte. La fonction à minimiser devient : $f(\mathbf{q}(t), \dot{\mathbf{q}(t)} + k.||AB||, t)$; des termes supplémentaires, pénalisant la déviation, sont ajoutés à la fonction objectif f ; la valeur de ces termes augmentant à mesure qu'on s'éloigne du respect de la contrainte. En général, le terme supplémentaire est calculé comme un ressort entre la configuration qui respecte la contrainte et la configuration courante. La mise en place d'un ressort suppose qu'on dispose d'une mesure de la déviation de contrainte et du paramétrage de la raideur k du ressort. L'usage de la pénalité pose parfois problème ; en effet, un paramétrage trop petit ne suffit pas à séparer des objets et une raideur trop importante tend à faire diverger le processus d'intégration numérique.
- *la projection :* la technique de la projection consiste à ignorer les contraintes dans la phase de résolution du mouvement et à résoudre celles-ci dans une étape de *post-correction* [Fau98]. Les méthodes par projection s'appuient sur un algorithme itératif qui parcourt l'ensemble des contraintes et corrigent celles qui seraient violées.
- *les multiplicateurs de Lagrange :* la méthode des multiplicateurs de Lagrange consiste à traiter les contraintes en ajoutant de nouvelles inconnues au système. Chaque inconnue supplémentaire ayant pour tâche de faire respecter une contrainte [Len04].

Le choix de l'une où l'autre méthode est soumis à discussion. Les méthodes à base de pénalité ont l'avantage de leur simplicité ; ce qui facilite l'intégration d'un grand nombre de contraintes, de natures diverses, dans un système mécanique et ce, sans gestion particulière. Cependant, la méthode à pénalité n'assure pas le respect des contraintes et, le fait d'utiliser un ressort dans le système mécanique, ajoute un terme de raideur k dont le paramétrage est délicat [Jou97]. En outre, cela change la fréquence de vibration de l'objet (les déformations sont analogues à des ondes qui se propagent dans l'objet), compliquant la résolution numérique du système. Les méthodes basées sur la projection satisfont les contraintes au prix d'une dégradation des propriétés physiques de l'animation. Les multiplicateurs de Lagrange sont une méthode générale assez robuste pour traiter les contraintes d'un système mécanique. Cependant, leur utilisation nécessite l'évaluation des dérivées des forces suivant les degrés de liberté, travail coûteux [Len04].

1.3.2 Modélisation des interactions

On parle de lois d'interaction pour désigner la manière dont les objets se comportent lorsqu'ils interagissent. Ces lois sont modélisées sous la forme d'un système de contrainte. On classe les lois d'interaction en deux catégories. La première contient les méthodes qui modélisent la non-intersection des objets, tandis que la seconde modélise les comportements de surface (comme la friction). Dans le cadre de cette thèse, nous n'abordons que le traitement de la contrainte de non-intersection (les lecteurs intéressés par les comportements de friction sont invités à consulter [Dur04]). La figure (1.16) représente les différentes étapes de deux objets entrant en collision. Durant la première étape (dite de *compression*), les objets se déforment. La manière dont les objets se déforment est fonction de la loi de comportement intrinsèque des

objets et de son paramétrage : masses-ressorts plus ou moins rigides, coefficients de Young et de Poisson (pour les méthodes qui dérivent de la théorie de l'élasticité linéaire). Pendant cette phase, l'énergie cinétique des objets en mouvement est transformée en énergie potentielle de déformation interne. Cette phase se termine lorsque les objets ont cessé de se rapprocher. Une fois la phase de compression terminée, il y a *restitution* d'une partie de l'énergie potentielle accumulée par la déformation. Cette énergie est restituée sous la forme d'énergie cinétique qui éloigne les objets. La position finale de la collision présente : soit une position *d'équilibre* (les objets restent en contact – resting contact –), soit une situation de *décollement*. La manière dont les objets se déforme, ainsi que la durée des différentes phases dépendent, à la fois, de la quantité d'énergie de déformation que les objets peuvent accumuler et de la vitesse à laquelle les objets entrent en collision. On peut distinguer différentes modélisations de la collision : impulsions, contraintes de non-intersection, contraintes d'équilibre des forces de surface (pressions) et contraintes de distance.

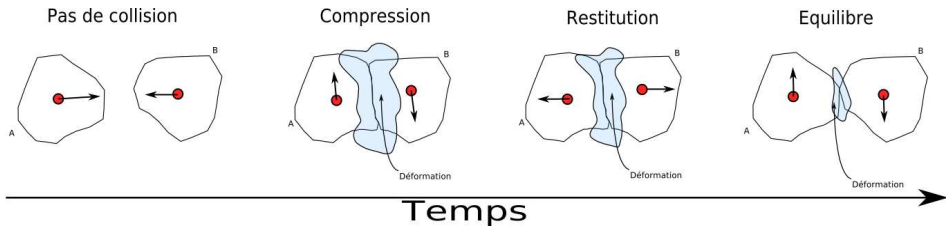

Fig. 1.16 : *1) Deux objets se rapprochent. 2) Les objets sont entrés en collision, les objets se déforment en accumulant de l'énergie 3) L'énergie accumulée pendant la phase de compression est reconvertie en énergie cinétique. 4) Un nouvel équilibre est trouvé entre les contraintes internes et externes.*

Collision entre objets rigides : les impulsions

Plus les objets sont rigides, moins ils peuvent emmagasiner d'énergie potentielle en se déformant. Lorsque les objets sont complètement rigides, la durée des phases de *compression/restitution* devient nulle. En pratique, cela se traduit par un changement instantané de la direction du vecteur-vitesse des objets. C'est à partir de ce constat qu'est construite la méthode des impulsions présentée dans [Mir96]. La méthode des impulsions dérive de modélisation des chocs entres objets rigides (comme deux boules de billard). En contrepartie, la méthode des impulsions est assez peu adaptée pour modéliser la collision d'objets déformables ou en contact permanent. On trouve néanmoins dans [MC95], une technique à base de trains d'impulsion. Ces trains, succession de petites impulsions, traitent les cas de collision entre objets déformables ou en contact prolongé.

Contrainte en position : l'intersection nulle

Quand on modélise des objets déformables, l'hypothèse de "phases de *compression/restitution* instantanées" ne peut plus être posée. C'est pourquoi on cherche à modéliser plus finement le comportement des objets et en particulier le fait qu'ils ne doivent pas s'intersecter. Pour cela, on définit une contrainte *d'intersection nulle*. Ainsi, pour

modéliser le fait que deux objets ne peuvent occuper le même espace au même instant, on emploie une équation de contrainte $g = vol(A \cap B)$ qui traduit le volume de l'intersection. Le respect strict d'une telle contrainte n'est pas envisageable dans un simulateur temps-réel , car le volume d'intersection doit être calculé (ce qui est très coûteux). Dans ce contexte, on remplace plutôt la mesure de l'intersection par des mesures simplifiées telles : la distance de pénétration [Cam97] ou la distance de croissance [OHH00]. L'avantage de ces mesures est leur moindre coût en calcul. Outre le temps de calcul propre à l'évaluation de la fonction g, il est important de noter que l'équation de contrainte qui en résulte n'est pas linéaire et la résolution d'un tel système par un algorithme itératif de type Newton/Descente de Gradient est bien trop lente pour être employée dans un contexte temps-réel . C'est pourquoi certaines simplifications sont proposées. On peut par exemple :

- utiliser la méthode de la pénalité pour résoudre le système contraint ; on ne peut alors pas garantir que la contrainte soit respectée et un nouveau paramètre, la raideur du ressort de rappel (qu'il est parfois délicat de paramétrer), est introduit. En outre, certains auteurs lui accordent une signification physique [Mes03][Rem00][Jou97]. Ils associent la raideur du ressort de pénalité à une modélisation de la déformation des objets. Le ressort devient alors une modélisation des phases de compression/restitution et non une méthode de résolution numérique.
- remplacer le système non-linéaire global par un ensemble des sous-systèmes indépendants. Bien évidemment, la résolution de sous-systèmes indépendants n'est pas équivalente à la résolution du système contraint initial, et il arrive que le système ne puisse empêcher les intersections. En général, les sous-systèmes s'appuient sur des mesures locales de l'interpénétration [KOLM02][RL06] (fig. 1.17) : .

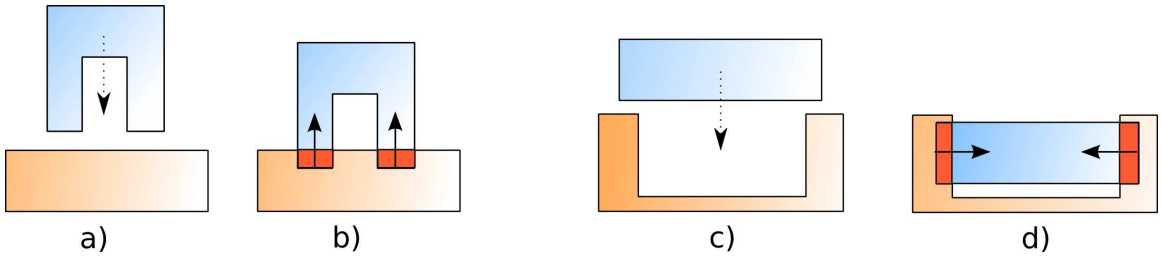

Fig. 1.17 : *Sur a) et b), un calcul de réponse local permet de repousser correctement les objets. Ce n'est pas le cas sur c) et d).*

Contrainte en pression : équilibre des pressions de surface

Lorsque la contrainte d'intersection vide est résolue grâce à une méthode de minimisation, on observe qu'une fois la phase de minimisation terminée, les forces internes exercées sur les surfaces en contact sont en équilibre. Cette propriété est employée dans [CD97] pour modéliser les contacts entre objets fortement déformables représentés par des surfaces implicites à squelettes. On retrouve cette idée dans la thèse de Christian Duriez qui [Dur04] modélise les interactions entre des objets déformables à l'aide de la loi de Signorini. La loi de Signorini traite la situation d'équilibre des surfaces comme une condition limite du système à résoudre. Une telle gestion des contacts suppose la connaissance a priori des zones de contact, puisque

c'est à cet endroit que se définit l'équation d'équilibre. La boucle de simulation est décomposée en deux étapes sucessives. La première, appelée *"mouvement libre"*, calcule le mouvement de l'objet soumis aux forces connues (gravité, forces externes, forces internes), tandis que pendant la seconde, les intersections entre les objets sont détectées et localisent les surfaces en contact. C'est sur ces surfaces en contact que l'équation d'équilibre des pressions est définie, puis résolue numériquement par une méthode d'Éléments Finis 2d. Une telle approche garantit qu'au niveau des surfaces en contact, il n'y ait pas d'intersection (à l'échelle de la discrétisation). Par contre, la méthode proposée ne résout pas le problème de l'apparition de nouvelles surfaces de contact pendant la phase de résolution. Un processus itératif doit alors être mis en place comme dans [LCDN06].

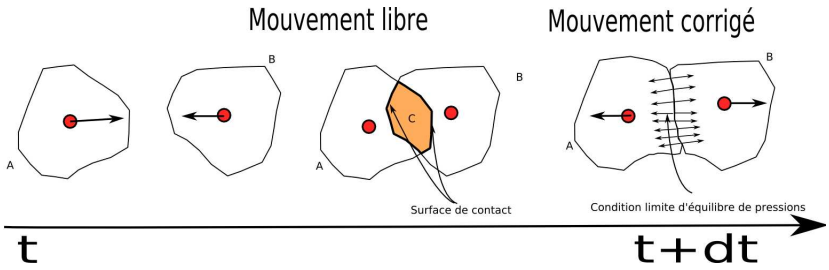

Fig. 1.18 : *Gestion du contact exact entre objets. Les zones de contact ne sont pas connue a priori et doivent être détectées pendant la simulation, c'est le rôle de la phase de détection des intersections.*

Quelques autres contraintes : équation de plan, distance nulle

D'autres formulations de contraintes permettent la modélisation des interactions entre objets. Ces formulations sont moins courantes. Cependant, certaines d'entre-elles présentent des propriétés intéressantes : ainsi dans [BW97], les collisions sont modélisées à l'aide de contraintes de plan ; la figure (1.19) nous renseigne sur les différents problèmes qui apparaissent lorsque les plans restreignent trop le mouvement de l'objet. Pour contourner ce problème, les auteurs de [LCDN06] proposent d'employer la carte de distance de l'objet. Cette carte (utilisée aussi pour détection de collision 1.3.4.0), associe à tout point de l'espace la distance qui le sépare du bord l'objet. Avec le gradient de la carte de distance, les auteurs en déduisent l'équation des plans de contraintes 1.20. A chaque itération de l'algorithme de résolution numérique, la contrainte est réévaluée et modifiée de manière à limiter l'influence des espaces inaccessibles.

1.3.3 Le pipeline de collision

L'évaluation des lois de comportements extrinsèques suppose l'identification des zones de contact entre les objets ; on a alors recours à une phase de détection des intersections. Ce processus de détection des intersections est en général découpé en une succession d'étapes affinant de plus en plus la précision du calcul de collision. On représente le processus de gestion des collisions sous la forme d'un pipeline composé de quatre étapes : 1) phase de détection globale (broad phase) ; 2) phase de détection approchée (narrow phase) ; 3) phase de détection

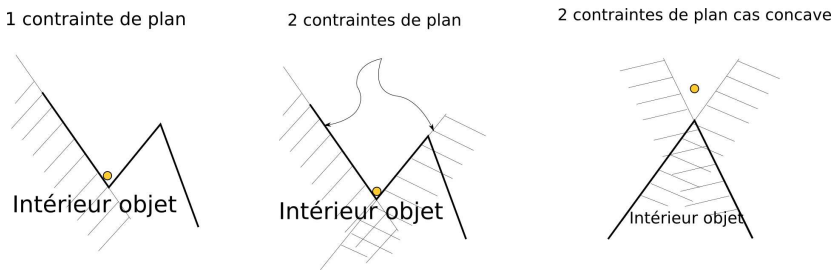

Fig. 1.19 : *Contrainte de non pénétration exprimée à l'aide des normales à la surface de l'objet.*

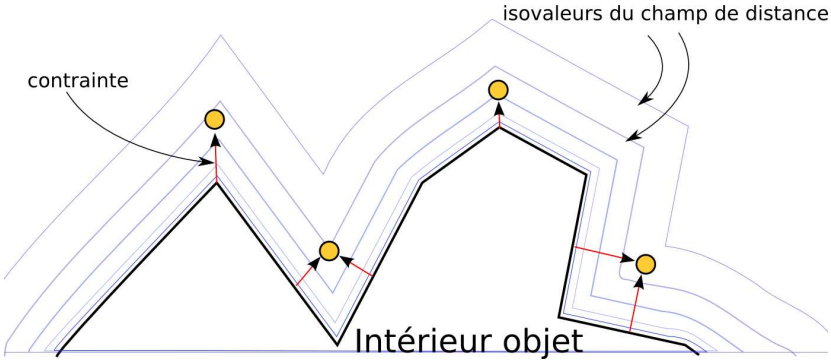

Fig. 1.20 : *Les contraintes (planaires) sont définies non plus à partir des primitives de la surface de l'objet mais à partir du gradient de la distance qui le sépare de la primitive la plus proche. On peut ainsi traiter aussi bien les objets convexes que concaves.*

exacte, 4) calcul de réponse. La broad phase consiste à passer d'un algorithme n-corps à un algorithme 2-corps, la narrow phase localise quant à elle les primitives potentiellement en collision en utilisant, par exemple, une hiérarchie de volume englobant. Enfin, dans la troisième phase, les intersections entre primitives sont effectivement calculées [FKM03]. On observe encore que certains auteurs considèrent la phase de réponse à la collision comme faisant partie du pipeline de collision. Nous préférons plutôt considérer que la phase de réponse, qui est analogue à une modélisation mécanique, est différenciée du pipeline de détection des intersections (phases 1,2,3). Par analogie avec la modélisation des contraintes intrinsèques aux objets (modélise le mouvement propre), le gestionnaire de collision permet de modéliser les contraintes extrinsèques. L'ensemble des contraintes sont à résoudre comme un seul et même système. Comme la résolution de ce système est difficile, on procède à des simplifications. Par exemple, dans [MDH+02], on fait l'hypothèse que de nouvelles contraintes ne peuvent être ajoutées à l'intérieur d'un pas de temps. D'autre auteurs proposent d'intégrer les forces de manière explicite, en ne prenant en compte que la position des objets à l'instant précédant, pour calculer les forces de collision.

1.3.4 La détection des intersections

La gestion des collisions est un domaine aux applications nombreuses ; que ce soit en robotique, en simulation physique, ou encore en informatique graphique, tous ces domaines ont développé des méthodes de détection de collision. Les multiples domaines d'application ont

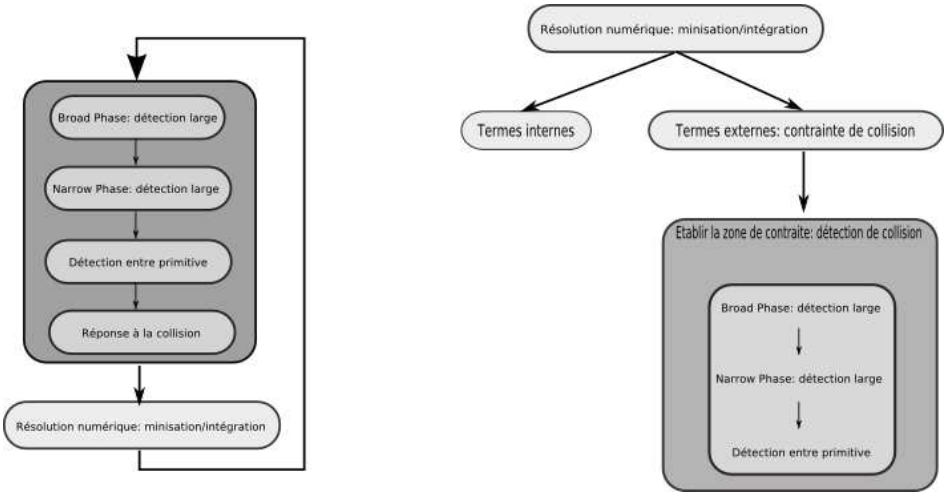

Fig. 1.21 : *Le pipeline de collision, la partie détection des intersections est décomposée en quatre phases. A droite, une autre manière de voir la place du pipeline de collision dans le processus de simulation. Cette vision a l'avantage de mettre en valeur le rôle symétrique des contraintes intrinsèques (déformations) et extrinsèques (collisions-frictions).*

entraîné l'apparition d'un très grand nombre de méthodes de détection de collision nécessitant parfois des modélisations très différentes des objets. Faire un inventaire exhaustif des publications dans ce domaine semble hors de portée dans le cadre de cette thèse. Des survols traitant des collisions entre objets rigides sont disponibles dans [LG98][JTT01] et [LM03]. Un état de l'art assez complet, dédié aux objets déformables, a été récemment publié [TKH+05]. Lorsqu'on simule des objets déformables, il faut détecter les cas d'auto-intersections, en plus des collisions entre les objets. Cette phase d'auto-collision est extrêmement coûteuse et conduit à l'utilisation d'algorithmes ad-hoc reposant sur les propriétés de l'objet simulé ; on peut consulter [VCMT95][VMT06] pour les auto-collisions de tissus ou encore [Ber06] pour les auto-collisions de cheveux. Les lecteurs francophones peuvent aussi se reporter au rapport d'activités [FKM03] qui résume les travaux d'une Action Spécifique du CNRS dédiée à la collision.

Lorsqu'on veut simuler des objets déformables, il faut combiner une loi de comportement à une géométrie donnée. De même, les contraintes qui modélisent les intersections entre objets doivent être appliquées sur une géométrie donnée. Or, cette géométrie est inconnue a priori et change en fonction des zones en contacts. Il faut donc réévaluer quelle part de la géométrie des objets est concernée dans le calcul d'une contrainte d'intersection. C'est l'objectif de la *détection des intersections* (*détection des collisions*). Les algorithmes utilisés dans cette étape dépendent essentiellement de la représentation géométrique des objets (maillage, union de sphère, voxel). Cependant le choix d'une méthode de détection des intersections se raccroche aussi au type des données désirées en sortie de la phase de détection. Par exemple, dans le cadre de l'interaction non physique ou des méthodes à base de contacts surfaciques (contrainte d'équilibre des pressions 1.3.2.0), la simple évaluation booléenne de la présence/absence d'intersection suffit. Par contre, dans le cas de la modélisation des interactions à l'aide d'une contrainte d'intersection vide, on cherche souvent à évaluer une mesure de l'interpénétration. La discrétisation temporelle est aussi un point important du processus de détection des

intersections. En effet, certains processus de détection de collision n'évaluent la présence de l'intersection qu'aux instants discrets de la simulation, c'est la *détection spatiale*. L'intersection spatiale peut manquer des collisions ; il faut alors avoir recours à une détection spatio-temporelle des intersections ; ces-dernières prenant en compte le mouvement intermédiaire des objets (obtenu par interpolation) entre deux instants discrets.

Les techniques de détection spatiale

L'intersection entre paires de primitives : bien souvent les objets sont représentés par des assemblages de primitives (triangles, tétraèdres, sphères). On utilise alors des méthodes de détection spatiale des collisions s'appuyant sur un test d'intersection entre paires de primitives. Ainsi, lorsqu'aucune des primitives n'est en intersection avec une autre, les objets ne sont pas en collision. Une première approche consiste à explorer tous les couples de primitives qui composent les objets et à tester leur intersection. Un test d'intersection arête-arête (primitive 2D) est disponible dans [SE03]. Les objets sont souvent représentés par une surface triangulée et le test d'intersection triangle-triangle fait l'objet de nombreux travaux : [MT97][GD03]. Le test d'intersection entre tétraèdres de [FG03] s'apparente au test d'intersection triangle-triangle. La simplicité du test d'intersection sphère-sphère (quelques opérations élémentaires [Mes03]) conduit certains auteurs à représenter les objets non plus à l'aide de maillage, mais avec des sphères [Mes03][Hub95], tout en continuant à afficher des objets triangulés. Ainsi, la géométrie de la collision, représentée à l'aide de sphères, est découplée de celle de l'objet mécanique affiché (voir *multireprésentation* dans le chapitre (2)). Apparaît alors la difficulté de mettre à jour les différentes représentations pendant l'animation.

L'intersection entre objets convexes : les premiers travaux portant sur la détection de collision furent initiés dans le domaine de la robotique. Les roboticiens cherchaient alors à déplacer des robots mobiles dans des environnements solides [Can85][Can87]. Ils ont observé que lorsque les objets sont convexes, la complexité de la détection de collision est considérablement diminuée. Deux catégories d'approches sont proposées : la première catégorie contient les algorithmes qui dérivent d'un algorithme appelé GJK ; ce dernier permet de calculer la différence de Minkowski [Weic] entre deux objets [MW88][Cam97] ; la seconde se base sur la recherche d'un minima à la distance qui sépare les deux objets. Le minima de la distance est obtenu à partir du constat suivant : soit X et Y, deux primitives appartenant à deux objets convexes. Soit $x \in X$ et $y \in Y$, les points les plus proches de X et Y. Si x appartient à la région de Voronoï de Y et y à la région de Voronoï de X, alors $||x, y||$ est la distance minimale entre les deux objets. Cette approche est introduite dans [Lin94] [LMP94], puis améliorée dans la librairie V-Clip [Mir98]. Le améliorations portent sur un test d'intersection numériquement plus robuste et sur un calcul d'une distance négative traduisant une mesure de l'intersection des objets. Ces deux méthodes sont extrêmement efficaces en $O(n)$, avec n le nombre de primitives. En plus, elles tirent parti de la cohérence temporelle d'une animation (petits changements entre deux instants de simulation) pour abaisser la complexité en $O(1)$ en moyenne. Cependant, ces deux méthodes ne fonctionnent que pour des objets convexes. Lorsque les objets ne le sont pas, les auteurs proposent une décomposition des objets en parties convexes. Chaque partie convexe est alors traitée indépendamment. Les approches de ce genre sont réservées

aux objets rigides pour lesquels la décomposition peut être précalculée, car les algorithmes de segmentation convexe [qhu] sont trop complexes pour une utilisation pendant la simulation.

Les cartes de distance : ces méthodes s'appuient sur l'utilisation de la fonction de distance signée $d(\mathbf{x})$ [EHK+00] pour identifier l'intérieur et l'extérieur des objets. Ainsi la notion d'intersection entre un objet et un point est définie naturellement comme :

$$\begin{cases} d(\mathbf{x}) & > 0 \quad \text{pas de collision} \\ d(\mathbf{x}) & = 0 \quad \text{contact} \\ d(\mathbf{x}) & < 0 \quad \text{pénétration} \end{cases} \tag{1.20}$$

Comme l'évaluation d'une telle fonction d n'est pas triviale, on utilise souvent une grille de voxels contenant la discrétisation de la carte de distance précalculée [FSG00]. Cette carte associe à tout point de l'espace un scalaire indiquant la distance qui sépare le point du bord de l'objet le plus proche. On trouve dans [HZLM02], [HZLM01], [KLM02] [KOLM02], des méthodes de détection de collision basées sur des cartes de distance ; les cartes de distance étant calculées avec le GPU. On trouve aussi dans [HLC+01], un algorithme de conversion d'objets maillés en carte de distance basé sur la propagation de front (feu de forêt). Lorsque le stockage complet d'une grille 3D consomme trop de mémoire, celle-ci peut être remplacée par une version adaptative nommée ADF (Adaptive Distance Field [FPPJ00]). Dans [BMF03], les objets sont représentés par un ADF ; ADF utilisé pour traiter la collision entre des objets volumiques et des tissus. Lorsqu'une collision est détectée, le tissu est repoussé, de manière plausible, hors de l'objet (fig. 1.22).

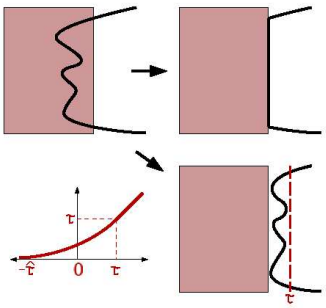

Fig. 1.22 : *Un tissu, en collision avec un solide, est repoussé de manière plausible. (Extrait de [BMF03]).*

L'utilisation d'une discrétisation de la fonction de distance peut poser des problèmes de précision. [HLC+01] propose de remplacer cette carte par une table qui associe à chaque voxel la primitive la plus proche. La grille de voxel sert alors uniquement de table d'accès rapide à la primitive la plus proche et la distance est calculée explicitement entre le point et la primitive géométrique. Une approche similaire est utilisée par [GS05] qui discrétise l'espace autour de l'objet et projette la discrétisation en coordonnées polaires [Weid] sur une sphère englobante. La sphère sert donc de table d'accès rapide (Lookup Table dans [EL00]) associant aux coordonnées polaires les primitives des objets les plus proches. L'utilisation de structures précalculées, pour évaluer rapidement la fonction de distance, limite l'usage des cartes de

distance aux seuls objets rigides. Néanmoins, l'utilisation de la carte s'est aussi étendue à la gestion des objets déformables [HFL00][FL01].

Fig. 1.23 : *Les modèles à base de carte de distance représentent souvent les intersections entre objet rigide et tissu. (extrait de [FSG00])*

Un dernier point important est le choix du maillage des objets. En effet, la fonction de distance est évaluée en un certain nombre d'échantillons qui correspond, en général, aux sommets du maillage des objets à tester. On illustre sur la figure (1.24) des cas où la collision est manquée. Dans le chapitre (4), on présente une solution à ce problème en ne calculant plus la pénétration sur les points des objets mais sur les segments en intersection.

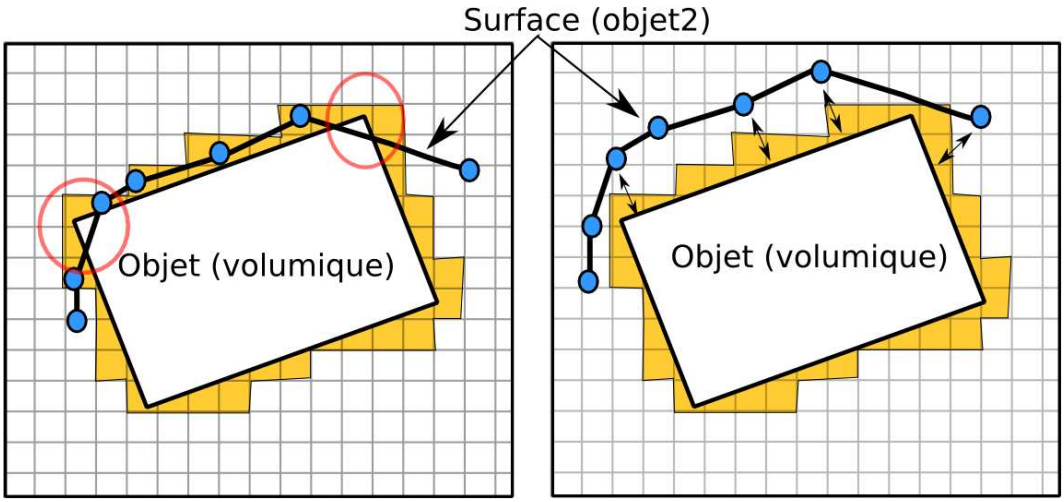

Fig. 1.24 : *L'utilisation des cartes de distance suppose que les objets soit finement discrétisés pour éviter de manquer des collisions (à gauche - entouré en rouge) ; certains auteurs proposent de rallonger la distance de détection (à droite).*

Intersection volumique : les méthodes que nous venons de présenter permettent de retourner une *mesure* de l'intersection. Cette mesure est bien souvent une distance, locale ou globale,

entre les deux objets. Bien que moins courante, l'utilisation du volume comme mesure de l'intersection est aussi possible. Dans [HTG03], on trouve une méthode de calcul du volume d'intersection de deux objets ; cette méthode s'appuie sur l'utilisation du hardware 3D pour obtenir une discrétisation volumique des objets et de leur intersection. La méthode est intéressante, mais les temps de calcul trop importants semblent être un frein à son utilisation. En plus, un calcul de réponse par contrainte d'intersection vide (pénalité, multiplicateurs de Lagrange), suppose qu'on puisse évaluer le gradient suivant les différents degrés de liberté. Comme il s'agit d'une formulation purement discrète, il faut procéder par différenciation numérique, ce qui est coûteux ([Weie]).

Détection spatio-temporelle

La détection spatio-temporelle propose de prendre en considération le mouvement intermédiaire des objets entre deux pas de temps [WW86] [Wan00]. C'est un aspect qui n'est pas pris en compte par les méthodes purement spatiales. Ne pas en tenir compte peut conduire à des simulations incohérentes, les objets se traversant les uns les autres sans qu'aucune collision ne soit détectée. La quantité de collision manquée est d'autant plus sensible que les objets sont rapides ou bien alors très fins (comme les tissus). Comme le mouvement intermédiaire n'est pas connu, il est approché à l'aide de techniques d'interpolation (fig. 1.25). Dans le cadre d'une simulation de tissu, [BFA02] s'appuie sur une méthode spatio-temporelle. Les tissus sont représentés par des maillages triangulés. Le mouvement intermédiaire est obtenu par interpolation linéaire du mouvement des triangles. Une telle approche donne de bons résultats visuels. Elle reste néanmoins trop lente pour être exploitée en temps réel. L'interpolation linéaire du mouvement est inadaptée pour interpoler des objets soumis à une rotation. C'est pourquoi, dans [RKC02], le mouvement intermédiaire est considéré comme un mouvement de vissage bien adapté pour interpoler le mouvement des objets rigides.

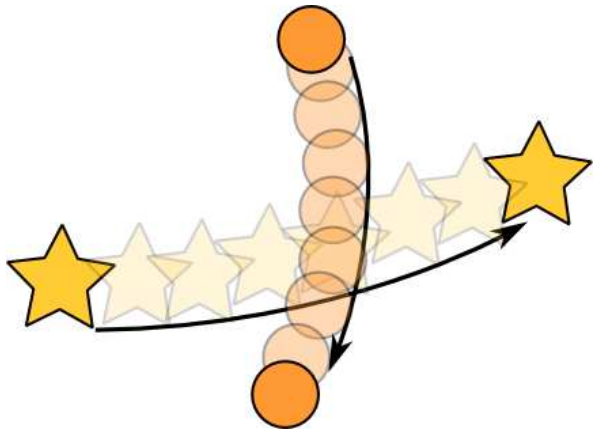

Fig. 1.25 : *Entre deux instants discrets, deux objets peuvent entrer en collision sans que celle-ci soit détectée.*

1.3.5 Accélération de la détection des intersections

Les algorithmes de détection des intersections sont en général très coûteux. Par exemple, l'intersection géométrique de surfaces triangulées déformables nécessite l'exécution d'un test d'intersection triangle-triangle pour toutes les paires de triangles. Une large part de la recherche en détection des intersections s'est concentrée sur l'accélération de la phase de détection des intersections. Les méthodes d'accélération ont deux objectifs : le premier est l'établissement rapide de la non-intersection de deux objets composés de nombreuses primitives, le second, la localisation des primitives effectivement en intersection.

Partitionnement spatial

Les méthodes de partitionnement spatial divisent l'environnement global en cellules, de manière à localiser plus efficacement les primitives en intersection. Il existe un grand nombre de techniques de partitionnement spatial dont les propriétés sont assez connues pour être largement étudiées dans le domaine de la visualisation et du rendu, en particulier dans les domaines du lancer de rayons [CDP95] et de la radiosité. Parmi les méthodes de partitionnement spatial, on distingue celles qui effectuent un partitionnement absolu de l'espace (comme les grilles de voxel) de celles basées sur la géométrie de l'environnement comme les BSP [faq] ou les portails [LG95]. Les techniques de BSP/portails découpent l'environnement en cellules de manière à réduire le nombre de test à effectuer ; deux objets ne peuvent être en collision que s'ils appartiennent à la même cellule (fig. 1.27). Les méthodes à base de BSP/portails sont à privilégier lorsque l'environnement est quasi statique. Par contre, pour un environnement dynamique, on priviligie plutôt un partitionnement absolu comme grille volumique ou un octree (partionnement spatial) ; les primitives des objets y sont insérées. Pendant l'insertion, on teste l'intersection entre les primitives déjà présentes dans la cellule de la grille avec les primitives en cours d'insertion. On peut observer, sur la figure (1.26), différentes stratégies de grilles de voxel. Avec une résolution de grille trop faible, celle-ci est ineffective . Si la résolution est trop grande, l'insertion d'un unique objet risque de recouvrir un nombre trop important de primitives. Dans [BGAM04], une méthode de dimensionnement automatique de la grille de voxel est présentée pour éviter les problèmes de sur/sous-échantillonnage. Dans [THM+03], la grille de voxel est remplacée par une table de hachage. Cela économise de la mémoire tout en étendant l'utilisation des grilles de voxel à des environnements non bornés. [Edw] propose d'éviter les problèmes liés au dimensionnement de la grille en utilisant une approche multirésolution (octree). Une approche mixte est encore présentée dans [CDP95] où les objets sont triés par taille, puis insérés dans des grilles de résolution adaptées à leur taille.

Le sweep and prune

Le *sweep and prune* [CLMP95] est une technique d'accélération de la détection de collision efficace, basée sur la réduction dimensionnelle : soit ns objets dont on veut tester les intersections paire à paire. Une approche brutale nécessite $\frac{n(n+1)}{2}$ tests d'intersection entre paires d'objets. L'utilisation du *sweep and prune* ramène cette complexité à $O(n.logn)$ dans le pire des cas, et en $O(n)$ lorsque le mouvement des objets est peu rapide. Le principe du *sweep*

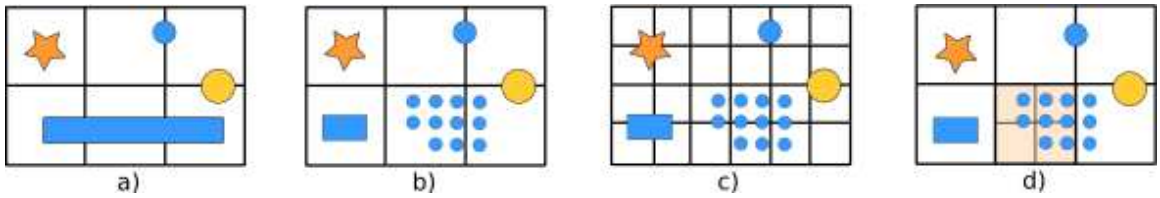

Fig. 1.26 : *Grille de voxel : a) grille uniforme et distribution uniforme. b) grille uniforme et distribution non uniforme. c) Grille uniforme de résolution plus fine. d) Octree.*

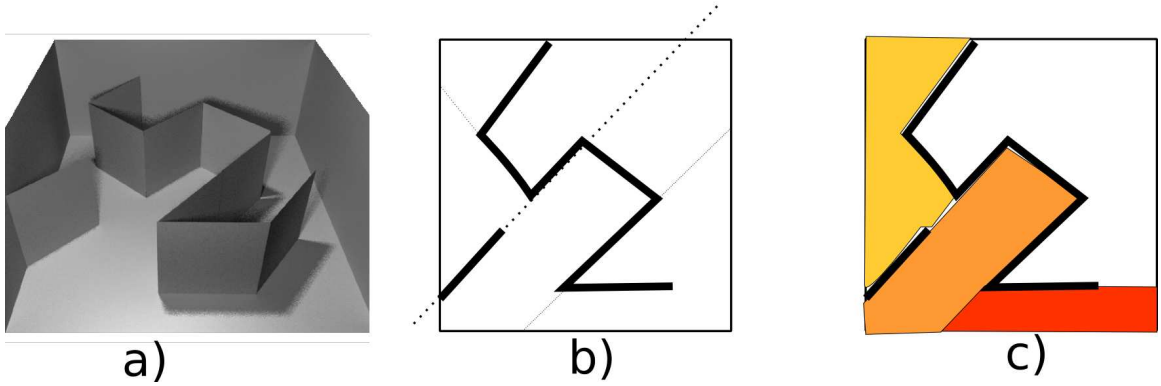

Fig. 1.27 : *a) un environnement statique ; b) découpage en BSP : on choisit un plan qui divise l'espace en 2 sous-espaces ; c) la segmentation "portail" est une segmentation logique qui divise l'espace en zones connectées par des ouvertures.*

and prune est de projeter les bornes des objets sur un axe, puis de recenser les paires d'objets dont les bornes s'intersectent à l'aide d'un algorithme de tri. Lorsque le nombre d'objets en collision est élevé, un tri en temps linéaire, comme le *radixsort*, est recommandé. Lorsque la cohérence temporelle est importante, l'usage d'un "tri bulle" permet de s'approcher de la borne $O(n)$ en moyenne. La méthode du *sweep and prune* ne peut identifier les paires en intersection mais seulement les paires qui ne le sont pas. Pour augmenter l'efficacité de la méthode, on répète, sur plusieurs axes, l'algorithme du *sweep and prune*. Les objets qui ne s'intersectent pas sur tous les axes sont rejetés hors du test d'intersection (fig. 1.28).

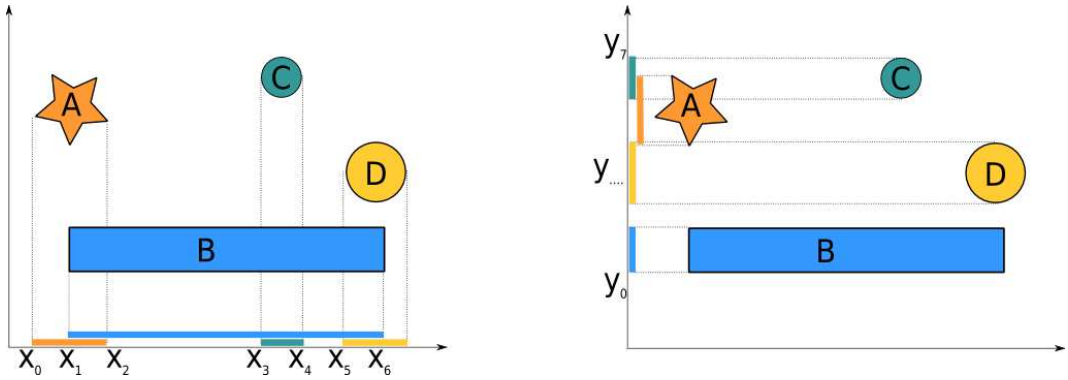

Fig. 1.28 : *Deux étapes du sweep and prune, on teste les intersections entre les projections des objets.*

Le principe de la réduction dimensionnelle est repris dans l'algorithme Cullide [GRLM03]. La projection des objets sur un axe 1d est remplacé par la projection sur un plan (fig. 1.29). La phase de projection sur le plan bénéficie du hardware de rendu 3d et des nouvelles extensions comme le test d'occlusion. Deux objets, dont les projections ne s'intersectent pas, sont forcément disjoints. La méthode est très efficace du point de vue des temps de calcul et est retenue dans [GKJ+05] et [JKG+05]. Cependant, on remarque que, contrairement à la méthode du sweep and prune, le fait d'utiliser le rendu graphique suppose une discrétisation spatiale des objets pouvant induire des erreurs ; ce dernier point est traité dans [GLM04].

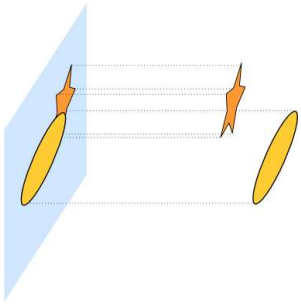

Fig. 1.29 : *Cullide : les objets sont projetés sur un plan à l'aide du hardware 3D. Les tests d'intersection entre leurs projections sont réalisées à l'aide de la carte 3D.*

Les volumes englobants

La manière la plus courante de détecter rapidement la non-intersection de deux objets complexes consiste à englober ces-derniers dans un volume dont le test d'intersection est simple. Ainsi, au lieu de tester l'intersection de chacune des primitives des objets A et B, il suffit de tester l'intersection entre leurs volumes englobants. Le choix du volume englobant fait l'objet de nombreuses études. Généralement, on estime que la qualité d'un volume englobant est définie par les propriétés suivantes :

- la puissance d'approximation d'une géométrie donnée ; elle se mesure avec la distance de Hausdorff ; [Weib].Plus l'objet est bien approché par sa primitive englobante, plus le nombre de faux positifs (indiquant une collision alors qu'il n'y en a pas) est faible ;
- la complexité du test d'intersection ;
- la complexité du calcul de construction et de mise à jour du volume englobant à partir de la géométrie de l'objet.

On observe sur la figure (1.30) quelques volumes englobants couramment utilisés. Dans le cadre de la détection de collision entre objets déformables, on privilégie les volumes simples à mettre à jour (comme les boîtes alignées sur les axes (AABB)[vdB01]), ou dont le test d'intersection est rapide (comme les sphères [Hub93]). Cependant, ces primitives ne possèdent pas la même efficacité lorsqu'il s'agit d'approcher une géométrie donnée ; la différence entre la forme de l'objet et la primitive englobante est plus importante avec des sphères ou des AABB que pour des boîtes orientées (OOBB) [GLM96] ou des K-Dop [KHM98]. Sur la figure (1.30), nous présentons quelques volumes englobants courants (il en existe d'autres, comme

[KPLM97][BCG+96]). On représente sur l'axe vertical de la figure, les processus de mise à jour suivant les possibilités du mouvement des objets (rigides, déformables, découpables).

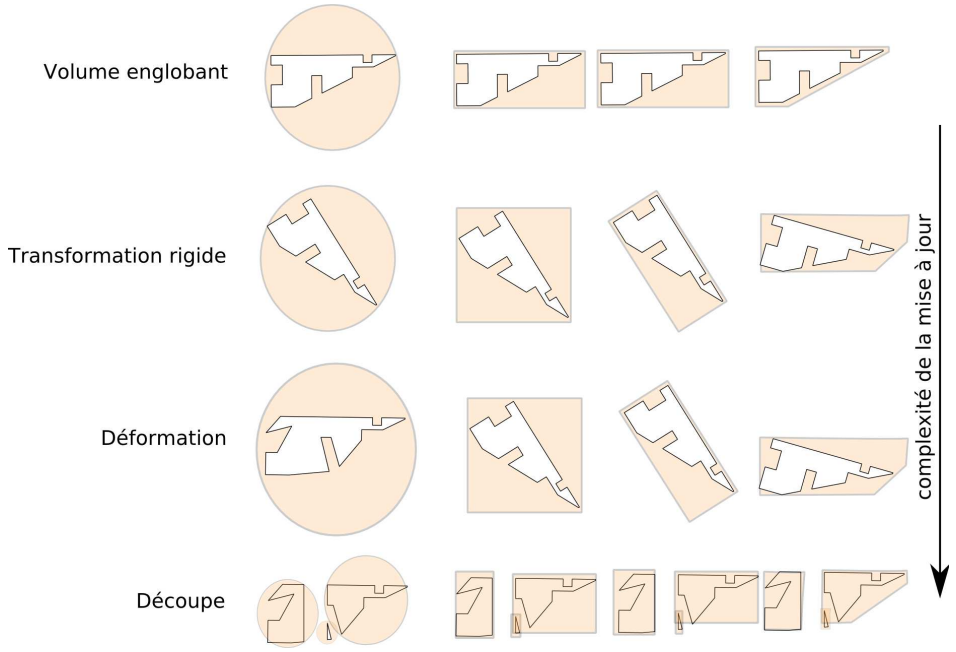

Fig. 1.30 : *Quelques volumes englobants : Sphère, AABB, OOBB, K-Dop.*

Hiérarchie de volume englobant

Les hiérarchies généralisent l'utilisation des volumes englobants et localisent les primitives des objets en intersection avec une complexité en $O(n.logn)$. On peut évaluer les performances d'une hiérarchie à partir des critères suivants :

- la puissance d'approximation d'une hiérarchie dépend de la primitive englobante utilisée (voir la section précédente) ;
- l'existence d'algorithme de mise à jour racine-feuilles (top-down) ou feuilles-racine (bottom-up) ;
- l'algorithme de construction.

Dans [Hub95] et [Hub96], Hubbard propose d'employer des hiérarchies de sphère pour tester les intersections entre objets rigides. Dans [BO02], [BO04], différents algorithmes de construction de hiérarchie de sphère à partir de modèles triangulés sont présentés et expérimentés.Les hiérarchies de sphère ont la particularité d'être utilisées comme des hiérarchies de volume englobant mais également comme primitives de modélisation à partir desquelles un calcul de réponse est possible [ODGB01][OD01], [OD99]. Cette dualité entraîne le développement de méthodes *interruptibles* [DO00], dans lesquelles la profondeur d'exploration de la hiérarchie est bornée par le temps de calcul alloué au processus de collision. Si les hiérarchies de sphère ont de bonnes propriétés, leur usage se limite bien souvent à la détection d'intersections entre objets rigides. On peut cependant noter les travaux publiés dans [JP04], présentant une technique de mise à jour très efficace de hiérarchies de sphère pour les objets dont les déformations

s'expriment comme une combinaison linéaire des modes de déformations obtenus par Analyse Modale. L'usage des hiérarchies de volume englobant est quasi systématique lorsque les objets

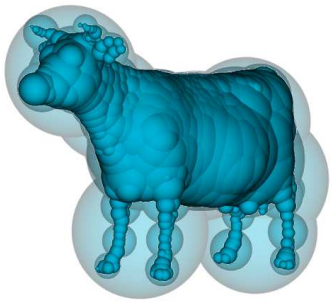

Fig. 1.31 : *SphereTree produit par l'algorithme de [BO02]*

sont rigides ; les K-Dops ou les OBB étant de bonnes primitives en raison de leurs capacités à représenter la géométrie des objets. Dans [vdB01], puis [vdB97], Van Den Bergen propose

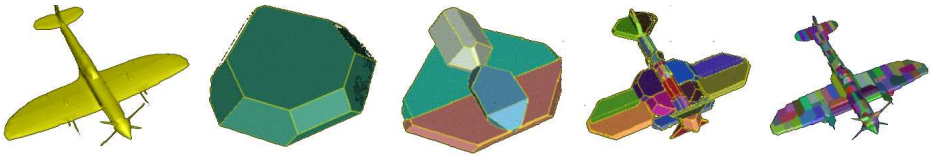

Fig. 1.32 : *Hiérarchie de K-Dop [KHM98].*

l'utilisation de hiérarchies de boîtes englobantes alignées (AABVH) sur les axes pour tester la collision d'objets déformables. Dans le contexte des objets déformables, il semble que les hiérarchies à base d'AABB soient à privilégier. Dans [MKE02], les auteurs étudient l'usage des K-Dops pour la détection de collision de tissus.

Comme le remarque [Mes03], les méthodes à base de hiérarchies ne traitent pas le cas des objets découpables. En effet, changer la topologie de l'objet suppose de changer la topologie de la hiérarchie qui doit alors être reconstruite. Différents travaux récents traitent de ce point particulier ; dans [SGG+06], les auteurs proposent une nouvelle méthode d'accélération de la détection de collision basée sur les cartes 3d qui permet de traiter les objets qui se fracturent ; dans [LAM06], les auteurs proposent un algorithme original de création de hiérarchie, qui à la volée, permet la construction de la hiérarchie. L'intérêt de l'approche de [LAM06] est d'être compatible avec les architectures courantes de collision. L'approche de [SGG+06] semble par contre plus performante. On notera enfin que l'algorithme de segmentation dynamique présenté au chapitre (5) peut aussi servir pour mettre à jour une hiérarchie de volume englobant bien que nous ne l'ayons pas développé dans ce but.

Autres techniques d'accélération

Il existe d'autres techniques d'accélération de la détection des intersections dont l'usage est plus marginale.

L'accélération cinématique : l'intégration d'informations supplémentaires, comme le mouvement des objets, accélère la phase de détection de collision par la construction de prédicats de non intersection. Ainsi, lorsque deux objets ne sont pas en collision et qu'ils s'éloignent l'un de l'autre, on peut se dispenser de tester leur intersection [BEG+99].

Les techniques stochastiques : des techniques stochastiques sont introduites afin de diminuer la complexité inhérente au calcul de détection des intersections entre des objets composés d'un grande nombre de primitives. Dans [SKSH03], des particules mobiles sont déposées à la surface des objets. Ces particules cherchent ensuite à se déplacer à la surface du maillage de manière à minimiser la distance qui sépare les objets. Pour fonctionner, une telle approche suppose d'utiliser un nombre suffisant de particules pour détecter correctement toutes les collisions entre des objets concaves. En pratique, on peut estimer qu'avec une particule par composante convexe, on arrive à détecter les collisions. Les techniques stochastiques se rapprochent des méta-heuristiques de minimisation globale [Wikc]. Les particules peuvent tomber dans des minima locaux et ne plus en ressortir. [KNF04] et [RCFC03] font l'étude de règles de déplacements de particules afin de garantir un nombre suffisant de particules pour détecter les collisions.

1.4 La résolution des équations du mouvement

Nous avons présenté comment sont modélisées les déformations et les collisions entre objets. Le calcul effectif du mouvement des objets est le dernier élément essentiel pour composer un moteur de simulation. Le calcul de l'animation finale est obtenu par résolution numérique des équations différentielles obtenues à partir des objets composant la scène et de la modélisation des collisions. On distingue deux formulations différentes coexistantes : dans la première, le calcul du mouvement dynamique des objets suppose un processus d'intégration temporelle qui, à partir d'une configuration donnée, permet de calculer une nouvelle configuration ; dans la seconde, l'aspect dynamique de l'animation est ignoré, la simulation est perçue comme un problème de minimisation énergétique statique (résolu à l'aide de méthodes de minimisation). La figure (1.33) représente de manière intuitive ce qui se passe lorsqu'on simule un objet. Les lois de comportements intrinsèques décrivent les propriétés matérielles d'un objet, tandis que les lois de comportements extrinsèques traduisent la manière dont deux objets vont évoluer lors des situations de contact. Ces lois sont discrétisées spatialement au travers d'une représentation géométrique. En général, la simulation mécanique aboutit à l'expression suivante :

$$M\ddot{\mathbf{x}} + D\dot{\mathbf{x}} + K(\mathbf{x}).\mathbf{x} = \mathbf{f} \tag{1.21}$$

où $\mathbf{x}$ représente la position, $\dot{\mathbf{x}}$ représente la vitesse, et $\ddot{\mathbf{x}}$ l'accélération des degrés de liberté de l'objet. La matrice M représente la matrice de masse et D la matrice d'amortissement.

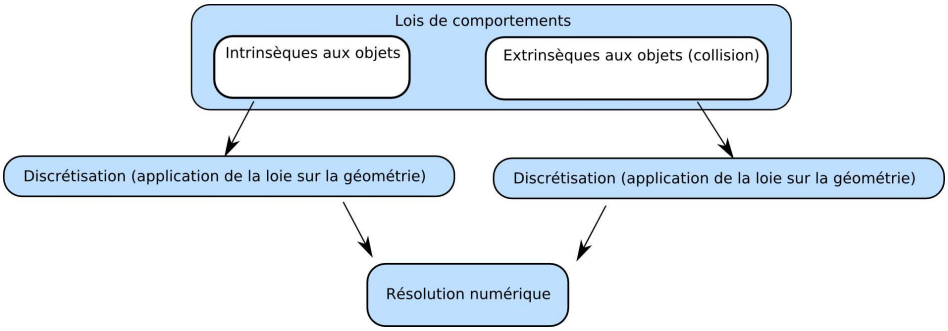

Fig. 1.33 : *Résolution des équations décrivant le mouvement des objets simulés.*

La matrice K représente la matrice de raideur de l'objet et f les forces. On priviligie les matrices diagonales, bandes ou creuses, car elles simplifient les calculs. On parle de résolution dynamique du système, lorsqu'on résout directement l'équation 1.21. Si les termes mettant en jeu les dérivées temporelles sont supprimés (eqn. 1.22), on arrive alors à une formulation statique du problème.

$$\underbrace{M\ddot{\mathbf{x}} + D\dot{\mathbf{x}}}_{\text{dérivée suivant le temps}} + K(\mathbf{x}).\mathbf{x} = \mathbf{f} \tag{1.22}$$

1.4.1 Le cas de la résolution dynamique

Dans le cadre dynamique, le mouvement des objets se calcule comme une suite de positions à des instants t successifs. L'équation différentielle d'ordre 2 (eqn. 1.21) est reformulée sous la forme de deux équations d'ordre 1 appelée forme de *Cauchy*. A partir d'une équation du type $\ddot{x} = f(t, x, \dot{x})$, on pose :

$$\begin{cases} \dot{x} &= v \\ \dot{v} &= f(t, x, v) \end{cases} \tag{1.23}$$

$$V = \begin{bmatrix} x \\ v \end{bmatrix} \tag{1.24}$$

$$\frac{\delta}{\delta t}\begin{bmatrix} x \\ v \end{bmatrix} = f(t, \begin{bmatrix} x \\ v \end{bmatrix}) \tag{1.25}$$

$$\frac{\delta V}{\delta t} = f(t, V) \tag{1.26}$$

Cela revient à résoudre un système d'équations différentielles ordinaires du premier ordre. De même, le vecteur noté V, dans l'équation 1.24, est communément appelé *vecteur d'état* ; la simulation consistant à calculer les vecteurs d'état (notés $V(t_i)$) successifs. Pour cela, il y a deux possibilités :

– la première utilise, pour le calcul de $V(t_i)$, les dérivées aux instants précédents : on parle dans ce cas de méthodes *explicites*. Suivant le schéma de discrétisation adopté, il existe différentes méthodes explicites plus ou moins précises. La précision des méthodes explicites est un critère important. Ces erreurs tendent à augmenter l'énergie de la simulation à chaque pas de temps. Il en résulte un phénomène de divergence qui entraine l'arrêt de la simulation. En pratique, on retiendra que les systèmes raides sont aussi plus difficiles à résoudre ; l'erreur s'y accumulant plus vite. La raideur du système est conditionnée par la rigidité des objets mécaniques (mou/rigide) et par la rigidité des collisions. Face à ce problème de stabilité, on peut diminuer le pas de temps, ce qui évite la divergence de la simulation si on fait de la détection de collision spatiale ; diminuer le pas de temps réduit aussi le risque de manquer des collisions. Par contre, il implique une augmentation des calculs.
– la seconde résout un système d'équations dont les dérivées $V(t_i)$ à l'instant courant (c'est-à-dire t_i) sont des inconnues : il s'agit des méthodes *implicites*. Les méthodes implicites supposent la résolution d'un système non-linéaire : par linéarisation du système, puis utilisation de méthodes exactes (Gauss ou décomposition LU), ou encore par des méthodes itératives (technique du gradient conjugé) [gsl]. Quelle que soit la méthode choisie, la résolution du système implicite est nettement plus coûteuse. En contrepartie, l'intégration implicite du mouvement ne souffre pas des mêmes problèmes de stabilité que ceux rencontrés avec les méthodes explicites [BW98].

Une étude complète des différentes techniques d'intégration est disponible dans [Hil02]. La résolution des interactions entre les objets suppose que les vecteurs d'état soient regroupés et résolus de manière globale. Cette contrainte est assez forte, car elle suppose l'utilisation d'un pas de temps commun à tous les objets et le plus petit possible pour garantir la stabilité du système. Pour ces raisons, on découple [Hil02] les objets de comportement mécanique différent ; cela permet de n'utiliser un petit pas de temps que pour les objets de raideur élevée. Cependant les collisions doivent toujours être résolues de manière globale. [Deq05] propose une architecture qui découple complètement les objets. Il contraint les collisions à une méthode de pénalité, les forces de pénalité étant ajoutées de manière explicite au bilan des forces de l'objet.

1.4.2 Le cas de la résolution statique

Lorsque les termes dépendant du temps sont négligés (1.21), on parlera de résolution statique. Le système à résoudre se ramenant à :

$$K(\mathbf{x}).\mathbf{x} = \mathbf{f} \tag{1.27}$$

On cherche un état d'équilibre, une configuration unique $\mathbf{x}$, minimisant l'énergie du système en fonction des forces externes $\mathbf{f}$. Le défaut principal de ce genre d'approche est celui de passer instantanément d'une position à une autre, sans position intermédiaire, sans cohérence temporelle. Pour émuler le comportement dynamique de l'objet, on utilise un algorithme de résolution de système linéaire itératif. Chaque étape de résolution est alors associée à l'avancement d'un pas de temps. En général, l'approche statique est bien adaptée à la simulation d'objets déformables fixés ou contrôlés par l'utilisateur.

1.4.3 Contraintes temporelles d'un environnement virtuel simulé

L'utilisation d'un moteur de simulation pour modéliser un environnement virtuel implique un certain nombre de contraintes relatives au paramétrage du pas de temps. En effet, les seuls critères de paramétrage du pas de temps d'une simulation non-interactive sont le temps de calcul de la simulation et sa stabilité. Ces critères ne suffisent plus lorsque l'utilisateur agit sur la simulation. Pour cela, il faut poser une certain nombre de contraintes :

- *la synchronicité des temps :* les actions de l'utilisateur doivent être synchrones avec l'environnement simulé. Il faut que le temps tel qu'il est vu au niveau de la simulation soit identique au temps réel écoulé pour l'utilisateur. Cette contrainte implique que l'intégralité du mouvement pendant un pas de temps Δt soit calculée en un temps inférieur à Δt.
- *la latence :* le critère de synchronicité n'est pas suffisant pour avoir une simulation interactive de bonne qualité. Pour cela, il faut ajouter la contrainte de latence. Cette latence, mesurée en ms, est la différence entre une modification de la scène et sa perception par l'utilisateur. Lorsque l'utilisateur n'agit pas sur la scène, une borne maximale de la latence peut être définie à partir de la persistance rétinienne, soit $\frac{1s}{25} = 40ms$. Cette borne devient insuffisante pour un périphérique haptique dont le seuil de persistance est situé au delà des 300Hz[7]. Cela signifie qu'un utilisateur qui agit sur la simulation s'attend à percevoir le résultat en moins de $\frac{1}{300} = 3.3ms$.

Lorsque les contraintes de synchronicité et de latence sont respectées, le pas de temps (et donc le temps de calcul) est inférieur à $3.3ms$, on parle de simulation temps-réel . Malheureusement, une durée de $3.3ms$ n'est pas suffisante pour simuler des environnements complexes (une dizaine d'objets), c'est pourquoi on a parfois recours au terme de *temps interactif*. Cela signifie que les contraintes ne sont pas respectées, mais que cela nuit peu (notion toute relative) à l'interaction. Nous avons observé dans nos tests qu'une désynchronisation des temps d'un facteur deux entre le temps simulé et le temps utilisateur constitue une limite acceptable pour l'interaction. Une étude plus fine, basée sur la perception et la psychologie, serait nécessaire pour quantifier de manière plus précise une telle limite d'acceptabilité de la désynchronisation.

1.5 Découpe de modèles mécaniques

Un objet découpable est un objet qui subit des changements topologiques de *suppression de voisinage* ; l'opération de découpe consistant à briser une relation entre deux éléments voisins. Les incidences d'une découpe sur la simulation mécanique sont nombreuses. D'une part, la notion de voisinage est intimement liée aux processus de modélisation mécanique (mécanique des milieux continus, maillage) et d'autre part, l'efficacité des techniques d'animation temps-réel et d'accélération de la détection de collision reposent sur des structures de données qui sont précalculées avant la simulation. Dans cette thèse, le même terme de découpe regroupe plusieurs situations réelles différentes. Ce peut être à la fois le résultat obtenu par la fracture d'un objet ou un organe découpé par le scalpel d'un chirurgien ; la différence réside dans la manière dont est généré le profil de découpe. En pratique, ce n'est qu'une fois ce profil calculé qu'on applique réellement la découpe au modèle mécanique. Suivant que le modèle est simulé à

[7] On trouve dans la littérature un certain nombre de mesures de ce seuil de persistance haptique. Suivant les études, le phénomène de persistance commence pour des fréquences de 300Hz jusqu'à 1000Hz.

l'aide d'une technique avec/sans maillage, l'application réelle de la découpe est plus ou moins complexe. La découpe d'objet virtuel a déjà fait l'objet d'un certain nombre de travaux, dont nous nous sommes inspirés, pour faire cet état de l'art [Cot98][For03] [Nie03][MBF04].

1.5.1 Calcul du profil de découpe

Dans la suite du mémoire, nous distinguerons deux types de profil de découpe : celui produit par un outil, et celui produit par une fracture de l'objet. Dans le cadre de la simulation médicale, il est important de modéliser le processus de découpe des organes. En général, on procède de façon discrète, en approchant le chemin suivi par l'outil. C'est ce chemin qui définit le profil de la découpe [GCMS00b]. Le profil de découpe est fourni par l'utilisateur, ce qui n'est pas le cas lorsqu'on modélise la fracture ; ce processus de fracture intervenant lorsqu'un objet est soumis à des efforts mécaniques trop importants. La modélisation des fractures s'appuie sur la mécanique des milieux continus ; le profil de la découpe étant obtenu par les théories physiques adaptées [Ngu90]. Dans [Nie03], on trouve des approches intermédiaires entre la découpe géométrique et la fracture. Elles modélisent de manière physique l'interaction entre l'outil du chirurgien et le matériau, matériau qui se déforme et finit par se fracturer. L'approche suivie dans la thèse concerne uniquement l'insertion d'une aiguille dans un objet déformable. De telles modélisations ad-hoc peuvent traduire les fractures qui se forment et permettre d'obtenir une simulation crédible.

1.5.2 Découpe dans les modèles maillés

Les méthodes de simulation les plus courantes s'appuient sur l'utilisation d'un maillage. La découpe de tels objets est possible en modifiant directement le maillage. Cependant le maillage doit rester de qualité pour le calcul numérique (ch. 1.2.2.0). Cette contrainte de qualité est particulièrement sensible lorsqu'on utilise des Éléments Finis pour résoudre les équations de la mécanique des milieux continus. Pour garantir une bonne précision à la méthode de résolution, il faut que les éléments soient de taille identique et qu'il n'y ait pas d'éléments dégénérés.

Dans le cadre de la méthode des Éléments Finis, le maillage devrait être recalculé pour tenir compte de la nouvelle forme de l'objet. Mais recalculer complètement le maillage n'est pas envisageable : les temps de calculs pouvant être de l'ordre de l'heure, si on utilise certains algorithmes [Mol04] ! C'est pourquoi des alternatives au remaillage sont proposées : 1) la suppression d'éléments, 2) la séparation de faces, 3) la subdivision d'éléments, 4) le remaillage local, 5) la multireprésentation.

Suppression d'éléments

La suppression d'éléments est probablement la manière la plus simple d'implémenter des opérations de découpe et de fracture [Cot98]. Suivant la technique d'animation utilisée, on supprimera un ressort, une facette, ou bien un tétraèdre du maillage servant de support aux calculs mécaniques. Par cette opération, le nombre de noeuds mécaniques diminue et par là même, les temps de calcul, ce qui présente un intérêt certain. Lors de découpe par suppression d'éléments, il faut faire attention à garder la cohérence du maillage initial. Par exemple, la

suppression d'éléments dans un maillage tétraédrique entraîne l'apparition de points singuliers qui doivent être traités de manière adéquate [FDA02].

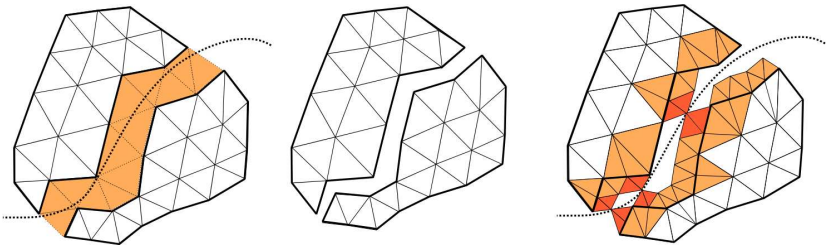

Fig. 1.34 : *Découpe par suppression d'éléments. De gauche à droite : a) Les éléments intersectés par la découpe sont supprimés. b) L'objet découpé a perdu une partie de sa masse, le front de coupe n'est pas visuellement satisfaisant. c) Découpe améliorée, les éléments intersectés sont subdivisés ; les éléments restant en intersection sont supprimés.*

La suppression des éléments entraîne aussi l'apparition d'artefacts visuels (c'est grossièrement découpé !) et mécaniques : il y a suppression de matière (fig. 1.35).

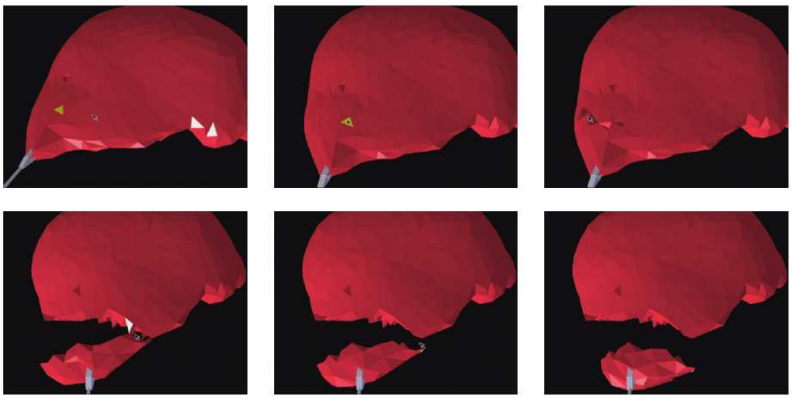

Fig. 1.35 : *Découpe par suppression d'éléments. On observe que la suppression d'éléments est satisfaisante lorsque la découpe est irrégulière, mais qu'elle ne permet pas de représenter une découpe fine. (Extrait de [Cot98])*

Séparation de faces

La technique de séparation de faces possède le même objectif que celui de la suppression d'éléments, à savoir : permettre la découpe d'un objet déformable sans entraîner de baisse de performance de la simulation.La technique consiste à séparer les éléments du modèle le long des bords de ces-derniers (fig. 1.36). Le nombre d'éléments à simuler reste constant, ce qui est très avantageux en terme de performance. En plus, il n'y a pas de disparition de matière dans le système mécanique, ce qui est une propriété recherchée.

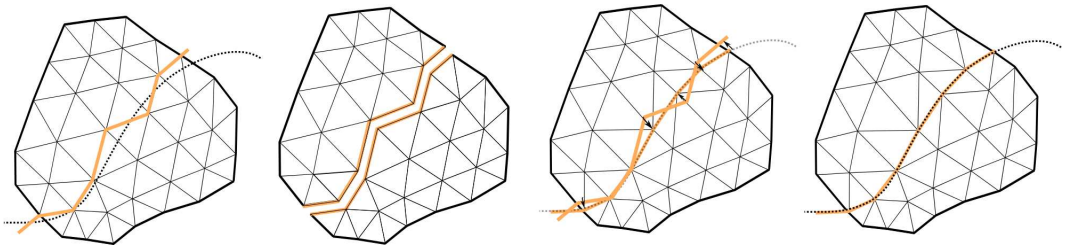

Fig. 1.36 : *Découpe par séparation, de gauche à droite : a) Recherche des arêtes/faces qui approchent le front de coupe. b) Séparation des arêtes. c) Amélioration du résultat visuel en projetant les faces séparées sur la découpe.*

La technique de séparation de faces donne de bons résultats en 2d ; ainsi, dans le cadre du développement d'un simulateur de capsulo-rexis, nous avons expérimenté cette approche pour découper un tissu animé à l'aide d'un maillage masses-ressorts. Pour adoucir les bords de la découpe, on a recours à une technique de subdivision (Spline) (fig. (1.37). Si les résultats sont très convaincants en 2d, on remarque que l'extension de la séparation de faces à la découpe volumique est problématique, et n'a, semble-t-il, pas été traitée. La difficulté de l'extension 3d provient de l'identification de la liste de faces à séparer et de la manière dont ces faces devraient être projetées sur le chemin de l'outil sans créer de distorsions du maillage.

Fig. 1.37 : *Découpe du rhexis (tissu à la surface l'oeil). On utilise la technique de la séparation de faces.*

Découpe par subdivision d'éléments

La découpe par subdivision consiste à ne découper que les éléments du maillage qui intersectent le front de découpe (fig. 1.38). Suivant le type d'éléments composant le maillage, différents profils de découpe sont utilisés. Dans [MK00], les auteurs proposent une liste de motifs de découpe d'éléments triangulaires et tétraédriques.

On peut remarquer que le nombre d'éléments augmente à chaque opération de découpe. Pour éviter qu'il ne croisse trop vite, [BGTG04] propose de ne subdiviser effectivement un tétraèdre que lorsqu'il n'est plus en intersection avec l'outil de découpe. Un automate évalue alors, pour les tétraèdres en cours de découpe, quel motif de subdivision est le plus adéquat.

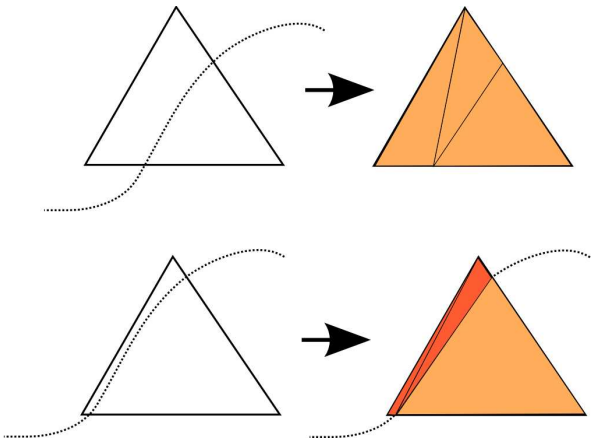

Fig. 1.38 : *Découpe par subdivision : suivant le profil de la découpe, des éléments mal conditionnés – représentés en rouge – peuvent apparaître.*

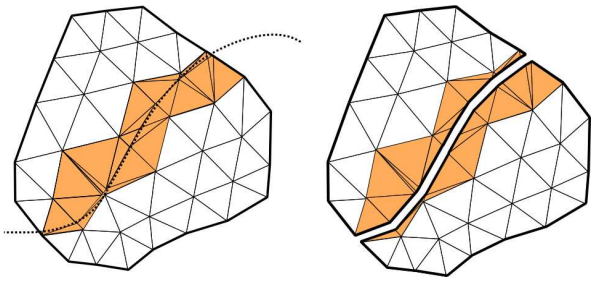

Fig. 1.39 : *Découpe par subdivision : les primitives qui intersectent le front de découpe sont subdivisées. La zone découpée est visuellement satisfaisante. On remarque l'apparition rapide de primitives non régulières (en rouge sur le dessin).*

La découpe basée sur la subdivision reproduit fidèlement (fig 1.39)la forme de la découpe en un temps de calcul compatible avec la simulation temps-réel . En contrepartie, les éléments générés à l'issue de la subdivision sont de très mauvaise qualité (fig. 1.38). L'animation de maillage comportant des éléments mal conditionnés suppose une technique basée soit sur les masses-ressorts, soit sur les d'Éléments Finis inversibles [ITF06].

Remaillage

Hors du contexte de la simulation temps-réel, un certain nombre d'auteurs ont proposé des techniques de remaillage local [OH99][OBH02]. Ces méthodes régularisent le maillage aux alentours du front de découpe ; cela permet de diminuer le nombre d'éléments mal conditionnés.

On citera enfin les travaux suivants [GO01][GCMS00a], qui sont intermédiaires entre les méthodes par subdivision et les méthodes par remaillage : d'une part, il y a subdivision des éléments découpés et d'autre part, remaillage par la création d'un modèle multirésolution du maillage à simuler.

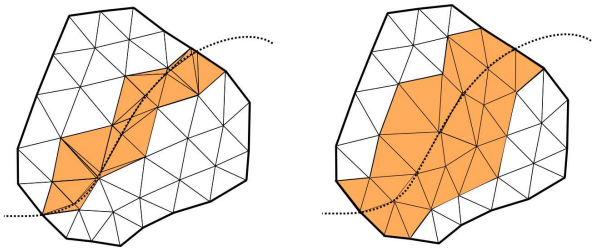

Fig. 1.40 : *Découpe par remaillage : elle entraîne une augmentation rapide du nombre d'éléments à simuler.*

Le Virtual node : la multireprésentation

On décrit la multireprésentation plus en détail dans le chapitre suivant (2.1.3). L'idée de base est de découpler le volume de matière simulé du volume de matière affiché ; l'affichage est alors contrôlé par le déplacement des points mécaniques. Dans l'approche de Molino [MBF04], les objets sont immergés dans un maillage tétraédrique et simulés à l'aide de la méthode des Éléments Finis. Pendant les découpes, le maillage affiché et le maillage simulé se différencient. Cela permet d'avoir des découpes/fractures visuellement satisfaisantes (fig. 1.41) tout en offrant les avantages suivants :

- stabilité : aucun élément mal conditionné n'est inséré dans la simulation.
- rapidité : le nombre d'éléments n'augmente que peu.

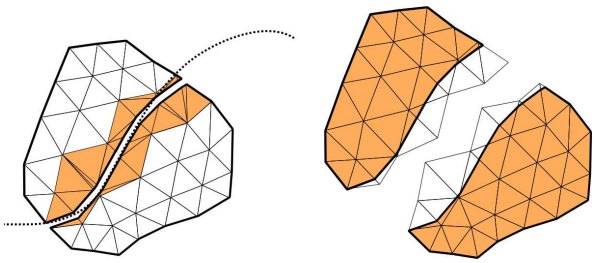

Fig. 1.41 : *Découpe par la technique du virtual node ; à gauche, sans virtual node, des éléments mal conditionnés apparaissent. A droite, l'objet (en orange) est découplé du maillage ; les tétraèdres découpés étant dupliqués.*

Une approche assez semblable est présente dans [MTG04] où des objets complexes sont immergés dans une grille de voxel et animés par la méthode des Éléments Finis. Les faces de la grille de voxel peuvent se séparer et ainsi simuler des comportements de fracture, l'affichage suivant les fractures du maillage mécanique.

De notre point de vue, la multireprésentation est actuellement la seule approche qui permette, à la fois d'utiliser une méthode de simulation physiquement validée (FEM), et une découpe fine des objets, le tout, dans une simulation temps-réel.

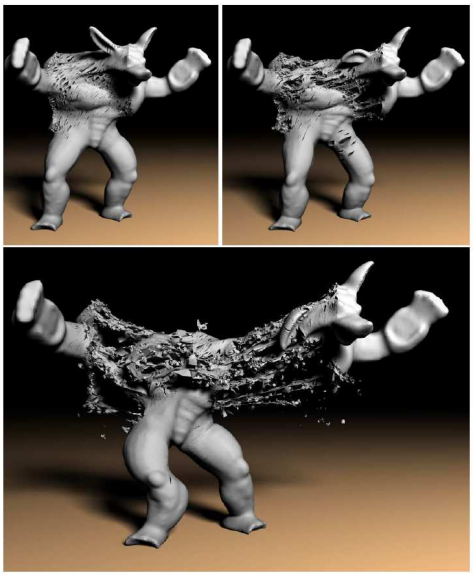

Fig. 1.42 : *Extrait de [MBF04] : résultat obtenu par la méthode du virtual node.*

Découpe par contrôle de la fonction de base

La plupart des techniques de découpe modifient directement le maillage de l'objet. Pourtant, une découpe peut être insérée à l'aide du contrôle de la base de l'interpolation. On décrit plus en détail le principe de la découpe par contrôle de la fonction de base dans le chapitre suivant. Pour l'instant, on retiendra que c'est une méthode largement employée en mécanique, sous le nom d'Éléments Finis Étendus (XFEM) [SMMB00]. L'avantage de la découpe par contrôle de la fonction de base est que le nombre d'éléments simulés (et leur forme) ne change pas. On peut trouver une utilisation de la méthode des XFEM pour la simulation en temps réel d'objets déformables [VVW04].

1.5.3 Découpe dans les modèles sans maillage

Les modèles animés par des méthodes sans maillage se prêtent plus naturellement à la découpe. En effet, pour garantir la stabilité de la simulation, les méthodes qui sont basées sur un maillage imposent que celui-ci soit de bonne qualité (sans éléments dégénérés) hors cela suppose des temps de calcul très importants. En ne reposant pas sur maillage pour représenter la géométrie des objets, les méthodes sans maillage peuvent intégrer la découpe nettement plus facilement. Dans [MKN+04], un graphe est construit, traduisant la proximité des sphères utilisées par le processus d'interpolation du champ de déplacement. Lors des phases de changement topologique, le graphe est supprimé puis reconstruit pour fixer la nouvelle forme de l'objet. Dans un tel objet, la découpe consiste simplement à supprimer des arêtes du graphe. Le domaine de la mécanique s'est aussi intéressé à la découpe dans les modèles sans maillage. Différentes méthodes de contrôle topologique modifient la fonction de distance (et donc implicitement la topologie) utilisée par le processus d'interpolation de la géométrie. Certaines approches proposent de remplacer la distance euclidienne par la distance

géodésique : le plus court chemin ne sortant pas de l'objet. D'autres s'appuient sur une notion de transparence qui décrit la visibilité entre deux noyaux d'interpolation. La fonction de poids prend alors en compte ce facteur de visibilité et diminue d'autant l'influence d'un échantillon.

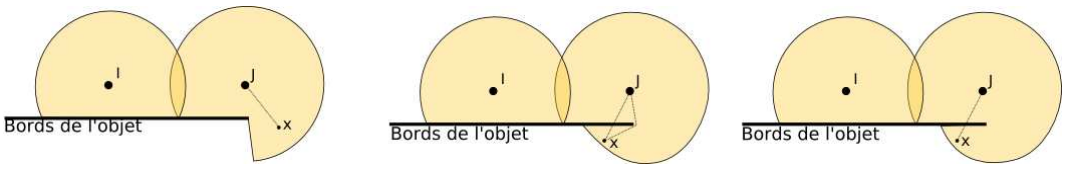

Fig. 1.43 : *Différents critères de contrôle topologique : la visibilité, la diffraction et la transparence.*

On trouve dans [PKA+05] l'utilisation de telles méthodes appliquées au domaine de la simulation de fracture pour l'informatique graphique (fig. 1.44).

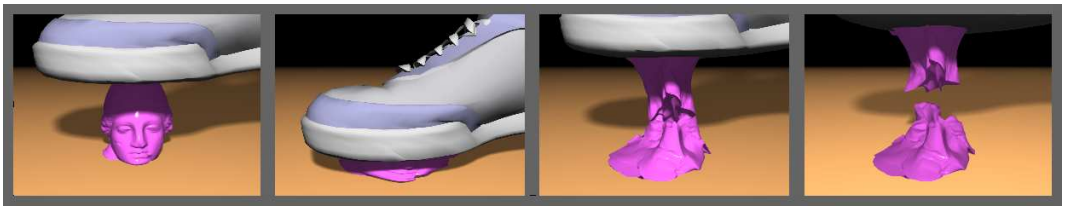

Fig. 1.44 : *Extrait de [PKA+05] qui s'appuie sur un contrôle du mélange.*

Conclusion

Cette description des différentes composantes d'un moteur de simulation montre la grande diversité des problèmes à résoudre (collision, déformation, intégration) ainsi que le grand nombre d'approches possible pour chacun d'eux.

Au niveau de la simulation mécanique des déformations, les méthodes qui dérivent de la mécanique des milieux continus ont l'avantage de fournir un cadre formel à la description précise des lois de comportements : linéaire/non linéaire, isotrope/anisotrope, matériaux ductiles/fragiles.... En outre, on a vu au travers de nombreuses méthodes, que des simplifications sont envisageables et permettraient la simulation en temps réel. Face aux méthodes qui dérivent de la mécanique des milieux continus, on trouve les méthodes à géométrie discrète. L'utilisation de ces méthodes nécessite un investissement théorique moindre et ces-dernières peuvent facilement être implémentées dans un simulateur. En revanche, la description de comportement précis suppose une modélisation géométrique/mécanique ad-hoc et les phases d'intégration temporelle du mouvement demandent des intégrateurs robustes ou de petits pas de temps (ralentissant la simulation).

Du point de vue de la découpe, les méthodes à géométrie discrète semblent, dans un premier temps, plus appropriées ; il suffit de briser les relations liant les particules. Ainsi, dans le cas d'un maillage masses-ressorts, la suppression d'une arête du maillage entraîne la suppression des forces de rappel qui lient les particules. En contrepartie, si on cherche à préserver les propriétés mécaniques (comme la raideur), il faut mettre en place des mécanismes particuliers assez complexes. Plus précise, la découpe de modèles issus de la mécanique des milieux continus, se heurte bien souvent à l'utilisation d'un maillage qu'il faut mettre à jour de manière adaptée. Les méthodes sans maillage sont alors sensiblement plus simples pour modéliser des processus de découpe, mais leur temps de calcul est aussi plus élevé.

Enfin, au niveau de la détection de collision, on observe une évolution de l'activité de recherche qui tente de relâcher de plus en plus les contraintes géométriques et topologiques pesant sur les objets simulés. On s'intéressait initialement aux rigides-convexes puis non-convexes ; maintenant on s'intéresse aux objets déformables et pour certaines méthodes récentes aux objets découpables/fracturables.

Chapitre 2

La multireprésentation pour la simulation

Sommaire

Introduction

Comme nous l'avons mis en évidence dans le premier chapitre, les avancées des dernières années en simulation visent à proposer de nouvelles solutions algorithmiques ou numériques. Pour construire un logiciel permettant de simuler ensemble des objets variés ainsi que leurs interactions, il faut concevoir une architecture souple et évolutive et donc nous intéresser au génie logiciel. Les notions classiques du génie logiciel comme la généricité, la réutilisation de code ou l'efficacité sont des aspects qu'il est intéressant d'étudier dans le cadre des moteurs logiciels de simulation. Même si le développement de simulateurs n'est pas notre objectif premier, le fait de participer, avec l'équipe Alcove, à la création du moteur de simulation Sofa [sof] a fortement inspiré et structuré le présent travail. Par opposition au chapitre précédant, nous présentons maintenant, une vue de la simulation d'objets déformables, « teintée » de génie logiciel. Dans un premier temps, nous recensons et décrivons les différentes manières de structurer les objets déformables, trouvées dans la littérature : monolithique, découpage par aspect/composant, utilisant la multireprésentation ou reposant sur le principe du multimodèle. En proposant cette classification, nous espérons favoriser la définition d'un vocabulaire commun, qui ne semble pas exister pour l'instant.

Un point particulier de cette classification tend à recentrer la structuration des objets virtuels dans les simulateurs autour d'une représentation géométrique centrale. En effet, nous considérons que la géométrie est le dénominateur commun de l'ensemble des aspects d'un objet virtuel. Étudier les représentations géométriques retiendra notre attention. Cet intérêt est relié aussi à la modélisation des objets découpables. La découpe est une opération de changement topologique, ce qui implique un lien serré avec la représentation géométrique utilisée. Nous énumérerons donc les représentations géométriques usuelles de l'informatique graphique (employées pour la mécanique, l'affichage et la collision) que nous ramènerons progressivement à une modélisation unique, sous la forme d'un problème d'interpolation. Notre approche, proposant de mettre l'interpolation géométrique en avant, induit une nouvelle vision des découpes d'objets. Ce chapitre, en donnant une place importante aux développements logiciels et en mettant la géométrie et les interpolations au cœur de la simulation, introduit nos contributions des chapitres suivants (3, 4 et 5).

2.1 Structure des objets virtuels dans un simulateur

La manière de programmer des objets virtuels est une composante importante du développement d'un moteur de simulation. Dans les architectures modernes, [sof][MDH+02][CGTS04][Fau], on fait de plus en plus référence au découplage des différents aspects d'un objet virtuel : mécanique, collision et affichage. L'objectif de ce découplage est de favoriser la réutilisation de code (par exemple : code de simulation, algorithme de détection de collision, code associé à un objet particulier) d'un environnement virtuel à un autre. En faisant cela, on rejoint les objectifs du génie logiciel : généricité, réutilisabilité et stabilité. Parmi les différentes évolutions des architectures d'objets virtuels, on distingue les approches monolithiques, les approches par aspects, les approches multimodèle et enfin les approches mono et multireprésentation.

2.1.1 Structure monolithique vs. découpage en aspect

En se penchant sur les premiers travaux en simulation, on remarque que les objets y sont conçus sous la forme d'une seul bloc logiciel (fig. 2.1.a). A un objet virtuel correspond un *objet* au sens du génie logiciel. Plus tard, pour simplifier les développements, les différents aspects d'un objet virtuel sont séparés dans des entités logicielles différentes (fig. 2.1.b). Les aspects les plus connus au niveau de la modélisation d'objets virtuels sont ceux de détection de collision, de calcul de l'animation et d'affichage. Le découpage sous forme d'aspects est lié au fait que chacun des aspects nécessite des algorithmes « métier » complexes ; il correspond en général aux différents domaines d'expertises impliqués. Ce découpage apporte une plus grande clarté de programmation. Du point de vue d'une méthodologie de programmation, le découpage par aspect (son nom ne l'indique pas[8]) se rapproche de la programmation par composants.

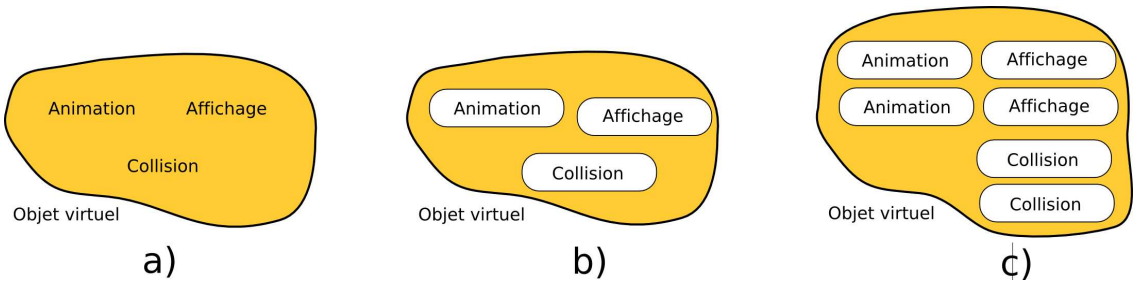

Fig. 2.1 : *a) approche monolithique, b) découpage en aspect, c) multimodèles.*

2.1.2 Multimodèle

Certains travaux, [Gri05][Deq05], explorent de nouvelles façons de modéliser les objets virtuels au travers d'approches appelées *multimodèle*. L'idée à la base du multimodèle est d'associer plusieurs modélisations différentes d'un même aspect. Ainsi, un objet virtuel est doté de plusieurs modèles mécaniques : rigide, déformable, etc ... ; ces modèles étant activés en fonction du contexte d'utilisation. Voici quelques domaines d'application au multimodèle :

- performance : utiliser une mécanique rigide lorsqu'un objet est vu de loin et une mécanique déformable lorsque l'objet est plus proche ;
- extension du champ d'application : on étend le domaine d'utilisation d'un modèle en concaténant différents sous-modèles mécaniques : déformables linéaires, non-linéaires ;
- simplification des calculs : utiliser une modélisation des collisions par contact pour certaines paires d'objets et par pénalité pour d'autres.

Le concept de multimodélisation se rapproche de la programmation d'agents. Les objets sont dotés d'une certaine autonomie, permettant le choix du modèle à employer en fonction du contexte. On peut se demander si le multimodèle ne supposerait pas une description sémantique des propriétés des modèles ainsi que de l'environnement.

[8] Même si le mot peut prêter à confusion, il ne faut pas confondre le découpage par aspect d'une architecture d'objet virtuel à la programmation par aspect telle que définie dans [SPDC06].

2.1.3 Mono et multireprésentation

La mise sous forme d'aspect est essentiellement un découpage logiciel d'un code monolithique. L'apport de l'utilisation d'un découpage par aspect est localisé au niveau de la clarté du code produit et de sa maintenance ; il n'y a pas d'influence au niveau des techniques de modélisation. Cette approche ne permet pas de décrire complètement certaines architectures comme [MDH+02]. En effet, dans cette dernière, on autorise chaque aspect à utiliser une représentation géométrique particulière ; ainsi, on utilise systématiquement des sphères pour la gestion des collisions alors que les affichages sont des surfaces triangulées, l'aspect "simulation mécanique" n'a par contre par de représentation prédéfinie (maillages, particules, matrice d'inertie ...). Cela nous conduit donc à séparer les approches qui privilégient une géométrie particulière (monoreprésentation), des approches qui utilisent plusieurs représentations géométriques. Quelques exemples vont illustrer l'intérêt de découpler les représentations géométriques.

Exemple 1 *Nous sommes beaucoup plus sensibles à la complexité visuelle d'un objet qu'à son comportement dynamique. Cette information nous permet, par exemple, de faire les calculs dynamiques en s'appuyant sur une représentation géométrique peu raffinée, puisque c'est la partie la plus consommatrice de temps de calcul. A contrario, il est préférable d'utiliser des géométries précises pour l'affichage.*

Exemple 2 *La dimension de la représentation géométrique (1d, 2d ou 3d) n'est pas forcément identique suivant l'aspect traité. Ainsi, le calcul de simulation des déformations suppose assez généralement une représentation volumique, tandis que l'affichage peut se contenter d'une géométrie de dimension deux (une surface) ; ou encore, un modèle mécanique 1d pour simuler des objets volumiques filiformes [Len04][RGF+04].*

Ces considérations entraînent le développement, notamment dans le projet Sofa [sof], de ce que nous appelons *multireprésentation* (sous-entendu : multireprésentation géométrique). La mise à jour des différentes représentations constitue alors un point tout à fait particulier propre aux approches multireprésentation. En général, c'est le moteur de simulation qui calcule les déplacements/déformations des objets, et la représentation mécanique agit comme un maître en déformant les autres représentations.

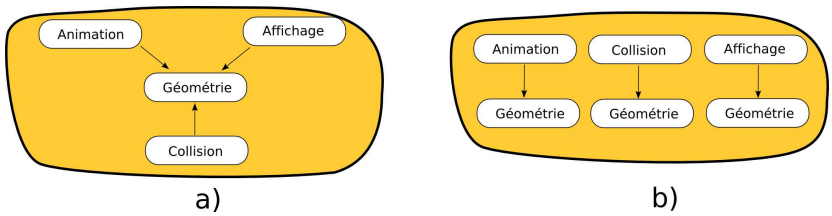

Fig. 2.2 : *a) monoreprésentation : une seule représentation commune de la géométrie b) multireprésentation : les géométries de chacun des aspects peuvent différer.*

Ce processus ressemble aux techniques de modélisation par fonctions de transformation, comme les FFD [SP86] ou le skinning [Blo02]. De manière plus formelle [IF02], soit $\mathbf{u}(x)$, la

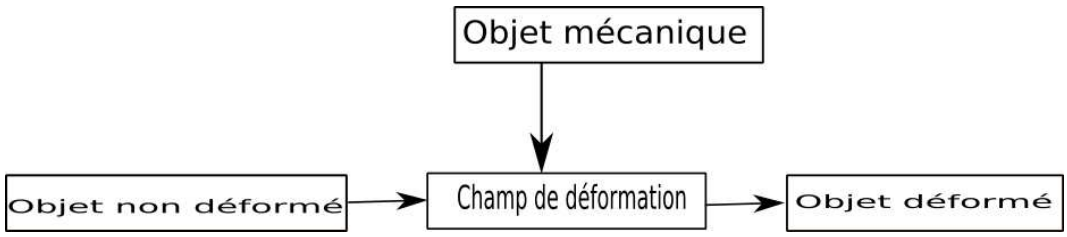

Fig. 2.3 : *Multireprésentation : un objet est déformé par un champ de déformation. Le champ de déformation étant contrôlé par la représentation mécanique.*

représentation géométrique qui, à une paramétrisation x donnée (pas forcément de la même dimension que $\mathbf{u}(x)$) , associe un point de l'objet ou de sa surface dans R^3, et soit $\mathbf{f}(x)$, la transformation qui va déplacer/déformer la géométrie de l'espace au repos vers l'espace déformé X.

$$X = \mathbf{f}(\mathbf{u}(x)) \tag{2.1}$$

En pratique, la fonction $\mathbf{f}(x)$ est modélisée par :
- une combinaison de transformations rigides, c'est la technique du skinning ;
- à partir de points de contrôle à l'aide d'une technique d'interpolation comme les *Déformations de Formes Libres (FFD)* ou encore *l'interpolation aux moindres carrés mobiles* [MKN+04].

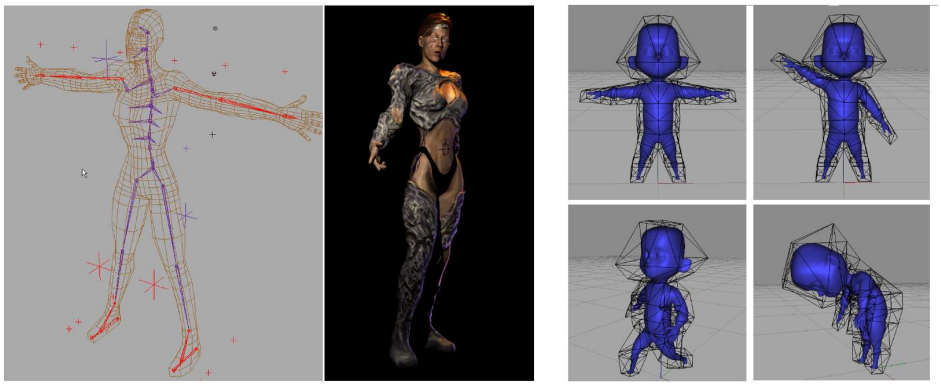

Fig. 2.4 : *A gauche, la transformation est définie par un squelette (extrait de [SK00]). A droite, la transformation est définie comme interpolation à partir de coordonnées barycentriques calculées sur une décomposition convexe (extrait de [DM06]).*

En pratique, les points de contrôle de la fonction de déformation sont pilotés par le modèle mécanique de l'objet. La multireprésentation est une solution très souple, car elle favorise l'adaptation de la représentation, mais aussi la résolution de chacune des géométries en fonction du réalisme (ou rapidité) désiré. C'est pourquoi la multireprésentation est souvent utilisée dans les approches mutlirésolution ou les approches à niveaux de détail. Elle permet par exemple de changer la résolution de la géométrie mécanique sans influer sur la géométrie de

l'affichage. On compare ainsi les approches multireprésentation et multirésolution de [Deb00] ou [DMG05] avec une approche monoreprésentation multirésolution de [GCMS00b].

La souplesse apportée par l'usage de la multireprésentation a un coût non négligeable quant à la complexité de la programmation du moteur de simulation et des objets virtuels [sof]. En outre, la découpe est aussi nettement plus complexe, car il faut non seulement découper chacune des représentations, mais aussi découper les transformations qui couplent les représentations. Ce point est discuté dans la section (2.3).

2.2 Modélisation géométrique

On a montré l'importance de la multireprésentation géométrique pour le développement des objets virtuels. Penchons nous maintenant, plus spécifiquement, sur les représentations géométriques. La communauté de l'informatique graphique s'intéresse depuis longtemps à la manière de représenter la géométrie des objets. Les différentes représentations géométriques proposées reposent sur un nombre très réduit de concepts mathématiques, à savoir la géométrie euclidienne (espace affine euclidien) et la topologie de l'espace de paramétrisation des objets. Cela nous permet d'élaborer une vue abstraite d'une représentation géométrique, regroupant les différentes représentations sous une forme unifiée. Cette vue abstraite assimile la représentation informatique d'une géométrie à un processus d'interpolation qui, à partir d'un petit nombre d'éléments discrets, permet la construction d'objets géométriques continus. Dans le cadre de la simulation, l'objectif des modélisations est de représenter des objets réels. Pour nos sens, on peut considérer qu'un tel objet est « continu » (bien que ce soit sans fondement physique) et qu'il peut donc être assimilé à un champ matériel continu M. Or, l'ordinateur, de mémoire finie, ne peut que représenter ce milieu de manière discrète, à partir d'un ensemble fini de paramètres. C'est à partir de ces paramètres qu'un processus d'interpolation est utilisé pour approximer la géométrie initiale. Sur la figure 2.5, on montre comment un objet matériel est modélisé (la surface ou le volume), puis discrétisé en un nombre fini d'échantillons $N = \{u_0, u_1, u_2, ...\}$. Ce sont ces échantillons qui sont interpolés de manière à reformer un champ continu, noté $\tilde{u}(x)$, approchant le champ matériel initial M.
Cette formulation a l'avantage de donner une éclairage original sur les différences entre les représentations avec maillage ou sans. En plus, considérer les représentations géométriques comme un problème d'interpolation favorise l'étude du contrôle de la continuité. En effet, nous considérons que l'interpolation est le fait de créer du continu à partir de valeurs discrètes dont l'inverse est l'insertion de discontinuités (de découpe).

2.2.1 Quelques représentations géométriques usuelles

Suivant leurs formulations mathématiques, les représentations géométriques sont en général classées en deux catégories : [Rog01], les formulations explicites et les formulations implicites. Un autre critère de classification possible est la présence ou l'absence de maillage. Cette classification rejoint le positionnement d'un grand nombre de travaux récents, comme les méthodes sans maillage pour la simulation [MKN+04], la reconstruction de surface sans maillage [AGP+02][Reu03][Lev03] ou encore le rendu à base de points. Mais commençons, tout d'abord, par rappeler les notions de formulations explicite et implicite avant de s'intéresser

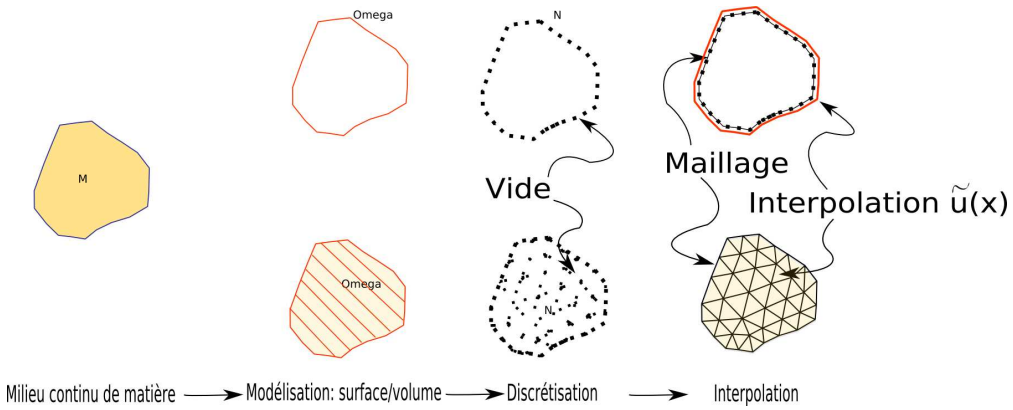

Fig. 2.5 : *Processus de modélisation d'un objet. Identification de ce qui est à modéliser, discrétisation, puis interpolation.*

aux représentations avec ou sans maillage. Pour simplifier, nous présenterons ces notions dans le cas 2d.

Les formulations explicites et implicites

On utilise des fonctions mathématiques pour représenter la géométrie des objets. On distingue deux formulations différentes : les formulations explicites et les formulations implicites :

- *les formulations explicites :* elles décrivent la géométrie comme un ensemble de points y qui peuvent être calculés à partir d'une fonction $y = f(x)$ (par évaluation de la fonction $f(x)$). Par exemple, pour représenter un segment de droite, on utilise l'équation analytique suivante :

  $$y = a.x + b$$

  Le contrôle de la droite s'effectue par le biais de coefficients a et b et de l'intervalle de définition du paramètre x. L'usage de fonctions analytiques est peu répandu en modélisation, celui des fonctions paramétriques lui étant préféré ; ces-dernières offrant un contrôle de la géométrie nettement plus intuitif. Par exemple, le segment de droite peut être mis sous une forme paramétrique, comme l'ensemble des points x, y calculés à partir des équations $f : \mathbb{R} \rightarrow \mathbb{R}^2$.

  $$f(t) = \begin{cases} x = b_x + t.(a_x - b_x) \\ y = b_y + t.(a_y - b_y) \end{cases}$$

  Le segment est alors défini à partir des extrémités a et b. Celui-ci peut ainsi être manipulé plus intuitivement en déplaçant les extrémités a et b qui deviennent des *points de contrôle.*
- *les formulations implicites :* elles décrivent la géométrie de l'objet comme l'ensemble des couples x, y qui vérifient l'équation $f(x, y) = 0$; c'est la résolution de l'équation qui donne les points de la courbe. Pour résoudre une telle équation, on a recours à des méthodes de recherche de racines assez coûteuses [wol]. Dans le domaine

de l'informatique graphique, les formulations implicites sont peu utilisées car assez complexes à manipuler, à l'exception notable des surfaces implicites à base de squelette comme les Blobs et autres MétaBalls.

On associe aussi aux différentes représentations (implicites ou explicites), une classe de continuité (notée C^k) qui indique si les dérivées d'ordre de 1 à K de la fonction existent et sont continues. C'est une donnée importante en modélisation, car elle décrit le caractère "lisse" ou non de la géométrie obtenue. C'est aussi un point à ne pas négliger lorsque la géométrie est employée pour discrétiser des quantités physiques. En effet, ces quantités sont ensuite impliquées dans des calculs nécessitant l'évaluation de leurs dérivées (comme pour la mécanique des milieux continus). On définit les classes de continuité suivantes :

- C^{-1} : la fonction décrivant la géométrie est discontinue ;
- C^0 : la fonction est continue sur l'ensemble du domaine mais n'est pas dérivable. C'est par exemple le cas des maillages tétraédriques des méthodes Éléments Finis utilisant l'interpolation linaire ou encore les maillages de surface pour le rendu ;
- C^1 : la fonction est dérivable une fois : certains types de splines ;

Représentations basées sur un maillage

Les représentations basées sur un maillage sont bien adaptées pour modéliser des objets structurés (non fluides). Elles s'appuient sur un encodage explicite de la topologie : le maillage. Ce maillage permet la construction de représentations géométriques peu coûteuses. Parmi les techniques basées sur un maillage, on trouve les assemblages de polyèdres (ou polygones en 2d et segments en 1d), les courbes et surfaces paramétriques comme les Splines, les courbes, surfaces et volumes à subdivision, ainsi que les représentations discrètes comme les grilles de voxel.

Maillage de polyèdres : les maillages de polyèdres sont les modèles les plus communs actuellement. Ils sont simples à manipuler, à implémenter et leur affichage est peu coûteux ; ce sont les primitives de base des cartes graphiques actuelles. Le maillage peut être obtenu par pavage du domaine à modéliser à l'aide de primitives élémentaires. On distingue les maillages formés d'éléments simplexes (segment en 1d, triangle en 2d, tétraèdre en 3d) et les maillages utilisant des primitives plus complexes (mais souvent convexes). L'avantage de l'utilisation de maillages simpliciaux (composés de simplexes) est double : d'une part, on dispose d'algorithmes efficaces de pavage (souvent basés sur la triangulation de Delaunay [cga][gts]), et d'autre part, chaque élément simplexe, formé de v_i sommets, dispose d'une paramétrisation locale simple à calculer sous la forme des coordonnées barycentriques $\alpha_i[x]$:

$$\begin{aligned} \alpha_i[x] &\leq 0 \\ \sum_i \alpha_i[x] &= 1 \\ \sum_i v_i.\alpha_i(x) &= x \end{aligned}$$

Cette paramétrisation ouvre la voie à un processus d'interpolation (des positions, des couleurs, des textures ...). Le choix d'autres primitives que celui des simplexes se traduit par une

augmentation de la complexité des algorithmes de pavage ; on peut aussi noter que la paramétrisation d'une cellule convexe est encore un sujet actif de recherche [JT93][DKT].

Les Splines : il s'agit de la famille de modèles la plus ancienne, dont les premières formes sont venues en solution à l'interpolation polynomiale par morceaux d'un nombre fini de points [Sch46]. Il existe plusieurs modèles de Splines [Gri99] se définissant tous comme une combinaison linéaire de fonctions prédéfinies :

$$C(t) = \sum_{i=0}^{n} q_i b_i(t)$$

Les constantes q_i codent les positions de points dans l'espace. Ces valeurs sont pondérées par la fonction de poids $b_i(t)$, puis mélangées (par la somme). Dans le cas 1d, la paramétrisation de t, est effectuée à partir d'un assemblage de segments qui, mis bout à bout, forment l'abscisse paramétrique de la fonction. A partir de la formulation 1d, les courbes splines sont étendues en 2d et 3d à l'aide du produit tensoriel. L'utilisation du produit tensoriel entraîne implicitement la création d'un maillage sous la forme d'une grille 2d ou 3d, ce qui est problématique pour représenter des objets de forme quelconque. Des travaux récents étendent les Splines aux domaines de paramétrisation basés sur des triangles [HB03]. Ces travaux sont en lien avec les méthodes de résolution d'équations différentielles de type Éléments Finis s'appuyant sur des fonctions de formes non linéaires.

Les courbes et surfaces à subdivision : la technique de subdivision construit la courbe $\tilde{u}(x)$ à partir d'un maillage initial subdivisé récursivement [ZSL+00]. Une étape de cette récursion revient à subdiviser le maillage, comme sur la figure 2.6, et à l'interpoler des valeurs portées par les noeuds. A la fin de cette étape, un nouveau maillage est produit et il sert à une nouvelle étape de subdivision. La subdivision est une méthode très efficace et qui produit des maillages

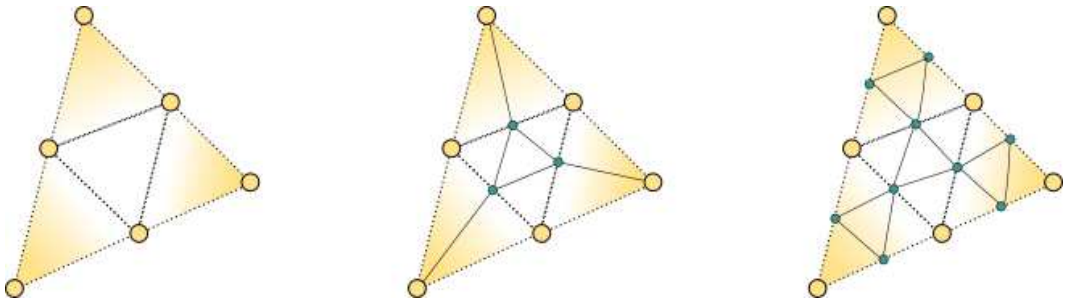

Fig. 2.6 : *Le maillage (à gauche) peut être subdivisé partiellement (au centre) ou compètement (à droite).*

lisses, tout en offrant un grand contrôle de la géométrie. Toute modification sur le maillage initial se répercute sur l'ensemble des subdivisions, et le contrôle est alors particulièrement intuitif.

Représentations sans maillage

Les méthodes sans maillage peuvent représenter des géométries fortement déformables : l'absence de maillage permettant une adaptation facile aux changements de topologie des objets. C'est pourquoi les fluides et/ou les objets a-topologiques (comme de la pâte à modeler) s'appuient sur des méthodes sans maillage.

Les méthodes implicites sans maillage : les fonctions implicites définies par squelette [Blo95][She99] sont probablement les méthodes de représentations géométriques les plus connues, car retenues dans la majorité des modeleurs 3D ; de telles représentations étant très utiles pour modéliser des objets non structurés (comme des fluides ou des éléments organiques). L'avantage des méthodes à base de squelette (type Blobs ou MétaBalls) est la simplification du contrôle de la forme obtenue. La fonction implicite $f(x)$ y est calculée comme un mélange (somme ou convolution) de fonctions simples à support compact. La primitive la plus simple est une sphère dotée d'une fonction de forme gausienne (fig. 2.7). Cependant, la méthode s'est étendue aux primitives plus complexes (cylindres, triangles). Dans le cas des sphères, la fonction $f(x)$ se calcule à partir des centres x_i et des rayons r_i des sphères. L'équation implicite à résoudre est alors :

$$f(x,y) = \sum_i f_i(x) \tag{2.2}$$

$$f_i(x,y) = w_i(\frac{||xx_i||}{r_i}); \tag{2.3}$$

La résolution des équations implicites suppose l'emploi d'un algorithme de recherche de racines pour extraire la surface à afficher [BF95].

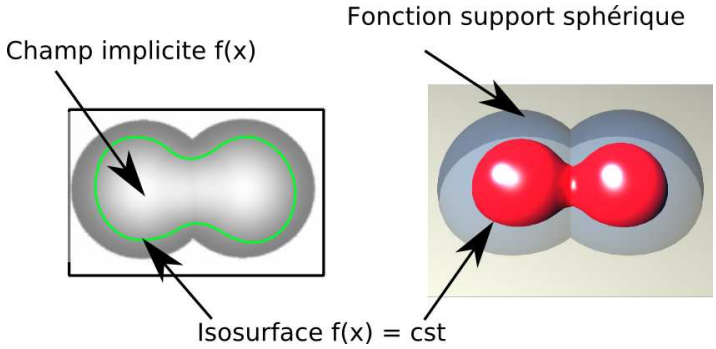

Fig. 2.7 : *Surface implicite à squelette (ici, le squelette est composé de points).*

Les méthodes explicites sans maillage : celles-ci s'appuient sur un processus d'interpolation, comme les fonctions à bases radiales (RBF) [RTSD03] ou l'interpolation aux moindres carrés mobiles. Il faut différencier les approches définissant un objet de même dimension que l'espace d'interpolation (volume dans $\mathbb{R}^3$), comme c'est le cas avec la méthode des moindres carrés mobiles, de celles qui construisent un objet de dimension inférieure (surface dans $\mathbb{R}^3$) [Lev03].

2.2.2 La géométrie : mélange d'interpolation et de topologie

Nous avons exploité les principales représentations géométriques de l'informatique graphique. Introduisons dès lors, une représentation unifiée de ces différentes représentations sous la forme d'un processus d'interpolation. L'intérêt de cette vue unifiée est de mettre en valeur les moyens de manipuler la continuité de la géométrie, et donc la découpe. Le processus d'interpolation s'appuiera sur l'approximation aux moindres carrés mobiles, présentée ci-dessous.

Approximation aux moindres carrés

L'approximation aux moindres carrés permet de construire un polynôme $\tilde{u}(x) = \alpha_1 + \alpha_2.x + \alpha_3.x^2...$ qui approche un ensemble fini d'échantillons localisés. Dans le cas 1d, nous notons la position de ces échantillons $x_0, x_1, x_2...$ et leurs valeurs $U = u_0, u_1, u_2, ...$ L'approximation aux moindres carrés tire son nom de la minimisation du carré de l'erreur entre les valeurs connues et le résultat de la fonction d'interpolation $\tilde{u}(x)$. Plus formellement, nous cherchons à minimiser la valeur de S définie par :

$$S = \sum_{i=0}^{n} ||u_i - \tilde{u}(x_i)||^2 \tag{2.4}$$

Dans la suite, le polynôme d'interpolation est représenté sous forme vectorielle pour faciliter l'extension à des fonctions non linéaires ou à plusieurs variables :

$$\tilde{u}(x) = b(x)^T.c \tag{2.5}$$

avec $b^T(x) = [b_1(x), b_2(x), ..., b_i(x) = [1, x, ..., x^i]$, fonctions de base de l'interpolation et $c = [\alpha_1, \alpha_2, ...,$ coefficients du polynôme. Le minimum de l'équation 2.4 est obtenu lorsque les dérivées partielles (suivant les inconnues c de la fonction) s'annulent ; ce qui revient à résoudre le système d'équations linéaires $\nabla S = 0$ suivant :

$$\begin{aligned}
\frac{\delta S}{\delta \alpha_1} = 0 \Leftrightarrow \sum_i 2b_1(x_i)[b(x_i)^T.c - u_i] = 0 \\
\frac{\delta S}{\delta \alpha_2} = 0 \Leftrightarrow \sum_i 2b_2(x_i)[b(x_i)^T.c - u_i] = 0 \\
\dots \dots \dots \\
\frac{\delta S}{\delta \alpha_i} = 0 \Leftrightarrow \sum_i 2b_i(x_i)[b(x_i)^T.c - u_i] = 0.
\end{aligned} \tag{2.6}$$

et sous forme vectorielle :

$$\begin{aligned}
p = \sum_i b(x_i).u_i \\
A = \sum_i b(x_i).b(x_i)^T.
\end{aligned} \tag{2.7}$$

Le système pour trouver les coefficients est alors :

$$c = A^{-1}.p \tag{2.8}$$

Le choix de la base $b^T(x) = [b_1(x), b_2(x), ..., b_i(x)]$ est important. La version la plus simple de l'approximation aux moindres carrés, appellée régression linéaire, s'appuie sur une base $b^T(x) = [1, x]$. La fonction $\tilde{u}(x)$ obtenue est alors une droite. L'utilisation de polynômes d'ordre plus élevé est possible, pour cela il suffit d'augmenter la base de l'interpolation. L'approximation aux moindres carrés peut aussi s'étendre aux fonctions à plusieurs variables $X = [x, y]$ en 2d ou $X = [x, y, z]$ en 3d. Pour cela, on utilise une base de la forme $b^t(X) = [1, x, y]$ pour une base linéaire ou encore $b^t(X) = [1, x, y, x^2, y^2, xy]$ pour une base quadratique en dimension deux.

On observe que la taille de la matrice A dépend de la taille de base $b(x_i)$. Plus la base est importante, plus l'inversion de la matrice A est coûteuse. En outre, un nombre trop petit d'échantillons rend la matrice A non inversible.

Approximation aux moindres carrés mobiles

L'approximation aux moindres carrés mobiles (Moving Least Square, MLS) est expliquée dans [Nea04]. Elle suggère de pondérer l'influence des échantillons à l'aide d'une fonction de poids $w_i(x)$. La fonction d'interpolation (eq. 2.5) est remplacée par : $\tilde{u}^{mls}(x) = b^t(x).c(x)$. La minimisation de l'erreur pour trouver les coefficients devient :

$$A(x).a(x) = \sum_{i=0}^{n} w_i(x).b(x_i).u_i \tag{2.9}$$

avec :

$$A(x) = \sum_{i=0}^{n} w_i(x).b(x_i).b(x_i)^T \tag{2.10}$$

On observe que la fonction n'est plus définie globalement à partir des échantillons, mais localement pour un point x donné. Les termes de cette fonction sont réorganisés pour séparer ceux qui dépendent du paramètre x de ceux qui dépendent des échantillons :

$$\tilde{u}^{mls} = \sum_{i=0}^{n} \phi(x).u_i \tag{2.11}$$

avec :

$$\begin{aligned} \phi(x) &= c^t(x)b(x_i)w_i(x) \\ c(x) &= A^{-1}(x)b(x) \end{aligned} \tag{2.12}$$

En pratique, on emploie des fonctions de poids gausiennes à support compact. Il s'agit souvent de sphères centrées sur l'échantillon (du même type que celles utilisées dans les méthodes implicites à base de squelette). Le rayon des sphères définit la largeur du support et doit être paramétré avec attention car :

- des largeurs « trop petites » rendent la fonction $A(x)$ non inversible.
- des rayons « trop grands » tendent à lisser fortement les détails de la courbe ; celle-ci s'éloigne alors d'une courbe d'interpolation.

On trouve, en annexes (A), quelques expérimentations portant sur l'approximation et l'interpolation aux moindres carrés mobiles. Dans la suite de la thèse, nous nommerons $\phi(x)$ *fonction de forme*, $b(x)$ *base de l'interpolation* et $w_i(x)$ *fonction de poids.*

Spline, maillage et interpolation aux moindres carrés mobiles

On sait que l'approximation aux moindres carrés mobiles est à la base des principales méthodes sans maillage [Lev03]. Cependant, il existe d'autres liens entre les représentations géométriques et l'interpolation. En effet, l'utilisation de la base constante $b = [1]$ aboutit à l'équation [9] :

$$\tilde{u}^{sheppard}(x) = \sum_{i=0}^{n} \frac{w_i(x)}{\sum_{i=0}^{n} w_i(x)}.u_i \tag{2.13}$$

Cette équation ressemble à celle des surfaces implicites à squelette (au terme de normalisation près). Reprenons maintenant la formulation d'une spline donnée dans [Len04, page 32] : « *Une courbe spline de l'espace* $\mathbb{R}^3$ *est une courbe paramétrique définie à l'aide de* $n+1$ *points ponctuels* q_i *de l'espace* $\mathbb{R}^3$ *(appelés points de contrôle de la courbe) pondérés par des fonctions d'influence* b_i *:* $[0,1] \rightarrow \mathbb{R}$ »

$$P(s) = \sum_{i=0}^{n} b_i(s) q_i \tag{2.14}$$

A nouveau, cette formulation est très proche de celle de l'équation 2.11, des échantillons connus $q_i = u_i$ sont mélangés à l'aide de la somme par des fonctions de pondération (ou de forme dans le cas des moindres carrés). Cette proximité n'est guère surprenante puisque l'objectif des splines était de résoudre un problème d'interpolation. Cette analogie entre les formulations Spline et la méthode aux moindres carrés mobiles est intéressante et nous amène à étudier plus précisément le paramètre X utilisé dans les formules précédentes. De manière informelle, disons que pour les méthodes sans maillage, le paramètre X est défini sur $\mathbb{R}^3$, alors que pour celles avec maillage, X est défini à partir des coordonnées barycentriques et des relations d'adjacence du maillage ; de cette distinction naîtront les termes de topologies implicite et explicite. Ces termes décrivent non pas la topologie de la géométrie formée mais la topologie de la paramétrisation (les paramètres) de la fonction d'interpolation qui, elle, produit la géométrie. Il peut arriver que la topologie de la paramétrisation sois identique à celle de la géométrie (comme pour les maillages triangulaires), mais ce n'est pas obligatoire comme le montre la méthode des XFEM (découpe dans un maillage tétraèdrique par utilisation de fonction de formes discontinues).

- *topologie implicite :* la topologie implicite (en relation avec les représentations géométriques sans maillage présentées précédemment) est générée par l'usage de la distance euclidienne pour mesurer la proximité entre deux points. En pratique, la topologie implicite est associée à l'espace euclidien doté de la distance euclidienne. Un exemple d'utilisation de topologie implicite est l'approximation aux moindres carrés mobiles qui repose sur des fonctions de poids impliquant le calcul de la distance euclidienne.

[9] Cette équation est parfois appelée interpolation de Sheppard.

– *topologie explicite :* la topologie explicite est construite comme un assemblage d'éléments simples (une discrétisation en élément) ou à l'aide d'un graphe. En outre, ces éléments sont dotés d'une paramétrisation locale sous la forme de coordonnées barycentriques. On se sert de ces coordonnées pour exprimer une « distance locale » relative à l'élément. Des relations de voisinage qui lient les éléments les uns aux autres et permettent de construire une paramétrisation globale sur l'ensemble du maillage. Parmi les topologies explicites, on trouve :
  – les maillages simpliciaux : chaîne de segments, maillage triangulaire, maillage tétraédrique ;
  – les maillages générés par le produit tensoriel : ils comprennent les chaînes de segments, les grilles 2d, les grilles 3d. De tels maillages sont à la base des extensions 2d/3d des Splines ;
  – les pavages convexes : l'objet est formé d'un assemblage de cellules convexes ; par exemple, le diagramme de Voronoï d'un ensemble de points produit un tel pavage ;
  – les grilles de voxel.

Une vue abstraite d'une modélisation géométrique

Nous venons de mettre en parallèle les représentations géométriques avec l'approximation au moindres carrés mobiles. Le résultat est une vue abstraite, unifiée, des différentes représentations (splines, maillages, surfaces implicites à squelette, grilles de voxel). Cette nouvelle vue est formée de trois couches successives (fig. 2.8) :
– la première couche est la topologie paramétrique qui décrit les relations de voisinage entre les échantillons ;
– la deuxième, appelée *fonction de forme*, s'appuie sur la topologie pour interpoler une série d'échantillons. La fonction de forme est elle-même décomposée en fonctions de poids et de base ;
– la dernière couche, résultat de l'interpolation, est la géométrie finale de l'objet. On peut remarquer que la géométrie finale dispose aussi d'une topologie et que celle-ci n'est pas forcément corrélée à la topologie paramétrique.

2.3 Découpage de la géométrie

On a présenté dans (1.5) les différentes approches de découpe des objets virtuels. La simulation des objets est indissociable de la géométrie des objets et donc de la manière dont celle-ci est représentée. La vue abstraite des représentations géométriques peut donc nous aider à évaluer, classer ou développer des méthodes de découpe. Pour modéliser de la matière continue, il faut que les conditions suivantes soient respectées :
– une fonction de base de l'interpolation continue ;
– une fonction de poids non discontinue. En général, on emploie une fonction de poids à l'allure gausienne, car elle produit un mélange *lisse* des primitives ;
– une topologie paramétrique connexe.

En changeant l'un ou l'autre de ces éléments, on introduit des discontinuités dans la géométrie produite (fig. 2.9). Lorsque la discontinuité est à l'ordre C^{-1}, il y a découpe.

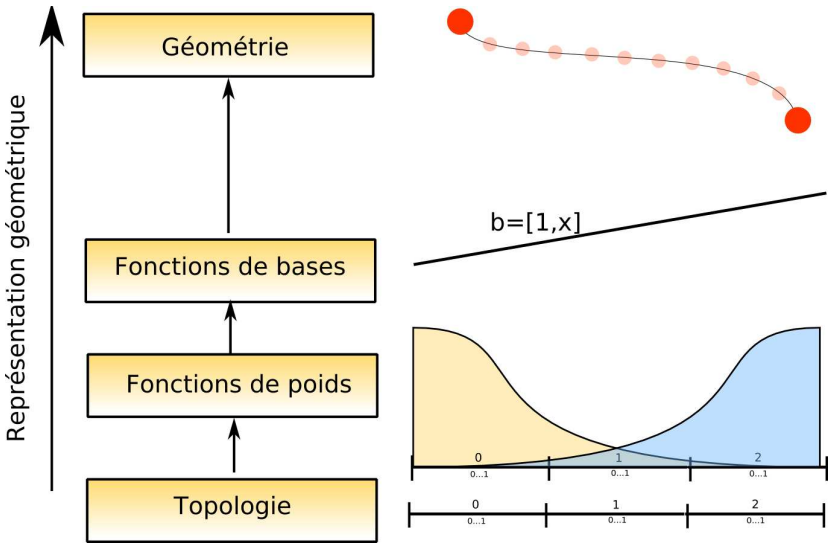

Fig. 2.8 : *Représentation de la géométrie basée sur un modèle à trois couches : la topologie (le maillage), la fonction de forme qui se décompose en fonctions de poids et de base, et enfin la géométrie finale produite.*

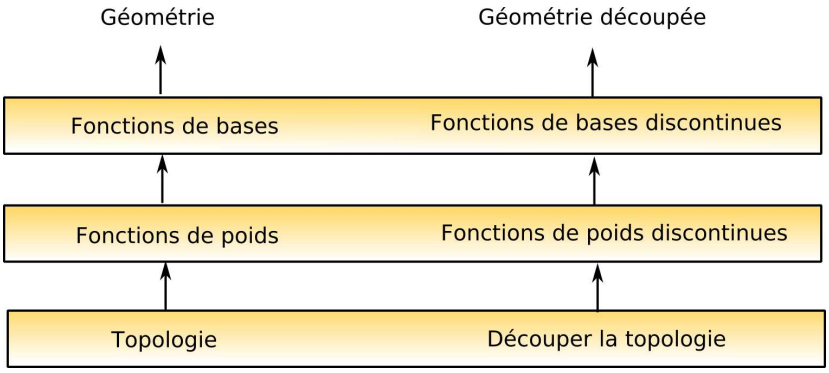

Fig. 2.9 : *On peut insérer des discontinuités dans la géométrie des objets en modifiant les différents niveaux de la représentation géométrique.*

Découpe par contrôle de la fonction de forme

En enrichissant la fonction de forme, on introduit des discontinuités dans le champ géométrique interpolé. On suppose que l'objet est obtenu au travers d'un processus d'interpolation analogue à l'équation aux moindres carrés mobiles (2.11). Pour contrôler la fonction de forme, deux approches sont possibles suivant l'enrichissement de la fonction de base ou de la fonction de poids :

– *l'enrichissement de la fonction de base :* l'insertion de discontinuité par contrôle de la base est une méthode issue du domaine de la simulation mécanique de la fracture. Prenons l'exemple d'un objet modélisé dans $\mathbb{R}^2$ dont le processus d'interpolation est construit à partir d'une fonction de poids radiale. La base de l'approximation aux moindres carrés mobiles $b^t(X) = [1, x, y]$ est remplacée par la base suivante [Duf04] :

$$b^T(x) = [1, x, y, \sqrt{r}] \tag{2.15}$$

L'enrichissement de la fonction de base n'est pas propre aux fonctions à base radiale. Par exemple, dans la méthode des Éléments Finis Étendus (XFEM), où les objets sont modélisés par assemblage d'éléments simples (tétraèdres, hexaèdres), l'interpolation linéaire à l'intérieur d'un élément est remplacée par une forme discontinue. L'application d'une telle méthode est présentée dans [VVW04].

– *l'enrichissement de la fonction de poids :* un résultat assez proche est obtenu en manipulant les fonctions de poids $w_i(x)$ de manière à y insérer des discontinuités. Ainsi, dans sa thèse, Marc Duflot [Duf04] propose :

$$w_c(x) = \alpha\sqrt{r}.cos(\frac{\omega}{2}).f_4(s) \tag{2.16}$$

Le contrôle de la continuité géométrique au travers de la fonction de poids ou de la fonction de base se traduit par des résultats assez semblables. Le contrôle de la fonction de forme autorise la représention d'une géométrie complexe avec relativement peu d'échantillons. On peut ainsi représenter des découpes sans augmenter le nombre d'échantillons ; ce qui, dans le contexte de la simulation, est intéressant puisque le nombre d'échantillons est corrélé au nombre de degrés de liberté de l'objet et donc au temps de simulation.

La découpe par manipulation de la topologie paramétrique

La manipulation directe de la topologie sur laquelle est construite l'interpolation offre un moyen efficace de découper les objets. C'est aussi cette méthode qui est privilégiée dans le domaine de l'informatique graphique, en raison de l'usage quasi systématique des représentations géométriques basées sur des maillages. Suivant que la topologie de l'interpolation est représentée implicitement ou explicitement, on observe différentes approches :

– *le cas des topologies explicites (maillage) :* lorsque la topologie est définie explicitement à l'aide d'un maillage, la méthode la plus simple consiste à modifier directement ce maillage. C'est probablement la méthode de découpe la plus courante en informatique graphique. Lorsque le maillage sert de base à un processus d'interpolation, sa modification permet aussi de contrôler la forme de la géométrie produite : dans [LGMC05], des Splines dynamiques sont découpées par insertion de noeuds dans la paramétrisation spline. On peut aussi citer les travaux de [SE04] qui proposent un système de FFD discontinues en 2d et 3d.

– *le cas des topologies implicites :* si la topologie paramétrique est implicite, engendrée par une fonction de distance, l'approche privilégiée consiste à modifier artificiellement ladite fonction de distance. Pour cela, on *déforme* la mesure de distance de manière à éloigner artificiellement des poids initialement proches. Différents critères de *déformation* sont proposés, comme la diffraction et la transparence [OFTB96] ou bien encore la visibilité [BGL94]. On trouve aussi dans la littérature des approches, hybrides, plus difficilement classables, s'appuyant sur des surfaces implicites à squelette (topologie implicite) mais y ajoutant un graphe de mélange [Des97][Tri01] permettant de contrôler simplement (explicitement) la topologie produite (fig. 2.10). Nous considérons que l'usage de ce graphe revient à utiliser une déformation de la mesure de distance sous une forme binaire : tout ou rien.

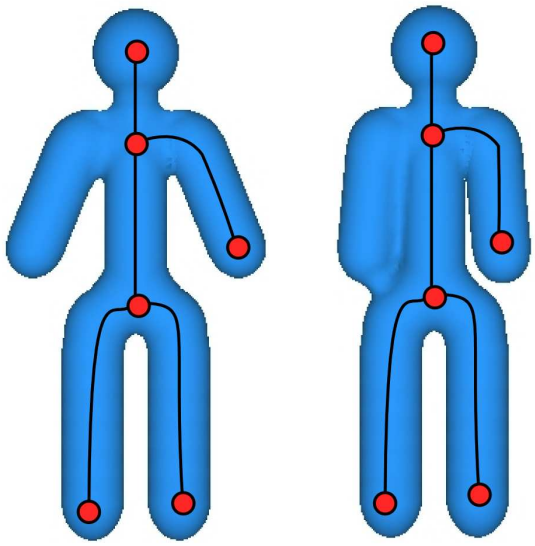

Fig. 2.10 : *Un objet modélisé à l'aide de métaballs. Pour contrôler la topologie finale de l'objet, on utilise un graphe de mélange qui définit explicitement quelles primitives seront fusionnées (Extrait de [Tri01]).*

Conclusion

Dans ce chapitre, nous avons proposé une classification des structures logicielles d'objets virtuels simulés. Cette classification combine des idées issues du génie logiciel telles la généricité et la modularité avec les contraintes du développement d'objets pour la simulation. La classification proposée a aussi l'avantage de rendre explicites les choix d'implémentation pris dans les différents simulateurs [sof][MDH+02] développés dans l'équipe. On observe que la tendance actuelle est de privilégier les approches découpées en aspect avec monoreprésentation ou multireprésentation, les objets multimodèle étant encore assez peu développés.

La seconde contribution de ce chapitre est une réflexion sur le rôle des représentations géométriques pour les environnements virtuels. Cette réflexion nous invite à mettre davantage en valeur les rôles joués par la topologie et la géométrie dans la simulation d'objets virtuels. La conclusion de cette réflexion est l'utilisation d'une nouvelle structuration des objets virtuels décrite par le schéma (2.11). Dans cette nouvelle structuration, on emploie une représentation centrale de la géométrie qui est convertie en représentations géométriques spécifiques (à la collision, à la mécanique, à l'affichage). On peut noter que le modèle proposé dans le chapitre (3) ne tient pas compte de cette réflexion : il n'y a pas de représentation topologique centrale, ce qui conduit, comme nous le verrons, à des difficultés pratiques lors de la découpe. A contrario, dans le chapitre (5), le modèle proposé intègre cette séparation conceptuelle des structurations, ce qui nous permet de proposer un système de découpe flexible et efficace.

L'importance du contrôle topologique n'est apparue que tardivement au cours de la thèse et il serait sans doute intéressant de poursuivre le raisonnement commencé dans ce chapitre en étudiant des représentations géométriques plus complexes comme les G-cartes [Dis]. En outre, il nous semble important d'avoir une meilleur représentation de cette topologie si on

veut pouvoir manipuler des objets déformables qui peuvent se briser ou se découper. Les travaux en géométrie différentielle discrète [DKT] sont un prolongement intéressant en lien avec l'interpolation de la géométrie des objets et les calculs numériques.

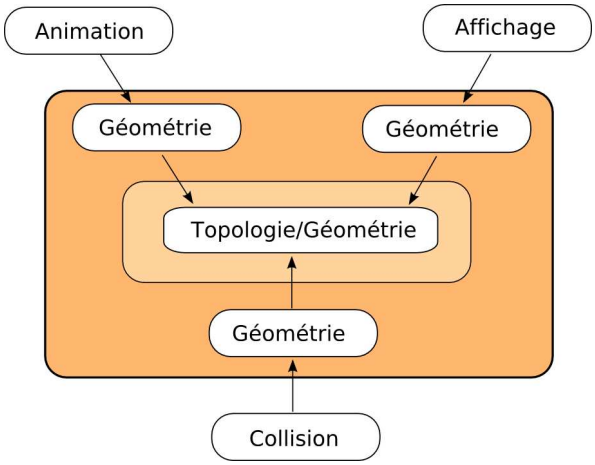

Fig. 2.11 : *Nouvelle proposition de structure pour les objets virtuels. Il est important que la découpe soit gérée au niveau de la représentation géométrique centrale.*

Chapitre 3

Modèle mécanique adaptatif

Sommaire

Introduction

Dans ce chapitre, nous présentons une structure d'objet virtuel illustrant bien le concept et l'intérêt de la multireprésentation dans le cadre de la simulation. Ce travail, réalisé en collaboration avec Jérémie Dequidt, présente le découplage d'un modèle d'objet déformable en ses différents aspects et représentations. Ce travail a été publié dans [DMG05], dans une forme un peu différente ; j'ai en effet retiré les travaux spécifiquement effectués par Jérémie Dequidt de la présente thèse.Découpler les représentations utilisées pour la mécanique, l'affichage et la collision offre une grande souplesse de conception et favorise le choix de la « meilleure » représentation pour une tâche donnée. Ainsi, on peut animer des objets visuellement très complexes, comportant beaucoup de détails, tout en gardant une représentation mécanique simple, adaptée à une simulation en temps réel. Le découplage favorise aussi le développement de modèles multirésolution permettant l'adaptation automatique du temps de calcul. Dans l'approche que nous présentons, les objets virtuels sont animés à l'aide de la méthode des Éléments Finis ; le maillage utilisé étant composé d'éléments hexahédriques découpés en octree. L'affichage des objets est effectué à l'aide de surfaces triangulées, tandis que la détection de collision et les calculs de réponse sont réalisés par des sphères. La détection de collision s'appuie, elle aussi, sur une représentation multirésolution : l'arbre de sphères (SphereTree). L'intérêt de la multirésolution est de permettre l'adaptation, en cours de simulation, du niveau de détail de chacune des composantes du simulateur (voir la thèse : [Deb00]). Le choix de la résolution est effectué par un Oracle décisionnel à partir des temps de calcul des pas de temps précédents, de manière à respecter une contrainte de temps de calcul.

Le chapitre s'organise de la manière suivante : dans un premier temps, nous faisons une présentation générale du moteur de simulation et de l'approche multireprésentation (sec. 3.1) ; nous décrivons, dans le second temps, les détails des composantes mécaniques (sec. 3.2) et de collision (sec. 3.3). La conclusion s'étendra par une réflexion sur l'ajout de la découpe à un tel modèle. Cette discussion est liée au chapitre 5 contenant la proposition complète d'un modèle d'objet découpable.

3.1 Structure d'un objet multireprésentation

La structure d'objet que nous présentons s'appuie sur un découpage en aspect des différentes composantes logicielles (mécanique, collision, affichage). Chacune de ces

composantes étant associée à une représentation géométrique particulière. Les différentes représentations sont reliées les unes aux autres, de manière à pouvoir transférer les quantités (les déformations et les forces) d'une représentation à l'autre. En pratique, ce transfert est modélisé par une transformation qui passe de l'espace non-déformé – la position de repos de l'objet – à l'espace déformé. Dans cette approche, la transformation est pilotée par la représentation mécanique. Cette transformation doit être inversible afin de transférer les forces de collision, exprimées sur la représentation de collision (potentiellement déformée), vers la représentation mécanique. La structure finale adoptée est illustrée sur la figure (3.1).

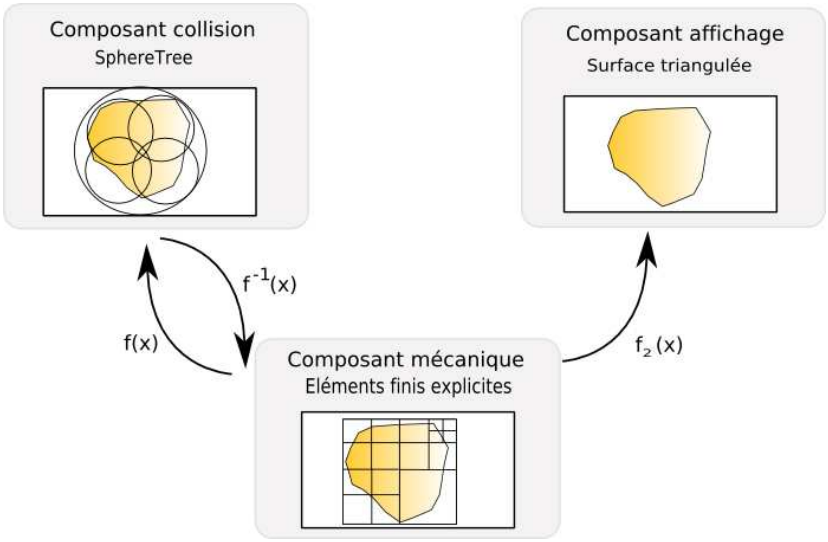

Fig. 3.1 : *Organisation d'un objet dans le cadre de la multireprésentation. On distingue les trois représentations et les transformations qui les lient.*

La transformation choisie se base sur une technique d'interpolation trilinéaire qui s'apparente aux déformations de forme libre (FFD). Les points de contrôle de la déformation sont pilotés par la déformation mécanique. Cela permet de construire un champ d'interpolation global, noté $f(X)$, qui est obtenu par un assemblage par morceaux, à la manière de l'interpolation utilisée par la méthode Éléments Finis. Les déformations de l'affichage et de la collision sont calculées sur l'ensemble de la cellule par interpolation linéaire des vecteurs de déplacement. La figure (3.1) présente la numération des sommets de la cellule. A partir de ces sommets, on cherche à interpoler des valeurs sur l'ensemble de la cellule ; soit $X_0 = [x, y, z]$, la position au repos d'un point associé à une cellule. Les coordonnées du point sont converties en coordonnées relatives, notées s, t, w, respectivement au repère formé par trois axes de la cellule. A partir de ce triplet, la position du point après déformation est calculée comme suit :

Les valeurs sont d'abord interpolées suivant le paramètre w :

$$i_1 = w.u_0 + (1 - w).u_1 \tag{3.1}$$
$$i_2 = w.u_2 + (1 - w.)u_3 \tag{3.2}$$
$$j_1 = w.u_4 + (1 - w).u_5 \tag{3.3}$$
$$j_2 = w.u_6 + (1 - w).u_7 \tag{3.4}$$

Puis, suivant le paramètre t :

$$k_1 = t.i_1 + (1 - t).i_2 \tag{3.5}$$
$$k_2 = t.j_1 + (1 - t).j_2 \tag{3.6}$$

Enfin, elles sont interpolées suivant le paramètre s :

$$X = s.k_1 + (1 - s).k_2 \tag{3.7}$$

Nous avons choisi l'interpolation linéaire car simple à inverser (par opposition à une interpolation du type MLS) et permettant de transférer les forces de pénalités, calculées sur la représentation de collision, vers la mécanique (point présenté plus en détail dans la section 3.3). En plus d'être facilement inversible, l'interpolation trilinéaire est aussi peu coûteuse.

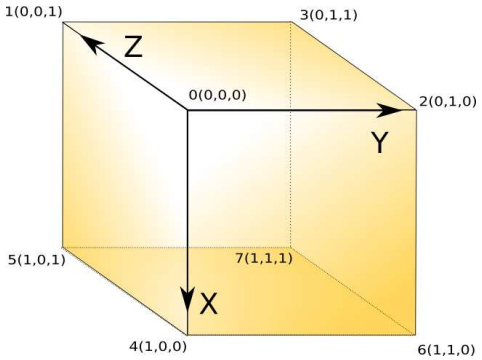

Fig. 3.2 : *Numérotation utilisée pour l'interpolation trilinéaire.*

3.2 Le composant mécanique : Éléments Finis explicites

Le composant mécanique s'appuie sur la théorie de l'élasticité linéaire : le tenseur de déformation de Cauchy est utilisé et le matériau respecte la loi de Hooke (les contraintes du matériau sont proportionnelles aux efforts qu'il subit). La résolution numérique des équations de l'élasticité linéaire se base sur la méthode des Éléments Finis explicites ; les éléments étant de formes cubiques ; le cube se prêtant bien au développement d'un processus de multirésolution (par subdivision régulière : en octree). En outre, sur une grille, il existe des méthodes efficaces de codage des cellules, tels les codes de Morton [Mor66], avec lesquels on peut retrouver rapidement (temps constant) les cubes voisins dans une hiérarchie (voir [Deq05], pour plus de détails).

3.2.1 Génération du maillage mécanique

Avant la simulation, on convertit les objets surfaciques en un maillage volumique composé de cellules cubiques. Pour cela, nous employons l'algorithme de calcul de la carte de distance vu dans [HTJ+99]. En sortie de cet algorithme, nous disposons d'une carte de distance, discrète et signée. Chaque élément de distance négative correspondant à un voxel. Pour les tests, nous utilisons une résolution de 64*64*64 qui suffit à produire des maillages suffisamment fins pour la simulation mécanique de l'objet (fig. 3.3).

A partir de la "voxellisation" de l'objet ainsi obtenue, on génère la hiérarchie octree finale utilisée pour la simulation multirésolution. Cette hiérarchie est produite à partir de la boîte englobante de l'objet. Cette boîte est calculée et correspond à la racine de l'octree. Elle est ensuite subdivisée récursivement en huit sous-boîtes, la récursion se poursuivant tant qu'il reste des voxels inclus dans la boîte englobante et que la taille de ces voxels est supérieure à la taille des boîtes.

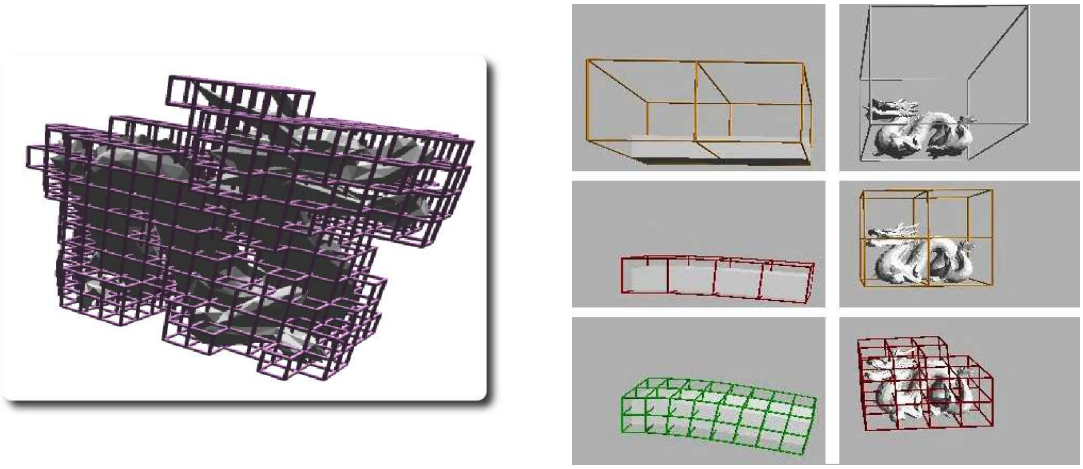

Fig. 3.3 : *Objets discrétisés sous la forme de cellules cubiques. Ces cellules servent ensuite à construire la hiérarchie mécanique.*

3.2.2 Éléments Finis explicites

On choisit d'utiliser les Éléments Finis explicites pour une question d'efficacité. Le tenseur de déformation de Cauchy (linéaire) est employé, ainsi que la loi de comportement de Hooke (linéaire) : ce qui simplifie considérablement les calculs. En effet, avec des tenseurs linéaires, la matrice de rigidité qui lie les forces aux déplacements ne dépend pas de la configuration courante de l'objet ; elle peut donc être précalculée. Les forces internes reçues par les particules mécaniques sont notées $F = [f_0, f_1, ..., f_n]^T$. Ces forces, traduisant les contraintes matérielles, s'expriment alors linéairement en fonction des déplacements des mêmes particules $U = [u_0, u_1, ..., u_n]^T$ et de la matrice de rigidité notée K ; matrice de rigidité calculée à partir de la formule donnée dans [MTG04] et ne dépendant que de la largeur de la cellule et des deux coefficients décrivant le comportement de la matière (Young et Poisson). En pratique,

les forces à l'instant $t+1$ sont calculées à partir des positions à l'instant courant. Pendant le *bilan des forces,* on effectue l'opération suivante sur l'ensemble des cellules :

$$F = K.U \tag{3.8}$$

Nous avons retenu une formulation des Éléments Finis dite sans assemblage (masses-tenseurs). Bien que l'utilisation d'une matrice de rigidité assemblée soit plus efficace (une partie des opérations y étant factorisées), la méthode des masses-tenseurs est plus souple [Cot98]. Cette souplesse rend possible les modifications du maillage – changement de résolution, découpe – de manière locale.

3.2.3 Extension aux grandes rotations

L'utilisation du tenseur linéaire limite le domaine de validité de la simulation mécanique. On parlera de "petits" déplacements et de "petites" déformations pour désigner le domaine de validité des tenseurs linéaires. Dans le contexte de l'informatique graphique, la contrainte de petites déformations est peu gênante[10]. Par contre, celle des petits déplacements limite considérablement le mouvement de l'objet qui ne peut alors effectuer que de grandes translations et de petites rotations (comme cela est illustré dans l'état de l'art). Pour contourner cette contrainte, nous choisirons une méthode corotationnelle dérivée de [NPF05] qui supprime la composante en rotation du champ de déplacement U, au moment du calcul du bilan de force. Cette rotation est ensuite restaurée une fois les forces internes calculées. La méthode employée est basée sur le calcul d'un repère rigide local aux cellules (fig. 3.4). A partir de ce repère, nous retirons les composantes de rotation pour le calcul des forces de manière à ne garder que les déformations de l'objet.

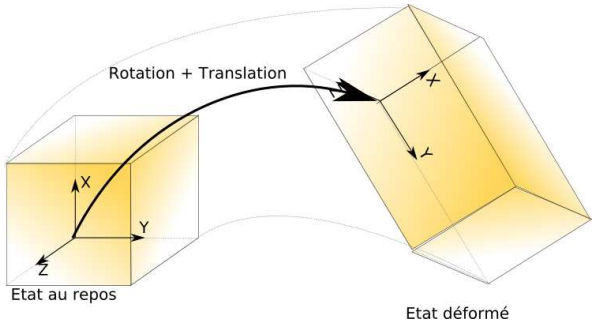

Fig. 3.4 : *Méthode corotationnelle, la composante rigide du mouvement est supprimée de manière à ne laisser que la composante en déformation.*

Reprenons la formulation de la force interne calculée sur une cellule. Soit f_e le vecteur de taille $8*3$ contenant les forces générées aux huit sommets de la cellule. Celle-ci est calculée à partir du déplacement des huit sommets u_e et de la matrice de rigidité élémentaire K_e :

[10] Faire l'hypothèse de petites déformations est assez courante en informatique graphique. Cependant, pour certains type de simulation (comme la simulation médicale), cela semble une simplification trop importante. En effet, les lois de comportement des matériaux biologiques sont en général mal approchées par des lois linéaires [Sch03][Pic01].

$f_e = K_e\ u_e$. Dans le cadre d'une formulation corotationnelle par élément, le déplacement rigide, $u_e = u_0, ..., u_7$, est décomposé en une rotation R_e et une translation T_e. L'expression de la force devient alors :

$$f_e = R_e\ K_e\ (R_e^{-1}\ - u_0) + T_e \tag{3.9}$$

avec :

$$T_e = u_0; \tag{3.10}$$
$$R_e = NN_0 \tag{3.11}$$

3.2.4 Simulation multirésolution

Le découplage des représentations facilite le développement de méthodes multirésolution. Les changements de résolution d'une représentation sont masqués pour les autres représentations. Cela permet de changer la résolution du maillage mécanique, indépendamment de celle du maillage affiché. L'approche ici développée s'appuie sur la structure d'octree de la représentation mécanique ; les différentes cellules sont simulées indépendamment avec la méthode des masses-tenseurs. Le fait que chaque cellule puisse être simulée indépendamment permet de subdiviser localement celle-ci en fonction du réalisme désiré et de l'état de déformation de l'objet (fig. 3.6). Nous avons envisagé plusieurs méthodes de subdivision :

- *critère statique de subdivision par distance au bord :* ce critère est évalué au démarrage de la simulation et associe aux bords de l'objet des cellules plus petites. Les bords de l'objet sont simulés précisément tandis que les comportements internes le sont plus grossièrement (fig.3.5). Cette approche s'inspire du calcul des cartes de distance adaptatives (ADF) de [FPPJ00]. On peut noter qu'un tel critère de subdivision a l'avantage de coupler implicitement la forme de l'objet avec le maillage utilisé pour sa simulation, cela limite les effets de couplage non désirés.
- *critère dynamique de subdivision par évaluation du Laplacien :* le Laplacien est défini comme la somme des dérivées secondes d'un champ continu. Dans sa thèse [Deb00], Gilles Debunne propose d'utiliser le Laplacien du champ de déplacement comme critère de subdivision. En effet, le Laplacien mesure le caractère non-linéaire du champ de déformation : un Laplacien élevé traduit un champ de déformation non-linéaire. Son calcul, pour un noeud I du maillage octree, est :

$$(\Delta u)^I = \frac{2}{\sum_{J \in voisin(I)} ||IJ||} \sum_{J \in voisin(I)} \frac{u(J) - u(I)}{||IJ||} \tag{3.12}$$

  où l'on inspecte les voisins J de I avec $||IJ||$ qui représentent la distance entre I et J et $u(x)$, la valeur du champ de déplacement en un point x. A chaque itération, on calcule la valeur absolue du Laplacien en chaque sommet. Les contributions des huit sommets sont cumulées pour caractériser la déformation générale de la cellule. Cette mesure est ensuite injectée dans l'Oracle pour évaluer la pertinence de la subdivision/simplification d'une cellule. L'Oracle s'appuie sur l'évaluation du temps de calcul nécessaire au pas de

temps précédant. En fonction d'un temps de calcul objectif, il déduit si le modèle doit être ou non-raffiné/simplifié. Le Laplacien est utilisé pour choisir sur quelle cellule se portera la subdivision/simplification.

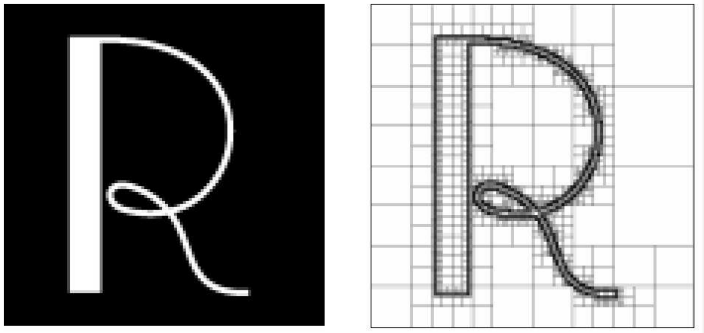

Fig. 3.5 : *Illustration d'un ADF, on utilise la carte de distance comme paramètre pour choisir la taille des cellules : plus on s'éloigne de la surface, plus les cellules sont grandes.*

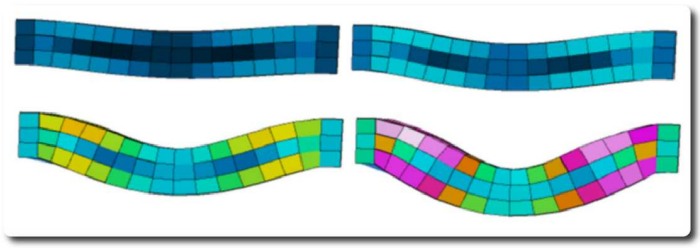

Fig. 3.6 : *Illustration du calcul du Laplacien pour une poutre à différents niveaux de déformation. Le Laplacien calculé y est illustré sous la forme d'une échelle thermique.*

En informatique graphique, on appelle "jonctions-T" les zones d'un maillage où un sommet est confondu avec une arête ou une face (fig. 3.7). Dans le cadre du modèle développé, on mélange des cellules de résolutions différentes, entraînant l'apparition de telles jonctions-T (voir les fig. 3.7 et 3.8). Ces jonctions-T posent problème et nous font sortir du cadre de la méthode Élément Finis qui les interdit, précisément. Nous identifions deux points de difficulté liés aux jonctions-T. La première difficulté est l'introduction de degrés de liberté supplémentaires d'un seul côté de la jonction. La cohérence volumique de l'objet n'est alors plus respectée et des *"trous"* peuvent apparaître. La deuxième difficulté est liée au calcul mécanique des forces internes. En effet, en introduisant une discontinuité dans le champ d'interpolation – qui est à la base de la méthode des Éléments Finis – les jonctions-T entraînent la production de forces différentes de part et d'autre de la jonction ; ces forcent ne respectent plus le principe d'action-réaction. Pour gérer les jonctions-T, nous avons adopté une approche pragmatique qui consiste à contraindre par projection le mouvement des degrés de liberté supplémentaires[11]. Les forces calculées sur la jonction-T sont pondérées et réparties

[11] D'autres méthodes auraient permis un résultat équivalent sans avoir recours à une méthode de projection qui peut amener des comportements « non-physiques ». Ces méthodes consistent à redéfinir un champ

sur les particules mécaniques voisines de la jonction. Le mouvement des différents sommets est toujours calculé par intégration numérique, mais celui d'une particule est suivi d'une étape de post-correction au niveau des jonctions-T. Celles-ci sont projetées sur la face/arête adjacente, afin de ne pas créer de trous dans le maillage volumique.

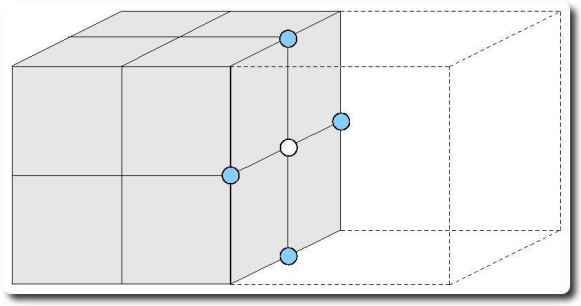

Fig. 3.7 : *Exemple de jonction-T entre deux cellules simulées à différents niveaux de résolution.*

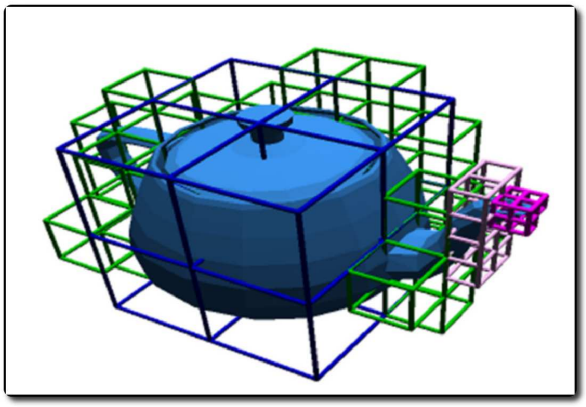

Fig. 3.8 : *On observe sur cette image un nombre de jonctions-T assez élevé.*

3.3 Gestion des collisions

La détection de collision est souvent le calcul le plus coûteux de la simulation. Suivant le simulateur et la méthode de détection de collision choisis, on estime que ce calcul peut occuper jusqu'à quatre-vingts pour cent du temps total de simulation. Nous avons présenté dans l'état de l'art 1.3), les différentes techniques d'accélération de la détection de collision. Ces techniques supposent l'utilisation de structures de données additionnelles dont la mise à jour,

d'interpolation continu en prenant en compte la particule ajoutée. C'est l'approche suivie dans [NFP06]. Une autre technique assez simple pour gérer de la mécanique multirésolution serait d'utiliser la mécanique sans maillage en organisant les rayons d'influence comme présenté dans l'annexe (A), de manière à avoir un champ d'interpolation correctement défini.

pendant la déformation, est coûteuse. Dans le présent travail, nous optons pour une technique à base d'arbre de sphère [Hub95] ; c'est un modèle simple à implémenter, multirésolution et qui illustre bien le découplage des représentations. Cet arbre de sphère est construit directement à partir de la hiérarchie de simulation mécanique. Chaque sphère est associée à une cellule mécanique qu'elle englobe. En pratique, la profondeur de la hiérarchie de sphère est plus grande que celle dédiée à la simulation mécanique (6-8 niveau contre 3-4). L'intérêt majeur des arbres de sphère est de permettre le développement de techniques de détection de collision et de réponse multirésolution. En effet, un parcours en largeur de l'arbre permet de raffiner la précision de la détection de collision (fig. 3.9) ; la profondeur d'exploration de l'arbre, pendant la phase de détection, est paramétrée par le temps de calcul disponible [DO00]. Une fois le dernier niveau de la hiérarchie atteint, ou le temps alloué à la détection écoulé, une réponse par pénalité est calculée à partir des sphères en intersection.

Fonction `getIntersection`(*SphereTree* S_1, *SphereTree* S_2)

Données : SphereTree : S_1, S_2
Résultat : ensemble des couples de sphères en intersection
début
  Fifo<tuple_sphere> : f ;
  f.enfiler(S_1, S_2) ;
  a := S_1 ; b := S_2 ;
  pour *intersection(a.sphere, b.sphere)* **et** *temps_restant* ≥ 0 **faire**
    pour chaque *as* $\in$ *a.fils* **faire**
      pour chaque *bs* $\in$ *b.fils* **faire**
        f.enfiler(as, bs) ;
    a,b := f.defiler() ;
fin

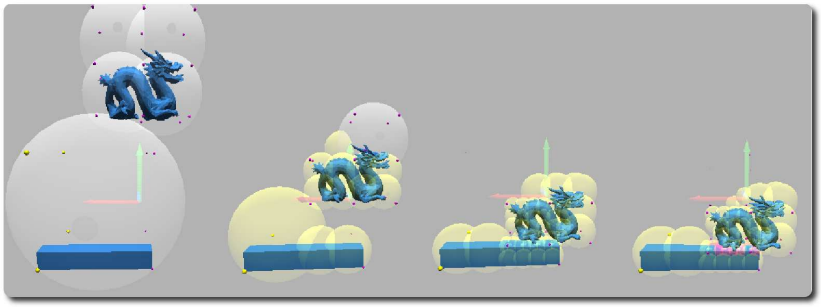

Fig. 3.9 : *Chaque niveau de la hiérarchie peut être choisi pour calculer une réponse à la collision. La forme de l'objet est de mieux en mieux approchée à mesure que l'on descend dans la hiérarchie.*

3.3.1 Algorithme de mise à jour de la représentation de collision

Nous nous sommes concentrés sur l'étape de mise à jour du modèle de collision d'objets déformables, car elle est essentielle pour les performances de la simulation. Deux approches

sont envisageables : de la racine vers les feuilles (top-down) [LAM01] ou des feuilles vers la racine (bottom-up) [vdB97]. Un algorithme top-down ayant l'avantage de pouvoir être interrompu si l'objet n'est pas en collision ; on parle ainsi de mise à jour à la demande [JP04]. L'algorithme présenté dans [JP04] utilise une telle approche qui, combiné à une mise à jour en un temps constant des sphères-racine, aboutit à un pipeline de collision extrêmement efficace. Malheureusement, cette efficacité est liée au fait que les auteurs utilisent un modèle mécanique particulier, qui exprime les déformations de l'objet par un jeu de coordonnées réduites obtenues par l'analyse modale. Comme nous utilisons la méthode des Éléments Finis,cette approche pour la mise jour complète de la hiérarchie ne nous convient pas. Cependant, nous avons développé un algorithme hybride qui s'en inspire.

Cet algorithme hybride répartit les sphères de collision en trois ensembles. Le premier ensemble, nommé `UPset`, contient les sphères associées à des cellules de niveaux supérieurs aux cellules effectivement simulées. Le deuxième, nommé `ACset`, est associé aux cellules effectivement simulées. Le troisième ensemble, `DOWNset`, correspond aux sphères associées aux cellules de niveaux inférieurs à celles qui sont simulées. L'algorithme de mise à jour employé dépendra de l'ensemble auquel appartient la sphère.

La mise à jour des sphères de `UPset` s'appuie sur une approche de type « feuilles vers racine » de complexité $O(n.log.n)$, avec n, le nombre de cellules effectivement simulées. Elle utilise l'algorithme de mise à jour de hiérarchie de volume englobant présentée par Van Den Bergen dans [vdB97]. Les ensembles `ACset` et `DOWNset` sont mis à jour par une évaluation directe des déformations à la manière de [JP04].

3.3.2 Calcul de réponse à la collision

Pour séparer des objets en collision, on utilise des forces de rappel calculées à l'aide d'une méthode de pénalité. En pratique, la réponse modélise la répulsion des objets en intersection à l'aide d'un ressort de raideur k donné. La pénalité est calculée à partir des sphères en intersection (fig. 3.3.2) suivant la formule d'un ressort linéaire. Le point d'application des forces ainsi calculées est le centre des sphères de collision. Le report des forces au centre des sphères est lié au fait que seul ce point central possède un jeu de coordonnées relatives (u, v, w) qui permet le transfert des forces vers la représentation géométrique simulée (les noeuds de la grille de voxels). Les forces sont reportées sur les nœuds du maillage mécanique : à partir des coordonnées relatives du centre de la sphère, on répartit la force sur les particules simulées en employant les coordonnées relatives comme pondération.

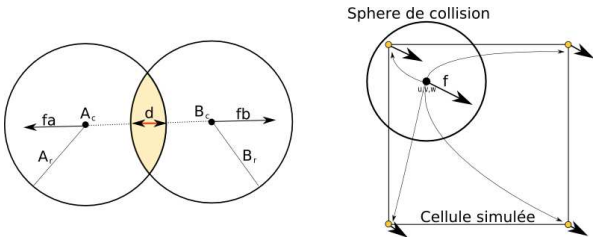

Fig. 3.10 : *A gauche, calcul de la pénalité à partir des couples de sphères en intersection. A droite, répartition d'une force appliquée au centre d'une sphère de collision sur les noeuds mécaniques.*

On peut noter que la paramétrisation du ressort de collision est délicate. Un ressort de trop forte raideur tend à repousser trop violemment les objets en intersection ; un ressort de trop faible raideur ne sépare pas complètement les objets [Mes03]. Le fait d'utiliser une approche multirésolution nous conduit à utiliser des coefficients de raideurs multiples en fonction des différents niveaux de résolution. La valeur de ces coefficients dépend à la fois des propriétés mécaniques de l'objet et de la résolution utilisée pour la simulation. Chaque niveau de la hiérarchie possède donc un paramètre de raideur. Paramètre obtenu de manière empirique par comparaison visuelle avec une simulation "test". En plus du paramétrage difficile de la raideur à la collision , nous avons remarqué que la définition de force de réponse à partir de sphères peut être problématique [MAC03]. Dans les calculs de réponse à base de distance, on sait que la réponse est mal calculée lorsque les objets sont en forte intersection, lorsque la pénétration dépasse le milieu de l'objet. En raison du caractère local de la réponse calculée à l'aide des sphères, le milieu de l'objet est alors équivalent au centre des sphères. Plus celles-ci sont petites, plus la méthode risque de produire des réponses mal orientées. La figure 3.11 illustre ce qui se passe lorsqu'on veut ajouter des sphères afin d'augmenter la finesse de la collision. Seule une petite zone proche de la surface de l'objet est utilisable pour calculer une direction pour la réponse à la collision. A mesure qu'on augmente la finesse de la détection de collision (utilisation de sphères plus petites), la taille de la zone où la réponse est bien orientée diminue. On peut éviter cela en calculant la pénalité de manière globale et non plus par couple de sphères en intersection. En pratique, cela revient à approcher la carte de distance à l'aide de primitives à support compact. Or, c'est précisément ce que permet de faire la méthode d'interpolation MLS. Le principal désavantage en est le coût nettement supérieur des calculs de réponse. Ces raisons nous conduirons, dans la suite de nos travaux, à abandonner les approches à base de sphères.

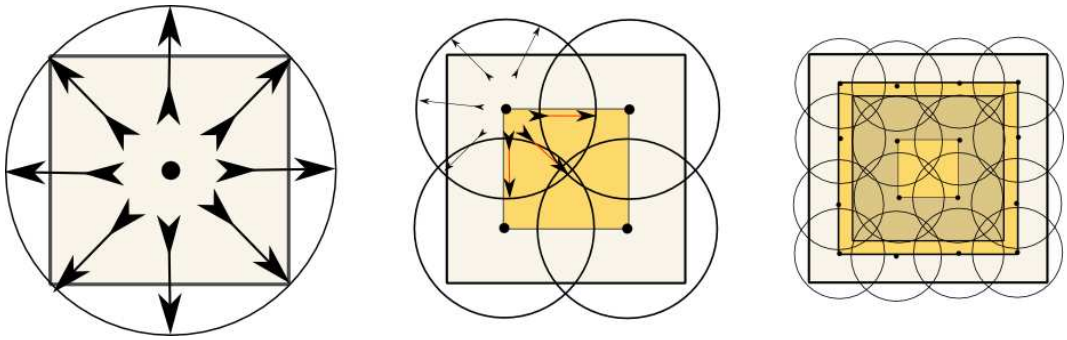

Fig. 3.11 : *Un cube est approché par des sphères de collision à différents niveaux de détail. On suppose qu'une particule entre en collision avec cet objet. On a coloré en jaune les zones de l'objet où la réponse n'est plus corrélée à la distance de pénétration et où la réponse calculée ne va pas empêcher les objets de s'interpénétrer.*

3.3.3 Mesure de performance

Pour évaluer les performances de la méthode de détection de collision, des tests sont réalisés dans des scènes pré-définies. Ces scènes possèdent différentes densitées d'objets et on recense les collisions (tab. 3.12). Cinq cents itérations sont enregistrées pour ces trois scènes,

en sélectionnant des cellules mécaniques issues de différents niveaux de la hiérarchie. Les temps mesurés sont rapportés sur la figure 3.13. Il apparaît que le temps total de collision est fortement lié au temps nécessaire pour mettre à jour l'ensemble `UPset`, ce qui confirme l'inefficacité d'une méthode de type « des feuilles vers la racine ». Le temps de mise à jour de l'ensemble `ACset` est plus faible, ce qui s'explique par le fait qu'il ne concerne que peu de nœuds (en comparaison à totalité de l'arbre). Le coût de l'ensemble `DOWNset` est, de manière générale, plus faible que celui du `UPset` (le nombre d'opérations à réaliser pour la mise à jour d'une sphère étant moindre).

	Scene 1	Scene 2	Scene 3
Nombre de collisions	0	11500	1224000
Nombre de tests d'intersection	10500	1598500	4310500

Fig. 3.12 : *Trois scènes de tests dont la densité de collision est différente.*

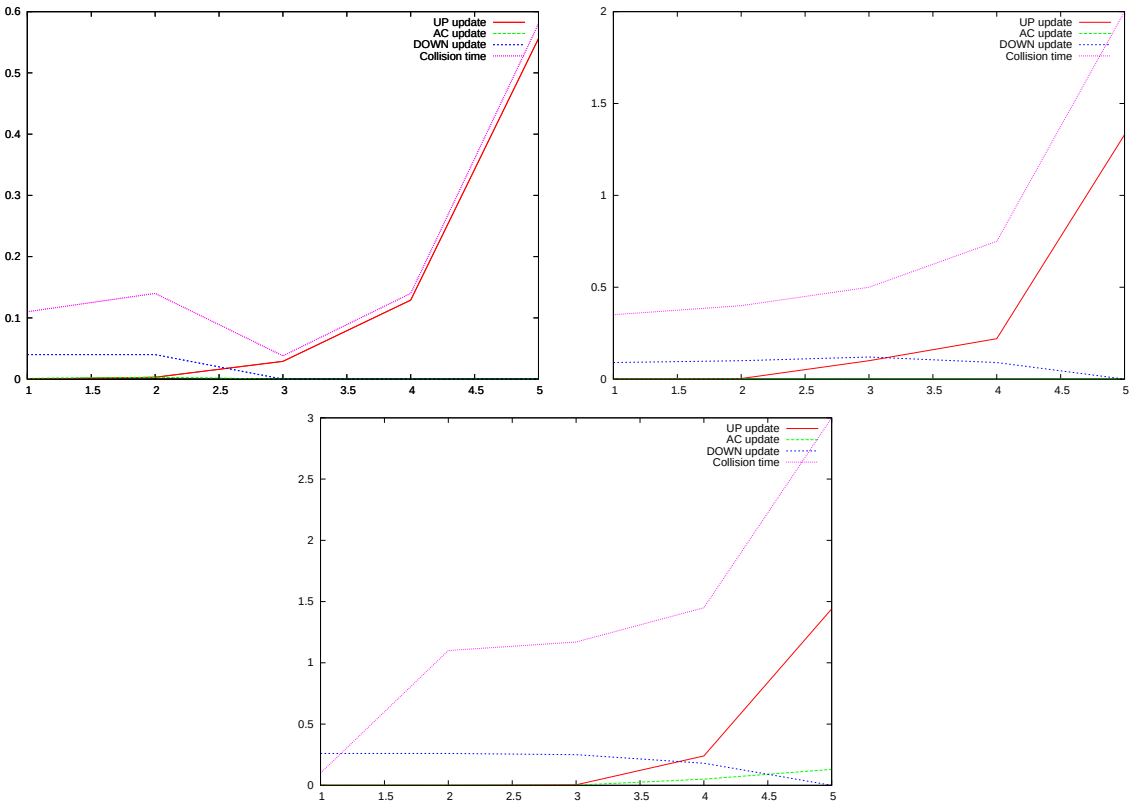

Fig. 3.13 : *Evaluation des performances du modèle de collision pour les trois scènes de tests (le temps est cumulé et l'échelle est en millisecondes).*

Pour avoir un aperçu des performances de la méthode, on peut se référer aux vidéos disponibles à l'adresse suivante – `http://www.lifl.fr/~marchal/these/video` – qui montrent l'utilisateur interagir avec différents objets virtuels. Sur ces vidéos, on affiche en jaune les sphères de la hiérarchie de collision qui sont mises à jour. Les sphères rouges

représentent les feuilles de la hiérarchie du modèle de collision. Les vidéos illustrent bien la séparation entre les différents aspects de l'objet virtuel simulé.

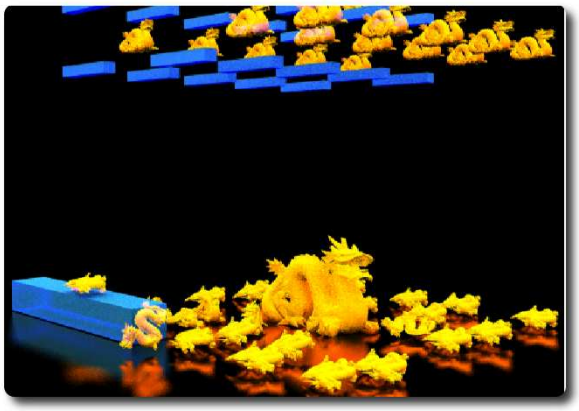

Fig. 3.14 : *Simulation d'un grand nombre d'objets déformables (12fps).*

3.4 Quelques réflexions sur la découpe

La combinaison de la découpe avec la multireprésentation pose problème. En effet, la représentation utilisée pour la simulation et la transformation entre les représentations sont uniquement basées sur des informations géométriques (la position des points/facettes de l'objet), ignorant totalement les aspects topologiques. Or, l'opération de découpe est une opération à la fois géométrique et topologique. Le problème principal que nous identifions est lié à la déconnexion entre la topologie du maillage affiché et la topologie du maillage simulé. C'est la force et la faiblesse de cette approche. C'est une force, car cela permet d'animer des objets visuellement complexes en des temps de calcul raisonnables. C'est aussi une force, puisque le découplage rend possible l'animation d'objets mal-formés comme des surfaces non-orientables ou des objets non-fermés (sans intérieur/extérieur). Mais c'est une faiblesse, car la découpe devient alors particulièrement complexe : doit-on découper toutes les représentations ? De plus, le fait d'utiliser des cellules cubiques ne favorise pas l'insertion de découpe, car l'approximation d'un chemin de découpe linéaire n'est pas possible (fig. 3.4). Le profil de découpe est approché à l'aide de découpes alignées sur les axes des cellules, ce qui est assez grossier. Enfin, pour découper un tel objet, il faut aussi faire remonter la découpe au niveau des cellules-parents de manière à traduire correctement la découpe quel que soit le niveau de la hiérarchie. Pour cela, il semble intéressant d'utiliser des raideurs non-uniformes sur les cellules mécaniques simulées à la manière de [NPF06]. Cela permettrait de tenir compte de la manière dont la matière peut remplir, de manière incomplète, le volume de la cellule.

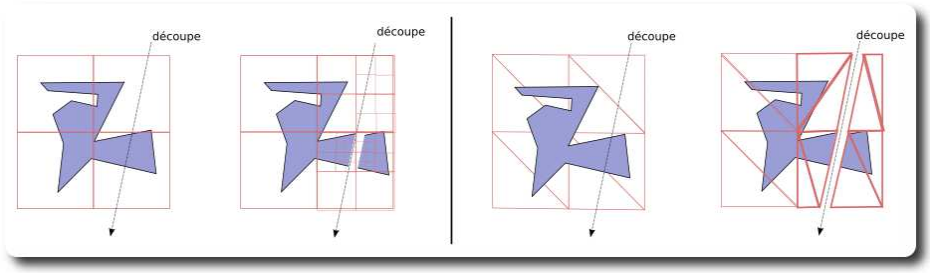

Fig. 3.15 : *La découpe d'une grille implique de générer un très grand nombre de cellules le long du front de coupe. Ce ne serait pas la cas si le maillage de l'objet se composait de tétraèdres.*

Conclusion

Dans ce chapitre, nous avons présenté un modèle complet d'animation d'objets déformables illustrant le principe de la multireprésentation et les avantages de la multirésolution. Ce modèle est utilisé pour simuler des objets visuellement complexes dans des temps de calcul compatibles avec une utilisation en temps interactif. Le modèle proposé est équipé d'un système d'Oracle décisionnel qui adapte, en cours de simulation, la précision des calculs en fonction du temps disponible. On a aussi cherché à découper un tel modèle d'objet, cependant, les obstacles à sa réalisation nous conduisent à envisager d'autres méthodes exposées dans le chapitre (5).

Chapitre 4

Calcul de carte de distance et détection de collision

Sommaire

Introduction

Dans ce chapitre, nous présentons nos travaux concernant un algorithme efficace de calcul de carte de distance pour la détection de collision entre objets découpables. L'objectif de ces travaux était de traiter les collisions entre des objets déformables découpables représentés par des maillages tétraèdriques ; de tels maillages étant très courants en informatique et servant de base à la simulation Éléments Finis ou aux méthodes masses-ressorts.

Le principe de l'approche est d'utiliser la carte de distance des objets en intersection pour calculer les forces de réponse. Comme le calcul de la carte est une opération coûteuse, celle-ci est approximée sous la forme d'une interpolation linéaire par morceaux à partir des tétraèdres formant le maillage. Dans [FL01], les auteurs proposent d'utiliser un algorithme de Fast-Marching pour calculer efficacement la distance au bord. Cet algorithme nécessite une grille de calcul en plus du maillage, et ne permet pas de traiter les objets découpables. Cette carte de distance est utilisée pour traiter les collisions et les auto-collisions entre des objets déformables. Nous avons donc proposé un nouvel algorithme (publié dans [MAC04]) qui s'appuie directement sur le maillage pour calculer cette carte. Les points forts de cet algorithme sont :

- la grille de calcul n'est pas nécessaire, elle est remplacée par le maillage tétraèdrique de l'objet ;
- la rapidité ;
- la précision par rapport à la version du Fast Marching à base de différences finies qui est exploitée dans l'article [FL01].

Pour évaluer cette méthode, nous avons développé un modèle d'objets déformables, à l'aide d'un maillage masses-ressorts, et découpables par suppression d'éléments ; modèle qui a été inséré dans le simulateur Spore développé par l'équipe.

Le chapitre est organisé de la manière suivante : une présentation générale de la méthode (notamment le pipeline de collision (sec. 4.1)), le calcul rapide de la carte de distance (présentant l'algorithme original de Fast-Marching que nous comparons à notre propre proposition (sec. 4.2)), le détail de l'insertion concrète de cet algorithme dans le pipeline de collision, et enfin, le calcul de réponse que nous avons adopté (sec. 4.3).

4.1 Gestion des collisions par carte de distance

Dans le cadre de ce travail, on suppose que les objets sont représentés par un maillage tétraèdrique. Ce maillage est utilisé pour le calcul des déformations – par une méthode à base de masses-ressorts – et pour le calcul des collisions. Aux sommets de ce maillage, on stocke la distance entre le sommet et le bord de l'objet le plus proche. A partir de ces sommets, la distance au bord en tout point de l'objet est calculée par interpolation linéaire. De cette distance, une réponse à la collision tend à repousser les objets en intersection. Pendant que l'objet se déforme, et qu'il est découpé, il est nécessaire de mettre à jour les informations de distance portées par les sommets du maillage (fig. 4.1). C'est le rôle des algorithmes de calcul de carte présentés dans la section (4.2).

Pour tester la validité de l'algorithme, en situation réelle, nous avons implémenté une opération simple de découpe : la suppression de tétraèdres. On l'a vu dans l'état de l'art,

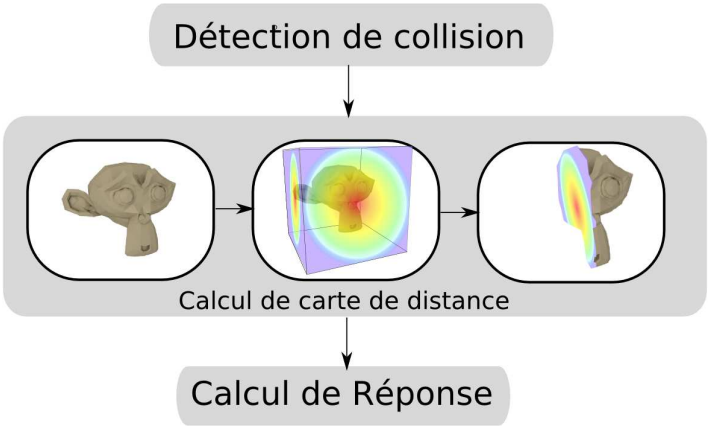

Fig. 4.1 : *Schéma de l'approche de [FL01]. Nos contributions concernent la phase de calcul de carte de distance et la phase de réponse à la collision.*

la suppression d'éléments ne permet pas la production de découpes fines. On peut noter que cette technique de découpe, par suppression d'éléments, est reprise dans Spore pour modéliser certaines opérations chirurgicales (fig. 4.1).

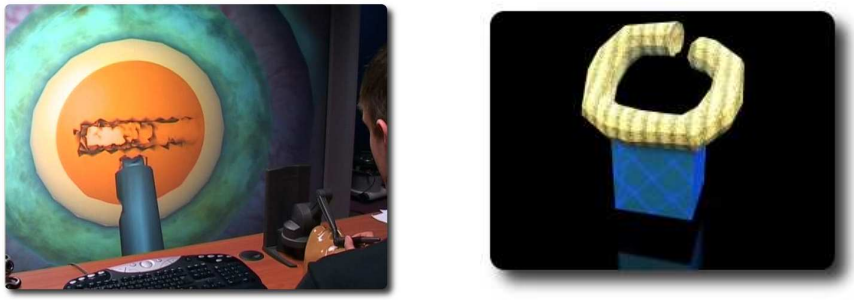

Fig. 4.2 : *Opération de l'oeil où la cornée doit être creusée. Cette image illustre bien la découpe par suppression d'éléments.*

4.2 Calcul rapide de carte de distance par Fast-Marching

Pour bien poser le problème, nous allons, dans un premier temps, présenter la méthode du Fast Marching introduite par James Sethian dans [Set96], qui résout l'équations Eikonale de la forme :

$$\sum_{i=1}^{n}(\frac{\delta u(x)}{\delta x_i})^2 = F(x) \tag{4.1}$$

Le point qui nous intéresse n'est qu'un cas particulier de cette équation : lorsque $F(x) = 1$. Résoudre l'équation Eikonale est alors équivalent à calculer la distance géodésique sur un

domaine géométrique donné. Ainsi on peut calculer la distance au bord d'un maillage comme le font les auteurs de [FL01].

4.2.1 Calcul de distance sur une grille par Fast-Marching

La méthode s'appuie sur une technique de propagation de front, très proche de l'algorithme du plus court chemin dans un graphe de Dikjstra. Cette méthode permet de calculer la distance entre un ensemble de primitives (formant le bord d'un objet) et le reste du domaine. En partant des primitives-source, l'information de distance est propagée de proche en proche jusqu'à couvrir l'ensemble du domaine de calcul. Le domaine est discrétisé sous la forme d'une grille de voxel, les noeuds du bord étant marqués. Chaque nœud de cette grille est associé à un scalaire d qui stocke la valeur courante de la fonction et un statut $s \in init, actif, inactif$ indiquant si le nœud a déjà reçu une valeur valide. Pendant l'initialisation, tous les nœuds de la grille ont leur distance positionnée à $d = +inf$ et leur statut $s = init$. Les nœuds-source sont alors marqués avec les valeurs suivantes $d = 0$ $s = actif$ et insérés dans un minimier[12]. Le rôle du minimier est la sélection du nœud à utiliser dans le processus de mise à jour (fig 4.3). Le calcul de la valeur mise à jour est réalisé à l'aide d'un schéma aux différences finies qui approxime la valeur recherchée à partir des noeuds voisins connus.

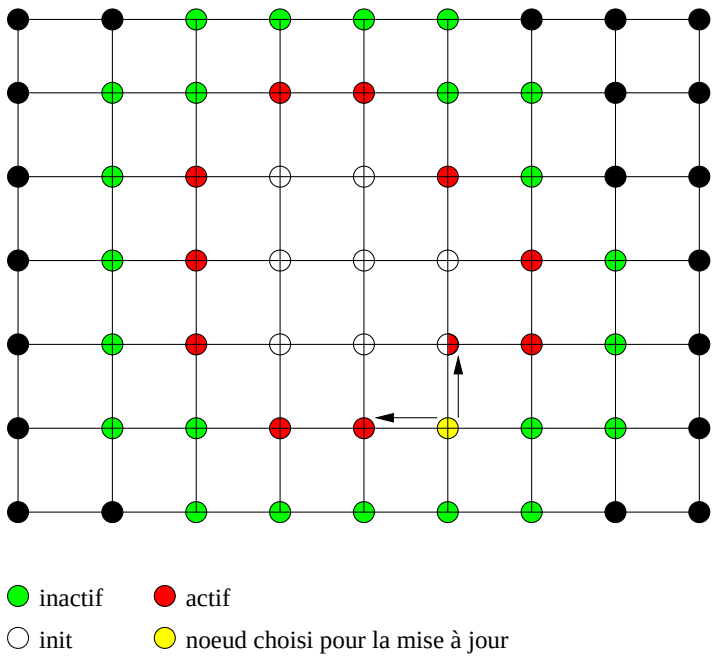

Fig. 4.3 : *Une étape de la méthode de Fast Marching. La cellule de valeur d minimale est extraite du minimier. A partir des valeurs connues, les voisins de la cellule sont mis à jour par un calcul de différences finies.*

[12] Un minimier est une structure de type tas (heap) dont la racine stocke l'élément de valeur minimale [Wikb]. L'insertion d'une nouvelle valeur dans le tas se fait en $O(n.logn)$ et la récupération de la valeur minimale du tas $O(1)$

Dans le contexte de la détection de collision, l'algorithme de Fast-Marching s'utilise avec succès dans [BMF03], pour l'interaction entre objet rigide et tissu, et dans [FL01], pour la gestion de collision d'un objet déformable animé par la méthode des Élément Finis. Ces deux méthodes utilisent l'algorithme du Fast-Marching pour calculer la distance entre les noeuds d'une grille 3d et la surface d'un objet 3d. Pour cela, les méthodes discrétisent la surface de l'objet 3d dans la grille de calcul. C'est à partir de cette discrétisation que la carte est calculée par Fast Marching. [BMF03] emploie une carte adaptative (ADF) : la grille de calcul étant subdivisée par un octree. Cet octree sert ensuite dans le calcul d'une réponse plausible à la collision d'un tissu avec un objet volumique. Dans [FL01], les objets sont représentés par un maillage tétraèdrique. Il faut donc une étape supplémentaire pendant laquelle on convertit, par interpolation trilinéaire, la distance calculée sur la grille en une distance aux sommets du maillage.

L'usage d'une grille pour le calcul n'est pas complètement satisfaisant :

- précision liée au calcul numérique : l'algorithme s'appuie sur un schéma aux différences finies qui introduit une erreur à chaque étape du calcul ; l'algorithme se basant sur une technique de propagation de front, l'erreur est elle-même propagée ;
- précision de la conversion « maillage vers grille » : la résolution de la grille de calcul doit être suffisante pour capturer correctement la complexité de la forme de l'objet. Trop grossière, la distance calculée est complètement erronée ; trop fine, elle consomme alors trop de mémoire et sa mise à jour devient assez longue. Pour donner un exemple, lors de nos tests, passer d'une grille 20x20 à une grille 100x100 divise l'erreur par 8, mais multiplie le temps de calcul par 30 et la consommation de mémoire par 25 (voir 4.2.3) ;
- calculs inutiles : sur la figure 4.4, on remarque que les méthodes de calcul employant une grille calculent la distance externe ; calcul inutile pour l'utilisation que nous faisons de la distance.

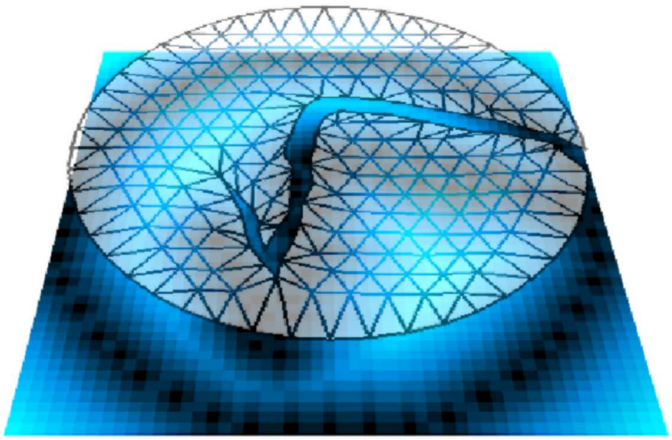

Fig. 4.4 : *Un objet (le disque en haut) et sa grille de calcul. On observe qu'une grande partie de la grille de calcul n'est associée à aucun sommet du maillage. Le calcul de distance y est alors superflu.*

4.2.2 Calcul de distance sans grille

Nos premières tentatives conduisaient à dupliquer la grille pour tenir compte des différentes composantes non connexes ; de plus, pour maintenir une précision suffisante la résolution de la grille était adaptée en cours de simulation. Ces constatations nous conduisent à proposer un nouvel algorithme de calcul de la carte s'affranchissant de cette grille 3d et calculant directement cette carte à l'aide de la structuration offerte par le maillage tétraèdrique. Il est intéressant de noter que l'algorithme du Fast-Marching est en réalité très proche de l'algorithme de Dikjstra (calcul du plus court chemin dans un graphe) et que son auteur, James Sethian, avait déjà étendu l'algorithme pour que celui-ci fonctionne sur des triangulations [KS98]. La méthode proposée par Sethian résout l'équation Eikonale dans sa forme générale et s'appuie toujours sur une méthode de différences finies pour calculer les valeurs encore inconnues. Cet algorithme est assez complexe à mettre en œuvre car il suppose que la variété triangulée ne contienne pas d'angle obtus. Cette méthode se distingue de celle que nous proposons. En effet, notre objectif n'est pas de résoudre l'équation Eikonale dans le cas général, mais uniquement dans le cas particulier qui correspond à la fonction de distance. Cette simplification est déjà utilisée dans [Tsa02]. Cependant il est important de noter qu'en résolvant l'équation Eikonale, on calcule la distance géodésique entre deux points du domaine, ce qui diffère du calcul de la distance euclidienne. La différence entre les deux résultats est illustrée sur la figure 4.5. On montre cependant que pour calculer une distance interne, remplacer le schéma aux différences finies par un calcul explicite de la distance euclidienne est faisable :

Soit $d_g(x,p)$ la distance géodésique obtenue à l'aide de l'algorithme de Fast-Marching et $ch(x,p)$ le chemin liant les points x et p. Le bord du domaine est modélisé à l'aide d'éléments linéaires et le chemin peut être décomposé suivant qu'il passe ou non par les bords du domaine : $ch(x,p) = ch(x,x_1)+ch(x_1,x_2)+...+ch(x_n,p)$. La distance associée est, elle aussi, décomposée en une somme de longueurs de segments successifs : $d_g(x,p) = d_e(x,x_1) + d_e(x_1,x_2) + ... + d_e(x_n,p)$. Supposons qu'on veuille maintenant calculer la carte de distance d'un objet : cela suppose que pour tout point p, on associe la plus petite distance au bord de l'objet : soit un point p quelconque et $ch(x,p)$ l'ensemble des chemins géodésiques liant un point du bord à p. On a :

$$min(d_g(x,p)) = min(d_e(x,x_1) + d_e(x_1,x_2) + ... + d_e(x_n,p))$$

Comme x_n appartient à l'ensemble des primitives du bord, on sait que le minimum correspond au seul dernier terme du chemin $(d_e(x_n,p))$. Une autre façon de voir consiste à considérer que la distance géodésique est équivalente à la distance euclidienne si l'espace est localement convexe, c'est-à-dire quand le segment qui relie deux points x_1 et x_2 n'intersecte pas les bords du domaine. Or, le calcul de la carte génère une segmentation convexe de l'objet. Nous pouvons donc remplacer le calcul de l'équation Eikonale par le calcul explicite de la distance euclidienne.

L'algorithme de calcul de la carte sans grille est le suivant (fig. 4.6) : soit Ω le domaine sur lequel on calcule la distance interne. Ce domaine, un sous-ensemble de R^2 ou R^3, est pavé à l'aide d'un maillage composé de triangles, dans le cas 2d, ou de tétraèdres, dans le cas 3d. Pour cela, le maillage est assimilé à un graphe $G(V,E)$. L'ensemble des sommets du maillage est associé aux nœuds du graphe. Les arêtes de ce graphe correspondent aux arêtes des tétraèdres du maillage. Chaque nœud n du graphe stocke la position du sommet dans

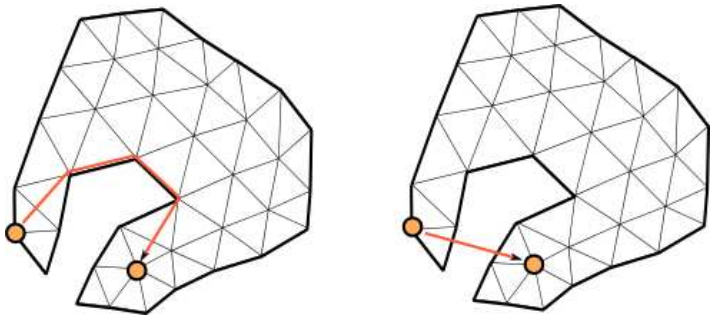

Fig. 4.5 : *Différence entre distance géodésique (à gauche) et distance euclidienne (a droite).*

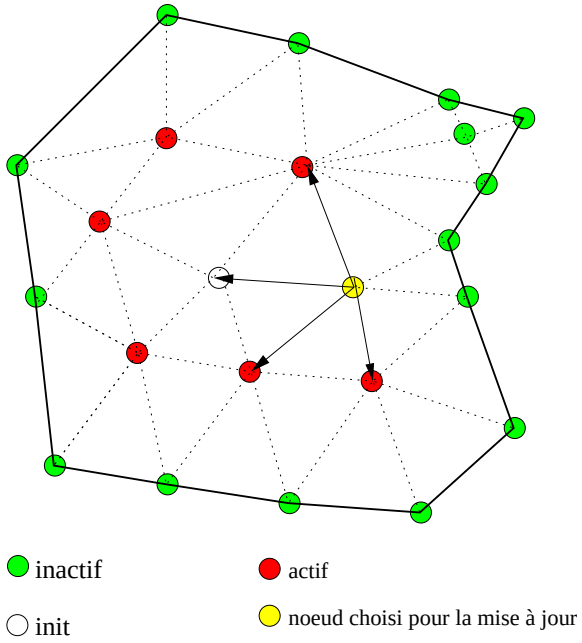

Fig. 4.6 : *Algorithme de calcul de la carte de distance sans grille.*

R^3, notée n_x, la plus petite distance au bord connue n_d et un pointeur vers la primitive du bord à laquelle correspond cette distance n_f. Tout comme l'algorithme du Fast-Marching les nœuds du maillage sont initialisés avec $n_d = +\inf$, et la primitive la plus proche est laissée vide ($n_f = null$). Les noeuds du bord de l'objet sont initialisés à $n_d = 0, n_f =$ n et insérés dans un minimier. La boucle centrale de l'algorithme consiste à prendre la valeur minimale du minimier et à s'en servir pour mettre à jour les informations stockées sur les noeuds voisins. Le schéma de mise à jour des noeuds voisins s'appuie sur le calcul de la distance euclidienne $dist(n_x, f)$ et sur les règles suivantes :

$$n_f = (n_d > dist(n_x, f))?f : n_f \tag{4.2}$$
$$n_d = (n_d > dist(n_x, f))?dist(n_x, f) : n_d \tag{4.3}$$

Les primitives du bord de l'objet peuvent être des points, des arêtes ou des triangles. La fonction de distance est définie à partir des fonctions de distance propres à ces primitives (pour le calcul voir [SE03]) :

$$dist(x,p) = \begin{cases} d_{pts}(x,P) & \text{si p est de type point} \\ d_{seg}(x,S) & \text{si p est un segment} \\ d_{tri}(x,T) & \text{si p est un triangle} \end{cases} \quad (4.4)$$

Nous avons implémenté deux versions de cet algorithme. La première calcule réellement la distance par rapport aux sommets, aux arêtes et aux triangles de la surface. La seconde, plus simple, ne le fait que par rapport aux nœuds de la surface. Cette version simplifiée ne traduit pas correctement la distance au bord de l'objet. Néanmoins, nous observons, dans les différentes expérimentations menées, que l'erreur générée est contrebalancée par des temps de calcul inférieurs.

4.2.3 Comparaison des deux méthodes

L'évaluation de l'algorithme porte sur deux aspects : la précision de la carte de distance obtenue, ainsi que la rapidité des calculs. Pour cela, nous avons comparé l'algorithme avec les méthodes suivantes :

- méthode naïve : la distance est calculée en évaluant l'ensemble des distances possible et en choisissant la minimale ; la complexité d'un tel algorithme est de $O(n.m)$, avec n le nombre de nœuds à évaluer et m le nombre de primitives composant le bord ;
- Fast-Marching : la distance est calculée par l'algorithme du Fast-Marching. Nous avons testé plusieurs résolutions de grille de calcul : les résultats que nous présentons sont calculés sur des grilles de 20x20 et 100x100 ;
- CFFM : la distance est calculée par notre algorithme en propageant la primitive – point, segment, triangle – la plus proche ;
- CPFM : la distance est calculée par notre algorithme mais en ne propageant que la primitive point.

La distance est calculée pour les 300 images de l'animation de tissu de résolutions différentes qui se déchire (fig. 4.7). Pour chacune des méthodes, nous avons mesuré le temps de calcul ainsi que l'erreur maximale et l'erreur moyenne. Les résultats sont présentés dans les tableaux 4.8 et 4.9. Ces résultats montrent que les algorithmes CFFM et CPFM sont les plus précis mais aussi les plus rapides. Les mauvais résultats de l'algorithme de Fast Marching sont liés d'une part à la phase de discrétisation de l'objet sous la forme d'une grille, et d'autre part à l'utilisation d'un schéma aux différences finies qui n'offre qu'une approximation des opérateurs différentiels. En outre, le caractère incrémental de la méthode tend à propager cette erreur liée à l'ensemble du calcul. L'algorithme que nous proposons n'est pas sensible à cela. On observe que notre méthode est aussi nettement plus rapide. Néanmoins, si on reporte le temps de calcul sur le nombre de nœuds du maillage (la grille ou le maillage tétraèdrique), on remarque que c'est la méthode du Fast-Marching (sur la grille) qui reste la plus efficace. Cependant, cette efficacité est contrebalancée par le très grand nombre de nœuds nécessaire pour obtenir des résultats suffisamment précis (d'après nos tests, une grille de taille 20x20 est un minimum).

Ces résultats encourageants nous amènent à réaliser des tests de performance de la méthode CPFM en 3d sur des maillages utilisés dans le simulateur (figure 4.10). Ces tests montrent que, même sur le plus gros maillage dont nous disposions (trois cent mille tétraèdres

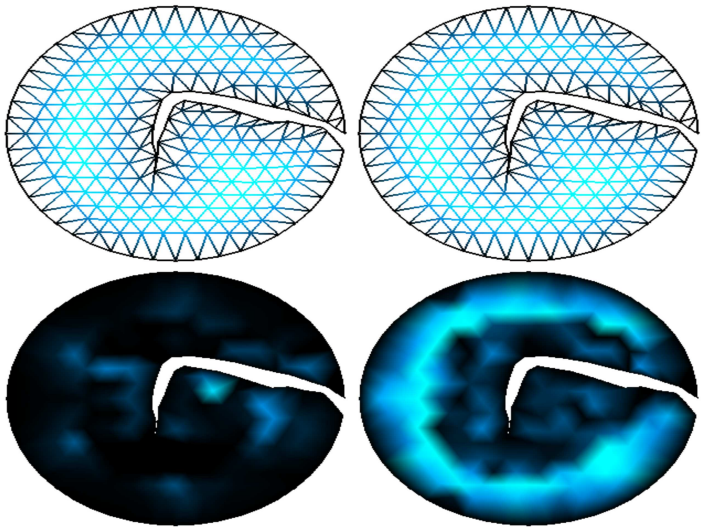

Fig. 4.7 : *Logiciel de test, un disque est animé et découpé. On effectue ensuite le calcul de la carte de distance. A gauche, le calcul à l'aide de la méthode CPFM, à droite, la même carte de distance calculée avec l'algorithme de Fast-Marching et une grille de résolution 100x100. Sur la ligne du bas, on représente, à l'aide d'un gradient de couleur, l'erreur de calcul. Les couleurs claires indiquent des erreurs plus importantes.*

Algorithme	temps en ms	erreur maximale	erreur moyenne
Naïf(exact)	9.930		
Fast-Marching 20x20	0.288	0.326	0.085
Fast-Marching 100x100	9.728	0.149	0.011
CFFM	0.630	0.02	8.10^{-5}
CPFM	0.196	0.12	0.008

Fig. 4.8 : *Comparaison des performances pour les 300 images de l'animation d'un disque composé de 486 noeuds.*

/ cinquante mille noeuds), la carte de distance pouvait être calculée en moins de $0.1s$. Ce temps de calcul est tout à fait satisfaisant comparé à celui nécessaire à la simulation avec la méthode des masses-ressorts. Nous avons ensuite fait varier la résolution des maillages et avons mesuré les temps de calcul de la méthode CPFM pour les objets *Cheval* et *Pion*, mais cette fois-ci, en faisant varier le nombre de noeuds de 1000 à 50000.

Dans le contexte de la détection de collision d'objets déformables, on remarque encore que les méthodes CFFM et CPFM supportent, sans modification, la découpe du maillage.

Algorithme	temps en ms	erreur maximale	erreur moyenne
Naïf(exact)	28.0		
Fast-Marching 20x20	0.600	0.34	0.098
Fast-Marching 100x100	12.0	0.081	0.016
CFFM	5.0	0.002	3.10^{-6}
CPFM	2.3	0.06	0.001

Fig. 4.9 : *Comparaison des performances pour les 300 images de l'animation d'un disque composé de 1000 noeuds.*

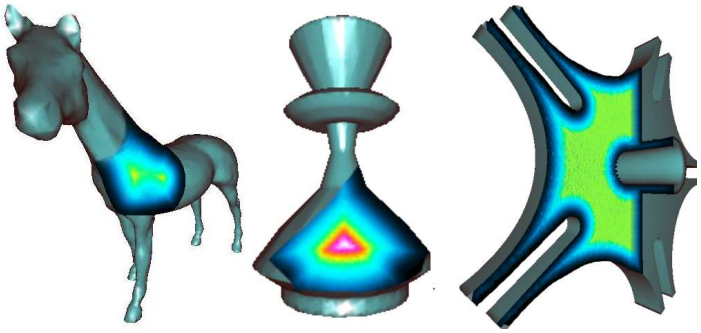

Fig. 4.10 : *Même des objets composés de plus de 300 000 tétraèdres (50 000 noeuds) ont leur carte calculée en 0.1s.*

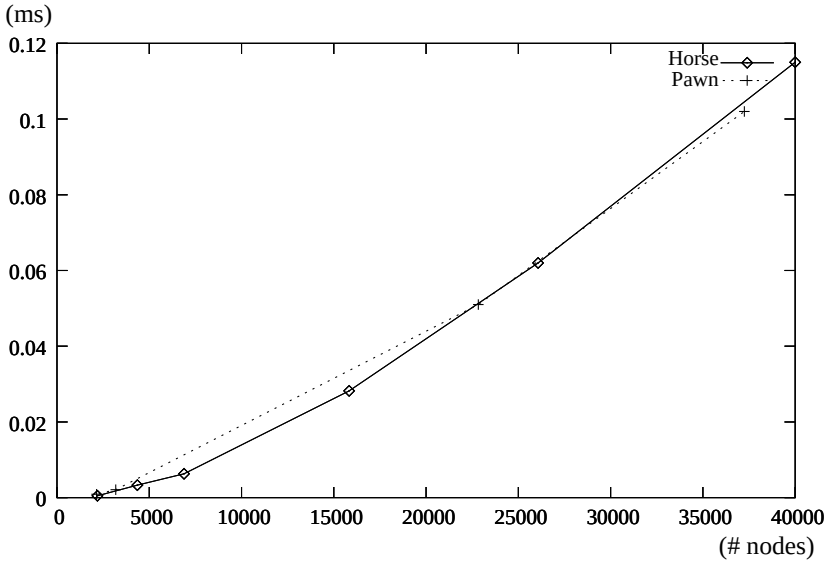

Fig. 4.11 : *On fait varier le nombre de nœuds des maillages Pion et Cheval (axe horizontal) et on mesure le temps de calcul (en s) de la méthode CPFM.*

4.3 Application à la détection de collision

Le pipeline de collision employé dans ce chapitre est représenté sur la figure (4.12). Il s'appuie sur une grille de voxel, à la manière de [THM+03], afin de réduire le nombre de tests d'intersection tétraèdre-tétraèdre. Cette grille de voxel se différencie de la grille de calcul du Fast-Marching car elle est globale à la scène et non plus spécifique à l'objet. Il est intéressant de remarquer qu'on pourrait aussi utiliser l'algorithme du Fast-Marching sur cette grille globale. Cependant, la résolution nécessaire pour avoir des résultats précis avec la méthode du Fast-Marching suppose une grille nettement plus fine que la grille globale employée. Une fois terminée la localisation des couples de tétraèdres potentiellement en intersection, nous utilisons le test d'intersection exact (présenté avec son code source dans [GPR02]). Lorsqu'une intersection est détectée, la réponse à la collision est calculée à partir de la carte (fig. 4.13). Nous avons mis en place un système de mise à jour "à la demande" de la carte de distance ;

celui-ci ne recalcule qu'une partie de la carte, la quantité de mise à jour est fonction de la profondeur de pénétration des objets.

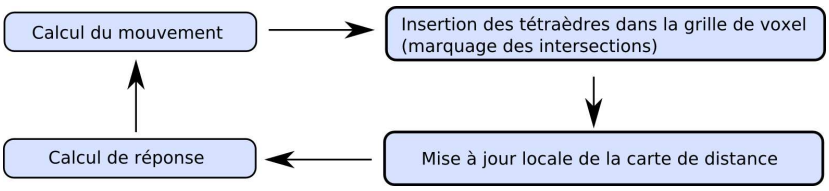

Fig. 4.12 : *La boucle de simulation comprenant l'étape supplémentaire de mise à jour de la carte entre la détection et la réponse à la collision.*

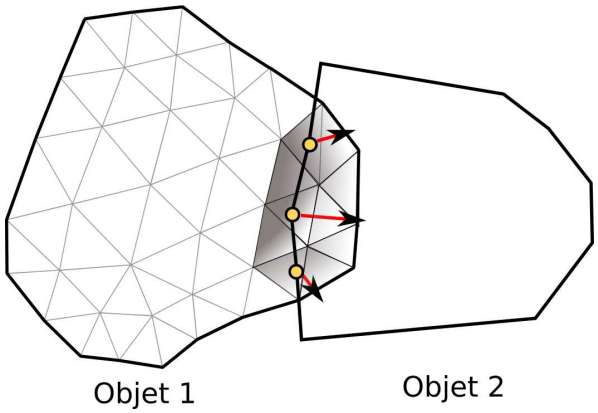

Fig. 4.13 : *A partir de la carte de distance, on calcule des forces de réponse par la méthode de pénalité.*

4.3.1 Calcul de la carte de distance "à la demande"

Le calcul "à la demande" de cette carte a pour objectif de limiter la quantité de nœuds qui doit être mise à jour pour obtenir la réponse à la collision. Ainsi, lorsque les objets ne s'intersectent pas, le calcul complet de la carte est inutile. Plus précisément, un calcul de la carte peut être interrompu lorsque la distance est calculée pour l'ensemble des tétraèdres en intersection. Pendant la phase de détection de collision, on compte les tétraèdres en intersection et les noeuds qui y sont associés ; ce sont les nœuds pour lesquels il faut calculer la distance. A chaque fois qu'un de ces noeuds passe de l'état *vivant* à l'état *mort*, le compteur de nœuds est décrémenté. La boucle centrale de l'algorithme s'arrête lorsque le compteur atteint zéro. Le calcul à la demande de la carte réduit considérablement le temps de calcul ; les objets qui ne sont pas en intersection n'ont pas leur carte recalculée et, lorsqu'il y a interpénétration, celle-ci est en général concentrée sur la surface de l'objet. En pratique, des pénétrations importantes des objets se traduisent simplement par des calculs plus longs (fig. 4.14).

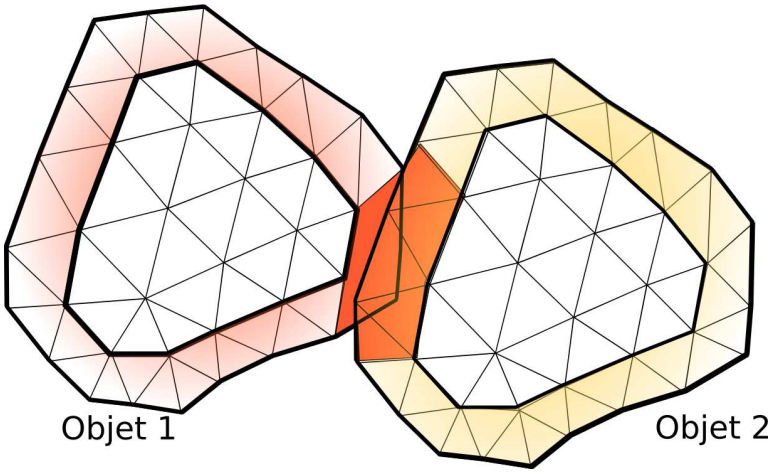

Fig. 4.14 : *Calcul à la demande de la carte de distance. En général, seule une petite bande à la périphérie du bord doit être recalculée.*

4.3.2 Calcul de réponse par pénalité

Une fois toutes les collisions détectées et la carte de distance mise à jour, des forces de pénalité sont calculées de manière à repousser les objets qui s'interpénètrent. La méthode de la pénalité produit des forces en réponse à une mesure de l'interpénétration. Dans ce travail, cette mesure de l'interpénétration est évaluée à partir de la carte portée par les tétraèdres. Dans un premier temps le calcul dit de « point dans un volume » [FL01] est employé. Ce calcul associe à chaque nœud mécanique inclus dans un tétraèdre, une force de pénalité qui repousse le nœud hors de l'objet. Les auto-collisions peuvent être traitées de la même façon, en testant les intersections entre les couples de tétraèdres non-adjacents d'un même objet et en construisant une réponse par pénalité [FL01]. Le fait de traiter les auto-collisions rajoute un grand nombre de tests d'intersection qui ralentit la simulation. L'accélération des auto-collisions est un point de recherche encore ouvert. Le calcul de réponse par sommet est relativement peu coûteux, mais il suppose que les objets soient finement discrétisés (c'est-à-dire qu'il y ait beaucoup de nœuds). Lorsqu'il n'y a pas assez de noeuds, on risque de manquer des collisions. Dans le cadre de la simulation temps-réel, nous ne pouvons simuler des objets trop finement discrétisés. Nous avons donc développé un autre calcul de réponse dit « segment dans un volume », qui a l'avantage de ne manquer aucune collision.

Réponse de type « point dans un volume »

Le calcul de réponse par point est basé sur l'approximation de la distance de pénétration d, à partir de la carte de distance. Cette distance de pénétration d est calculée comme une combinaison linéaire des valeurs de distance portées par les nœuds du tétraèdre. Dans un premier temps, le point où la fonction doit être évaluée est converti en coordonnées locales au tétraèdre. On note $u = u_{1..3}$, les coordonnées locales d'un point p dans le repère dont l'origine coïncide avec x_4 et dont les axes sont portés par les arêtes adjacentes à x_4 (fig. 4.15). Les coordonnées u sont calculées de la manière suivante :

$$u = G^{-1}[p - x_4] \tag{4.5}$$

où G est une matrice 3x3 définie par :

$$G = [p - x_1, p - x_2, p - x_3] \tag{4.6}$$

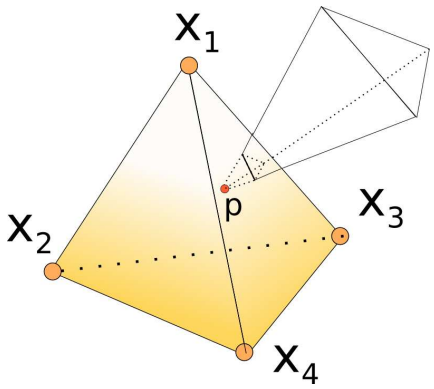

Fig. 4.15 : *Intersection point - tétraèdre. La distance est interpolée au point p, par interpolation des valeurs portées par le tétraèdre.*

Un point p inclus dans un tétraèdre remplit les conditions suivantes : $u_1 \geq 0, u_2 \geq 0, u_3 \geq 0$ and $u_1 + u_2 + u_3 \leq 1$. Pour un tel point, les informations de distance portées par les nœuds du tétraèdre, et notées $d_{0..3}$, peuvent être interpolées à partir des coordonnées barycentriques :

$$\tilde{d} = u_1.d_1 + u_2.d_2 + u_3.d_3 + (1 - u_1 - u_2 - u_3).d_4 \tag{4.7}$$

Cette information de distance nous renseigne sur l'intensité de la pénalité à appliquer. La seule information d'intensité ne suffit pas à générer une force de pénalité, et il faut lui adjoindre une direction. Cette direction est calculée comme gradient de la carte de distance ; ce qui correspond à calculer les normales aux isovaleurs de la carte. En pratique, comme ce champ de distance est calculé par interpolation linéaire à partir des valeurs portées par les sommets du tétraèdre, le gradient est constant pour l'ensemble du tétraèdre. Ce gradient est obtenu par :

$$\nabla\tilde{d} = \begin{pmatrix} \frac{\partial\tilde{d}}{\partial x} \\ \frac{\partial\tilde{d}}{\partial y} \\ \frac{\partial\tilde{d}}{\partial z} \end{pmatrix} = G^{-1} \begin{pmatrix} u_1 - u_4 \\ u_2 - u_4 \\ u_3 - u_4 \end{pmatrix} \tag{4.8}$$

Une fois le gradient obtenu, la force de pénalité est générée à partir de l'équation du ressort :

$$F = k.d.\nabla\tilde{d} \tag{4.9}$$
$$\tag{4.10}$$

Cette force est appliquée en p et intégrée au bilan des forces de l'objet.

Réponse du type « segment dans un objet »

Le calcul de réponse par sommets génère une réponse si l'un des sommets d'un maillage est inclus dans un tétraèdre du maillage de l'autre objet. Des collisions peuvent être manquées lors des intersections le long des arêtes des tétraèdres. Pour contourner le problème, il faut utiliser des objets très finement discrétisés, à la manière de [BMF03]. Cependant, l'augmentation de la discrétisation est bien souvent incompatible avec la simulation interactive. Nous proposons donc une méthode de calcul de pénalité par segment, plus coûteuse en temps de calcul, mais qui autorise l'utilisation de maillages nettement plus grossiers. Dans un premier temps, les arêtes en intersection sont détectées (fig. 4.16). Les extrémités de l'intersection sont calculées et la force de pénalité y est évaluée. La direction de la force ne change pas sur l'ensemble du segment, on se contente d'évaluer son intensité. Cette intensité est ensuite intégrée sur l'ensemble du segment. Une fois la force obtenue sur le segment, elle est répartie sur les sommets mécaniques les plus proches.

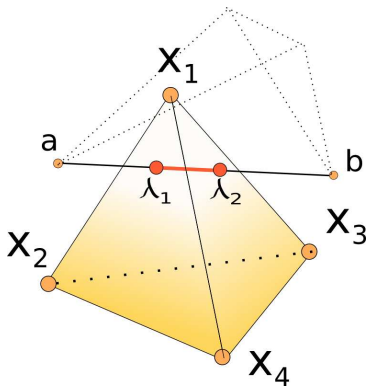

Fig. 4.16 : *Intersection segment - tétraèdre. La distance est interpolée aux extrémités de l'intersection entre le segment et le tétraèdre.*

Soit un segment $[a, b]$ exprimé sous forme paramétrique :

$$x = a_x + \lambda(b_x - a_x) \tag{4.11}$$
$$y = a_y + \lambda(b_y - a_y) \tag{4.12}$$
$$z = a_z + \lambda(b_z - a_z) \tag{4.13}$$

Les deux extrémités de ce segment sont projetées dans le référentiel local au tétraèdre.

$$A = G^{-1}(a - t_4) \text{ and } B = G^{-1}(b - t_4) \tag{4.14}$$

Ce qui donne l'expression du segment dans le référentiel du tétraèdre :

$$u_x = A_x + \lambda(B_x - A_x), u_y = A_y + \lambda(B_y - A_y), u_z = A_z + \lambda(B_z - A_z) \tag{4.15}$$

Il y a une intersection entre le segment et le tétraèdre, s'il existe un nombre λ dans l'intervalle $[0, 1]$, dont les coordonnées dans le repère du tétraèdre vérifient l'équation suivante : $u_1 \geq 0, u_2 \geq 0, u_3 \geq 0, u_1 + u_2 + u_3 \leq 1$. Le tétraèdre est un convexe, l'intersection avec un segment se réduit à un unique intervalle $[\lambda_1, \lambda_2]$. Si cet intervalle est vide, il n'y a pas d'intersection. A partir de ces deux abscisses, il est possible de calculer la distance de pénétration aux deux extrémités de l'intersection par interpolation linéaire des valeurs portées aux sommets du tétraèdre. Une fois calculées, ces valeurs sont ensuite intégrées le long de l'intersection de manière à mettre en corrélation la quantité de matière en intersection et la quantité de force de pénalité. Cette étape d'intégration est réalisée analytiquement et revient à calculer l'aire d'un trapèze formé par le segment et les distances approchées.

$$\tilde{d} = \int_{l \cap t} d = \frac{1}{2}(\tilde{d_1} + \tilde{d_2})||x_1 x_2||$$

La force de pénalité $\overrightarrow{f}$ est obtenue à partir de la mesure de l'intersection $\tilde{d}$. Elle est ensuite répartie en deux forces (f_1,f_2) appliquées aux extrémités de l'intersection (λ_1,λ_2). La répartition des forces est proportionnelle à ($\tilde{d_1}$,$\tilde{d_2}$). L'étape finale du calcul de force consiste à reporter, sur les extrémités du segment, les forces de pénalité obtenues aux extrémités de l'intersection (fig. 4.17). Pour cela, on utilise la position sur l'abscisse paramétrique des forces (f_1,f_2) comme coefficients de pondération. La formule suivante nous donne les forces de pénalité appliquées aux degrés de liberté :

$$a_f = \lambda_1 * \overrightarrow{f_1} + \lambda_2 * \overrightarrow{f_2}$$

$$b_f = (1 - \lambda_1) * \overrightarrow{f_1} + (1 - \lambda_2) * \overrightarrow{f_2}$$

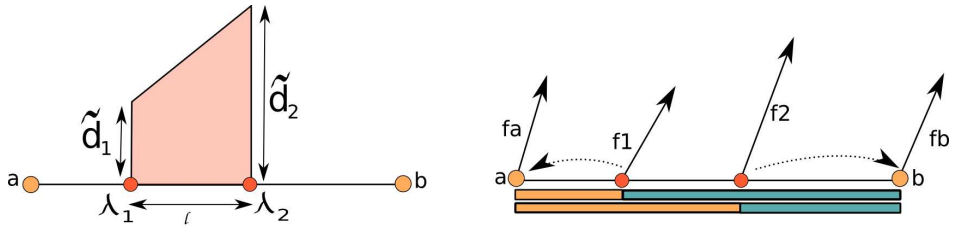

Fig. 4.17 : *Forces calculées sur le segment ab. La force est intégrée sur l'intersection, puis reportée aux extrémités a et b, en utilisant les valeurs λ_1, λ_2 comme pondération.*

Nous avons comparé les deux méthodes de calcul de réponse. La méthode de réponse par segment est sensiblement plus lente que celle par point (fig. 4.19). En contrepartie, elle est plus précise et ne rate pas de collision à cause d'un échantillonnage trop faible des objets.

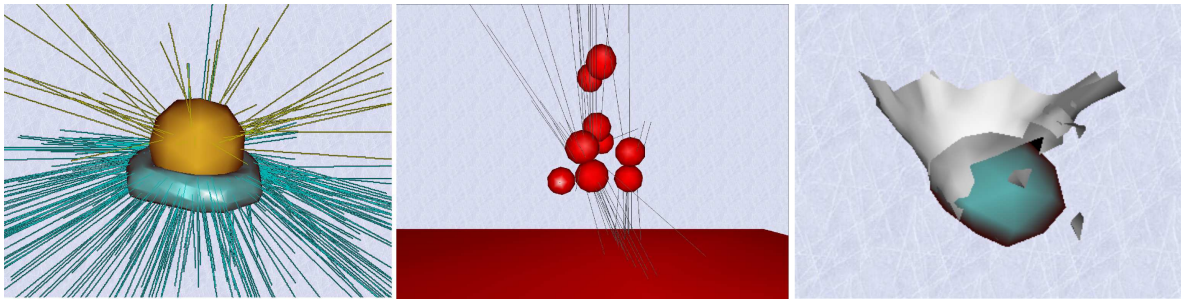

Fig. 4.18 : *Illustrations de [MAC04] : gauche et milieu : représentation des forces calculées ; droite : interaction entre un tissu et un objet déformable.*

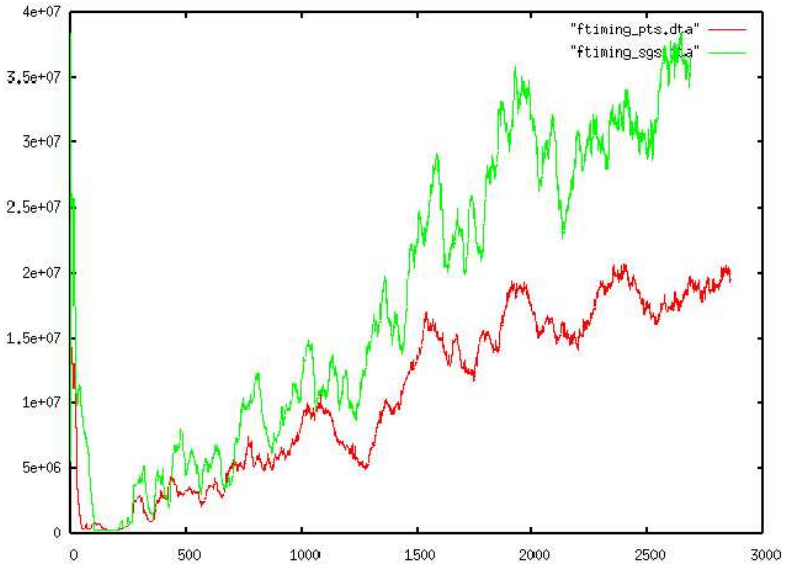

Fig. 4.19 : *Sur une scène complexe, le temps de calcul de la phase de réponse est près de deux fois plus élevé.*

Conclusion

Dans ce chapitre, nous avons présenté un algorithme efficace de calcul de la carte de distance pour des objets représentés par un maillage tétraédrique. Cet algorithme est utilisé pour gérer la collision entre objets déformables et a été présenté dans [MAC04]. Par rapport à [FL01], notre algorithme est à la fois plus rapide et plus précis. En outre, comme il ne nécessite plus de grille de calcul additionnelle, on peut gérer la collision d'objets découpables. L'efficacité du calcul de carte de distance est illustré sur la vidéo disponible à l'adresse `http://www.lifl.fr/~marchal/these/video/chap4_fastmarching.mpg`. En plus du calcul de carte de distance, nous proposons aussi une méthode de calcul de pénalité basée sur les segments en intersection qui supprime les artefacts liés à la méthode de réponse à base de point. La validation de ce travail s'appuie sur le développement d'un modèle d'objets déformables à base de masses-ressorts qui supporte la découpe par suppression d'élément. Concernant la méthode masses-ressorts, celle-ci est simple à implémenter, mais elle est aussi inefficace

pour simuler des objets volumiques déformables en temps interactif. En effet, l'intégration numérique du mouvement, avec des intégrateurs explicites d'ordre faible, est complètement instable (même pour des faibles raideurs) et l'usage des intégrateurs plus évolués, comme le Runge-Kutta4 ou l'Euler implicite entraîne un augmentation considérable du temps de calcul. Cela nous a conduis à travailler sur d'autres méthodes de simulation comme le Shape-Matching que nous avons utilisé dans le cadre du modèle déformable et découpable présenté dans le chapitre suivant.

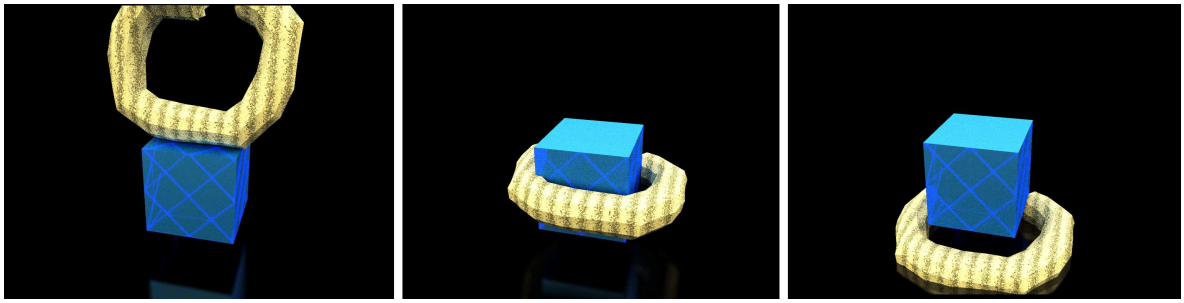

Fig. 4.20 : *Illustrations de [MAC04] : le tore est découpé et entre en collision avec un cube.*

Chapitre 5

Un modèle découpable pour le temps réel

Sommaire

Introduction

Dans ce chapitre, nous illustrons l'utilisation de la multireprésentation avec un modèle d'objets déformables supportant des opérations de découpes fines. Pour cela, nous proposons un modèle d'objets simulables en temps réel, grâce à la technique du Shape-Matching [MHTG05], déformables et découpables. Pour le traitement de la découpe, sera retenue une représentation topologique centrale, sous la forme d'un maillage tétraédrique, à partir de laquelle seront déduites les autres représentations. Un maillage tétraédrique servira de représentation commune, car c'est une structure volumique simple à manipuler et facile à découper. L'algorithme de découpe du maillage est basé sur l'utilisation d'une machine à état qui minimise le nombre de tétraèdres insérés [BGTG04]. Une fois le maillage découpé, les aspects mécaniques, la collision, et l'affichage, seront mis à jour. Les algorithmes de mise à jour dépendent des informations nécessaires aux différents aspects. L'animation à base de Shape-Matching s'appuie sur l'utilisation de groupes de particules simulées. Pendant la découpe, la mise à jour de ces groupes repose sur un algorithme de segmentation ; algorithme qui s'appuie sur un calcul de distance par la méthode du Fast-Marching (voir chapitre 4).

Le chapitre est organisé de la façon suivante : dans un premier temps, nous présentons la structure générale de la méthode (5.1) ; ensuite l'algorithme de découpe du maillage tétraédrique (5.2), suivie par la méthode d'animation basée sur le Shape-Matching (5.3). La particularité de cette méthode d'animation repose sur une segmentation de l'objet en groupes de simulation. Pour traiter la découpe, nous introduirons un nouvel algorithme de segmentation (5.4).

5.1 Structure globale du modèle proposé

La structure de la méthode s'inspire du modèle multireprésentation (voir chapitre (3)). Comme remarqué sur la figure (5.1), la structure multireprésentation du chapitre (3) est maintenant enrichie d'un composant supplémentaire qui stocke une représentation commune encodant la topologie de l'objet. Nous pensons que cette représentation commune est nécessaire pour éviter les ambiguïtés et les incohérences topologiques entre les différentes représentations. Dans l'approche suivie, seule cette représentation-maîtresse est découpée ; les représentations particulières, liées aux aspects (mécanique, affichage, collision) étant mises à jour lors des découpes.

Représentation topologique : le maillage tétraédrique sera notre représentation centrale de la topologie et de la géométrie de l'objet. C'est le modèle de topologie explicite le plus simple à manipuler. On dispose d'un grand nombre d'algorithmes de découpe. En outre, un schéma d'interpolation globale – l'interpolation linéaire par morceaux – facilite l'extension de la méthode : par exemple, interpoler des coordonnées de texture, des couleurs ou bien des masses mécaniques.

Animation : nous avons fait le choix de la méthode du Shape Matching pour le composant mécanique. C'est une méthode qui est qualifiée de "sans maillage" par ses auteurs. Nous qualifions plutôt cette méthode d'a-topologique au sens où elle ne repose sur aucune structure

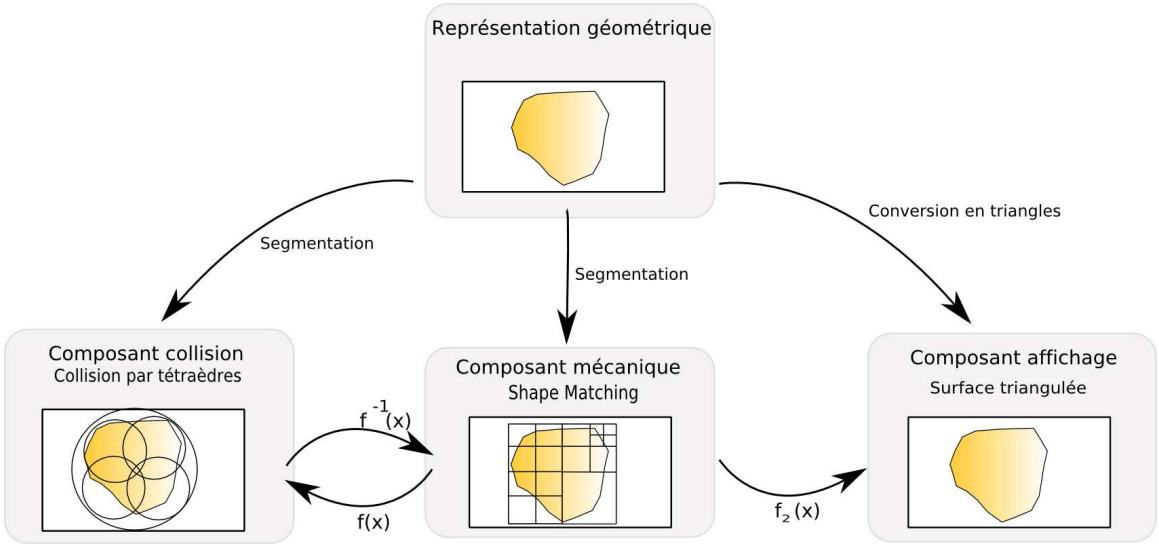

Fig. 5.1 : *Structure d'objet : représentation géométrique/topologique centrale. A partir de cette représentation, différents algorithmes (segmentation, conversion) permettent la mise à jour des représentations mécanique, d'affichage, de collision.*

topologique, implicite ou explicite, contrairement aux méthodes sans maillage du domaine de la mécanique des milieux continus.

Gestion des collisions : pour la gestion des collisions, nous employons des hiérarchies de boîtes englobantes. Ces hiérarchies sont mises à jour pendant les déformations à l'aide de l'algorithme de Van Den Bergen [vdB97]. La mise à jour de la topologie de la hiérarchie pendant les découpes repose sur l'algorithme de segmentation hiérarchique.

Affichage : l'affichage consiste simplement à visualiser la surface du maillage tétraédrique.

Choix d'une représentation géométrique

On a vu que les méthodes qui dérivent de la mécanique des milieux continus resposent sur une représentation volumique de l'objet. La méthode du Shape-Matching n'a pas cette contrainte et se contente d'un nuage de points pouvant décrire aussi bien un volume qu'une surface d'objet. Cependant, nous avons fait le choix d'une représentation volumique maillée. En effet, lorsque l'objet est déformé, sans représentation volumique précise, il est difficile de reconstruire la surface obtenue après découpe (fig. 5.2).

Plus formellement, si la géométrie non-déformée est notée $\tilde{u}(T)$, la déformation par une fonction $f(X)$ de cet objet s'écrit :

$$\tilde{u}_{def}(T) = f(\tilde{u}(T)) \tag{5.1}$$

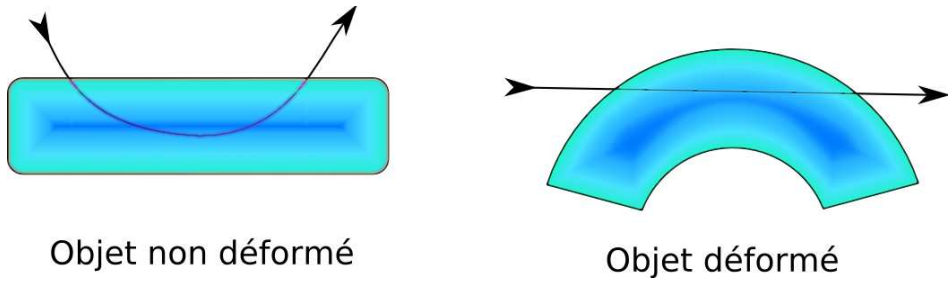

Fig. 5.2 : *Découpe d'un objet déformé. Représenté par sa seule surface, il devient difficile de calculer le chemin de la découpe dans l'espace non-déformé.*

Retrouver le chemin parcouru par l'outil dans l'espace non-déformé suppose que la fonction de déformation $f(X)$ soit définie sur l'ensemble de l'objet, inversible, et simple à calculer ; la plus simple étant l'interpolation linéaire par morceaux, obtenue à partir d'un maillage tétraédrique.

5.2 Découpe du maillage tétraédrique

Dans cette section, nous décrivons la façon de découper la représentation topologique centrale ; on laisse de côté la manière dont celui-ci est animé.

Nous modélisons l'outil de découpe comme un segment orienté, dont le mouvement est contraint par l'orientation (on coupe uniquement dans le sens de la lame). Le mouvement de cet outil est approché linéairement par une succession de triangles. A la fin d'un pas de temps, nous testons l'intersection entre les triangles décrivant le mouvement de l'outil et les tétraèdres du maillage. De ces intersections, on déduit la manière dont le tétraèdre va être subdivisé. On distingue deux cas : le premier où la découpe est *active*, traduit le fait que l'outil n'a pas encore quitté le tétraèdre qu'il est sensé découper. Durant cette phase active la représentation mécanique n'est pas modifiée. Le second cas survient lorsque l'outil quitte le tétraèdre intersecté. A ce moment, les représentations mécaniques et de collision sont effectivement mises à jour.

C'est une machine à état, associée aux tétraèdres, qui suit l'évolution des découpes. Elle suit l'évolution du profil de découpe pendant le mouvement de l'outil et indique lorsque celui-ci n'intersecte plus le tétraèdre. Cette approche est similaire à [BGTG04], bien que sa formalisation diffère au niveau de l'encodage de la machine à état. En effet, nous préférons décrire l'automate d'états au niveau des triangles (formant le tétraèdre) plutôt que directement au niveau des tétraèdres.

5.2.1 Découpe progressive d'un tétraèdre : la machine à état

La machine à état d'un tétraèdre implémente l'automate donné à la figure (5.3). Chaque face stocke son état en fonction des différentes intersections avec l'outil de découpe.

Les flèches modélisent les transitions possibles entre les états. L'état **A0** correspond à un triangle qui n'est pas entré en intersection avec l'outil. Les états **A1** et **A2** correspondent

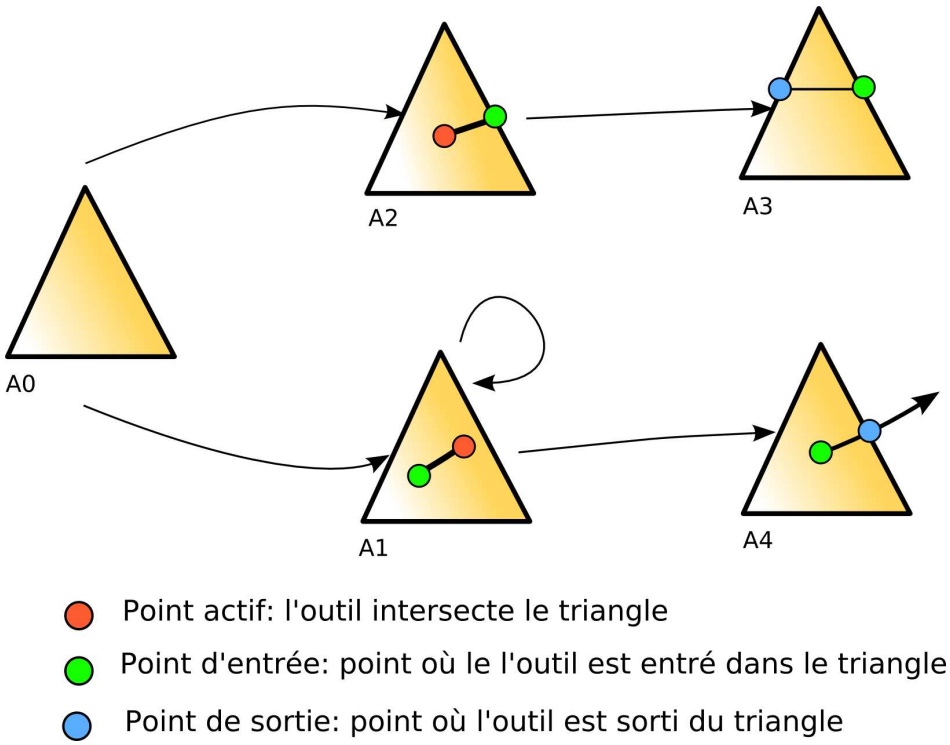

Fig. 5.3 : *Les différentes configurations encodées dans la machine à état.*

à des états temporaires. Les états **A0** , **A3** , **A4** sont qualifiés d'états de sortie. Une fois un état de sortie atteint, l'automate n'est plus mis à jour jusqu'à ce que le tétraèdre soit effectivement subdivisé.

5.2.2 Subdivision du tétraèdre

Tant que l'outil ne quitte pas le tétraèdre, celui-ci n'est pas mécaniquement subdivisé, seul son affichage est modifié de manière à permettre un retour à l'utilisateur. Une fois que l'outil quitte le tétraèdre, celui-ci est effectivement subdivisé, le maillage de l'objet est alors mis à jour tout comme les représentations de collision et de mécanique. La manière dont est subdivisé le tétraèdre (fig. 5.5) est calculée en fonction des états des différentes faces qui le composent (fig. 5.4).

5.3 Animation des déformations

Nous présentons maintenant la manière d'animer un objet déformable, modélisé par un maillage tétraédrique, à l'aide de la technique du Shape-Matching. Dans un premier temps, seuls les sommets du maillage sont utilisés pour l'animation. Ces sommets forment le nuage de particule à partir duquel est extraite la transformation rigide de l'objet – rotation, translation – et ses déformations. En plus d'être une méthode complètement sans maillage, la méthode est dite *inconditionnellement stable* au niveau de l'intégration numérique,

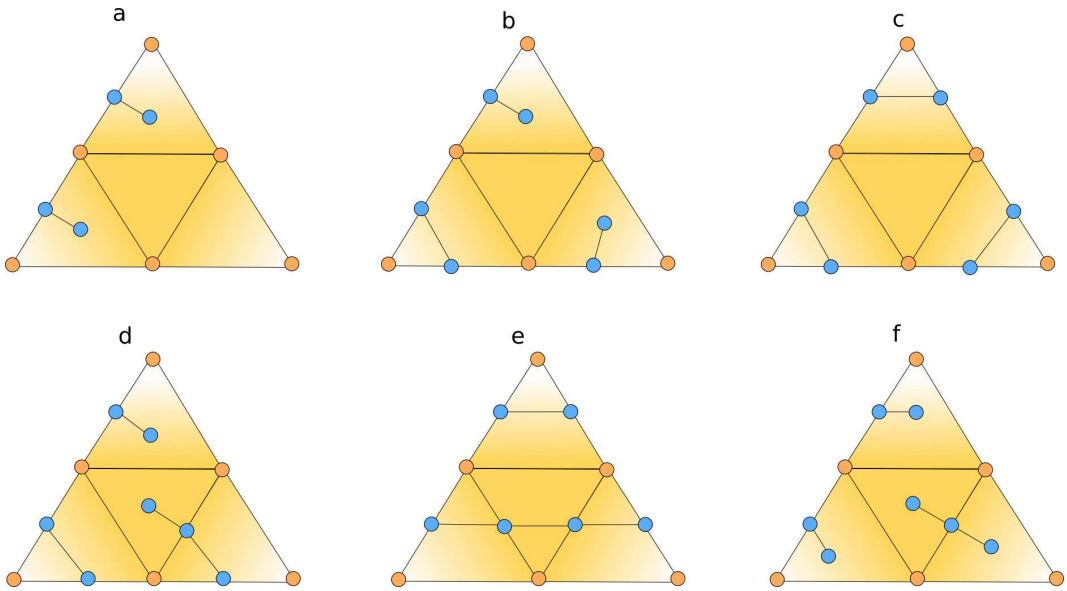

Fig. 5.4 : *Les faces d'un tétraèdre, les différents assemblages en fonction des états possibles des intersections par face.*

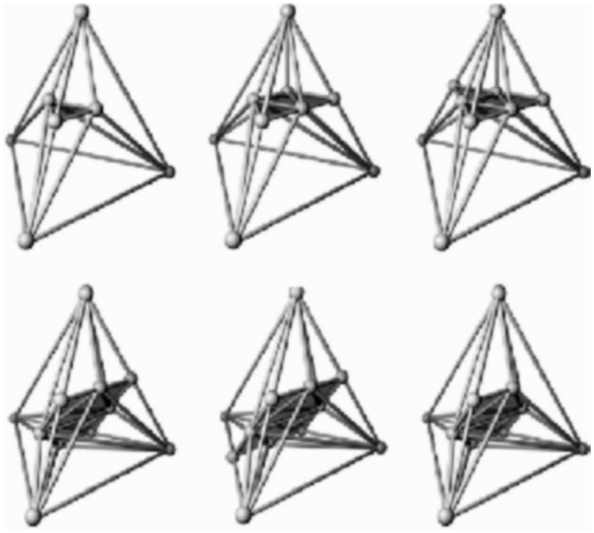

Fig. 5.5 : *Motif de subdivision exprimé sous forme canonique.*

(propriété assez rare au niveau des modèles simulés). Enfin, par opposition aux méthodes Éléments Finis, l'animation par Shape-Matching, utilisable avec un maillage, ne souffre pas du bon ou mauvais conditionnement de celui-ci. Les tétraèdres non-réguliers produits par la découpe ne lui posent pas de problème. En contrepartie, en associant un mouvement rigide unique à l'objet, la méthode du Shape-Matching ne permet pas de traduire toutes les déformations d'un objet ; pour augmenter le potentiel de représentation des déformations, la méthode est étendue par la décomposition des particules de l'objet en plusieurs sous-ensembles appelés clusters, interconnectés ; les clusters étant animés par la méthode du Shape-Matching. On peut aussi remarquer que, ne dérivant pas de la mécanique des milieux continus, la méthode

du Shape Matching est plus adaptée pour une utilisation dans un contexte de "réalité virtuelle plausible" plutôt que celui de la simulation précise de comportement (contexte médical).

5.3.1 Description générale du Shape-Matching

La méthode permet l'animation de nuage de particule à partir de la combinaison d'un mouvement rigide avec une déformation (fig. 5.6). La boucle d'animation est la suivante : à partir de la position courante des particules, la composante rigide du mouvement est extraite par l'algorithme de Shape-Matching. Chaque particule à simuler, de masse m_i, est associée à une position au repos x_i^0, une position courante x_i et une position rigide g_i. Cette position rigide est calculée à partir du nuage de particule par extraction de la composante rigide du mouvement (une rotation, une translation). La mouvement rigide le plus proche

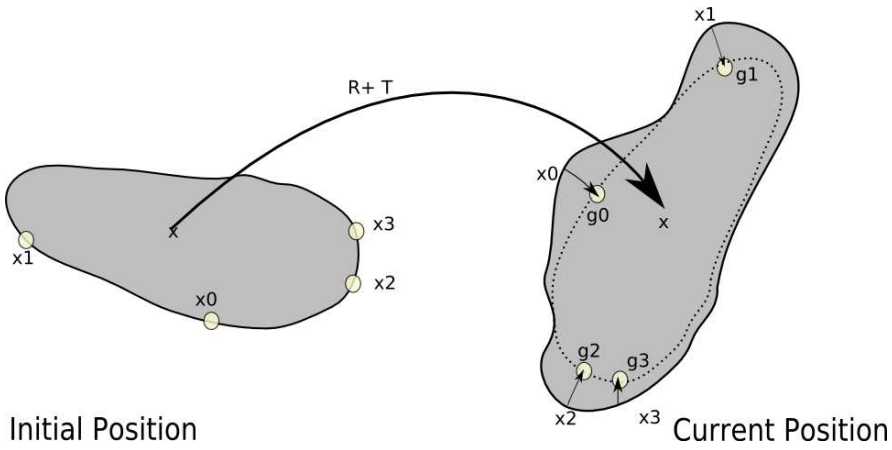

Fig. 5.6 : *La méthode Shape-Matching*

du mouvement est calculé à partir du nuage de particule à sa position de repos x_i^0 et de la position à un instant donné x_i. Le mouvement rigide consiste alors à trouver les vecteurs t, t^0 et la matrice de rotation R telle que (minimisation du carré de la différence entre les positions g_i et x_i) :

$$\sum_i m_i(R(x_i^0 - t_0) + t - x_i)^2. \tag{5.2}$$

Les vecteurs t et t^0 sont calculés à partir du barycentre des deux nuages de point.

$$T = t - t^0, t^0 = \frac{\sum_i m_i x_i^0}{\sum_i m_i}, t = \frac{\sum_i m_i x_i}{\sum_i m_i}. \tag{5.3}$$

Soient $q_i = x_i^0 - t^0$ et $p_i = x_i - t$, les positions des particules par rapport aux barycentres respectifs. Pour calculer la matrice de rotation R, les auteurs proposent dans un premier

temps, de calculer la transformation A qui minimise la fonction $\sum_i m_i(Aq_i - p_i)^2$. Cela revient à calculer :

$$A = (\sum_i m_i p_i q_i^T)(\sum_i m_i q_i q_i^T)^{-1} = A_{pq}A_{qq}. \tag{5.4}$$

Avec A_{qq} une matrice symétrique et donc ne contenant pas de rotation. La rotation optimale R est ainsi extraite de A_{pq} par décomposition polaire.

A partir de la matrice de rotation R et du vecteur de translation T, on calcule la position rigide g_i :

$$g_i = R(x_i^0 - t^0) + t. \tag{5.5}$$

A partir de g_i et de x_i, une force de rappel peut être générée qui ramène les particules libres vers leur position de repos. La connaissance de la destination g_i permet la construction d'un schéma d'intégration temporelle toujours stable.

Les auteurs proposent d'enrichir les déformations possibles en rajoutant des modes de déformation. Ainsi, les auteurs proposent de calculer g_i à partir non plus d'une transformation rigide mais d'une transformation linéaire ou quadratique. Dans la suite, seuls les modes de déformation rigide et linéaire sont employés ; le mode de déformation quadratique n'a pas été utilisé car il nécessite une quantité non-négligeable de précalculs qu'on ne peut alors plus effectuer pendant les découpes.

5.3.2 Animation par clusters

L'animation d'objets déformables par la méthode du Shape-Matching est très efficace du point de vue des performances. Par contre, la déformation n'est pas réaliste lorsque les objets ont une forme différente de celle d'une sphère (objets étoilés). Si l'objet à une forme non-étoilée, les modes de déformation linéaire et quadratique ne sont pas suffisants pour décrire efficacement la comportement de l'objet. Pour augmenter le potentiel de représentation des déformations, les auteurs de [MHTG05] proposent d'utiliser plusieurs composantes rigides, locales, donnant alors plus de flexibilité au modèle (fig. 5.7).

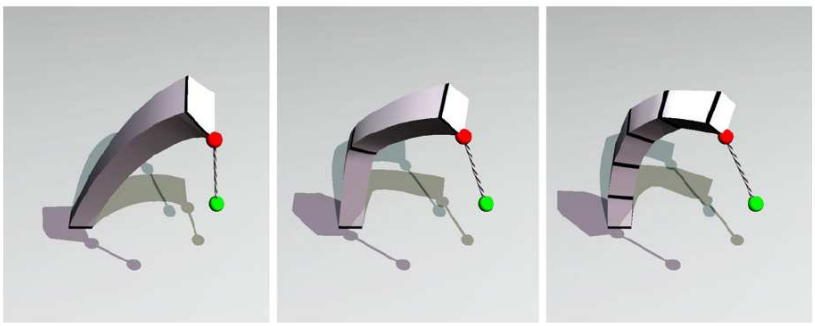

Fig. 5.7 : *La flexibilité de l'objet augmente avec l'emploi d'un plus grand nombre de clusters (extrait de [MHTG05]).*

La vitesse d'une particule se calcule à partir des positions par cluster g_i^c :

$$v_i = \sum_c \frac{\alpha_c g_i^c(t) - x_i(t)}{\sum_c \alpha_c h} \tag{5.6}$$

L'utilisation des clusters se traduit par une augmentation des déformations animables. Cependant, le paramétrage des clusters et la manière dont le mélange est effectué reste un aspect assez peu développé dans l'article initial. Un parallèle intéressant existe entre le mélange des clusters et les techniques de blending (utilisées lorsqu'on fait du Skinning 5.8). Comme nous avions déjà travaillé sur le sujet du Skinning dans [GM03], on établit quelques recommandations sur les propriétés de mélange entre les clusters :

- si les clusters ne se recouvrent pas assez, la mécanique est trop souple : les clusters tendent à évoluer de manière indépendante. Pour contourner ce problème, il faut qu'un nombre suffisant de particules soit partagé par les clusters ;
- une cassure assez nette est visible au niveau de la zone de mélange : pour lisser la transition entre les clusters, on utilise un facteur de pondération supplémentaire, à la manière des Splines (fig. 5.8). Ce poids va permettre le contrôle de la manière dont les clusters vont se mélanger.

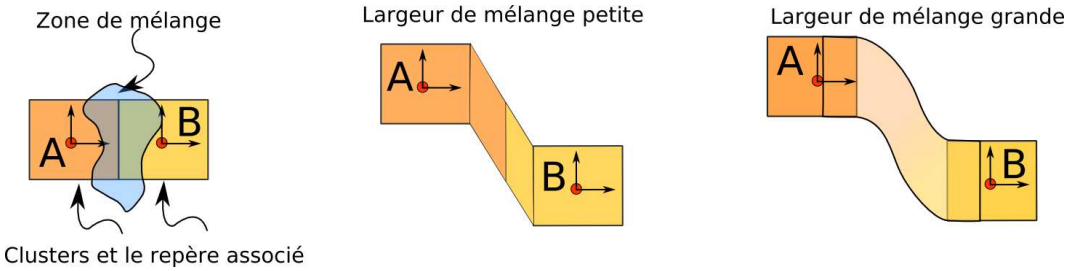

Fig. 5.8 : *Contrôle du mélange entre clusters par l'utilisation d'une fonction de poids qui évolue de manière continue dans la zone de mélange.*

5.4 Algorithmes de segmentation et découpe

La méthode du Shape-Matching repose essentiellement sur une structuration implicite sous la forme de groupes de clusters. Pour nous, ces clusters sont aussi intimement liés à la forme de l'objet et donc à sa découpe. Dans cette section, nous allons décrire les différents algorithmes envisageables pour générer les clusters à partir du maillage tétraédrique.

Dans l'article [MHTG05], les auteurs proposent un algorithme de clustering basé sur la proximité entre les particules. Le principe de l'algorithme est le suivant : l'objet est immergé dans une grille de voxel ; les nœuds appartenant à une même cellule de la grille sont regroupés en un cluster. Une telle segmentation donne des résultats acceptables, mais elle entraîne un couplage entre des particules non-forcément connectées[13] (fig. 5.9).

[13] Le couplage entre particules pour la découpe est discuté dans 3.4.

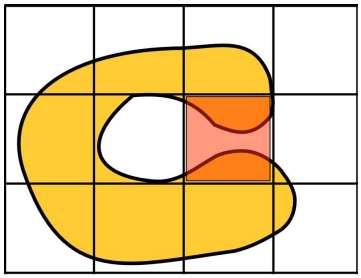

Fig. 5.9 : *La segmentation proposée dans [MHTG05] induit un couplage basé sur la distance entre des particules. Deux particules proches sont couplées, même si leur distance géodésique est importante. Ce peut être génant si les particules appartiennent à deux composantes non-connexes d'un même objet.*

Une segmentation basée sur la distance euclidienne ne tient pas compte de la forme de l'objet à simuler. Pour cela, nous choissons une autre méthode qui s'appuie sur une segmentation de l'objet en un ensemble de composantes convexes. L'avantage de cette approche est de définir clairement la relation qui lie la forme de l'objet et la segmentation obtenue (fig. 5.10) ; aux erreurs de calcul près, deux objets de forme identique seront segmentés de la même façon.

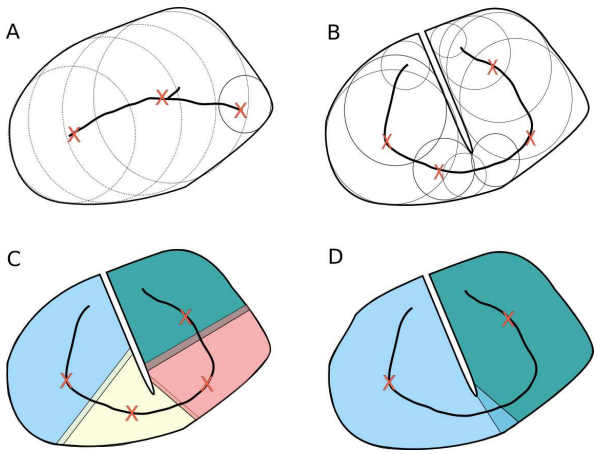

Fig. 5.10 : *La méthode de segmentation convexe permet d'associer de manière cohérente la segmentation et la forme de l'objet avant et après découpe. a) un objet est associé à son axe médian ; b) l'objet est découpé et l'axe médian est recalculé ; c) une nouvelle segmentation convexe est déduite à partir de l'axe médian ; d) lorsque l'algorithme produit trop de segments, ceux-ci sont fusionnés par un algorithme hiérarchique.*

5.4.1 Segmentation convexe par Fast-Marching

L'algorithme choisi s'appuie sur l'extraction de l'axe médian de l'objet. En général, le calcul d'axe médian, tout comme le calcul du squelette d'un objet, nécessite des algorithmes assez complexes et surtout coûteux. Cependant, le fait que les objets soient représentés par un maillage tétraédrique permet de développer un algorithme de segmentation qui est une adaptation de la technique de calcul de carte distance (introduite dans le chapitre (4)).

L'algorithme de segmentation convexe se compose de deux étapes. Dans la première, on calcule les nœuds du maillage tétraédrique qui appartiennent à l'axe médian. Dans la seconde, on associe à ces nœuds les particules de l'objet, en établissant le diagramme de Voronoï des nœuds de l'axe médian. Plus précisément :

- *l'axe médian :* l'axe médian d'un objet est défini comme le centre des sphères de rayon maximal. L'axe médian, version discrète du squelette de l'objet, est considéré comme un bon descripteur de la forme de l'objet.
- *le diagramme de Voronoï :* le diagramme de Voronoï d'un ensemble de primitives réalise une partition de l'espace en zone. Chaque zone associant à une primitive le domaine dans lequel elle est plus proche que toute autre primitive.

La formulation du diagramme de Voronoï et celle de l'axe médian s'appuient sur l'algorithme de calcul de distance ; nous pouvons donc réutiliser l'algorithme de Fast-Marching pour les calculer.

Extraction d'axe médian :

Le maillage tétraédrique de l'objet est assimilé à un graphe $G(V, E)$. Les nœuds du graphe sont associés aux sommets du maillage, tandis que les arêtes du graphe sont associées aux arêtes des tétraèdres composant le maillage. L'algorithme de calcul de distance propage itérativement l'information du bord le plus proche à ses voisins. Lorsque tous les voisins d'un nœud portent une distance inférieure à la valeur du nœud courant, un maximum local est atteint. Chaque maximum correspond à un centre de cluster.

Fonction `calculAxeMedian(`*G(V,E)*`)`

```
Données : Graph : G(V,E)
Résultat : Liste : L ; points formant l'axe médian
début
    pour v dans V faire
        si v.estBord alors
            tas.inserer(V) ;
    pour tas.nonvide faire
        bool estColine := false ;
        v := tas.prendreMin() ;
        pour n v E.voisins(v) faire
            estColine = estColine | non(metterAJourDistance( n )) ;
        si estColine alors
            L.ajouter( v ) ;
fin
```

Calcul du diagramme de Voronoï :

Une fois l'axe médian extrait, une segmentation convexe de l'ensemble des particules est obtenue en calculant le diagramme de Voronoï des centres des clusters. La même technique de propagation est utilisée, mais son sens est inversé ; on part de l'axe médian et on propage sur le maillage le point de l'axe médian le plus proche ; les différentes zones de Voronoï formant ainsi les clusters nécessaires au calcul des déformations. Les particules à la frontière entre deux clusters sont marquées ; à partir de ces particules, un nouvel algorithme de propagation est employé qui permet de définir les zones de mélange entre les clusters. La largeur de la zone de mélange est obtenue en paramétrant le nombre d'étapes de la phase de propagation : plus il y a d'étapes, plus la zone de mélange sera large.

5.4.2 Hiérarchie de segmentation

Cet algorithme de segmentation convexe génère un nombre trop élevé de clusters. C'est pourquoi nous utilisons un algorithme de fusion hiérarchique réduisant le nombre de clusters. Cet algorithme de fusion prend en paramètre un graphe topologique $G_2(V_2, E_2)$; les nœuds du graphe V_2 correspondent aux différents centres des clusters. Ces nœuds sont reliés par une arête s'ils sont des voisins, au sens du diagramme de Voronoï (ils partagent une frontière commune). Ce graphe $G_2(V_2, E_2)$, obtenu à partir de l'algorithme de segmentation convexe, est simplifié par fusion de ses arêtes ; fusions traduisant le regroupement de deux nœuds du graphe et donc de deux clusters voisins. En répétant le processus, on construit une hiérarchie de clusters (fig. 5.11).

Fonction `construireUnNiveau(` G_2`)`

Données : Graphe : $G_2(V_2, E_2)$;
Résultat : Une forêt d'arbres
début
  E_2.trier(nombre_de_particules) ;
  V_3 = fusionnerAretes(E_2) ;
  E_3 = construireAretes(E_2, G_2) ;
fin

Pour éviter la formation de hiérarchies mal formées, on utilise une heuristique d'équilibrage basée sur le nombre de particules contenues dans les clusters. Cette heuristique trie les arêtes en fonction du nombre de particules ; l'ordre de fusion privilégie alors la formation des petits clusters afin d'obtenir un effet d'homogénéisation. D'autres heuristiques plus adaptées, comme l'accroissement du volume englobant, sont envisageables mais leur calcul (nettement plus complexe) ne semblait pas pertinent dans un contexte de découpe temps-réel (fig. 5.12).

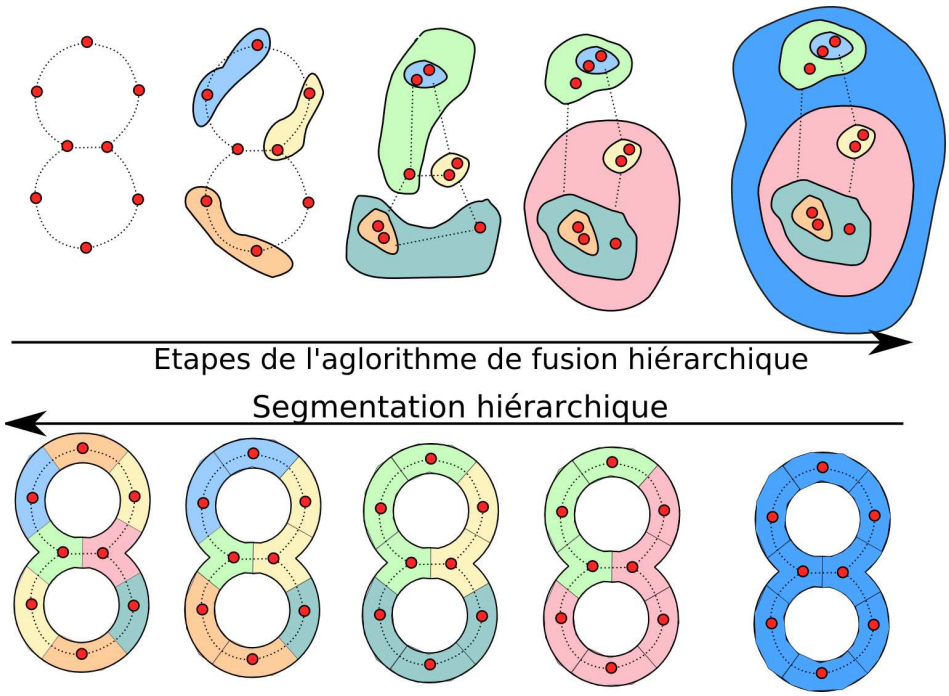

Fig. 5.11 : *Algorithmes de fusion hiérarchique et la segmentation de l'objet obtenue. Sur la ligne du haut, (de gauche à droite) on représente (par les zones colorées) les étapes de fusions successives. Sur la ligne du bas, on représente l'objet segmenté et les clusters correspondant aux étapes de fusion.*

5.4.3 Segmentation incrémentale : application à la découpe

L'algorithme de segmentation proposé ci-dessus se base sur un calcul rapide de la carte de distance. Les propriétés de localité de cette carte permettent de réduire la quantité de calcul quand elle doit être partiellement mise à jour.

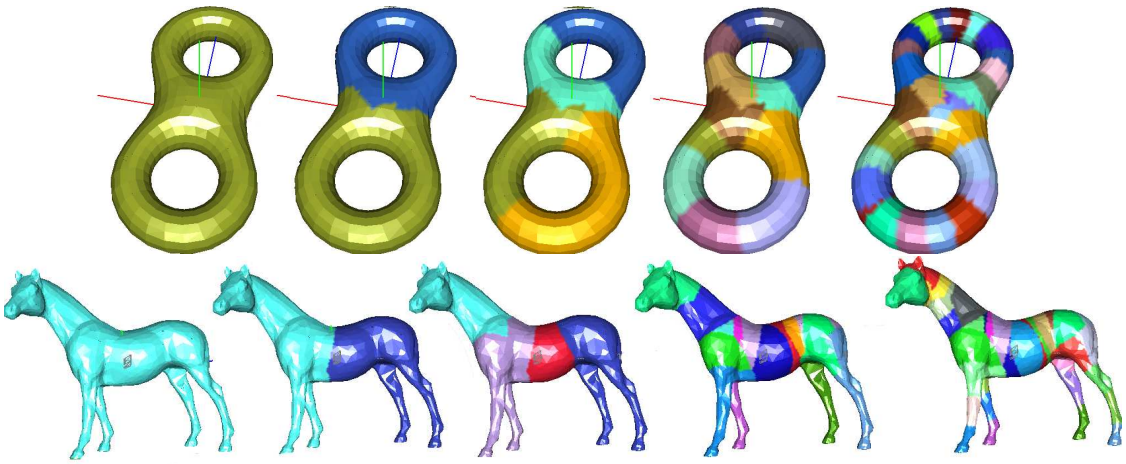

Fig. 5.12 : *Quelques exemples de hiérarchies de cluster produites par notre algorithme de segmentation.*

Propriétés de localité de la carte de distance

Contrairement à la carte de distance définie dans le chapitre précédant (recalculée à chaque déformation), la méthode de segmentation ici employée se contente d'une carte associée à l'état de repos de l'objet. Celle-ci peut être mise à jour facilement, car l'opération de découpe possède la propriété d'entraîner uniquement une diminution des distances : en quelque sorte, la découpe rapproche les bords de l'objet. En partant des bords nouvellement créés et en propageant l'information de distance à l'aide de l'algorithme du Fast-Marching, on peut mettre à jour la carte de manière locale : la propagation est stoppée lorsque les fronts ne permettent plus de faire décroître la distance au bord.

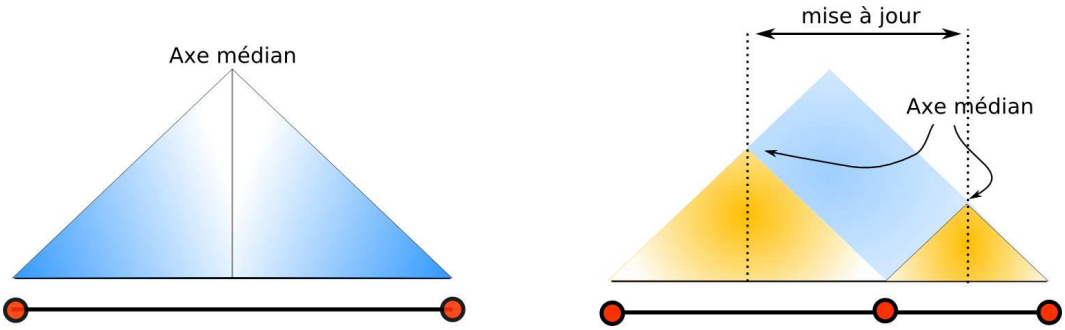

Fig. 5.13 : *Mise à jour de la carte de distance. A gauche, la carte de distance associée à un segment 1d. A droite, le segment est découpé et, en orange, les nouvelles cartes de distance.*

La figure (5.13) présente sur un exemple 1d, la mise à jour de la carte de distance. Sur cette figure, on voit que la moitié de la carte doit être mise à jour. Cette propriété nous permet d'établir, qu'au pire des cas, la mise à jour de la carte est d'une complexité en $\frac{O(n.logn)}{2}$, puisque son calcul complet (par Fast-Marching) est d'une complexité en $O(n.logn)$. Cependant, cette borne est rarement atteinte en pratique.

Extraction d'axe médian revisité

L'algorithme d'extraction est modifié de manière à bénéficier de la propriété de localité de calcul de la carte de distance. Les nœuds du maillage situés sur l'axe médian sont remplacés pendant la propagation du nouveau front de distance. Pendant cette étape, de nouveaux nœuds sont marqués comme faisant partie de l'axe médian et stockés dans une liste adéquate.

Calcul du diagramme de Voronoï revisité

Le calcul incrémental du diagramme de Voronoï est un peu plus complexe que celui de l'axe médian ; mais, comme la mise à jour de l'axe médian, il s'appuie sur une propriété de localité du diagramme (représentée sur la figure 5.14). Les modifications apportées à une cellule, pendant une opération de découpe, ne changent que localement le diagramme (elles ne s'appliquent qu'aux voisins immédiats). Pendant la phase d'extraction de l'axe médian, les nouveaux centres de la segmentation sont recensés, ainsi que l'ensemble de sommets voisins.

A partir de cette nouvelle liste, nous effectuons la mise à jour du diagramme de Voronoï. Un fois l'algorithme terminé, on dispose de deux listes de groupes de particules. Certaines sont associées à des groupes de particules qui n'ont pas changé, d'autres sont associés aux groupes créés ou modifiés. Ces deux listes sont utilisées pour mettre à jour la hiérarchie des clusters.

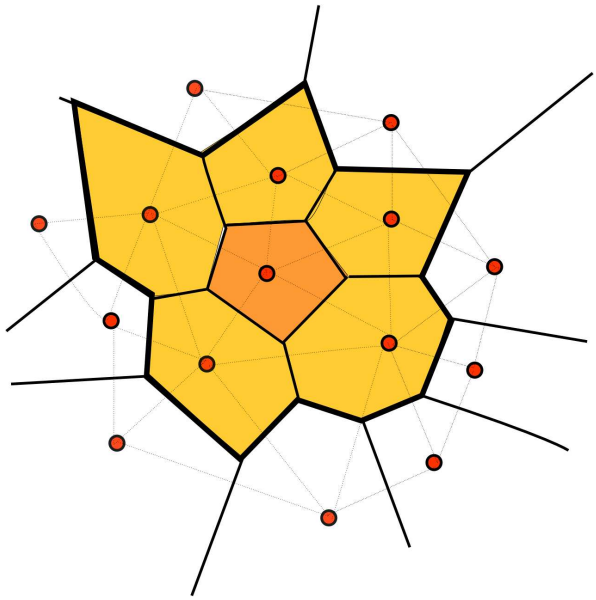

Fig. 5.14 : *Lorsqu'un nœud de l'axe médian est modifié, seuls ses voisins de Voronoï doivent être recalculés.*

Calcul revistié de la hiérarchie

La mise à jour incrémentale de la hiérarchie soulève un certain nombre d'interrogations. En effet, de part son algorithme de construction, elle tend à globaliser l'influence des modifications. Un changement local de la segmentation tend à se propager de niveau en niveau, et nécessite la mise à jour des données associées à chaque niveau ; cela tend à limiter l'intérêt d'un calcul incrémental pour la construction de la hiérarchie. Dans la suite du chapitre, nous allons donc recalculer complètement les hiérarchies de segment.

5.5 Expérimentations et résultats

Pour évaluer l'ensemble de l'approche, nous en avons réalisé l'implémentation. Le traitement de la découpe effectué au niveau du maillage tétraédrique de l'objet est suivi d'une phase de segmentation qui convertit la géométrie de l'objet en groupes de simulations utilisables par la méthode du Fast-Marching.

5.5.1 Intérêt de la segmentation par axe médian

On voit sur la figure (5.15) le rôle de la segmentation dans le processus de découpe des objets, un objet découpé (a gauche) et le résultat obtenu en utilisant différents niveau de segmentation. Le premier niveau revient à allouer un groupe de simulation par composante non-connexe de l'objet découpé. Cela ne permet pas de traduire correctement la découpe partielle de l'objet. Les niveaux inférieurs font apparaître celle-ci. Suivant le réalisme désiré et la rigidité des objets, nous sommes amenés à utiliser différents niveaux de la hiérarchie de segmentation. Nous pouvons aussi observer que la simulation au niveau de la racine des hiérarchies est suffisante lorsque les objets sont rigides. Dans nos expérimentations, nous nous sommes en général limités aux quatre niveaux suivant la racine ; l'augmentation du nombre de cluster se traduisant par une augmentation du temps de calcul. Enfin, la selection automatique du niveau de la hiérarchie à utiliser reste un problème ouvert.

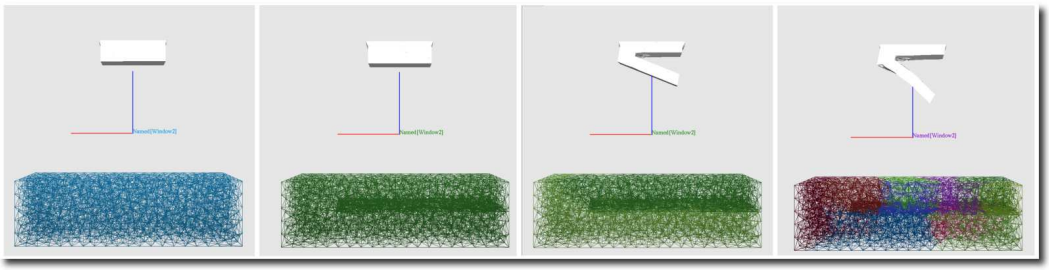

Fig. 5.15 : *Comparaison entre différents niveaux de segmentation.*

Découpe d'objets déformables

Nous avons implémenté l'ensemble des algorithmes décrits dans ce chapitre, et réalisé une série de mesures de performance pour évaluer la pertinence de l'approche. Pour cela, nous avons appliqué un mouvement de découpe prédéfini à un ensemble d'objets déformables de forme et de résolution différentes (fig. 5.16). Nous avons mesuré les temps de calcul correspondant aux différentes parties de la méthode présentée (fig. 5.17).

Temps de simulation (Sim) : mesure le temps du calcul d'un pas d'intégration temporelle en employant la méthode du Shape-Matching. Dans nos expériences, les différents objets sont animés à un pas de temps de 10 ms. On observe que les objets de moins de 30000 tétraèdres sont simulables en temps réel.

Temps de découpe (Cut) : le temps de découpe correspond au temps mis pour détecter les intersections entre le maillage tétraédrique et l'outil. Les benchmarks montrent qu'à cet endroit se consomme plus de la moitié du temps de calcul d'une opération de découpe. La majorité du temps est consommée par l'opération de calcul des intersections. Comme certains calculs sont redondants, nous envisageons d'en faire une implémentation plus efficace ; l'accélération d'un facteur 3 ou 4 ne semble pas hors de portée.

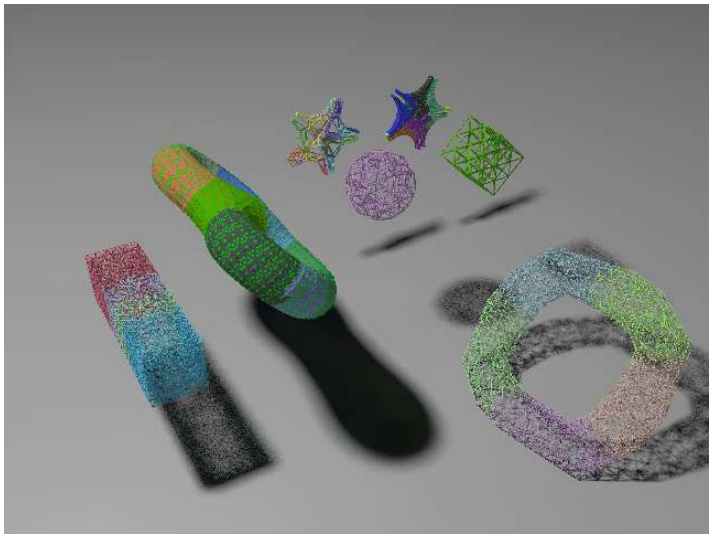

Fig. 5.16 : *Différents objets utilisés dans nos benchmarks*

Temps de mise à jour des matrices de déformation (Def) : la méthode du Shape-Matching permet de précalculer un certain nombre de matrices employé pour l'extraction de la composante rigide ou linéaire du mouvement. Ces précalculs sont liés à la segmentation et ils doivent donc être mis à jour pendant la découpe. La mise à jour de ces matrices est coûteuse ; c'est pourquoi une version incrémentale du calcul des matrices pourrait être bénéfique ; néanmoins, cela suppose la mise en place d'un algorithme de construction des hiérarchiques qui soit lui même incrémental.

Temps de segmentation (Seg/updt) : nous avons mesuré le temps nécessaire pour refaire la segmentation de l'objet à l'aide de l'algorithme de segmentation (Seg) et de sa version incrémentale (updt) : l'algorithme incrémental diminue considérablement le temps de calcul.

Object(#Tets)	Sim	Cut	Seg/updt	Def	#CutTets
cube(336)	0.1	2.3	0.5	0.3	77/279
piece1(333)	0.1	1.5	0.6	0.7	39/143
barre1(1160)	0.2	6	1	1.3	147/647
barre2(7000)	0.9	30	7/2	7	747/2638
torus(7000)	1.0	15	4/1	3	341/1185
horse(30000)	12	90	42/7	40	1239/4374
piece2(40000)	20	180	60/10	70	1360/4917

Fig. 5.17 : *Temps de calcul en millisecondes pour les différentes parties de la boucle de simulation. (Sim) correspond à la méthode du Shape-Matching ; (Cut) à la découpe du maillage tétraédrique ; (Seg/Updt) au temps mis par les algorithmes de segmentation (incrémental/non-incrémental) ; (Def) correspond à la mise à jour des matrices précalculées, utilisées par le Shape-Matching.*

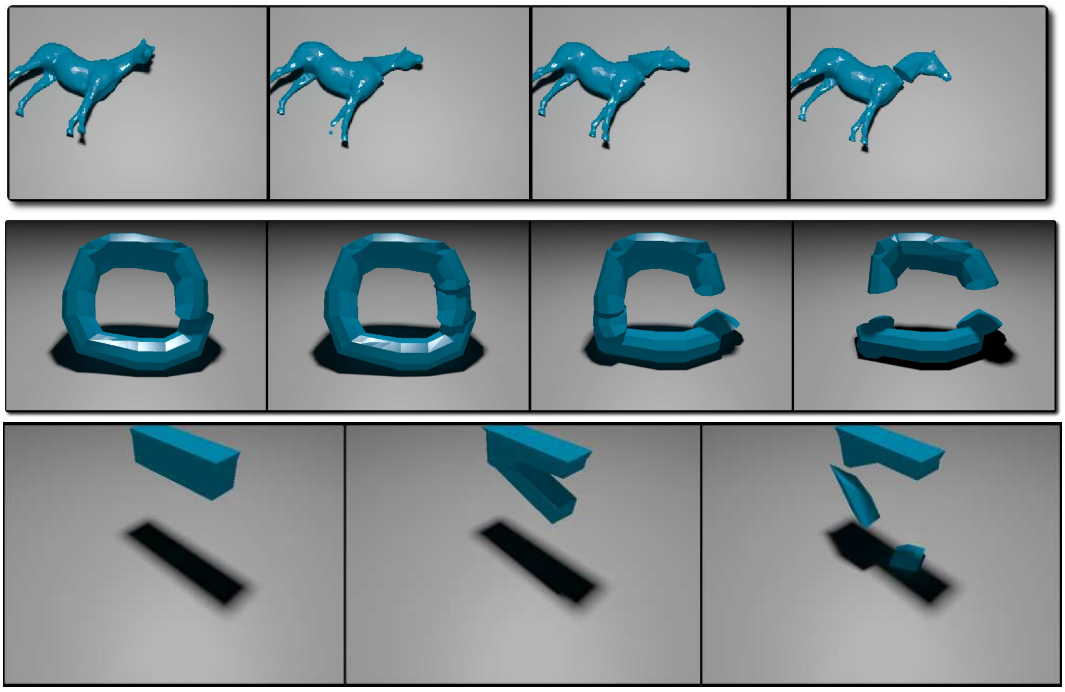

Fig. 5.18 : *Illustration des différents benchmarks de découpe*

Gestion des intersections

Pour accélérer la détection des intersections, nous aurions pu employer une approche par grille de voxel (comme dans le chapitre précédant). Nous avons cependant préféré expérimenter une autre approche à base de hiérarchie de boîte englobante (AABB). Chaque groupe de simulation est associé à une boîte englobante, la hiérarchisation des groupes de segmentation fournissant une hiérarchie de boîte englobante. L'intérêt de cette approche est que cette hiérarchie est alors mise à jour de manière implicite (automatique), pendant les phases de découpe des objets. Un second point intéressant est qu'il est possible de simplifier le problème des auto-collisions (très coûteux en temps de calcul). Pour cela, on fait l'hypothèse qu'il n'est pas nécessaire de tester les auto-collisions entre toutes les paires de tétraèdres composant l'objet mais uniquement entre les tétraèdres qui appartiennent à des groupes différents de simulations.

Conclusion et perspectives

Dans ce chapitre, un modèle d'objets découpables est proposé. Ce modèle est simulé de manière stable en temps réel et s'inspire du principe de multireprésentation. Par rapport aux approches précédentes, nous avons ajouté un aspect « topologique » fort qui structure le principe de découpe et la mise à jour des autres aspects tels : l'affichage, la collision et l'animation. Ce travail constitue aussi une première tentative d'utilisation de la méthode du Shape-Matching pour traiter des objets découpables. Les particularités de cette méthode d'animation ouvrent la voie au développement

d'une technique de segmentation rapide. Celle-ci repose sur le calcul de la carte par l'algorithme du Fast-Marching dont l'efficacité permet la re-segmentation de l'objet pendant les découpes. Une vidéo illustrant les résultats de cette méthode est disponible à http://www.lifl.fr/~marchal/these/video/chap5_decoupe_sca06.avi.

Parmi les points non-encore éclaircis, on peut noter :

- les expérimentations nous ont montré qu'il était difficile de contrôler la manière dont les groupes de cluster se mélangent. Peut-on avoir un meilleur contrôle ?
- est-il possible de mettre en place un système de sélection automatique du niveau de hiérarchie basé sur le comportement mécanique de l'objet ?
- il faut réaliser une étude plus détaillée de la continuité de l'animation. En effet, pour l'instant, les découpes et le changement de niveau de la hiérarchie simulée se traduisent par des discontinuités problématiques dans le comportement mécanique de l'objet.
- pour l'instant, seule la mise à jour de la carte de distance est incrémentale. Peut-on implémenter un calcul qui soit complètement incrémental, notamment au niveau du calcul des hiérarchies de cluster et de la mise à jour des matrices mécaniques ?
- nous avons remarqué récemment que le comportement de l'algorithme pour la segmentation d'objets peu discrétisés n'est pas prévisible. Par exemple, l'extraction de l'axe médian d'un tétraèdre unique n'est pas possible, l'algorithme sélectionnant un noeud du bord comme élément de l'axe. Cependant, dans le cadre de nos expérimentations sur les petits maillages, nous avons remarqué que la phase de création de la hiérarchie compense le problème. Nous envisageons de corriger le défaut en utilisant le centre des tétraèdres comme un noeud du maillage supplémentaire.

Conclusions et perspectives

Pour de nombreux auteurs dont nous avons consulté les travaux afin de rédiger cette thèse, la démarche semble suivre un seul fil conducteur : à partir d'une question initialement posée, une démarche inductive est suivie.

C'est avec une approche nettement plus divergente que nous avons abordé notre travail. Pour cela, nous avons cherché à établir des ponts entre les différentes thématiques (mécanique, modélisation géométrique...), les idées, les concepts. Grâce à cette approche, nous avons tenté, avec des échecs, mais aussi des réussites, d'avoir une vision un peu plus générale des liens entre quelques-uns des différents domaines d'intérêt de l'informatique graphique : simulation et animation, modélisation géométrique, interpolation, géométrie algorithmique.

Ainsi, dans le second chapitre, nous avons défendu une vision particulière d'objets simulés pour les environnements virtuels. Cette vision s'inspire des approches qualifiées de multireprésentation à laquelle s'est ajoutée une représentation géométrique-maîtresse, fortement liée à la structuration topologique des objets.

Les trois derniers chapitres de la présente thèse concernent des résultats plus concrets, publiés, ou en cours de publication ; résultats correspondant aux différentes expérimentations et développements menés.

Les résultats du chapitre (3) concernent le développement d'un modèle particulier d'objet multireprésentation adaptatif. Le modèle présenté permet de simuler facilement des objets (visuellement) très complexes ; cette comlpexité évolue pendant la simulation en fonction de critères comme le temps de calcul. Le contenu de ce chapitre a été publié dans [DMG05] sous une forme un peu différente, car présentant aussi certains travaux propres à Jérémie Dequidt (Oracle/code de Morton).

Les résutats du chapitre (4) portent sur l'utilisation d'un algorithme de Fast-Marching qui ne nécessite pas de grille de voxel. On utilise cet algorithme pour calculer la carte de distance d'objet volumique tétraédrique. La méthode est suffisamment efficace pour être utilisable en temps réel dans le cadre d'un pipeline de collision supportant les objets découpables. Au niveau de la gestion des collisions, on propose aussi une technique de calcul de réponse, dite de "segment dans un volume", plus précise que la réponse "point dans un volume" qui préexistait. Le contenu de ce chapitre a été publié dans [MAC04].

La troisième contribution, présentée dans le chapitre (5), est un modèle d'objet découpable. Ce modèle s'appuie sur une méthode d'animation sans maillage qui garantit sa stabilité pendant la phase d'intégration numérique. Pendant la découpe, les données associées à la mécanique doivent être recalculées ; on utilise pour cela un algorithme de segmentation suffisamment rapide pour être exécuté en temps réel. Cette contribution a été présentée sous la forme d'un poster dans [MG06].

La découpe d'objets déformables est loin d'être un sujet clos ; voici quelques-uns des différents travaux que nous aimerions entreprendre.

Les éléments décrits dans le chapitre (2), gagneraient à être mieux formalisés. Approfondir les idées qui y sont présentées permettrait peut-être d'ouvrir la voie vers de futurs résultats. Une formalisation rendrait aussi plus facile la mise en perspective du contenu de ce chapitre avec des travaux déjà existants ; nous pensons notamment aux G-Cartes [Dis] et à la géométrie différentielle discrète [DKT] qui pourraient trouver des applications à la simulation temps-réel d'objets déformables et découpables. Il nous semble aussi intéressant de continuer l'étude de l'interpolation MLS présentée dans les annexes : essayer de la généraliser sous la forme d'une technique permettant de calculer les valeurs (et les dérivées) d'une fonction interpolée sur un domaine géométrique, tout en laissant la possibilité de fixer certaines de ces valeurs et dérivées. Cet objectif rejoint d'ailleurs le problème HOPI (High Order Interpolation) introduit, pour le cas 2d, dans un récent article [XH06].

Au niveau de la découpe d'objets déformables, on peut noter que pour de faibles résolutions de maillage, l'algorithme de segmentation ne permet pas le calcul correct de l'axe médian (cas d'un tétraèdre unique par exemple). Nous envisageons une autre approche, utilisant les centres des tétraèdres comme nœuds supplémentaires dans le graphe. En outre, pour que le modèle de découpe soit pleinement utilisable, il faut qu'on ait un meilleur contrôle du mélange entre les différents clusters. En plus, le modèle de découpe présenté s'appuie sur la méthode du Shape-Matching dont les paramètres ne sont pas identifiables. En vue d'une utilisation dans un contexte médical, il nous semble important que le modèle mécanique soit prédictif et que l'identification des paramètres soit réalisable. Une méthode sans maillage basée sur la mécanique des milieux continus nous semble donc une voie intéressante (voir aussi l'article [SOG06] récement présenté). Toujours dans un cadre médical, on peut noter que la découpe géométrique ne traduit pas les interactions complexes de contact qui apparaissent entre l'outil et l'objet découpé. Cela peut poser problème si l'objectif est la formation de médecins. La gestion des contacts reste d'ailleurs encore un problème ouvert, notamment la gestion des collisions et auto-collisions d'objets découpables qui est une des étapes les plus coûteuse de la simulation. Un point particulier que nous sommes en train d'étudier est la remise à jour de la topologie d'une hiérarchie de volumes englobants pendant la découpe, l'approche envisagée s'appuie sur l'utilisation d'une structure de données dite « dynamique » mise à jour incrémentalement pendant la simulation.

Annexe A

Interpolation aux moindres carrés mobiles

Dans cette annexes, nous décrivons nos expériences portant sur la méthode de l'approximation aux moindres carrés mobiles (MLS) et nous faisons une proposition de construction d'une fonction réalisant une interpolation. Nous présentons ce travail dans le cadre d'une annexe car, bien qu'il nous semble intéressant (notamment en lien avec [XH06]), il n'est pas encore suffisamment mature pour être intégré comme une contribution à part entière.

La méthode MLS permet de résoudre le problème de l'approximation d'une fonction inconnue à partir d'un ensemble d'échantillons, problème déjà évoqué dans cette thèse (chapitres 2, 3, 5). Plus formellement, le problème de l'approximation consiste à construire une fonction $\tilde{u}(x)$, à partir d'un ensemble de valeurs connues $U = \{u_0 = u(x_0), \ldots, u_n(x_n)\}$ de la fonction inconnue $u(x)$, aux points d'échantillonage $N = \{x_0, \ldots, x_n\}$.

$$\tilde{u}(x) \approx u(x) \tag{A.1}$$

La technique de l'approximation aux moindres carrés mobiles permet la construction d'une telle fonction mais celle-ci ne passe pas exactement par les points de l'échantillon : elle ne réalise pas une interpolation. L'absence d'interpolation peut poser problème, par exemple pour imposer des conditions aux limites dans le cadre de la mécanique sans maillage. Nous avons donc cherché, à partir de la méthode MLS, à construire une telle fonction d'interpolation.

L'annexe est découpée comme suit : on présente l'approximation aux moindres carrés au travers d'illustrations de l'influence des différents paramètres. Ensuite, on s'intéresse à la manière de construire une fonction interpolative. Les expérimentations menées ont été réalisées en 1d, la fonction $\tilde{(u)}(x)$ étant comparée à une fonction $cos(x)$ échantillonnée (espacement de 0.5 unités entre deux échantillons), qui permet de visualiser le fonctionnement des différentes méthodes.

Approximation par la méthode des moindres carrés mobiles

La méthode des moindres carrés mobiles (Moving Least Square, MLS) dérive de la méthode des moindres carrés approchant une fonction inconnue par un polynôme. La méthode MLS s'en différencie en pondérant l'influence des échantillons dans le calcul de l'approximation. En informatique graphique, l'influence de l'échantillon est en général fonction de la distance entre l'échantillon et le point d'évaluation. Plus on est proche d'un échantillon, plus son influence augmente ; cette fonction est appelée fonction de poids w_i.

On reprend l'expression de la fonction d'interpolation donnée dans le chapitre (2).

$$\begin{aligned}\tilde{u}^{mls}(x) &= \sum_{i=0}^{n} \phi(x).u_i \\ \phi(x) &= c^T(x).b(x_i).w_i(x) \\ c(x) &= A^{-1}(x).b(x) \\ A(x) &= w_i(x).b(x_i).b(x_i)^T\end{aligned}$$

Quand on veut faire de l'interpolation aux moindre carrés mobiles, plusieurs paramètres sont à définir. Le premier étant la base $b(x)$ du polynôme d'approximation, le second, le choix de la fonction de poids $w_i(x)$.

Le choix de la base est déterminant, car il conditionne directement la précision de l'approximation et les temps de calcul. Plus la base est complexe (constante, linéaire, quadratique) plus la taille de la matrice A augmente et son inversion devient coûteuse. On peut noter que la base constante $b(x) = [1]$ (appelée interpolation de Sheppard) est la plus avantageuse en calcul :

$$\tilde{u}^{Sheppard}(x) = c(x) = \sum_i \frac{w_i(x).u_i}{\sum_i w_i(x)} \tag{A.2}$$

Cependant, elle ne permet pas l'interpolation d'une fonction linéaire, ce qui est illustré sur la figure (A.1), à chaque échantillon la fonction faisant un "plateau". Sur cette figure, on illustre aussi l'influence sur $\tilde{u}^{Sheppard}(x)$ du paramétrage de la fonction de poids (voir suite).

Le second paramètre sur lequel nous pouvons jouer est la fonction de poids w_i. Cette fonction, qui vient pondérer l'influence des échantillons, est basée sur une primitive sphérique (comme les MetaBalls). On note r_i le rayon de la sphère d'influence autour de l'échantillon i. Pour obtenir de bons résultats :

- la fonction $w(x)$ doit être continue et tendre vers 0.
- si la fonction est compacte, si elle vaut 0 au-delà d'un certain rayon r_i, alors l'approximation est locale (et A peut être une matrice creuse).

Sur la figure (A.2), on utilise la base linéaire $b = [1, x]$ et une telle fonction de poids. En faisant varier le rayon du support r_i on remarque que :

- un rayon $r_i < 0.5$ n'est pas utilisable, car la matrice A n'est alors pas inversible (au contraire de l'interpolation de Sheppard) ;
- un rayon $r_i = 0.5$ revient à réaliser une interpolation linéaire entre les échantillons ;
- les rayons $r_i > 0.5$ lissent la fonction qui ne réalise alors plus une interpolation.

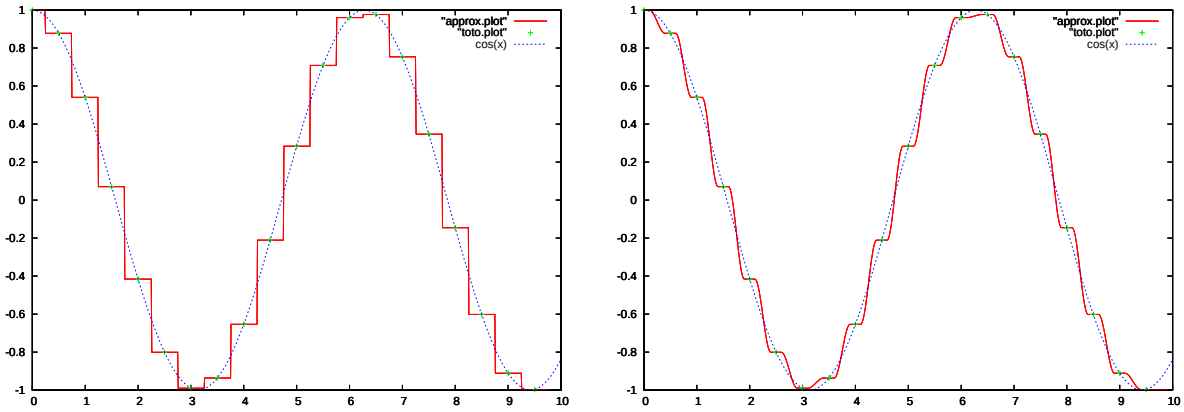

Fig. A.1 : *La fonction $cos(x)$ est tracée en bleu et comparée à l'approximation obtenue par la méthode Sheppard (MLS d'ordre 1). Nous faisons varier le rayon d'influence de la fonction de poids $(0.25, 0.5)$. Nous observons que la méthode de Sheppard réalise une interpolation pour des rayons compris dans l'intervalle $[0.25, 0.5]$. Lorsque le rayon dépasse 0.5, la méthode réalise alors une approximation.*

On pourrait être tenté de privilégier des fonctions de base plus complexes (quadratique : $b(x) = [1, x, x^2, ...]$). Cependant, la taille de la base influe sur les temps de calcul, puisqu'elle fait croître les dimensions de la matrice A. De même, la taille de la matrice A change suivant qu'on interpole les données en 1d, 2d ou 3d (par exemple en 2d on, les fonctions de base sont : linéaire : $b(x, y) = [1, x, y]$, quadratique : $b(x, y) = [1, x, y, x^2, y^2, xy]$). En outre, l'augmentation des dimensions de A, suppose qu'un plus grand nombre d'échantillons non nul soit évalué en x pour que cette matrice soit inversible. Cela implique que pour un ensemble d'échantillons donné, il faut que suffisamment de sphères d'influence s'intersectent. Or, plus les sphères sont grandes, plus le signal est lissé et s'éloigne d'une fonction d'interpolation.

De l'approximation à l'interpolation

Dans la mesure où l'interplation semble être une propriété intéressante (pour placer des conditions aux limites par exemple). Nous avons donc cherché à obtenir une fonction $\tilde{u}(x)$ qui soit interpolante au sens où elle passerait par les échantillons :

$$\forall_i u_i = \tilde{u}(x_i) \tag{A.3}$$

Une première approche utilise des fonctions de poids particulières qui vont forcer l'interpolation à passer par les échantillons. Cette approche ne nous satisfaisait pas ; il nous fallait expérimenter d'autres manières de "forcer" l'interpolation.

Interpolation par fonctions de poids singulières

On peut observer que la fonction est interpolante lorsqu'au niveau d'un échantillon, l'influence dudit échantillon devient prépondérante par rapport aux autres (par exemple : $w_i(x_i) = 1$ et $w_j(x_i) = 0$ avec $j \neq i$). Une première proposition pour rendre la méthode MLS interpolante consiste donc à utiliser une fonction de poids qui tend vers $+inf$, quand on

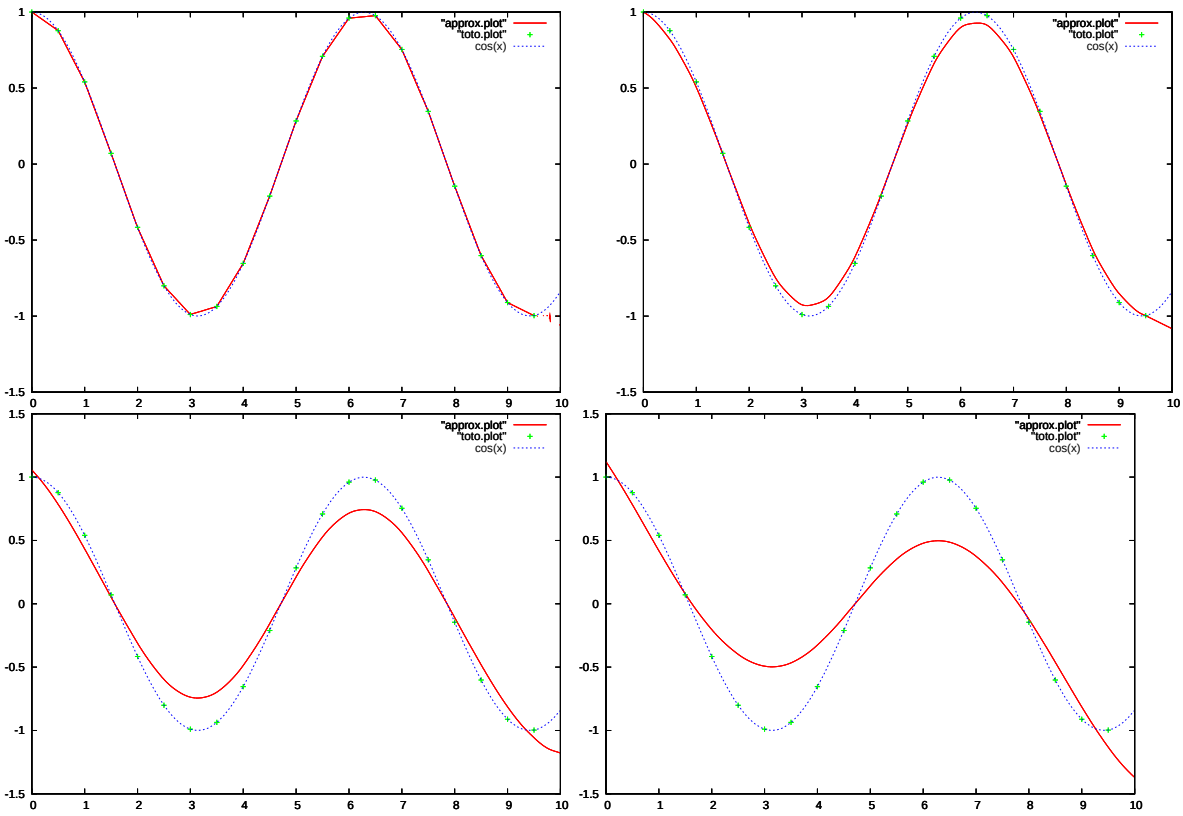

Fig. A.2 : *La même fonction cos(x) tracée en bleu est comparée à ses approximations utilisant la fonction de base linaire $b(x) = [1, x]$. Nous faisons varier le rayon des zones de mélange de (0.5, 1.0) et (2.0 3.0) ; on note que $r_i < 0.5$ n'est pas défini, car la matrice A n'est alors plus inversible. Avec $r_i = 0.5$, on a une interpolation linéaire, $r_i > 0.5$, approximation de plus en plus "lisse".*

s'approche d'un échantillon. C'est la méthode employée dans [SOS04]. L'approximation est alors contrainte de passer par les échantillons. Cependant, on voit sur la figure (A.3), que le fait d'utiliser des fonctions de poids qui tendent vers $+inf$ (singulières), entraîne l'apparition de singularités dans la fonction d'interpolation. Ces singularités rendent plus délicate l'utilisation de $\tilde{u}(x)$, si on a besoin d'évaluer ses dérivées (pour faire de la simulation mécanique, du rendu évolué,...).

Interpolation hybride

La méthode d'interpolation que nous introduisons maintenant est basée sur le constat suivant : l'interpolation de Sheppard est peu coûteuse en temps de calcul et très pratique, si on désire faire de l'interpolation. En effet, comme la base est de taille un, il suffit qu'une seule des pondérations soit non nulle pour que la matrice A soit inversible. Avec un seul échantillon, il est ainsi plus facile d'exprimer une contrainte d'interpolation en contrôlant le rayon des sphères d'interpolation.

La proposition consiste à utiliser l'interpolation de Sheppard non plus pour mélanger des constantes u_i, mais pour mélanger des fonctions $u_i^{Taylor}(x)$ passant par les échantillons :

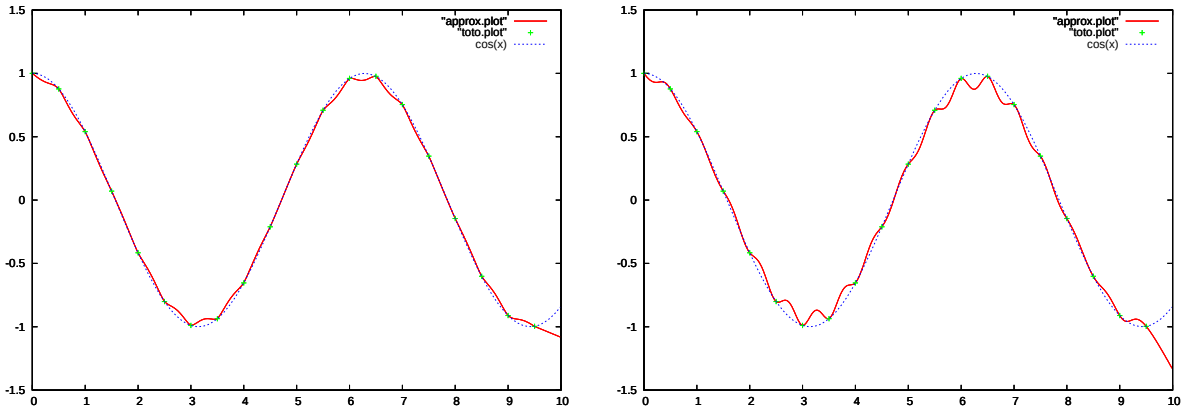

Fig. A.3 : *On utilise des fonctions de poids singulières. La fonction $\tilde{u}(x)$ est systématiquement interpolante quels que soient les rayons utilisés (image de gauche $r_i = 1.0$ et de droite $r_i = 3.0$).*

$$\tilde{u}^{Hybrid}(x) = \frac{\sum_i w_i(x).\tilde{u}_i^{Taylor}(x)}{\sum_j w_j(x)}$$

Pour garantir que les fonctions mélangées passent bien par les échantillons, on utilise le développement de Taylor qui nous indique comment approcher une fonction au voisinage d'un point x_i :

$$\tilde{u}_i^{Taylor}(x) = u_i + \nabla\tilde{u}_i(x_i) * ||x - x_i||$$

Les valeurs u_i sont connues ; ce sont nos échantillons. Par contre, les dérivées $(\nabla\tilde{u}_i(x_i))$ de la fonction sont inconnues. Nous les approchons donc à l'aide d'une interpolation MLS avec une base plus complète. Ainsi, avec un développement de Taylor à l'ordre 1, on utilise la fonction :

$$\nabla\tilde{u}_i(x_i) = \frac{\delta\tilde{u}_i^{Linear}(x_i)}{\delta x}$$

Cette fonction $\tilde{u}_i^{Linear}(x_i)$ étant calculée comme une approximation MLS avec la base linéaire $b(x) = [1, x]$. Pour être calculée, il faut utiliser un second jeu de rayons r_i^l de manière à garantir qu'au moins deux sphères s'intersectent aux positions où est évaluée cette approximation.

Pour que la fonction que nous présentons $\tilde{u}^{Hybrid}(x)$ réalise une interpolation, il faut que le rayon r_i^s ne recouvre pas les échantillons voisins. Plus formellement, on a :

$$\forall x_j, x_i || x_i x_j || >= r_i^s$$

La figure A.4 montre le résultat de l'interpolation obtenue avec la méthode hybride. Les différentes figures représentent les résultats obtenus en faisant varier le paramètre r_i^s entre la valeur minimale pour assurer la couverture du domaine (0.25), et la valeur maximale pour que la contrainte d'interpolation soit respectée (0.5). On constate que quelle que soit la valeur de r_i^s, la méthode hybride donne un résultat nettement amélioré par rapport à la méthode de Sheppard simple (fig. A.1). Sur la figure (A.5), on illustre l'influence du choix du paramètre r_i^l qui définit le rayon d'influence utilisé pour le calcul du gradient de la fonction.

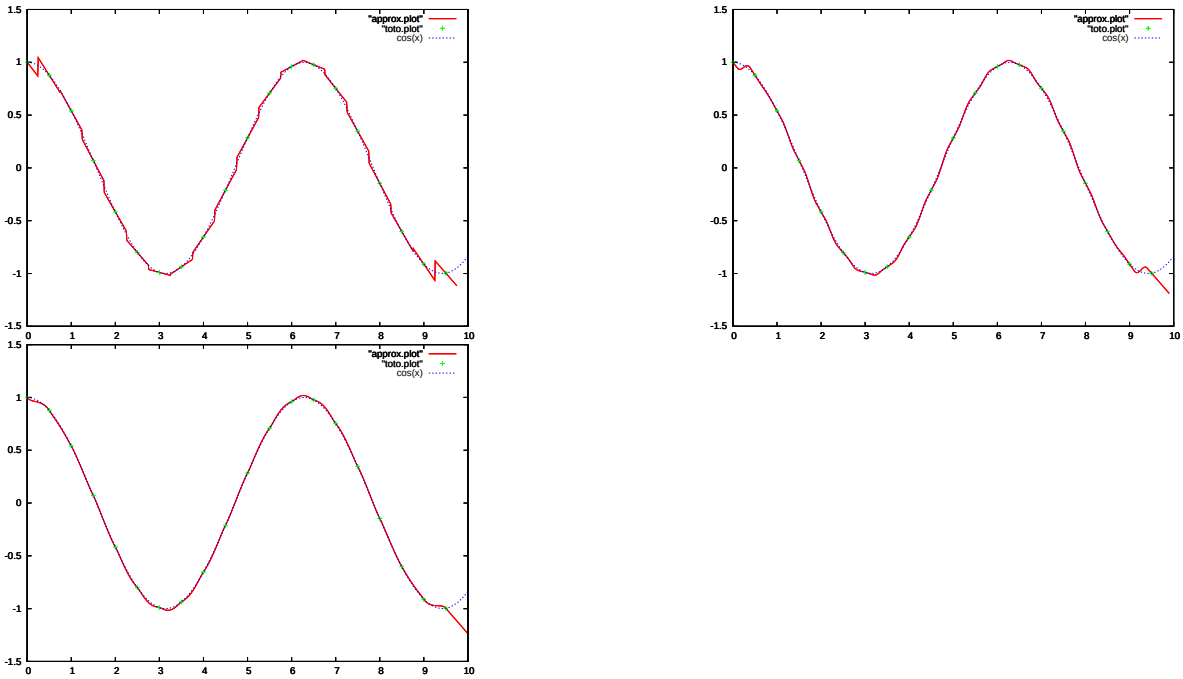

Fig. A.4 : *La fonction $cos(x)$ (en bleu) est comparée avec l'interpolation obtenue par notre méthode hybride. On donne au rayon r_i^s les valeurs suivantes : 0.25 (intersection des supports), 0.4 et 0.5.*

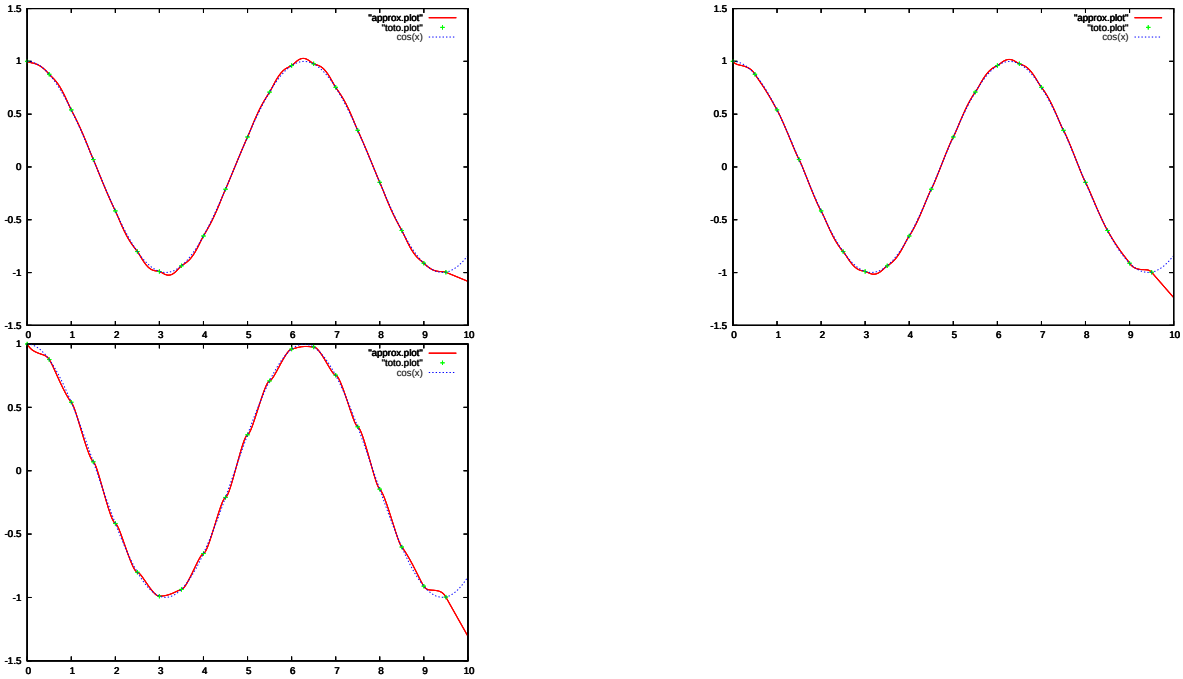

Fig. A.5 : *La fonction $cos(x)$ (en bleu) est comparée avec l'interpolation obtenue par notre méthode hybride. On donne au rayon r_i^l les valeurs suivantes : 1.0, 3.0, 4.0.*

Généralisation 2d, 3d

Il est décisif, pour que la méthode hybride réalise une interpolation, que la contrainte $\forall x_j, x_i || x_i x_j || >= r_i^s$ soit vérifiée ; c'est le cas pour les agencements suivants :

- en 1d, la contrainte peut toujours être respectée à la manière des paramétrisations splines.
- en 2d, la contrainte est respectée si on place les échantillons sur les sommets d'un triangle équilatéral ou au sommet d'une cellule carrée. Un maillage composé d'éléments identiques permet de respecter la contrainte.
- en 3d, la contrainte est respectée si on place les échantillons sur les sommets d'un triangle, ou aux sommets d'un cube. Un maillage composé d'éléments identiques permet de respecter la contrainte.

En 2d et 3d, la contrainte d'utiliser un maillage régulier peut être assouplie ; on peut aussi employer des maillages obtenus par subdivision régulière, si on contraint deux cellules voisines à avoir, au plus, un niveau de différence(fig. A.6).

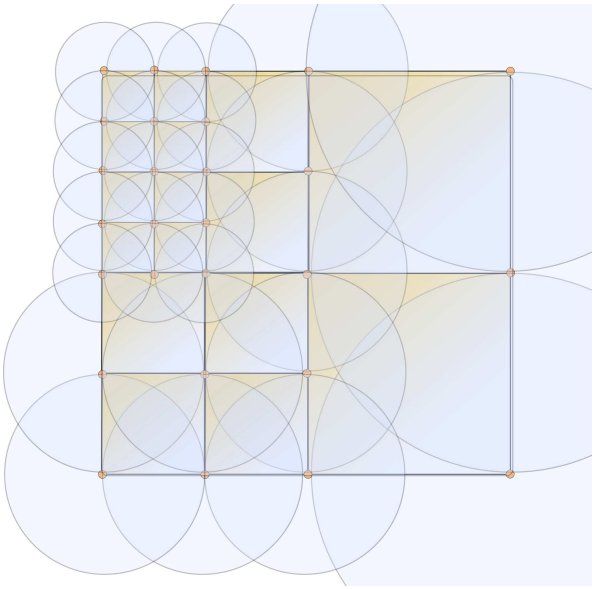

Fig. A.6 : *Placement des sphères d'interpolation sur une grille contrainte. Le domaine d'interpolation est complètement couvert et une seule sphère influe sur les valeurs des sommets.*

Conclusions et perspectives

Nous avons présenté quelques expérimentations 1d autour de la méthode MLS. Ces expérimentations ont débouché sur une technique d'interpolation que nous qualifions d'hybride car mélangeant la méthode MLS et un développement de Taylor. Cette méthode est avantageuse car :

- la qualité de l'interpolation est bien meilleure qu'une interpolation de Sheppard ;

– la vitesse de calcul est comprise entre celle d'une interpolation de Sheppard et celle d'une approximation MLS, l'inversion des matrices du MLS se faisant uniquement aux échantillons.

Pour poursuivre ce travail, voici quelques questions qui restent ouvertes :

– existe-t-il un intérêt à utiliser un développement de Taylor d'ordre plus élevé pour la fonction $\tilde{u}^{Taylor}(x)$? Quels sont les avantages/inconvénients ?
– quels sont les placements envisageables des sphères (fonctions de poids) en 2d et 3d qui permettent de conserver la propriété d'une interpolation ?
– le schéma de placement des sphères d'interpolation sur une grille subdivisée permet-il de résoudre le problème des fonctions-T référencées dans le chapitre (3) ?

De manière plus générale, on peut se demander si la méthode hybride ne pourrait pas servir à résoudre le problème HOPI [XH06].

A.1 Performance et parallélisme

Durant ce travail de thèse, notre équipe, en collaboration avec le CIMIT et les projets Asclepios et Evasion, a commencé le développement d'une nouvelle plate-forme de simulation. A terme, cette plate-forme devra servir de base logicielle commune aux différents développements dans le domaine de la simulation médicale. Les concepts de multireprésentation, de multimodélisation, et de découplage des différents éléments d'un simulateur sont largement employés ; ces concepts favorisant l'ouverture et la réutilisation des composants de la plate-forme. Bien que ce n'était pas directement le sujet de la thèse, le lien entre la parallélisation et le découplage des objets nous intéressait et nous nous sommes impliqués dans le développement de Sofa en proposant une technique simple de parallélisation.

C'est un point important, car il semble que la production de processeurs s'oriente de plus en plus vers l'utilisation de processeurs multicore. On trouve par exemple des processeurs dotés de quatre cœurs d'exécutions AMD,x86. Le Cell d'IBM qui équipe la console de jeu PS3, contient sept cœurs, pouvant chacun exécuter un thread indépendant. L'importance de cette tendance est que la mise en parallèle de plusieurs processeurs nécessite de modifier considérablement les codes de calcul pour tirer parti de la puissance de calcul disponible. Le travail effectué, en collaboration avec Pierre-Jean Bensoussan, est implémenté dans l'environnement SOFA, dédié à la simulation médicale [sof]. L'approche proposée se veut générique et non intrusive. En particulier, les codes de simulation et de détection de collision, particulièrement complexes, n'ont pas besoin d'être modifiés pour tirer profit des unités de calcul supplémentaires.

On peut distinguer plusieurs approches pour tirer parti des architectures multicores :

- *la parallélisation des codes de calcul :* c'est probablement la technique la plus utilisée dans le domaine de la simulation. Elle consiste à identifier dans un code de calcul existant, les zones qui peuvent être parallélisées. On trouve ainsi quelques implémentations d'algèbre linéaire qui tirent profit de plusieurs unités de calcul. La parallélisation du code de calcul est à privilégier lorsqu'on manipule des objets trop gros (plusieurs dizaine de milliers de degrés de liberté) pour être simulés sur un seul ordinateur ; on utilise alors des clusters regroupant plusieurs centaines d'ordinateurs mis en parallèle. On trouve quelques usages de cette approche pour la simulation interactive [ZFV04]. Mais pour être efficace, et c'est là le défaut de cette approche, elle suppose la modification des codes des objets ; pour profiter de la puissance de calcul supplémentaire, chaque objet de la simulation doit être parallélisé de manière ad-hoc.
- *la mise en pipeline :* les différents cores exécutent chacun une tâche bien précise du simulateur. Un core exécute le moteur de simulation, un autre se charge du rendu. Une telle approche est observée dans Spore [MDH+02].

Le code que nous avons développé, en collaboration avec Pierre-Jean Benssoussan, est sensiblement différent de ces deux approches et s'appuie sur le découpage en tâches de l'application à simuler.

Parallélisation par tâches

Le code de l'environnement virtuel et du moteur de simulation est découpé en petites tâches. Certaines de ces tâches peuvent s'exécuter en parallèle, d'autres pas. On utilise un encodage explicite des dépendances entre les différentes tâches à l'aide d'un automate. Chaque

tâche de la simulation est associée à un état de l'automate ; et les différentes relations de l'automate sont modélisées par les arêtes liant les états. Pour chaque arête, on associe une condition de passage, qui permet la synchronisation ou non des étapes. On peut observer sur la figure (A.7), un exemple simple d'un tel automate.

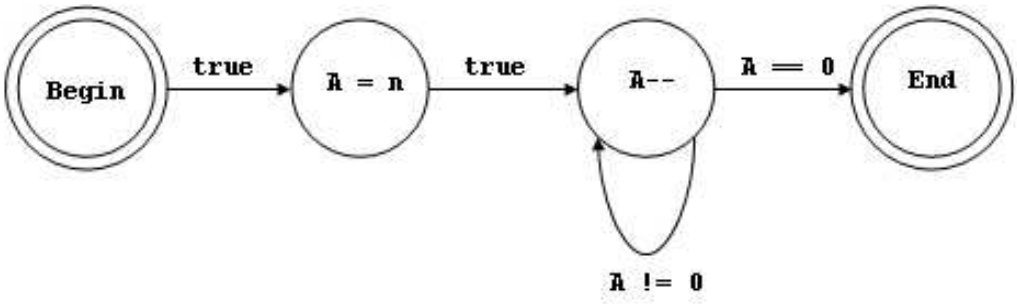

Fig. A.7 : *Exemple d'automate : les noeuds sont associés à des actions de la simulation ; les arêtes codent les éléments de contrôle et les relations de dépendance entre les opérations.*

Prenons, par exemple, une boucle de simulation triviale :

```
for i in objets:
   i.nextstep();
collision();
affichage();
```

Dans cette boucle de simulation, on remarque que les fonctions *nextstep* des n objets sont indépendantes et peuvent être exécutées par des threads différents. Au contraire, la collision et l'affichage font office de barrière qui synchronisent les différents processus. Une telle boucle de simulation peut donc être représentée par l'automate A.8. Pendant la simulation, cet automate nous renseigne sur :

- la tâche courante ;
- les tâches à venir ;
- l'ensemble des tâches indépendantes (exécutées en parallèle).

Automate et machine virtuelle

Dans l'implémentation réalisée, on emploie une machine virtuelle qui a pour tâche l'exécution de l'automate encodant la boucle de simulation. Cette machine virtuelle s'appuie sur le principe du client-serveur. Chaque thread de l'application (chaque processeur) est considéré comme un client demandant des tâches au serveur. Le serveur garantissant que les tâches distribuées correspondent bien à l'ordonnancement décrit par l'automate. Au niveau de l'implémentation, la machine virtuelle est composée de deux listes qui stockent l'ensemble des tâches prêtes à être exécutées. La première liste stocke les tâches séquentielles, la seconde, les tâches identifiées comme étant parallèles (fig. A.9). Les points forts de ce modèle client-serveur sont :

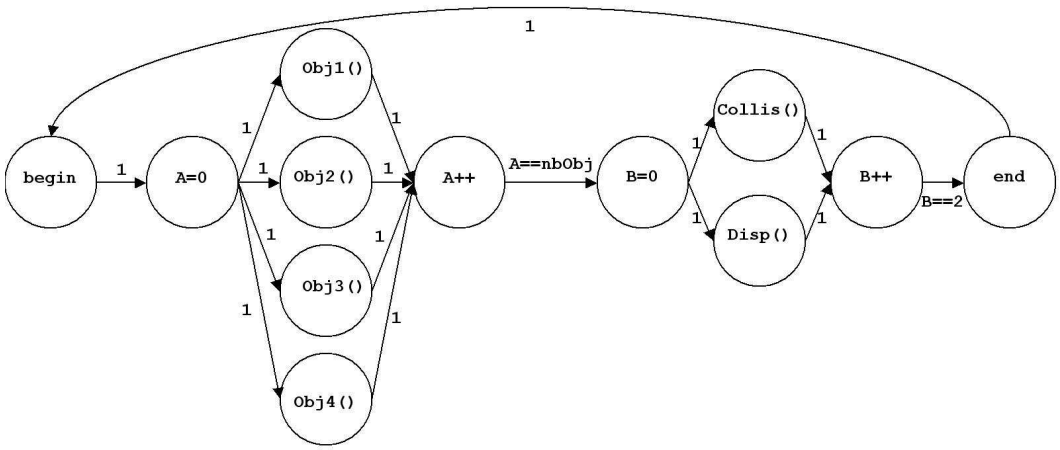

Fig. A.8 : *Mise sous forme d'automate d'une boucle de simulation triviale.*

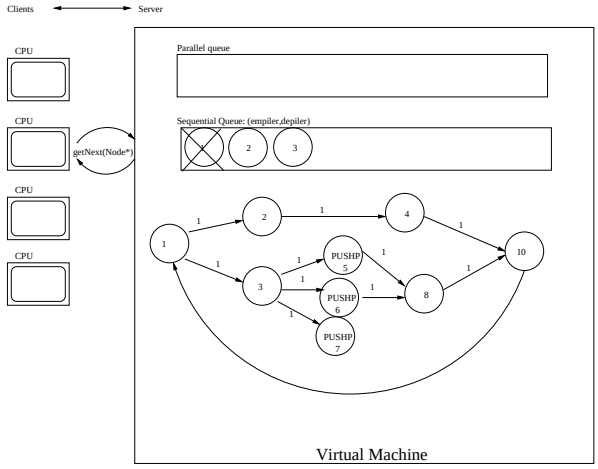

Fig. A.9 : *Machine virtuelle fonctionnant sous le mode du client-serveur.*

- l' équilibrage : la programmation parallèle implique de maximiser l'utilisation des processeurs, on parle d'équilibrage ; un bon équilibrage traduisant le fait que les threads se sont partagés les instructions de l'application. Un modèle client-serveur a l'avantage d'équilibrer naturellement les tâches. En plus, cet équilibrage est dynamique ce qui présente un intérêt certain lorsque les tâches sont bâties sur des algorithmes itératifs (dont on ne peut prédire la terminaison).
- le nombre réduit de mutex et de sémaphore qui limite le risque d'erreur et de blocage (dead-lock) ; un seul mutex est nécessaire au niveau du point d'entrée entre les clients et le serveur.

Résultats et perspectives

Un certain nombre d'expérimentations a été mené autour de cette architecture. On a noté une amélioration sensible des performances dès que la simulation comporte suffisamment

d'objets à simuler (de tâches indépendantes). En pratique, on arrive à une accélération de la simulation de l'ordre de 1.5x à 1.8x, quand on utilise une machine avec deux unités de calcul (dual-core,smp). Une parallélisation plus fine est possible en étudiant comment diminuer les phases de synchronisation (collision). Cependant, le fait de devoir décrire, à l'aide d'un automate, la boucle de simulation n'est pas forcément évident ; il faudrait pouvoir déduire directement l'automate à partir du code de simulation existante ; pour cela, un système tel que [GRR03] semble utilisable. Une description plus précise de l'implémentation de ce système est disponible à l'adresse web : `http://www.lifl.fr/~marchal/these/sofa_threading.doc`.

Bibliographie

[ACY+05] P. Alliez, D. Cohen-Steiner, M. Yvinec, et M. Desbrun. « Variational Tetrahedral Meshing ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 24(3), pages 617–625, 2005.

[AGP+02] M. Alexa, M. Gross, M. Pauly, H. Pfister, M. Stamminger et M. Zwicker. « Point-Based Computer Graphics ». Dans *Eurographics (tutorial)*, 2002.

[Bar89] D. Baraff. « Analytical methods for dynamic simulation of non-penetrating rigid bodies ». *Computer & Graphics*, volume 23(3), pages 223–232, 1989.

[BCG+96] G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell et A. Tal. « BOXTREE : A Hierarchical Representation for Surfaces in 3D ». *Computer Graphic Forum*, volume 15(3), pages 387–396, 1996.

[BEG+99] J. Basch, J. Erickson, L. J. Guibas, J. Hershberger et L. Zhang. « Kinetic Collision Detection Between Two Simple Polygons ». Dans *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 102–111, 1999.

[Ber06] F. Bertails. « Simulation de Chevelures Virtuelles ». Thèse de doctorat, *Institut National Polytechnique de Grenoble*, 2006.

[BF95] J. Bloomenthal et K. Ferguson. « Polygonization of Non-Manifold Surfaces ». Dans *ACM Siggraph*, pages 309–316. ACM Press, 1995.

[BFA02] R. Bridson, R. Fedkiw et J. Anderson. « Robust treatment of collisions, contact and friction for cloth animation ». Dans *ACM Siggraph*, pages 594–603. ACM Press, 2002.

[BGAM04] D. Borro, A. M. García-Alonso et L. M. Matey. « Approximation of Optimal Voxel Size for Collision Detection in Maintainability Simulations within Massive Virtual Environments ». *Computer Graphics Forum*, volume 23(1), pages 13–24, 2004.

[BGL94] T. Belytschko, L. Gu et Y. Y. Lu. « fracture and crack growth by element-free Galerkin methods ». *Modelling and Simulation in Material Science*, volume 2, pages 519–534, 1994.

[BGTG04] D. Bielser, P. Glardon, M. Teschner et M. Gross. « A State Machine for Real-Time Cutting of Tetrahedral Meshes ». *Journal of Graphical Models*, volume 66(6), pages 398–417, 2004.

[Blo95] J. Bloomenthal. « Skeletal Design of Natural Forms ». Thèse de doctorat, *Dept. of Computer Science, University of Calgary, Alberta*, 1995.

[Blo02] J. Bloomenthal. « Medial-Based Vertex Deformation ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 147–151. ACM Press, 2002.

[BMC05] F. Bertails, C. Ménier et M.-P. Cani. « A Practical Self-Shadowing Algorithm for Interactive Hair Animation ». Rapport Technique 5465, *INRIA*, 655 av de l'Europe ; 38334 Saint-Ismier Cedex ; FRANCE, 2005.

[BMF03] R. Bridson, S. Marino et R. Fedkiw. « Simulation of Clothing with Folds and Wrinkles ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 28–36. Eurographics Association, 2003.

[BO02] G. Bradshaw et C. O'Sullivan. « Sphere-tree construction using dynamic medial axis approximation ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 33–40. ACM Press, 2002.

[BO04] G. Bradshaw et C. O'Sullivan. « Adaptive Medial-Axis Approximation for Sphere-Tree Construction ». *ACM Transaction on Graphics*, volume 23(1), pages 1–26, 2004.

[BTH+03] K. S. Bhat, C. D. Twigg, J. K. Hodgins, P. K. Khosla, Z. Popović et S. M. Seitz. « Estimating Cloth Simulation Parameters from Video Authors ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 37–51. Eurographics Association, 2003.

[BW97] D. Baraff et A. Witkin. « Physically Based Modeling : Principles and Practice ». Dans *ACM Siggraph (courses notes)*, 1997.

[BW98] D. Baraff et A. Witkin. « Large steps in cloth simulation ». Dans *ACM Siggraph*, pages 43–54. ACM Press, 1998.

[Cam97] S. Cameron. « Enhancing GJK : Computing minimum distance and penetration distanses between convex polyhedra ». *IEEE International Conference on Robotic and Automation*, pages 3112–3117, 1997.

[Can85] J. Canny. « A voronoi method for the piano movers' problem ». Dans *IEEE Conference on Robotics and Automation*, 1985.

[Can87] J. Canny. « A new algebraic method for robot motion planning and real geometry ». Dans *IEEE Conference on Foundations of Computer Science*, 1987.

[CD97] M.-P. Cani et M. Desbrun. « Animation of Deformable Models Using Implicit Surfaces ». *IEEE Transaction on Visualisation and Computer Graphics*, volume 3(1), pages 39–50, Mar 1997. Published under the name Marie-Paule Cani-Gascuel.

[CDL+05] S. Cotin, C. Duriez, J. Lenoir, P. Neumann et S. Dawson. « New Approaches to Catheter Navigation for Interventional Radiology Simulation ». Dans *MICCAI*, pages 534–542, 2005.

[CDP95] F. Cazals, G. Drettakis et C. Puech. « Filtering, Clustering and Hierarchy Construction : a New Solution for Ray Tracing Very Complex Environments ». *Computer Graphic Forum (proceeding of Eurographics)*, volume 14(3), pages 371–382, 1995.

[CG02] P. Coussot et J.-L. Grossiord. *Comprendre la rhéologie*. EDP science, 2002.

[cga] « CGAL : Computational Geometry Algorithms Library ». http://www.cgal.org/.

[CGTS04] M. C. Cavusoglu, T. G. Goktekin, F. Tendick et S. S. Sastry. « GiPSi : An Open Source/Open Architecture Software Development Framework for Surgical Simulation ». Dans *Medicine Meets Virtual Reality XII (MMVR)*, pages 46–48, 2004.

[CLF93] C. Cadoz, A. Luciani et J. L. Florens. « CORDIS-ANIMA : A Modeling and Simulation System for Sound and Image Synthesis - the General Formalism ». *Computer Music Journal*, volume 17(4), 1993.

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

[Cot98] S. Cotin. « Modèles Anatomiques Déformables en temps-réel – Application à la simulation de chirurgie avec retour d'effort ». Thèse de doctorat, *Uuniversité de Nice - Sophia Antipolis*, 1998.

[CP] M. Á. Carreira-Perpiñán. « "Simplex Method." From MathWorld–A Wolfram Web Resource ». http://mathworld.wolfram.com/SimplexMethod.html.

[DC95] M. Desbrun et M.-P. Cani. « Animating Soft Substances with Implicit Surfaces ». Dans R. Cook, éditeur, *Computer Graphics (proceeding of Siggraph)*, volume 29, pages 287–290. ACM SIGGRAPH, Addison Wesley, aug 1995. Published under the name Marie-Paule Gascuel.

[DC96] M. Desbrun et M.-P. Cani. « Smoothed Particles : A new paradigm for animating highly deformable bodies ». Dans R. Boulic et G. Hegron, éditeurs, *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–76. Springer-Verlag, Aug 1996. Published under the name Marie-Paule Gascuel.

[DCA99] H. Delingette, S. Cotin et N. Ayache. « A Hybrid Elastic Model Allowing Real-Time Cutting, Deformations and Force-Feedback for Surgery Training and Simulation ». Dans *Computer Animation*, page 70. IEEE Computer Society, 1999.

[DDCB01] G. Debunne, M. Desbrun, M.-P. Cani et A. H. Barr. « Dynamic Real-Time Deformations Using Space & Time Adaptive Sampling ». Dans *ACM Siggraph*, pages 31–36. ACM Press / ACM SIGGRAPH, 2001.

[Deb00] G. Debunne. « Animation multirésolution d'objets déformables en temps-réel, Application à la simulation chirurgicale ». Thèse de doctorat, *Institut National Polytechnique de Grenoble*, 2000.

[Deq05] J. Dequidt. « Objets Autonomes en Simulation Physique Temps-réel ». Thèse de doctorat, *Université des Sciences et Technologies de Lille*, 2005.

[Des97] M. Desbrun. « Modélisation et Animation de Matériaux Hautement Déformables en Synthèse d'Images ». Thèse de doctorat, *Institut National Polytechnique de Genoble*, 1997.

[Dis] J.-M. Dischler. « Introduction à la Topologie et aux G-Cartes ». Cours de DEA de l'université de Strasbourg.

[DKT] M. Desbrun, E. Kanso et Y. Tong. Discrete Differential Forms for Computational Modeling. ACM Siggraph (courses notes).

[DM06] T. DeRose et M. Meyer. « Harmonic Coordinates ». Rapport Technique, *Pixar*, 2006.

[DMG05] J. Dequidt, D. Marchal et L. Grisoni. « Time-Critical Animation of Deformable Solids ». *Computer Animation and Social Agent (CASA)*, volume 16, pages 177–187, 2005.

[DO00] J. Dingliana et C. O'Sullivan. « Graceful Degradation of Collision Handling in Physically Based Animation ». *Computer Graphic Forum (proceeding of Eurographics)*, volume 19(3), pages 239–247, 2000.

[Duf04] M. Duflot. « Application des méthodes sans maillage en mécanique de la rupture ». Thèse de doctorat, *Université de Liège*, 2004.

[Dur04] C. Duriez. « Contact frottant entre objets déformable dans des simulations temps-réel avec retour haptique ». Thèse de doctorat, *Université d'Evry*, 2004.

[Ebe03] D. H. Eberly. *Game Physics*. Morgan Kaufmann, 2003.

[Edw] O.-A.-T. Edward. « Fast Collision Detection with an N-Objects Octree ». http://citeseer.ist.psu.edu/687360.html.

[EHK+00] B. Eberhardt, J. Hahn, R. Klein, W. Strasser et A. Weber. « Dynamic Implicit Surfaces for Fast Proximity Queries in Physically Based Modeling ». Rapport Technique WSI–2000–11, *Wilhelm-Schickard-Institut für Informatik, Universität Tübingen*, 2000.

[EL00] S. A. Ehmann et M. C. Lin. « SWIFT : Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching ». Rapport Technique, *Computer Science Department, University of North Carolina at Chapel Hill*, 2000.

[faq] « Binary Space Partitioning Trees FAQ ». http://www.faqs.org.

[Fau] F. Faure. « AnimAL, librairie de simulation (animation) ».

[Fau98] F. Faure. « Interactive Solid Animation Using Linearized Displacement Constraints ». Dans *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 61–72, 1998.

[FDA02] C. Forest, H. Delingette et N. Ayache. « Removing Tetrahedra from a Manifold Mesh ». Dans *Computer Animation*, pages 225–229. IEEE Computer Society, 2002.

[Fel00] C. A. Felippa. « A systematic approach to the elementindependent corotational dynamics of finite elements ». Rapport Technique, *Technical Report Report CU-CAS-00-03, University of Colorado*, 2000.

[FG03] C. R. Fabio Ganovelli, F. Ponchio. « Fast Tetrahedron-Tetrahedron overlap Algorithm ». *Journal of Graphics Tools*, volume 7, pages 17–26, 2003.

[FKM03] F. Faure, A. Kheddar et P. Meseure. « Compte rendu de l'AS collision », 2003. INRIA-Evasion.

[FL01] S. Fisher et M. C. Lin. « Fast Penetration Depth Estimation for Elastic Bodies Using Deformed Distance Fields ». Dans *IEEE International Conference on Robotics and Automation*, volume 1, pages 330–336. IEEE Computer Society, 2001.

[FM03] T.-P. Fries et H.-G. Matties. « Classification and Overview of Meshfree Methods ». Rapport Technique, *Technical University Braunscheiw (Germany)*, 2003.

[For03] C. Forest. « Simulation de chirurgie par coelioscopie : contributions à l'étude de la découpe volumique, au retour d'effort et à la modélisation des vaisseaux sanguins ». Thèse de doctorat, *École Polytechnique*, 2003.

[FPPJ00] S. F. Frisken, R. N. Perry, A. P.Rockwoord et T. R. Jones. « Adatpively Sampled Distance Fields : A General Representation of Shape for Computer Graphics ». Rapport Technique, *Mitsubishi Electric Research Laboratories (MERL)*, 2000.

[FSG00] A. Fuhrmann, G. Sobottka et C. Gross. « Distance Fields for Rapid Collision Detection in Physically Based Modeling ». Dans *Proceding of Graphicon*, 2000.

[GCMS00a] F. Ganovelli, P. Cignoni, C. Montani et R. Scopigno. « Enabling Cuts on Multiresolution Representations ». Dans *Computer Graphics International (CGI)*, pages 183–190, 2000.

[GCMS00b] F. Ganovelli, P. Cignoni, C. Montani et R. Scopigno. « A Multiresolution Model for Soft Objects Supporting Interactive Cuts and Lacerations ». *Computer Graphics Forum*, volume 19(3), pages 271–282, 2000.

[GD03] P. Guigue et O. Devillers. « Fast and Robust Triangle-Triangle Overlap Test using Orientation Predicates ». *Journal of Graphics Tools*, volume 8(1), pages 25–42, 2003.

[GKJ+05] N. K. Govindaraju, D. Knott, N. Jain, I. Kabul, R. Tamstorf, R. Gayle, M. C. Lin et D. Manocha. « Interactive Collision Detection between Deformable Models using Chromatic Decomposition ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 24(3), pages 991–999, 2005.

[GLM96] S. Gottschalk, M. C. Lin et D. Manocha. « OBBTree : A Hierarchical Structure for Rapid Interference Detection ». Dans *Computer Graphics (proceeding of Siggraph)*, pages 171–180, 1996.

[GLM04] N. K. Govindaraju, M. C. Lin et D. Manocha. « Fast and Reliable Collision Culling using Graphics Processors ». Dans *ACM VRST*, pages 2–9. ACM Press, 2004.

[GM03] L. Grisoni et D. Marchal. « High performance Generalized Cylinders Visualization ». Dans *Shape Modeling International*, pages 257–263, 2003.

[GO01] F. Ganovelli et C. O'Sullivan. « Animating cuts with on-the-fly re-meshing ». Dans *Eurographics (short paper)*, pages 243–247, 2001.

[GPR02] F. Ganovelli, F. Ponchio et C. Rocchini. « Fast Tetrahedron-Tetrahedron Overlap Algorithm ». *Journal of Graphics Tools*, volume 7(2), pages 17–26, 2002.

[Gri99] L. Grisoni. « Elements de multiresolution en modelisation geometrique ». Thèse de doctorat, *Université de Bordeaux I*, 1999.

[Gri05] L. Grisoni. « Vers une simulation physique temps reel multi-modèle ». Habilitation à diriger des recherchers, *Université des Sciences et Technologies de Lille*, 2005.

[GRLM03] N. K. Govindaraju, S. Redon, M. C. Lin et D. Manocha. « CULLIDE : Interactive Collision Detection between Complex Models in Large Environments using Graphics Hardware ». Dans *ACM-Eurographics Graphics Hardware*, pages 25–32. Eurographics Association, 2003.

[GRR03] T. Gautier, R. Revire et J. Roch. « ATHAPASCAN : an API for Asynchronous Parallel Programming ». Rapport Technique, *INRIA RT-0276*, 2003.

[GS05] T. Giang et C. Sullivan. « Closest Feature Map For Time Critical Collision Handling ». Dans *VriPhys*, 2005.

[gsl] « GSL : GNU Scientific Library ». http://www.gnu.org/software/gsl/.

[gts] « GTS : GNU Triangulated Surface Library ». http://gts.sourceforge.net/.

[Guaa] J. Guarrigues. « Cours de mécanique des millieux continus ». http://jgarrigues.perso.egim-mrs.fr/mmc.html.

[Guab] J. Guarrigues. « Cours de statique des coques ». http://jgarrigues.perso.egim-mrs.fr/coques.html.

[Guac] J. Guarrigues. « Cours de statique des poutres ». http://jgarrigues.perso.egim-mrs.fr/poutre.html.

[Guad] J. Guarrigues. « Cours d'élasticité linéaire isotrope en petites perturbations ». http://jgarrigues.perso.egim-mrs.fr/elas.html.

[Guae] J. Guarrigues. « Initiation à la méthode des éléments finis ». http://jgarrigues.perso.egim-mrs.fr/ef.html.

[GW] D. Gluss et E. W. Weisstein. « "Lagrange Multiplier." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. ». http://mathworld.wolfram.com/LagrangeMultiplier.html.

[Hav] « Havok Physics ». http://www.havok.com/.

[HB03] S. Hahmann et G.-P. Bonneau. « Polynomial Surfaces Interpolating Arbitrary Triangulations ». *IEEE Transaction on Visualisation and Computer Graphics*, volume 9(1), pages 99–109, 2003.

[HFL00] G. Hirota, S. Fisher et M. C. Lin. « Simulation of Non-penetrating Elastic Bodies Using Distance Fields ». Rapport Technique TR00-018, *University Of North Carolina*, 2000.

[Hil02] L. Hilde. « Algorithmes de résolution des équations du mouvement pour l'animation basée sur la physique ». Thèse de doctorat, *Université des Sciences et Technologies de Lille*, 2002.

[HLC+01] J. Huang, Y. Li, R. Crawfis, S. chiun Lu et S. Liou. « A Complete Distance Field Representation ». Dans *IEEE Visualization Conference*, pages 247–254. IEEE Computer Society, 2001.

[HS04] M. Hauth et W. Strasser. « Corotational Simulation of Deformable Solids ». Dans *WSCG*, 2004.

[HTG03] B. Heidelberger, M. Teschner et M. Gross. « Real-Time Volumetric Intersections of Deforming Objects, ». Dans *Vision, Modelling, Visualisation*, pages 461–468, 2003.

[HTJ+99] K. E. Hoff, T.Culver, J.Keyser, M. C. Lin et D. Manocha. « Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware ». Dans *ACM Siggraph*, pages 277–285, 1999.

[Hub93] P. M. Hubbard. « Interactive Collision Detection ». Dans *IEEE Symposium on Research Frontiers in Virtual Reality*, pages 24–32. IEEE Computer Society, 1993.

[Hub95] P. M. Hubbard. « Collision Detection for Interactive Graphics Applications ». *IEEE Transaction on Visualisation and Computer Graphics*, volume 1(3), pages 218–230, 1995.

[Hub96] P. M. Hubbard. « Approximating polyhedra with spheres for time-critical collision detection ». *ACM Transaction on Graphics*, volume 15(3), pages 179–210, 1996.

[HZLM01] K. E. Hoff, A. Zaferakis, M. C. Lin et D. Manocha. « Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware ». Dans *ACM Symposium on Interactive 3D Graphics*, pages 145–148. ACM Press, 2001.

[HZLM02] K. E. Hoff, A. Zaferakis, M. C. Lin et D. Manocha. « Fast 3D Geometric Proximity Queries between Rigid and Deformable Models Using Graphics Hardware Acceleration ». Rapport Technique, *UNC-CS*, 2002.

[IF02] S. Ilic et P. Fua. « Using Dirichlet Free Form Deformation to Fit Deformable Models to Noisy 3-D Data ». Dans *European Conference on Computer Vision*, pages 704–717, 2002.

[ITF06] G. Irving, J. Teran et R. Fedkiw. « Tetrahedral and Hexahedral Invertible Finite Elements ». *Graphical Models*, volume 68, pages 66–89, 2006.

[JKG+05] N. Jain, I. Kabul, N. K. Govindaraju, M. C. Lin et D. Manocha. « Multi-Resolution Collision Handling for Cloth-like Simulations ». Dans *Computer Animation and Social Agent (CASA)*, pages 141–151. John Wiley and Sons Ltd., 2005.

[Jou97] A. Joukhadar. « Simulation Dynamique et Applications Robotiques ». Thèse de doctorat, *Institut National Polytechnique de Grenoble*, 1997.

[JP04] D. L. James et D. K. Pai. « BD-Tree : Output-Sensitive Collision Detection for Reduced Deformable Models ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 23(3), pages 393–398, 2004.

[JT93] W. Joe et D. Tony. « Barycentric coordinates for convex polytopes », 1993.

[JTT01] P. Jiménez, F. Thomas et C. Torras. « 3D Collision Detection : A Survey ». *Computer & Graphics*, volume 25(2), pages 269–285, 2001.

[KHM98] J. Klosowski, M. Held et J. S. B. Mitchell. « Efficient collision detection using bounding volume Hierarchies of k-dops ». *IEEE Transaction on Visualisation and Computer Graphics*, volume 4(1), pages 21–36, 1998.

[KLM02] Y. J. Kim, M. C. Lin et D. Manocha. « DEEP : Dual-space Expansion for Estimating Penetration Depth between convex polytopes ». Dans *IEEE International Conference on Robotics and Automation*. IEEE Computer Society, 2002.

[KNF04] S. Kimmerle, M. Nesme et F. Faure. « Hierarchy Accelerated Stochastic Collision Detection ». Dans *Vision, Modelling, Visualisation*, Stanford, California, 2004.

[KOLM02] Y. J. Kim, M. A. Otaduy, M. C. Lin et D. Manocha. « Fast Penetration Depth Computation for Physically-based Animation ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 23–31. ACM Press, 2002.

[KPLM97] S. Krishnan, A. Pattekar, M. C. Lin et D. Manocha. « Spherical shells : A higher-order bounding volume for fast proximity queries ». Rapport Technique, *Department of Computer Science, University of North Carolina*, 1997.

[KS98] R. Kimmel et J. A. Sethian. « Fast Marching Methods on Triangulated Domains ». Dans *Proceedings of the National Academy of Sciences*, volume 95, pages 8341–8435, 1998.

[LAM01] T. Larsson et T. Akkenine-Möller. « Collision detection for continuously deforming bodies. ». Dans *Eurographics (short paper)*, page 30, 2001.

[LAM06] T. Larsson et T. Akenine-Möller. « A dynamic bounding volume hierarchy for generalized collision detection ». *Computer & Graphics*, volume 30(3), pages 451–460, June 2006.

[LCDN06] J. Lenoir, S. Cotin, C. Duriez et P. Neumann. « Interactive physically-based simulation of catheter and guidewire ». *Computer & Graphics*, volume 30(3), pages 417–423, 2006.

[Len04] J. Lenoir. « Modèle déformable 1D pour la simulation physique temps réel ». Thèse de doctorat, *Université des Sciences et Technologies de Lille*, 2004.

[Lev03] D. Levin. « *Geometric Modeling for Scientific Visualization* », Chapitre Mesh-independent surface interpolation, pages 37–49. Springer-Verlag, 2003.

[LG95] D. P. Luebke et C. Georges. « Portals and Mirrors : Simple, Fast Evaluation of Potentially Visible Sets ». Dans *Symposium on Interactive 3D Graphics*, page 105. ACM Press, 1995.

[LG98] M. C. Lin et S. Gottschalk. « Collision detection between geometric models : a survey ». Dans *IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[LGF04] F. Losasso, F. Gibou et R. Fedkiw. « Simulating water and smoke with an octree data structure ». *ACM Transaction on Graphics*, volume 23(3), pages 457–462, 2004.

[LGMC05] J. Lenoir, L. Grisoni, P. Meseure et C. Chaillou. « Adaptive resolution of 1D mechanical B-spline ». Dans *Graphite Conference*, pages 395–403, Dunedin - New Zealand, 2005.

[LIG06] F. Losasso, G. Irving et E. Guendelman. « Melting and Burning Solids into Liquids and Gases ». *IEEE Transaction on Visualisation and Computer Graphics*, volume 12(3), pages 343–352, 2006. Member-Ron Fedkiw.

[Lin94] M. C. Lin. « Effient Collision Detection for Animation and Robotics ». Rapport Technique ERL-94-13, *University of Chapell Hill*, 1994.

[LM03] M. C. Lin et D. Manocha. « *Collision and Proximity Queries* », Chapitre 35. CRC Press LLC, 2003.

[LMP94] M. C. Lin, D. Manocha et M. Ponamgi. « Fast algorithms for penetration and contact determination between non-convex polyhedral models », 1994.

[MAC03] D. Marchal, F. Aubert et C. Chaillou. « Détection et réponse aux collision à base de sphères ». Dans *Actes des journées de l'Association de l'Informatique Graphique Française*, 2003.

[MAC04] D. Marchal, F. Aubert et C. Chaillou. « Collision Between Deformable Objects Using Fast-Marching on Tetrahedral Models ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, 2004.

[MBF04] N. Molino, Z. Bao et R. Fedkiw. « A virtual node algorithm for changing mesh topology during simulation ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 23, pages 385–392, 2004.

[MC95] B. Mirtich et J. Canny. « Impulse based Simulation of Rigid Bodies ». Dans *ACM Symposium on Interactive 3D Graphics*, 1995.

[MCG03] M. Müller, D. Charypar et M. Gross. « Particle-Based Fluid Simulation for Interactive Applications ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 154–159, 2003.

[MDH+02] P. Meseure, J. Davanne, L. Hilde, F. Triquet, J. Lenoir et C. Chaillou. « SPORE : a Unified Framework for the Real-Time Simulation of Various Physically-Based Objects ». Rapport Technique, *Laboratoire d'Informatique Fondamentale de Lille*, 2002.

[med] « MedSim : Advanced Medical Simulations ». http://www.medsim.com/.

[Mes03] P. Meseure. « Animation basé sur la physique pour les environnements interactifs temps-réel ». Habilitation à diriger des recherchers, *Université des Sciences et Technologies de Lille*, 2003.

[MG06] D. Marchal et L. Grisoni. « Cutting through large deformables objects ». SCA (Poster and abstract), 2006.

[MHTG05] M. Müller, B. Heidelberger, M. Teschner et M. Gross. « Meshless deformations based on shape matching ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 24(3), pages 471–478, 2005.

[Mil88] G. Miller. « The Motion Dynamics of Snakes and Worms ». Dans *ACM Siggraph*, pages 169–173. ACM Press, 1988.

[Mir96] B. Mirtich. « Impulse-based Dynamic Simulation of Rigid Body Systems ». Thèse de doctorat, *University of California, Berkeley*, 1996.

[Mir98] B. Mirtich. « VClip : Fast and robust polyhedral collision detection ». *ACM Transaction on Graphics*, volume 17(3), pages 177–208, 1998.

[MK00] A. B. Mor et T. Kanade. « Modifying Soft Tissue Models : Progressive Cutting with Minimal New Element Creation ». Dans *Third International Conference on Medical Image Computing and Computer-Assisted Intervention*, volume 1935, pages 598–607. Springer–Verlag, 2000.

[MKE02] J. Mezger, S. Kimmerle et O. Etzmuss. « Improved Collision Detection and Response Techniques for Cloth Animation ». Rapport Technique WSI-2002-5, *Universität Tübingen*, 2002.

[MKN+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross et M. Alexa. « Point Based Animation of Elastic, Plastic and Melting Objects ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 141–151. ACM Press, 2004.

[MLFC04] P. Meseure, J. Lenoir, S. Fonteneau et C. Chaillou. « Generalized god-objects : a paradigm for interacting with physically-based virtual worlds ». Dans *Computer Animation and Social Agent (CASA)*, pages 215–222, 2004.

[MMD+02] M. Müller, M. McMillan, J. Dorsey, R. Jagnow et B. Cutler. « Stable real-time deformations ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, pages 49–54. ACM Press, 2002.

[Mol04] N. Molino. « Mesh generation and fracture for deformable bodies ». Thèse de doctorat, *Stanford University*, 2004.

[Mor66] G. M. Morton. « A computer oriented geodetic data base and a new technique in file sequencing ». Rapport Technique, *IBM Ltd. Ottawa, Canada*, 1966.

[MP89] G. Miller et A. Pearce. « Globular dynamics : A connected particle system for animating viscous fluids ». *Computers and Graphics*, volume 13(3), pages 305–309, 1989.

[MT] M. Müller et M. Teschner. « Volumetric Meshes for Real-Time Medical Simulations ».

[MT97] Moller et Trumbore. « Fast, Minimum Storage Ray-Triangle Intersection ». *Journal of Graphics Tools*, volume 2, 1997.

[MTG04] M. Müller, M. Teschner et M. Gross. « Physically-Based Simulation of Objects Represented by Surface Meshes ». Dans *Computer Graphics International (CGI)*, volume 00, pages 26–33. IEEE Computer Society, 2004.

[MW88] M. Moore et J. Wilhems. « Collision Detection and Response for Computer Animation ». *Computer Graphics (proceeding of Siggraph)*, volume 22(4), pages 289–298, 1988.

[Nea04] A. Nealen. « An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation ». Rapport Technique, *TU Darmstadt*, 2004.

[NFP06] M. Nesme, F. Faure et Y. Payan. « Hierarchical Multi-Resolution Finite Element Model For Soft Body Simulation ». *Lecture Notes in Computer Science*, volume 4072, pages 40–47, 2006.

[Ngu90] H. Nguyen Dang. *Un cours sur la mécanique de la rupture*. LTAS- Université de Liège, 1990.

[Nie03] H.-W. Nienhuys. « Cutting in deformable objects ». Thèse de doctorat, *Institute for information and computing sciences, Utrecht University*, 2003.

[NNB04] C. Ngo-Ngoc et S. Boivin. « Nonlinear cloth simulation ». Rapport Technique, *INRIA*, 2004.

[NPF05] M. Nesme, Y. Payan et F. Faure. « Efficient, Physically Plausible Finite Elements ». Dans *Eurographics (short paper)*, 2005.

[NPF06] M. Nesme, Y. Payan et F. Faure. « Animating Shapes at Arbitrary Resolution with Non-Uniform Stiffness ». Dans *Workshop in Virtual Reality Interaction and Physical Simulation (VRIPHYS)*. Eurographics Association, nov 2006.

[OBH02] J. F. O'Brien, A. W. Bargteil et J. K. Hodgins. « Graphical modeling and animation of ductile fracture ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 21(3), pages 291–294, 2002.

[OD99] C. O'Sullivan et J. Dingliana. « Real-time collision detection and response using sphere-trees ». Dans *15th Spring Conference on Computer Graphics*, pages 83–92, 1999.

[OD01] C. O'Sullivan et J. Dingliana. « Collisions and Perception ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 20(3), pages 151–168, 2001.

[ODGB01] C. O'Sullivan, J. Dingliana, F. Ganovelli et G. Bradshaw. « Collision Handling for Virtual Environments ». Dans *Eurographics (tutorial)*, 2001.

[OFTB96] D. Organ, M. Fleming, T. Terry et T. Belytschko. « Continuous meshless approximations for non-convex bodies by diffraction and transparency ». *Computational Mechanics*, volume 18(3), pages 225–235, 1996.

[OH99] J. F. O'Brien et J. K. Hodgins. « Graphical Modeling and Animation of Brittle Fracture ». Dans *ACM Siggraph*, pages 137–146. ACM Press/Addison-Wesley Publishing Co., 1999.

[OHH00] C.-J. Ong, E. Huang et S.-M. Hong. « A fast growth distance algorithm for incremental motions ». *IEEE International Conference on Robotics and Automation*, volume 16(6), pages 880–890, 2000.

[Pau03] M. Pauly. « Point Primitives for Interactive Modeling and Processing of 3D Geometry ». Thèse de doctorat, *ETH Zurich*, 2003.

[PDA02] G. Picinbono, H. Delingette et N. Ayache. « Modèle déformable élastique non-linéaire pour la simulation de chirurgie en temps réel ». *Les Comptes Rendus de l'Académie des Sciences (CRAS), C.R. Biologies*, volume 325(4), pages 335–344, 2002.

[Pic01] G. Picinbono. « Modèles géométriques et physiques pour la simulation d'interventions chirurgicales ». Thèse de sciences, *université de Nice Sophia-Antipolis*, 2001.

[PKA+05] M. Pauly, R. Keiser, B. Adams, P. Dutré, M. Gross et L. J. Guibas. « Meshless animation of fracturing solids ». *ACM Transaction on Graphics (proceeding of Siggraph)*, pages 957–964, 2005.

[qhu] « Q-Hull library ». http://www.qhull.org/.

[RCFC03] L. Raghupathi, V. Cantin, F. Faure et M.-P. Cani. « Real-time Simulation of Self-collisions for Virtual Intestinal Surgery ». Dans *International Symposium on Surgery Simulation and Soft Tissue Modeling*, numéro 2673 dans Lecture Notes in Computer Science, pages 15–26. Springer-Verlag, 2003.

[Rem00] Y. Remion. « Animation Dynamique : moteur lagrangien généraliste et applications ». Habilitation à diriger des recherchers, *Université de Reims Champagne-Ardenne, France*, 2000.

[Reu03] P. Reuter. « Reconstruction and Rendering of Implicit Surfaces from Large Unorganized Point Sets ». Thèse de doctorat, *Université Bordeaux 1, France*, 2003.

[RGF+04] L. Raghupathi, L. Grisoni, F. Faure, D. Marchal, M.-P. Cani et C. Chaillou. « An Intestine Surgery Simulator : Real-Time Collision Processing and Visualization ». *IEEE Transaction on Visualisation and Computer Graphics*, 2004.

[RKC02] S. Redon, A. Kheddar et S. Coquillart. « Fast Continuous Collision Detection between Rigid Bodies », 2002.

[RL06] S. Redon et M. C. Lin. « A Fast Method for Local Penetration Depth Computation ». *Journal of Graphics Tools*, volume 11(2), pages 37–50, 2006.

[Rog01] D. F. Rogers. *An Introduction to Nurbs*. Morgan Kaufmann Publishers, 2001.

[RTSD03] P. Reuter, I. Tobor, C. Schlick et S. Dedieu. « Point-based Modelling and Rendering using Radial Basis Functions ». Dans *Graphite Conference*, pages 111–118, 2003.

[Sch46] Schoenberg. « Contributions to the problem of approximation of equidistant data by analytic functions ». *Quarterly of Applied Mathematics*, volume 4(1 and 2), pages 45–99 and 121–141, 1946.

[Sch03] J.-M. Schwartz. « Simulation des déformations mécaniques dans une intervention de cryochirurgie ». Thèse de doctorat, *Univeristé de Laval*, 2003.

[SE03] P. J. Schneider et D. H. Eberly. *Geometric Tools for Computer Graphics*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, 2003.

[SE04] S. Schein et G. Elber. « Discontinuous Free Form Deformations ». Dans *Pacific Conference on Graphics and Applications*, pages 227–236, 2004.

[Set96] J. A. Sethian. « A Fast Marching Level Set Method for Monotonically Advancing Fronts ». *Proceedings of the National Academy of Sciences*, volume 93(4), pages 1591–1595, 1996.

[SGG+06] A. Sud, N. Govindaraju, R. Gayle, I. Kabul et D. Manocha. « Fast Proximity Computation Among Deformable Models using Discrete Voronoi Diagrams ». Dans *ACM Siggraph*, pages 1144–1153. ACM Press, 2006.

[She99] A. Sherstyuk. « Convolution Surfaces in Computer Graphics ». Thèse de doctorat, *School of Computer Science and Software Engineering, Monash University*, 1999.

[SK00] K. Singh et E. Kokkevis. « Skinning Characters using Surface-Oriented Free-Form Deformations ». Dans *Graphics Interfaces*, pages 35–42, 2000.

[SKSH03] M. Senin, N. Kojekine, V. Savchenko et I. Hagiwara. « Particle-Based Collision Detection ». Dans *Eurographics (short paper)*, 2003.

[SMB98] N. Sukumar, B. Moran et T. Belytschko. « The natural element method in solid mechanics ». *International Journal for Numerical Methods in Engineering*, volume 43(5), pages 839–887, 1998.

[SMMB00] N. Sukumar, N. Moës, B. Moran et T. Belytschko. « Extended Finite Element Method for Three-Dimensional Crack Modelling ». *International Journal for Numerical Methods in Engineering*, volume 48(11), pages 1549–1570, 2000.

[sof] « SOFA : Simulation Open Framework Architecture ». http://sofa-framework.org/.

[SOG06] D. Steinemann, M. A. Otaduy et M. Gross. « Fast Arbitrary Splitting of Deforming Objects ». Dans *ACM-Eurographics Symposium on Computer Animation (SCA)*, 2006.

[SOS04] C. Shen, J. F. O'Brien et J. R. Shewchuk. « Interpolating and Approximating Implicit Surfaces from Polygon Soup ». Dans *ACM Siggraph*, pages 896–904, 2004.

[SP86] T. W. Sederberg et S. R. Parry. « Free-Form Deformation of Solid Geometric Models ». Dans *Computer Graphics (proceeding of Siggraph)*, numéro 20 dans 4, pages 151–159, 1886.

[SPDC06] L. Seinturier, N. Pessemier, L. Duchien et T. Coupaye. « A Component-Based and Aspect-Oriented Model for Software Evolution ». *International Journal of Computer Applications in Technology (IJCAT), Special Issue on : Concern-Oriented Software Evolution*, 2006.

[Sta99] J. Stam. « Stable Fluids ». Dans *ACM Siggraph*, pages 121–128. ACM Press, 1999.

[TF88] D. Terzopoulos et K. Fleischer. « Modeling inelastic deformation : viscolelasticity, plasticity, fracture ». Dans *ACM Siggraph*, pages 268–278. ACM Press, 1988.

[THM+03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets et M. Gross. « Optimized Spatial Hashing for Collision Detection of Deformable Objects ». Dans *Vision, Modelling, Visualisation*, 2003.

[THMG04] M. Teschner, B. Heidelberger, M. Müller et M. Gross. « A Versatile and Robust Model for Geometrically Complex Deformable Solids ». Dans *Computer Graphics International (CGI)*, pages 312–319. IEEE Computer Society, 2004.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Thalmann-Magnenat, W. Strasser et P. Volino. « Collision Detection for Deformable Objects ». *Computer Graphic Forum (proceeding of Eurographics)*, volume 24(1), pages 61–81, 2005.

[TPBF87] D. Terzopoulos, J. Platt, A. H. Barr et K. Fleischer. « Elastically deformable models ». *Computer & Graphics*, volume 21(4), pages 205–214, 1987.

[Tri01] F. Triquet. « Habillage de modèles mécaniques : Facettisation temps réel de surfaces implicites ». Thèse de doctorat, *Université des Sciences et Technologies de Lille*, 2001.

[Tsa02] Y. R. Tsai. « Rapid and Accurate Computation of the Distance Function Using Grids ». *Journal of Computational Physics*, volume 178(1), pages 175–195, 2002.

[VCMT95] P. Volino, M. Courchesne et N. Magnenat-Thalmann. « Versatile and efficient techniques for simulating cloth and other deformable objects ». Dans *ACM Siggraph*, pages 137–144. ACM Press, 1995.

[vdB97] G. van den Bergen. « Efficient Collision Detection of Complex Deformable Models using AABB Trees ». *Journal of Graphics Tools*, volume 2(4), pages 1–13, 1997.

[vdB01] G. van den Bergen. « Proximity queries and penetration depth computation on 3d game objects ». Dans *Game Developer Conference*, 2001.

[VMT06] P. Volino et N. Magnenat-Thalmann. « Resolving Surface Collisions Through Intersection Contour Minimization ». *ACM Transaction on Graphics (proceeding of Siggraph)*, volume 25(3), pages 1154–1159, 2006.

[VVW04] L. M. Vigneron, J. G. Verly et S. K. Warfield. « Modelling Surgical Cuts, Retractions, and Resections via Extended Finite Element Method ». Dans *Proceedings of the 7th Int. Conf on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2004 (2)*, volume 3217 de *Lecture Notes on Computer Science*, pages 311–318, Saint-Malo, France, 2004. Springer Verlag.

[Wan00] K. Wang. « An $n^2 \log n$ Algorithm for Generating Swept Solids in NC Verification », 2000.

[Weia] E. W. Weisstein. « Mathworld entry for Calculus of Variations ». http://mathworld.wolfram.com/CalculusofVariations.html.

[Weib] E. W. Weisstein. « Mathworld entry for Hausdorff Measure ». http://mathworld.wolfram.com/HausdorffMeasure.html.

[Weic] E. W. Weisstein. « Mathworld entry for Minkowksi Sum ». http://mathworld.wolfram.com/MinkowskiSum.html.

[Weid] E. W. Weisstein. « Mathworld entry for Polar Coordinates ». http://mathworld.wolfram.com/PolarCoordinates.html.

[Weie] E. W. Weisstein. « "Numerical Differentiation." From MathWorld–A Wolfram Web Resource ». http://mathworld.wolfram.com/NumericalDifferentiation.html.

[Wika] Wikipedia. « Wikipedia entry for haptic ». http://en.wikipedia.org/wiki/Haptic.

[Wikb] Wikipedia. « Wikipedia entry for Heap data structure ». http://en.wikipedia.org/wiki/Heap_(data_structure).

[Wikc] Wikipedia. « Wikipedia entry for Metaheuristique ». http://fr.wikipedia.org/wiki/Métaheuristique.

[Wikd] Wikipedia. « Wikipedia entry for tactile ». http://fr.wikipedia.org/wiki/tactile.

[wol] « Mathworld entry for Root Finding (pointeur vers plusieurs méthodes) ». http://mathworld.wolfram.com/Root-FindingAlgorithm.html.

[WW86] W. Wang et K. Wang. « Geometric modeling for swept volume of moving solids », 1986.

[XH06] J. Xu et Z. Huang. « HOPI : A Novel High Order Parametric Interpolation in 2D ». Dans *Eurographics (short paper)*, pages 99–102, 2006.

[ZFV04] F. Zara, F. Faure et J.-M. Vincent. « Parallel Simulation of Large Dynamic System on a PCs Cluster : Application to Cloth Simulation ». *International Journal of Computers and Applications*, volume 202(3), pages 1464, 2004. special issue on cluster/grid computing.

[ZSL+00] D. Zorin, P. Schröder, D. Levin, L. Kobbelt, W. Sweldens et T. DeRose. « Subdivision for Modeling and Animation ». Dans *ACM Siggraph (courses notes)*, 2000.

Index des auteurs

P

R

S

T