Simulation Physique Interactive pour la Synthèse d'Images

François Faure

GRENOBLE UNIVERSITÉS

Simulation physique interactive
pour la synthèse d'images

François Faure

Thèse

présentée pour l'obtention de l'
Habilitation à Diriger des Recherches
de l'Université Joseph Fourier

spécialité
Informatique

préparée au sein du laboratoire
LJK (EVASION) UMR CNRS 5224 / INRIA

Composition du jury :

Hervé Delingette	Directeur de Recherche	INRIA	Sophia-Antipolis	Rapporteur
Abderrahmane Kheddar	Professeur	AIST-CNRS JRL	Tsukuba	Rapporteur
Jos Stam	Senior Scientist	Autodesk	Toronto	Rapporteur
Stéphane Cotin	Chargé de Recherche	INRIA	Lille	Examinateur
Laurent Daudeville	Professeur	UJF	Grenoble	Examinateur
Christophe Prud'Homme	Professeur	UJF	Grenoble	Examinateur

Table des matières

Table des figures

chapitre 1

Introduction

Nous verrons que l'intelligence humaine se sent chez elle tant qu'on la laisse parmi les objets inertes, plus spécialement parmi les solides, où notre action trouve son point d'appui et notre industrie ses instruments de travail, que nos concepts ont été formés à l'image des solides, que par là même, notre intelligence triomphe dans la géométrie, où se révèle la parenté de la pensée logique avec la matière inerte[...]
Henri Bergson

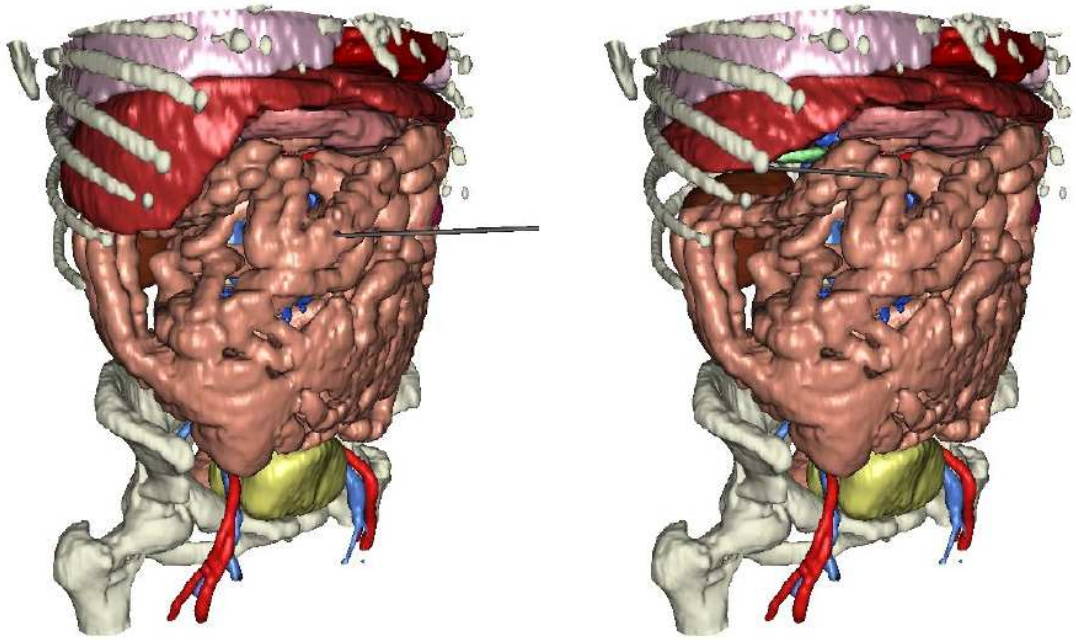


FIG. 1.1 – Simulation interactive complexe.

Des organes reconstruits d'après des images médicales interagissent et répondent aux actions d'un utilisateur. L'outil pousse le lobe du foie vers le haut, dégageant finalement la vésicule biliaire.

Les modèles physiques sont passionnants car ils sont une image de la réalité telle qu'elle vit et s'anime. En même temps, ce sont de pures créations intellectuelles, reflets de notre subjectivité, de nos désirs et de nos capacités. Ils interviennent dans quantités d'applications informatiques, par exemple : synthèse d'animations réalistes pour le cinéma ou le jeu vidéo, prototypes virtuels en CAO, outils de planification et d'entraînement en médecine. Depuis Archimède, les physiciens ont créé une grande variété de modèles mathématiques pour décrire la nature. La dynamique est bien connue depuis Newton. Les corps rigides sont étudiés depuis l'antiquité. Les milieux continus sont étudiés depuis des siècles, et la littérature qui continue s'accumuler sur la mécanique des fluides et des solides déformables est impressionnante. D'où vient que les modèles physiques, compris et étudiés depuis si longtemps, soient si peu présents dans les applications graphiques ? Pourquoi tant d'images et si peu de physique ?

Reconnaissons que les modèles physiques sont complexes à mettre en oeuvre et ne produisent pas encore beaucoup de résultats impressionnants, par rapport aux méthodes d'affichage qui ont fait de gigantesques progrès. Les équations physiques du rendu sont pourtant fort complexes. Les lois de l'optique mettent en jeu des interactions entre de très nombreux objets. Toutefois, par la peinture, la photo, le cinéma et la bande dessinée, nous sommes habitués à une infinie variété de styles graphiques, et nous admettons facilement des images physiquement erronées pourvu qu'elles véhiculent du sens. D'un point de vue physique, des simplifications très importantes peuvent alors passer pour un effet de style, et permettre des accélérations considérables. On peut alors organiser les calculs en un "pipeline" efficace, avec un temps de calcul proportionnel à la quantité de primitives géométriques. Les objets sont décrits par des surfaces dont les éléments peuvent être traités séparément. Les choses se compliquent quand on veut représenter des volumes (rendu volumique, effets athmosphériques) ou des interactions (ombre, reflet), mais l'essentiel de ces effets peuvent souvent être capturés par des précalculs ou par quelques bouclages du pipeline.

En mécanique, les objets sont intrinsèquement volumiques, et en interaction permanente avec leurs voisins. Nous admettons beaucoup moins de variété de style avec le mouvement physique. Hors du dessin animé, qui a popularisé une "cartoon physics" expressive et amusante, nous nous attendons généralement à voir "le vrai" mouvement. Le traitement des interactions est intrinsèquement long. Une difficulté supplémentaire est que la mécanique s'inscrit dans le temps, et que des erreurs peuvent s'accumuler à chaque

image. Il sera intéressant de suivre l'évolution de la physique dans les jeux vidéo. Ira t-elle vers le réalisme, le cartoon, au encore autre chose ?

Considérons la simulation en figure 1.1, qui représente une simulation médicale interactive. Chaque objet est représenté par des particules, qui échangent des forces suivant des lois propres aux matériaux dont les objets sont constitués. La résolution (finesse de discrétisation) des objets doit être adaptée aux effets recherchés. Plus on utilise de particules, et plus les mouvement peuvent être détaillés, mais plus les calculs sont longs. Il est important de trouver un bon compromis, et souvent utile de pouvoir raffiner là où c'est nécessaire, et simplifier là où c'est possible. Cela pose la question, encore très ouverte, du contrôle de la résolution, en relation avec le choix du modèle. Pour rendre les modèles physiques utilisables en pratique, il reste à développer des modèles dont on contrôle facilement la résolution.

Les objets sont visualisés par des surfaces dont les sommets ne sont pas les degrés de liberté indépendants, mais leurs sont attachés. Ces surfaces constituent des modèles "auxiliaires" dont les degrés de liberté ne sont pas indépendants, car entièrement définis par ceux du modèle mécanique, et qui doivent être remis à jour après chaque mouvement. La spécificité du domaine graphique est que les objets sont souvent définis d'abord par leur modèle visuel, dont on déduit ensuite un modèle mécanique. Les modèles visuels sont généralement beaucoup trop détaillés pour être directement utilisables en mécanique. De plus, ils ne représentent que des surfaces, alors que les modèles physiques sont généralement volumiques. Enfin, la modélisation mécanique nécessite des connaissances physiques relativement poussées. Pour rendre les simulations physiques facilement utilisables dans le domaine graphique, il faut donc des techniques de création automatique de modèles mécaniques d'après des modèles visuels.

Les objets interagissent à travers des modèles géométriques de contact, souvent des maillages. Ceux-ci ne peuvent pas être toujours identiques aux surfaces d'affichage, encore pour des problèmes de complexité. Toutefois, ils doivent en être suffisamment proches pour ne pas générer d'artefacts visibles. Ces modèles doivent être attachés au modèle mécanique, comme les surfaces d'affichage, mais il faut en plus qu'ils soient capables de transmettre les vitesses et les forces. Un grande variété de modèles de collision existe, ce qui complique l'implémentation et le choix des méthodes de calcul des réactions. De nombreuses méthodes ont été proposées, mais pas forcément combinables avec tout modèle de collision ni tout modèle mécanique. Par exemple, les méthodes résolvant des contraintes entre paires de particules ne s'appliquent pas directement aux modèles auxiliaires, dont les degrés de liberté ne sont pas indépendants. Enfin, les équations de contacts sont non linéaires (inégalités) et peuvent être complexes (frottement de Coulomb). Elles nécessitent des méthodes de résolution robustes, qui sont moins rapides que les solveurs linéaires. Pour développer l'utilisation des simulateurs physiques, le traitement des contacts reste à rendre plus automatique (création de modèles de contact d'après les modèles visuels), plus générique (méthodes s'appliquant à tous types de modèles), et plus efficace.

Le problème du contact soulève la question, plus générale, du couplage. Il existe d'excellents simulateurs de solides rigides, de solides déformables (généralement spécialisés dans un type de géométrie linéique, surfacique ou volumique), de fluides (eulériens ou lagrangiens), mais pratiquement pas de simulateurs génériques. Or, la plupart des scènes réelles ne sont pas composées d'un type unique d'objet, par exemple des solides ou des fluides. Pour simuler des objets de natures diverses en interaction, on est la plupart du temps obligé d'écrire un programme *ad hoc* implémentant un cas particulier. C'est pourquoi l'utilisation des simulations physiques reste principalement confinée à des applications dédiées à des études spécialisées. Même si l'on parvient à résoudre la plupart des cas particuliers de couplages aux contacts précédemment évoqués, il reste construire une architecture logicielle capable de combiner les modèles et appliquer automatiquement les couplages pertinents.

Un obstacle majeur à l'utilisation des simulateurs physiques est la lenteur des calculs. Indépendamment des performances des modèles et méthodes, et du soin apporté à leur implémentation, on peut faire moins de calcul en appliquant de plus longs pas de temps. C'est possible dans le cas des systèmes élastiques à l'aide d'intégration implicite. C'est plus complexe pour les contacts. La discontinuité du temps simulé et les corrections instantanées de position créent des configurations non physiques, comme des interpénétrations d'objets ou des retournements de tétraèdres, et de nombreuses approches ne résistent pas à des violations trop importantes. En partant d'une configuration juste, il est possible de rester dans des limites acceptables en réduisant les incréments, au moyen de retours en arrière ou de prédictions. Hélas, le temps de calcul nécessaire pour incrémenter le temps simulé d'une valeur donnée devient alors imprévisible, ce qui est inacceptable dans les applications interactives. En permettant d'appliquer des longs pas de temps,

la robustesse aux configurations non physiques est donc un élément clé pour l'efficacité des simulateurs. Il reste probablement beaucoup à faire dans ce domaine.

Une autre façon d'accélérer les calculs est de les paralléliser, surtout depuis que les constructeurs de processeurs ont renoncé à la course à la fréquence, au profit d'architectures parallèles. Non seulement la simulation physique, mais tous les domaines de l'informatique sont ainsi condamnés à effectuer dans les prochaines années une transition vers la programmation parallèle, afin d'exploiter au mieux le matériel. Trois niveaux de parallélisme sont actuellement disponibles : les grappes, les machines multicoeurs et les coprocesseurs parallèles (cartes graphiques). Les grappes permettent de traiter parallèlement des objets très divers, interagissant peu entre eux. Les machines multicoeurs traitent plus efficacement les interactions mais sont limitées en taille. Elles sont plus génériques que les coprocesseurs, qui excellent dans les grilles et les structures régulières. Les simulateurs du futur devront exploiter ces trois niveaux de parallélisme de la manière la plus transparente possible à l'utilisateur.

Nous pouvons identifier quelques critères de qualité d'une modélisation mécanique :

Simplicité de modélisation des objets (choix de la résolution, indépendance des modèles mécanique, géométrique et visuel), et de déploiement des calculs sur le matériel disponible

Généricité (n'importe quel objet peut interagir avec n'importe quel autre, indépendance solver/modèle)

Efficacite (rapidité, compromis réglable entre précision et temps de calcul, robustesse,)

Travaux effectués

Mes recherches visent à améliorer la performance et l'utilisation des modèles mécaniques, en explorant principalement les directions suivantes :

- modèles internes pour simuler la dynamique des objets déformables
- modèles de Contact (aussi indépendants que possible du modèle interne) pour détecter et établir des contacts entre objets
- architecture logicielle pour représenter et animer des scènes complexes
- implantation parallèle pour accélérer les calculs

Depuis ma prise de fonctions en 1999, j'ai encadré seul ou avec des collègues quatre thèses de doctorat (Florence Zara, Laks Raghupathi, Matthieu Nesme, Everton Hermann), dix stages de DEA, et de nombreux autres étudiants et ingénieurs. La plupart des travaux effectués concernent les directions évoquées ci-dessus.

Avec Matthieu Nesme, dont j'ai co-encadré le DEA et la thèse [Nes08] avec Yohan Payan de 2004 à 2008, nous avons d'abord travaillé sur l'efficacité des éléments tétraèdriques [NPF05] en améliorant la rapidité des calculs de matrices et la robustesse aux inversions. Puis nous avons montré une nouvelle façon d'utiliser des grilles régulières à la place des traditionnels tétraèdres. Ces modèles simplifient beaucoup la modélisation mécanique, tout en tenant compte de la répartition de matière et de la nature des matériaux [NPF06]. Une attention particulière a été portée à la validation [NMP+05]. Pour améliorer l'efficacité, nous avons finalement exploré les modèles multirésolution pour accélérer les calculs [NFP06]. Ces travaux sont présentés dans le chapitre 2.

Avec Laks Raghupati, dont j'ai encadré la thèse [Rag06] avec Marie-Paule Cani, ainsi que d'autres étudiants nous avons travaillé sur la détection en temps-réel des contacts en proposant une approche stochastique permettant de régler un compromis entre temps de calcul et complétude de la détection. Elle s'applique aux objets surfaciques et volumiques, a donné lieu à plusieurs publications [FAM+02, RCFC03, RGF+04, FLA+05] et nous l'avons encore améliorée par une approche hiérarchique [KNF04]. Nous avons ensuite cherché à améliorer la robustesse de la détection et du traitement des contacts [Rag06]. D'autre part, face à l'explosion combinatoire des méthodes de détection de collision entre les différents modèles géométriques, nous avons cherché à améliorer la généricité en proposant une méthode dans laquelle chaque modèle se projette dans une structure de données commune, une grille eulérienne de densité [FAN07]. Elle permet effectivement de faire interagir la plupart des modèles mais demeure lente car basée sur une structure volumique. Nous nous sommes donc tournés vers des approches basées sur les surfaces, et avons proposé une méthode pour améliorer la robustesse des réactions aux intersections profondes entre modèles déformables décrits par leurs surfaces [HFR08]. Finalement, nous avons combiné les avantages des deux approches précédentes à l'aide d'une nouvelle méthode à base d'images [FBAF08]. Ces travaux sont présentés dans le chapitre 3.

Les architectures logicielles permettant d'accroître la modularité et les performances des simulateurs ont toujours été un de mes centres d'intérêt. Avec plusieurs chercheurs de différentes équipes, nous avons développé une librairie de simulation modulaire et efficace pour faciliter la simulation médicale, SOFA [SOF, ACF+07, FAC+07]. Elle permet de modéliser les objets par assemblage de composants simples, de les hiérarchiser en modèle principal et auxiliaires, de détecter les contacts et d'appliquer des couplages efficaces. Au cours de la thèse de Florence Zara [Zar03], nous avons exploré la parallélisation des calculs sur grappes de PC. Nous avons identifiés les principaux problèmes posés par le cadre d'application en temps réel et proposé des solutions novatrices [ZFV02, ZFV04, ZG03]. Nous avons également couplé le simulateur parallèle avec des applications externes de visualisation et interaction [ARZ04, ZVF03]. La thèse d'Everton Hermann s'inscrit en continuité de ces travaux, en s'adaptant aux nouvaux contextes matériel et logiciel. Ces travaux sont présentés dans le chapitre 4.

La conclusion est tirée en section 5. L'annexe A présente des travaux anciens et des enseignements que j'en avais tiré pour la suite.

chapitre 2

Modèles mécaniques

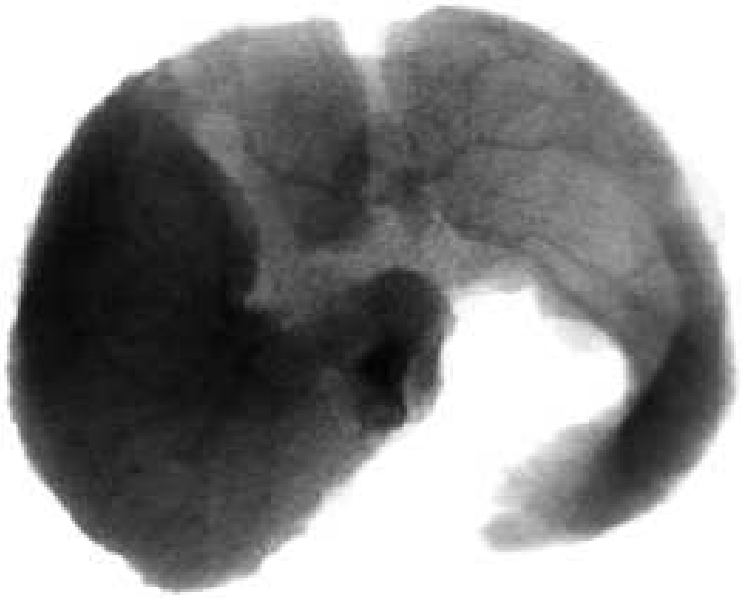


FIG. 2.1 – Modèle mécanique déformable.

Grille régulière d'éléments hexaédriques instanciés d'après des images médicales. Rendu volumique par textures déformées.

Ce chapitre reprend de larges extraits de la thèse de Matthieu Nesme [Nes08]. La première section est une introduction au domaine et une analyse des difficultés. La section 2 résume son travail de M2-R sur des modèles tétraèdriques robustes. La section 3 sert à motiver la suite du travail, centré sur la modélisation mécanique par grilles régulières. Puis les sections 4, 5, 6 résument nos contributions.

1 Introduction

L'animation physique en informatique est un domaine très large qui concerne un grand nombre de variétés d'objets et de comportements. Parmi les grandes familles de modèles physiques, on compte les corps rigides, les solides articulés, les solides déformables, les fluides (gaz et liquides). Chacune peut impliquer des lois très particulières reflétant des besoins très variés, et des algorithmes très différents. Leur unique point commun est d'implémenter les lois de la dynamique énoncées par Newton afin de créer un système dit *générateur*, dont l'état final résulte de lois appliquées à un état de départ pendant un cours laps de temps sans nécessairement d'intervention externe durant l'application des lois. Les lois physiques choisies peuvent aller des plus simples aux plus complexes. Dans l'encore plus vaste domaine de l'animation en informatique, l'animation physique se distingue clairement des approches purement cinématiques telles que l'animation par squelettes suivant des positions clés, principalement utilisée pour l'animation de personnages.

Parallèlement à l'animation de ces corps, divers domaines sont à prendre en considération. Par exemple, pour faire interagir plusieurs objets entre eux et avec leur environnement, il faut être en mesure de gérer les contacts, qui passent par l'étude de la gestion des collisions. Celle-ci consiste en la détection d'une intersection de matières, qu'il faut être capable de modéliser pour enfin y réagir.

La physique ne sert pas qu'à créer des animations visuellement réalistes. Elle ajoute une nouvelle dimension d'interactivité en réalité virtuelle en offrant plus de sensations réalistes pour une meilleure immersion. En plus du visuel, des retours haptiques [BJ71, BC94] ou sonores [Cad99, vdDKP01, TDLD07] peuvent aussi être modélisés en suivant des lois de la physique.

Tous ces domaines sont très complémentaires et doivent tous être étudiés et réunis dans l'idée de faire un "simulateur ultime". Chaque domaine est très pointu, mais peut être étudié assez indépendamment du fait de la diversité des lois mises en œuvre. On constate ainsi de nombreux grands thèmes de recherche très spécialisés en animation physique.

Dans ces travaux on ne s'intéresse qu'aux déplacements et déformations des solides déformables, et principalement pour leur rendu visuel, les parties haptique et sonore n'ayant pas été abordées.

Dans ce chapitre nous allons passer en revue les différentes méthodes proposées pour simuler des corps déformables.

1.1 Critères de qualité

L'animation physique peut avoir des objectifs très différents, allant du très simple pour faire un objet "plausible" aux simulations du réel avec une erreur maximale garantie. Afin de faire un état de l'art critique quand à la cohérence des méthodes existantes pour notre problématique de réalisme interactif, on doit tout d'abord mettre en avant nos propres critères de qualité.

Ces critères se basent essentiellement sur les propriétés qui nous semblent essentielles : la précision, la vitesse, la robustesse, la liberté du rendu visuel, la facilité de prendre en compte des géométries quelconques. De nombreux autres critères secondaires sont bien évidemment à prendre en compte, tels que la facilité d'implémentation, la possibilité d'adapter un algorithme de coupure ou de gérer facilement les collisions.

En terme de **précision** dans l'animation, on va différencier des grandes familles. Tout d'abord les modèles qui permettent de représenter un objet dont le mouvement est physiquement plausible, sans rigueur au niveau de la précision, tel que cela peut être le cas dans les jeux. Sans chercher la rigueur physique, il est possible à partir de lois simples de faire émerger des phénomènes complexes qui donnent illusion de la réalité [LC84]. Dans ce contexte, on va commencer à être satisfait si le modèle respecte une certaine plausibilité physique, sans garantir que c'est exactement le comportement qui serait arrivé, mais qui offre un comportement qui aurait pu arriver. À l'opposé, on va considérer les modèles très rigoureux mais complexes, qui tentent de se rapprocher le plus possible de la réalité.

Un point très important à considérer, est la possibilité de **paramétrer les matériaux** en intégrant des mesures expérimentales issues de la mécanique afin de facilement modéliser des matériaux connus, et des objets composés de plusieurs matériaux.

La **vitesse** de calcul est bien sûr un élément crucial du modèle désiré. Il doit être suffisamment rapide pour proposer des animations interactives. Offrir un compromis entre précision et vitesse est intéressant afin de proposer la meilleure solution approchée dans un laps de temps limité.

Il faut aussi considérer la **robustesse** du modèle, c'est-à-dire, sa résistance à ne pas diverger dans les configurations extrêmes ou numériquement mal-conditionnées. Il faut même noter que des configurations non-physiques telles que des inversions de matière ou d'aplanissement total peuvent arriver, d'autant plus dans le cadre d'applications interactives, dans lesquelles un utilisateur peut appliquer des contraintes irréalistes. Il est important que la simulation reste stable, continue à fonctionner et tente de fournir la solution la plus adaptée pour pouvoir continuer la manipulation.

L'interaction finale avec l'utilisateur passant par des activités sensorielles, il est important que celles-ci soient le plus détaillées possible. Dans notre cas, on pense principalement à la **qualité du rendu**. En effet, un simulateur ne pourra pas se contenter d'une mécanique irréprochable, il devra être en plus visuellement réaliste. Cela signifie généralement d'avoir une résolution d'affichage détaillée sur laquelle pourront être appliquées toutes les techniques de rendus réalistes mises au point en synthèse d'images. Il est donc important que l'objet animé soit capable de s'afficher en utilisant une résolution différente de celle utilisée par la mécanique, qui soit plus adaptée à l'affichage.

Principalement dans le cadre des simulations chirurgicales, il est très important d'être capable d'animer facilement n'importe quel morphologie afin de faire des applications **spécifiques au patient**. Cela peut-être aussi très pratique en graphisme pour ne pas compliquer la tâche de l'animateur qui est avant tout un artiste et qui n'a pas forcément à comprendre ce qui va être bien adapté du point de vue de la mécanique. On va préférer aussi les méthodes permettant de travailler avec différents types de données ; dans le médical on pense aux images scanner qui sont les principales sources de données sur la morphologie d'un patient.

Un dernier critère nous semble important d'être souligné, même s'il n'a pas été étudié explicitement : il s'agit de la facilité d'adapter une méthode de **découpe**. Celle-ci implique généralement la refonte du maillage sous-jacent, et il est important de penser à comment va se comporter le modèle mécanique : va t-il encore pouvoir fonctionner ? Va t-on devoir refaire énormément de calculs coûteux pour se mettre à jour ?

Dans la suite, on va lister des méthodes respectant au moins un des critères ci-dessus, en regardant les avantages que l'on peut en tirer, et ce qu'il faudrait y ajouter pour qu'elles respectent d'autres critères.

1.2 Modèles physiques

Afin de rendre les déformations plus réalistes, plutôt que d'utiliser des lois de déformations empiriques, des auteurs se sont tournés vers les hypothèses des mécaniciens qui tentent depuis des dizaines d'années d'évaluer la déformation de solides. Ils ont ainsi donné naissance aux premiers modèles purement mécaniques.

Terzopoulos est le pionnier dans cette approche. Dès 1987, il a proposé le premier modèle déformable mécanique en synthèse d'images [TPBF87]. Pour cela, il a pris telles quelles les méthodes complexes de résolutions issues de la mécanique avec pour objectif de fabriquer des images. En terme de qualité de déformations, les résultats étaient déjà très bons puisque les équations de la mécanique sont respectées. Par contre, vu la puissance limitée des machines de l'époque, l'ambition n'était pas du tout tournée vers le temps réel, et peu d'optimisations algorithmiques sont discutées.

Depuis ce travail préliminaire, une multitude d'approches a été proposée pour déformer des objets avec différents objectifs, que peuvent être la rapidité, la diversité des comportements, la robustesse...

A cette étape, on se doit de souligner le fait qu'il faut faire attention à ne pas réinventer la roue, car déjà beaucoup de choses ont été proposées par les mécaniciens, qui travaillent sur cette thématique depuis très longtemps. Par exemple, le 'stiffness warping' de Müller [MDM$^+$02] se rapproche des méthodes dites corotationnelles en mécanique [Fel00], ou encore l'approche multirésolution de CHARMS [GKS02] s'apparente aux éléments finis dits hiérarchiques en mécanique [ZKGB82]. À notre avis, il semble très ambitieux de vouloir faire mieux que les mécaniciens pour prédire ces mouvements. Nous pensons qu'il est préférable de comprendre les lois déjà proposées, de les analyser en tentant de voir comment elles peuvent s'implémenter, en tenant moins de place en mémoire, en étant très rapides à résoudre en utilisant du parallélisme, *etc*. Notre travail consiste donc en partie à jouer avec l'aspect numérique et algorithmique afin d'intégrer au mieux ces lois dans l'ordinateur.

Fréquemment on peut voir la catégorie des modèles physiques découpée en modèles continus et modèles discrets (*i.e.* milieu continu *vs* masses-ressorts). D'un point de vue mécanique, un modèle est dit discret lorsque les équations peuvent être écrites indépendamment pour chaque particule discrétisant l'objet, alors que dans les modèles continus le comportement va dépendre de l'intégralité de la matière et les états peuvent être évalués à n'importe quel point de l'objet. Mais d'un point de vue algorithmique, la différence n'est pas si claire, car même avec des modèles issus de la mécanique des milieux continus, une pratique commune est de "lumper" les masses, c'est-à-dire de concentrer la masse sur les particules afin de pouvoir gérer chaque particule indépendamment. Avec une telle vision, les éléments finis deviennent proches des systèmes masses-ressorts qui sont pourtant fréquemment présentés comme les deux grandes méthodes radicalement différentes.

De plus, dans tous les cas, pour être modélisé, l'objet doit être discrétisé. Les degrés de liberté sur lesquels vont être appliquées les équations sont couramment appelées particules. L'approche la plus simple est de considérer que ces particules représentent elles-mêmes l'objet. Elles peuvent aussi être des points de contrôles d'une surface paramétrique. Par exemple, très rapidement, est née l'idée de mettre de la physique sur les points de contrôles d'une FFD (Free Form Deformation) [FvdPT97]. Cette idée continue d'être exploitée, par exemple dans [CGC$^+$02] mais aussi dans nos travaux. Les particules peuvent aussi représenter des squelettes définissant une surface implicite : solides déformables [Can93], solides inélastiques et fluides avec le cas des SPH (Smoothed Particle Hydrodynamics) [DC95, MCG03, CBP05].

Qu'il soit discret ou continu, on peut considérer un modèle physique comme un ensemble de degrés de libertés (ddl) qui vont interagir en obéissant à des forces internes. Des chargements externes peuvent être appliqué sur ce modèle, principalement la gravité, des champs de forces, des contraintes (telles que fixer un ddl, contraindre sa position sur une ligne ou limiter sa vitesse). Un modèle physique a aussi une loi d'évolution. Elle peut être statique, pour retrouver un état d'équilibre annulant les forces sans prendre en compte

ni les masses ni le temps. C'est ce qui est utilisé quand seule la configuration finale est intéressante comme en ingénierie mécanique et dans les planificateurs chirurgicaux. Mais dès lors que l'on cherche à avoir un modèle interactif avec une évolution dans le temps, il faut se tourner vers des méthodes dynamiques qui prennent en compte les accélérations et les vitesses. Numériquement, cela se résume à être capable de calculer deux choses distinctes : d'un côté des **champs de forces internes** en fonction de contraintes extérieures, et de l'autre un nouvel état après un court laps de temps à l'aide d'une méthode d'**intégration du temps**. C'est pourquoi dans cette section nous nous appliquons à lister les méthodes numériques déjà proposées en informatique graphique dans ces deux domaines.

1.3 Champs de forces

La force agissant sur une particule dépend de la déformation de l'objet dans un voisinage plus ou moins grand autour de cette particule. Elle peut dépendre de toutes les particules de l'objet (Lennard-Jones par exemple), ou de simplement quelques particules voisines. Ce voisinage peut être prédéfini par un maillage (comme pour les masses-ressorts ou les éléments finis) ou bien être évalué et varier au cours du temps (méthodes dites *meshless*).

Paires en interaction C'est une des première lois physiques informatisées de façons plus ou moins approximée [MP89, TPF91, LJC+91]. Elle est plutôt associée aux comportements de type fluide. Dans ce cas, chaque particule est indépendante mais influence et est influencée par toutes les autres. De par cette globalité des interactions, les forces sont assez lentes à calculer. Des optimisations sont possibles pour ne pas calculer des influences insignifiantes en prenant par exemple un voisinage gaussien significatif. Il n'est pas facile de représenter des solides élastiques avec cette méthode, plutôt adaptée aux objets "pâteux" du type fluides visqueux. D'un point de vue macroscopique, il est de même très difficile d'associer des paramètres physiques réels aux matériaux

Masses-ressorts Il s'agit de la loi la plus simple faisant intervenir un maillage, limitant l'influence d'une particule à son voisinage direct [BHG92, CEO+93]. Cette force est généralement rapide à calculer. Plusieurs lois élastiques sont possibles entre les particules voisines. On peut retenir qu'un ressort est un élément fini 1D plongé dans un univers 3D donc très peu adapté pour représenter des objets volumiques et même des surfaces. On ne sait pas lui assigner des propriétés tridimensionnelles mécaniques mesurées. Des méthodes ont permis de les rendre très stables, par exemple en limitant leur élongation [Pro95]. Très rapides et faciles à implémenter, ils sont donc très bien adaptés aux simulations en synthèse d'images de type jeux vidéo mais sont peu adaptés aux simulations précises de type médical. [BC00] donne une solution permettant de gérer l'isotropie dans des objets volumiques simulés par des ressorts, tout en précisant qu'il est impossible de retrouver les paramètres mécaniques pour les ressorts correspondant à ceux issus de la mécanique des milieux continus dans un contexte général. Tout de même, pour certains cas particuliers, [LSH07] montre que les ressorts peuvent être équivalents aux éléments finis. Notons aussi qu'il est possible d'améliorer le comportement volumique de cette représentation 1D. Des méthodes récentes permettent de mieux prendre en compte les volumes grâce à des "mémoires de forme" [PBP96] ou même en considérant l'objet rempli d'un fluide parfait dans lequel les variations de volumes fournissent des pressions transformées en forces à la surface [SML02, MO03].

Méthodes des éléments finis Pour une meilleure précision, on préfère les modèles suivant les lois issues de la mécanique des milieux continus. Les équations étant plus complexes, ils sont généralement plus longs à calculer, mais ils sont bien plus précis, et surtout leur précision est évaluable, car ils permettent de prendre en compte des propriétés mesurées de matériaux réels. Diverses méthodes numériques permettent de calculer ces lois, les plus célèbres sont la méthode des éléments finis [CDC+96, DCA99, PDA00, MDM+02] et la méthode de différences finies [TPBF87, DDCB01] .

Les grandes avancées concernant l'animation d'objets déformables avec les contraintes d'interactivité, de précision et de qualité de rendu peuvent être rappelées chronologiquement :

– [CDC+96] propose un simulateur quasi-statique interactif en évaluant de petites déformations par des combinaisons linéaires de déformations unitaires précalculées.

– [DCA99] fournit une écriture en "masses-tenseurs" qui optimise l'intégration dynamique explicite, et simplifie les mises à jours dans le cas de changements topologiques.
– [PDA00] étend cette représentation à un calcul non-linéaire des déformations afin de gérer les déplacements en rotation.
– [DDCB01] présente un modèle multirésolution.
– [MDM+02] donne un calcul corotationnel des déformations plus rapide et simplifiant l'utilisation d'intégrations implicites plus stables.

Meshless Il s'agit d'une méthode très élégante basée sur la mécanique des milieux continus qui permet de calculer les déformations en différents points de l'espace d'après les voisinages et leurs dérivées spatiales à chaque pas de temps, sans maillage volumique. Elle est très bien adaptée aux coupures et aux milieux fluides-élastiques. Par contre, elle est relativement longue à calculer, et n'est pas encore adaptée à des simulations interactives [MKN+04, PKA+05].

Coordonnées réduites par décomposition modale Dans cette approche [HSO03, BJ05], l'idée est d'exprimer les déformations de l'objet par un jeu de coordonnées réduites obtenues par analyse modale. Il est en effet possible en prétraitement d'étudier tous les modes vibratoires de l'objet et de ne conserver que les composantes principales. On obtient ainsi des jeux de déplacements de base qui donnent des déformations globales de l'objet. L'idée de ne considérer qu'un ensemble de déplacements globaux avait été initiée par [PW89, WW90]. En interpolant ces déplacements, on peut obtenir toute une palette complète de déformations. L'avantage est qu'avec ce jeu très réduit de coordonnées, il est très rapide d'obtenir une déformation plausible pour des mouvements globaux, quelle que soit la complexité du modèle de départ (quels que soit le nombre de degrés de liberté initial). L'inconvénient est qu'il est difficile d'obtenir des déformations très locales, les déformations sont réduites à la palette de déplacements globaux dont on dispose. Cela rend cette approche très profitable en synthèse d'animations, car un résultat très réaliste peut être obtenu rapidement.

Shape-matching Il s'agit du dernier modèle proposé en graphique interactif en date [MHTG05]. L'idée est découper l'objet en sous-groupes (grossiers hexaèdres englobants et interpolants). Pour chaque groupe, une transformation géométrique entre la position au repos et la position déformée est évaluée. Cette transformation peut être un recalage rigide, ou un recalage élastique plus coûteux mais plus précis. Des déformations sont déduites dans des repères comparables à partir des positions des points de l'objet au repos et déformés après le recalage. Des forces sont ainsi créées sur les points pour tenter de les ramener vers leur position non déformée. Une version superposant ces groupes permet encore plus de souplesse dans la deformation [RJ07]. Ces modèles sont visuellement très impressionnants par rapport à leur coût de calcul, même s'ils n'incluent pas de force de restauration de volume. Malheureusement, ils peuvent difficilement être considérés lorsque l'on désire une grande précision car la qualité de leur déformation et le paramétrage des matériaux sont complètement empiriques. Ils semblent être la proposition la plus adaptée aux jeux vidéo, de par leur souplesse, leur rapidité et leur stabilité.

> Parmi tous ces moyens de calcul des forces internes d'un objet déformable, le seul moyen vraiment précis et qui cherche à refléter la réalité en permettant la prise en compte d'un paramétrage de vrais matériaux est la mécanique des milieux continus, à travers la décomposition en éléments finis. C'est pourquoi nous nous sommes tourné vers elle, avec les contributions présentées en sections 2,3,4,5,6.

1.4 Intégration du temps

Un système dynamique de particules correspond à un problème du second ordre de la forme :

$$\boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{C}\dot{\boldsymbol{u}} + \boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}_{ext} \tag{2.1}$$

où $\boldsymbol{u}$ correspond aux déplacements et $\boldsymbol{f}_{ext}$ aux forces externes appliquées sur toutes les particules. $\boldsymbol{M}$ est lié à la masse, $\boldsymbol{C}$ à l'amortissement et $\boldsymbol{K}$ à la rigidité, elles traduisent les interactions entre les particules.

Le temps est traité comme un paramètre. Les dérivées première $\dot{\boldsymbol{u}}$ et seconde $\ddot{\boldsymbol{u}}$ des déplacements représentent respectivement vitesses et accélérations. Les matrices $\boldsymbol{M}$, $\boldsymbol{C}$ et $\boldsymbol{K}$ s'obtiennent en regroupant les équations en chaque sommet ; elles peuvent dépendre de la géométrie et donc devoir être recalculées à chaque déformation. Le calcul des forces internes $\boldsymbol{Ku}$ dépend de la méthode utilisée, présentée en 1.3.

Plus généralement les forces internes peuvent être définies non-linéairement $\boldsymbol{f}_{int}(\boldsymbol{u})$; la résolution du système fait toutefois introduire les variations infinitésimales de forces représentées par les matrices $\boldsymbol{K} = \frac{\partial f}{\partial u}$ et $\boldsymbol{C} = \frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{u}}}$.

On résout donc un problème à chaque pas de temps, en partant d'un instant initial t_0 où les déplacements $\boldsymbol{u}(t_0)$ et les dérivées premières $\dot{\boldsymbol{u}}(t_0)$ sont supposées connues : ce sont les conditions initiales.

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

$$\boldsymbol{q}(t+h) = \boldsymbol{q}(t) + \int_t^{t+h} \dot{\boldsymbol{q}}(\boldsymbol{q}, t)dt$$

où $\boldsymbol{q}(t) = \begin{bmatrix} \boldsymbol{u}(t) \\ \dot{\boldsymbol{u}}(t) \end{bmatrix}$ représente l'état (position et vitesse) d'une particule au temps t et h le pas de temps.

Il existe quantité de schémas d'intégration, qui se distinguent par leur ordre de précision, leur convergence, leur stabilité et leur coût en terme de temps de calcul. Tous sont approximatifs car on ne sait pas calculer l'intégrale de la formule dans le cas général. La convergence signifie que plus le pas de temps h se rapproche de 0, plus la solution analytique se rapproche de la solution réelle. On peut distinguer deux grandes classes de méthodes :

méthodes explicites les dérivées premières et secondes déduites de l'état courant sont utilisées pour calculer la valeur suivante du champ inconnu. La stabilité de la solution demande généralement des pas de temps très petits.

> **Exemple :** le schéma d'Euler explicite fait l'approximation que la dérivée tout au long du pas de temps est égale à sa valeur au début du pas (la vitesse ne varie pas au cours d'un pas de temps) :
>
> $$\boldsymbol{q}(t+h) \approx \boldsymbol{q}(t) + h\dot{\boldsymbol{q}}(\boldsymbol{q}, t)$$

méthodes implicites les dérivées premières et secondes en fin de pas sont utilisées pour calculer la valeur suivante. La stabilité est meilleure, mais chaque pas de temps exige la résolution d'un système d'équations, souvent effectuée par un calcul itératif, avec les problèmes associés à tout calcul itératif : les paramètres algorithmiques et le choix de critères d'arrêt.

> **Exemple :** le schéma d'Euler semi-implicite décrit dans [BW98] (avec l'amortissement de Rayleigh, combinaison linéaire de la raideur et la masse $\boldsymbol{C} = \alpha\boldsymbol{K} + \beta\boldsymbol{M}$) :
>
> $$\begin{cases} ((1-h\beta)\boldsymbol{M} - h(h+\alpha)\boldsymbol{K})\Delta\dot{\boldsymbol{u}} = h(\boldsymbol{f}(t) + h\boldsymbol{K}\dot{\boldsymbol{u}}) \\ \dot{\boldsymbol{u}}(t+h) = \dot{\boldsymbol{u}}(t) + \Delta\dot{\boldsymbol{u}} \\ \boldsymbol{u}(t+h) = \boldsymbol{u}(t) + h\dot{\boldsymbol{u}}(t+h) \end{cases}$$

Ainsi, les méthodes explicites avancent directement vers l'inconnu d'après les états actuels pour fournir une solution assez rapidement. Par contre la stabilité n'est pas garantie, et les erreurs s'accumulant peuvent faire diverger le système. C'est pourquoi il est nécessaire d'utiliser des petits pas de temps car plus l'intervalle d'intégration est petit, moins l'erreur sera grande. Les pas de temps doivent par ailleurs être inversement proportionnels à la racine carrée du matériau simulé, plus le matériau est raide, plus le pas de temps doit être petit [Bel76, Mil05] (la condition de Courant impose que $h < d_{min}\sqrt{\frac{\rho}{\lambda+2\mu}}$ avec d_{min} la

plus petite distance entre deux particules voisines, ρ la masse volumique du matériau, et $\lambda + 2\mu$ relatif à la raideur du matériau *cf.* section 1.3).

Ces méthodes sont donc peu adaptées aux matériaux raides dans le cadre de simulations rapides. De l'autre côté, les méthodes implicites regardent en avant pour avancer, en choisissant un état d'arrivée compatible avec l'état courant. Elles sont ainsi inconditionnellement stables mais aussi plus lentes à calculer car elles demandent la résolution d'un système d'équations. Dans ce cas, la taille du pas de temps est moins un problème et de grands pas de temps et des matériaux raides peuvent être simulés. Par contre, dans ce cas, le système est plus difficile à résoudre, et obtenir une bonne solution approchée coûte cher, les méthodes de résolution convergeant lentement. Tout de même, un avantage des méthodes implicites est la possibilité de choisir la qualité de la solution approchée, tout en étant toujours stable. Plus la solution est approximative, plus le calcul est rapide et la simulation interactive est possible, mais plus le comportement des matériaux est ramolli en ne répondant plus assez rapidement aux contraintes. De plus, l'erreur due à l'approximation des solutions induit une certaine viscosité.

Il n'existe aucune recette miracle pour définir quel est le meilleur intégrateur, chacun ayant ses caractéristiques, le mieux est de les comparer concrètement pour en déduire le plus adapté en fonction des besoins. Pour donner des exemples, dans le cadre de simulation de crash tests automobile, ce sont des intégrateurs explicites qui sont utilisés [KSS01]. Vu la rigidité des matériaux, de très petits pas de temps sont utilisés. Quand aucune contrainte de temps de calcul est fixée, il semblerait qu'avec de très petits pas de temps, ce soient les méthodes explicites qui offrent les solutions les plus proches d'un comportement réel (Runge-Kutta2/MidPoint offrant la meilleure rapidité pour une précision donnée). Dans la communauté graphique, l'intégration implicite a été initialement utilisée [TPBF87]. Mais à cause de sa grande complexité due à une résolution très lente, les méthodes explicites lui ont été préférées pour le temps réel, se limitant à des matériaux mous [DCA99, DDCB01]. Depuis que des méthodes de résolutions rapides ont été proposées, l'intégration implicite a retrouvée une place privilégiée dans l'animation interactive grâce à sa stabilité qui ne nécessite pas d'adapter le pas de temps [BW98, DSB99, HE01, VMT05].

Un grand nombre de méthodes d'intégration du temps sont détaillées et comparées dans [EEH00, VMT01]. On peut se rendre compte que dans tous les cas, rien n'est bien adapté pour simuler précisément et rapidement des matériaux mal conditionnés (tels que raides ou quasi-incompressibles). Dans le cas explicite il faut faire énormément de petits calculs (car petits pas de temps) ; du côté implicite il faut calculer une coûteuse solution approchée correcte (demandant beaucoup d'itérations pour converger).

1.5 Améliorations

Dans le cas de simulations interactives, il est très important d'assurer une complexité minimale par rapport au problème.

La complexité d'un modèle physique dépend des éléments suivants :

- **la taille du système** : autrement dit le nombre de degrés de liberté
- **les lois d'interactions** : la construction du système traduisant les interactions entre les degrés de liberté peut être plus ou moins complexe (par exemple des ressorts sont plus faciles à traduire que des éléments finis), et le système en résultant peut être plus ou moins délicat à résoudre (linéaire ou non, conditionnement...)
- **les algorithmes de résolution** : une fois le système en place, l'algorithme calculant sa solution a une place très importante dans la complexité qui dépend de la convergence de la résolution. Certains algorithmes sont plus rapides, mais ne correspondent qu'à certains systèmes. Par exemple, concernant les systèmes linéaires symétriques, Gauss-Seidel ne fonctionne bien que sur des matrices à diagonale dominante. Si la matrice est définie positive, le gradient conjugué sera plus performant.

Par conséquent, il est possible de mettre en œuvre des procédés algorithmiques permettant d'accélérer les calculs relativement orthogonalement aux formulations des calculs des forces et d'intégration du temps. Même si elles sont très importantes, voire indispensables, du fait que ces méthodes peuvent relativement être appliquées à une grande variété de modèles, elles apparaissent à nos yeux comme des

améliorations. Une scène bien modélisée aura des degrés de liberté et des interactions de nature adaptée (modèles hybrides), un nombre de degrés de liberté raisonnable et bien placé (multirésolution, adaptativité), des méthodes de résolution à la meilleure convergence possible (méthodes hiérarchique, multigrid, parallélisme).

1.5.1 Modèles hybrides

Il est évidemment possible de mélanger plusieurs modèles mécaniques afin d'animer différents objets ensemble, mais ce mélange est possible pour un même objet. L'objectif principal est d'économiser en coût de calcul en utilisant la méthode adaptée la moins coûteuse. En interne, plusieurs modèles sont alors utilisés. Les couches peuvent être choisies au départ suivant des connaissances à priori : si on sait que telle partie est un os, on l'animera comme un solide, et la peau comme du déformable. Mais elles peuvent aussi être adaptatives et ainsi varier en fonction de critères. [CDA00] propose de simuler un objet déformable comme un corps rigide tant qu'il n'est pas déformé pour économiser des calculs, puis de résoudre ses petites déformations par la méthode des éléments finis linéaires et enfin d'utiliser un calcul non-linéaire des déformations lorsqu'elles sont importantes. [SSIF07] propose une méthode efficace pour coupler des objets déformables à des solides.

1.5.2 Multirésolution

La multirésolution est une approche classique en synthèse d'images, et en informatique en général, permettant de concentrer les calculs là où ils sont le plus nécessaires ou de manipuler à différentes échelles en raffinant les zones d'intérêt.

Nous proposons une classification, utilisant les termes relatifs à la multirésolution trouvés dans la littérature. Un schéma récapitulatif est donné en Figure 2.2.

Pour nous le terme **multirésolution** est le plus général et qualifie toutes méthodes utilisant plusieurs résolutions. Que la résolution soit fixe ou varie au cours de l'animation.

Une méthode **adaptative** est une méthode multirésolution pour laquelle les niveaux de résolutions utilisés varient au cours de l'animation, s'adaptant en fonction de critères. Ces critères peuvent être variés. Le plus classique en animation d'objets déformables est celui qui raffine les parties où les contraintes ou les courbures sont fortes [DDCB01]. [WDGT01] compare plusieurs critères basés sur les courbures, les états internes de déformations ou de contraintes. Dans le cas particulier où le critère concerne les ressources du système, la méthode est appelée **temps critique** et essaie de raffiner tant que les ressources permettent de respecter une limite de temps. On retrouve classiquement ce type de critère en détection de collisions [DO00, KZ03, MO05, MO06], mais aussi en animation de corps déformables [DMG05]. Il faut noter que pour le temps réel, l'adaptatif demande une structure très appropriée, permettant un changement de résolution rapide, sans latence.

La notion d'approche **hiérarchique**, à la différence de **nodale**, signifie que plusieurs niveaux de résolutions sont utilisés en parallèle pour résoudre le problème. Dans l'approche nodale plus classique, au contraire, seul le niveau le plus fin est considéré. Le hiérarchique traite donc le problème à un niveau nodal et global simultanément, ce qui permet de converger plus rapidement vers la solution, en gérant à la fois des déformations globales et locales.

La résolution **multigrid** est une approche hiérarchique particulière pour laquelle plusieurs niveaux de résolutions sont utilisés en suivant des règles spécifiques où l'idée générale est d'utiliser les niveaux plus grossiers comme correcteurs du niveau fin. Plus de détails concernant le multigrid seront présentés en section 3 et suivantes.

Les modèles **multi-couches** réfèrent généralement aux modèles hybrides utilisant différents modèles en interne, comme présenté en 1.5.1.

Dans la littérature, on retrouve plusieurs travaux multirésolution proches du domaine de l'animation physique de corps déformables, les plus notables sont décris ci-après.

[GD01] a proposé des interpolations hiérarchiques pour les FFD ; même si ces travaux ne touchent pas directement l'animation adaptative, le schéma de raffinement peut sûrement être exploité. [CF97] est le premier à animer des objets à différentes échelles. Il propose de continuer à faire vivre très grossièrement

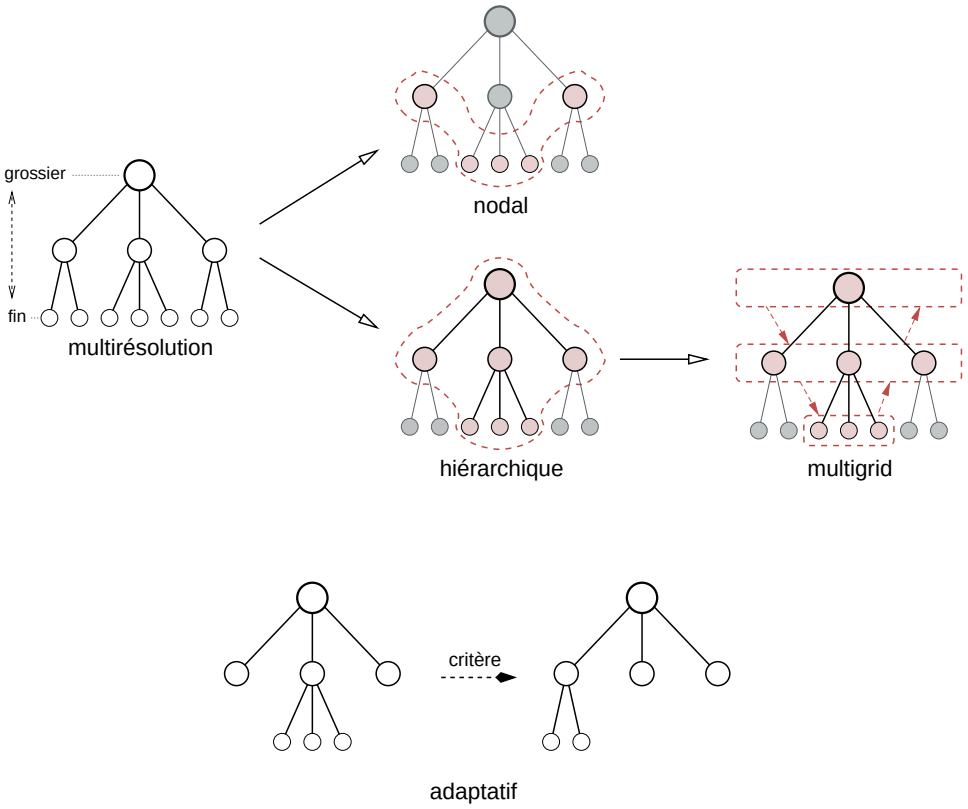


FIG. 2.2 – Diverses déclinaisons de la multirésolution

un objet qui n'est plus dans le champ de vision, afin que s'il y revienne, son état ait quand même évolué de façon cohérente tout en limitant les coûts.

En animation physique d'objets déformables, [HPH96, HH98] ont raffiné des surfaces 2D animées par des systèmes masses-ressorts. [GCS99, GCMS00b] a proposé une subdivision de systèmes masses-ressorts en trois dimensions, à partir d'arbres octaux puis de subdivisions tétraédriques, tout en essayant de préserver la cohérence des propriétés mécaniques à tous les niveaux.

La technique a été portée pour les éléments finis tétraédriques, en utilisant plusieurs maillages volumiques précalculés représentant l'objet à plusieurs résolutions [WDGT01]. [DDCB01] étend cette méthode en ajoutant le traitement de points fantômes interpolés, qui permettent d'utiliser des maillages tétraédriques non-emboîtés. Il ajoute aussi un pas de temps adaptatif. Son critère d'adaptation se déroule au cours du temps de façon géométrique, le raffinement s'effectuant là où les courbures des déplacements sont les plus fortes. Ce même critère d'adaptativité a été repris dans [DMG05] pour un octree d'éléments finis englobant l'objet déformé. Dans ce travail, ont en plus été ajoutés des critères concernant les ressources système, afin de permettre un taux de calcul minimal pour garantir l'interactivité. L'approximation des solutions données pour chaque pas de temps est donc variable et dépend de la difficulté à résoudre le système.

Une décomposition en ondelettes des fonctions d'interpolation d'éléments finis a été présentée dans [GKS02], à la manière des éléments finis hiérarchiques. Elle propose une approche très générique de la multirésolution, en limitant les problèmes soulevés aux jointures de maillages de différentes tailles.

Afin d'exploiter au mieux les différents niveaux de résolution, des approches multigrid ont été proposées sur des maillages tétraédriques non-emboîtés. [GW06] propose cette approche à la fois pour un calcul linéaire, corotationnel et non-linéaire des déformations. [OGRG07] propose une méthode multigrid adaptative basée sur [Bra77] pour laquelle il intègre une méthode de détection de collisions, afin de vrai-

ment exploiter la représentation hiérarchique, et propose ainsi un modèle d'animation multirésolution assez complet.

Dans tous ces travaux, les principaux problèmes liés à la multirésolution soulevés sont :
- la cohérence des propriétés mécaniques à chaque niveau. Les raideurs, masses... doivent simuler un comportement semblable (à une précision près) à tous les niveaux.
- la présence de nœuds en T aux jointures entre différents niveaux de résolution, car ces nœuds ne sont pas des ddl libres et subissent des contraintes d'interactions particulières.
- les transitions d'un niveau à l'autre dans le cas de méthodes adaptatives. Il est en effet important de trouver un critère qui lissera les changements de résolutions, et qui fera intervenir des opérations de passage peu coûteuses pour éviter les ralentissements.

Sont plus rarement soulevés les problèmes liés à la fabrication des maillages, étape pourtant déjà bien compliquée pour obtenir un seul niveau de résolution. Dans [GKS02, GW06], les niveaux fins sont obtenus en raffinant les niveaux grossiers, ce qui fait perdre à notre sens le réel intérêt d'une méthode multirésolution pour le temps réel. L'idée étant plutôt de simplifier certains calculs, donc de rendre certaines parties du système plus grossières, plutôt que d'encore plus complexifier certaines zones. Ce problème de maillage est pour nous un des facteur très limitant des méthodes multirésolutions en animation actuellement, même si [CGC+02, DMG05] ont ouvert la voie vers des maillages à différentes résolutions plus automatiques. [CGC+02] représente les objets comme étant des surfaces de subdivision interpolées dans des "éléments fictifs". Dans cette approche, les objets ne sont pas maillés directement, mais englobés dans un continuum plus facilement maillable, ce qui permet de facilement construire des maillages à différentes échelles, et il propose ainsi une approche multirésolution intéressante, dont les travaux qu'on propose s'inspirent fortement.

1.5.3 Parallélisme

L'évolution actuelle des architectures matérielles s'oriente de plus en plus vers le parallélisme. Celui-ci peut être mis en jeu de façons variées, comme les multiprocesseurs ou le multi-cores, les clusters de PC, ou les processeurs vectoriels tels que les récents GPU programmables. Pour être capable de simuler de grandes scènes complexes contenant plusieurs objets détaillés, il est indispensable de profiter de ces moyens de calcul. Profiter pleinement de ces architectures est à la fois très spécifique à l'architecture elle-même, mais aussi à l'algorithme à paralléliser. D'énormes gains ont pu ainsi être démontrés en animation physique, que ce soit sur un simulateur en général [ZVF03, ZFV04], sur un modèle déformable seul [GEW05] ou en détection de collisions [MOK95, SGHS98, HTG04a, GKLM07]. Plusieurs solutions de parallélisme sont étudiées dans la plate-forme d'animation SOFA [SOF]. Les études préliminaires montrent des résultats impressionnants en utilisant le GPGPU avec un gain supérieur à $\times 10$ pour les objets déformables. Nous avons toujours gardé en tête les propriétés efficaces sur GPU pour que nos méthodes soient en principe portables et efficaces sur GPU.

1.6 Déchirure / découpe

Bien que nous n'ayons pas explicitement abordé ce thème, une fonctionnalité indispensable à un simulateur généraliste concerne la découpe et la déchirure. Découpe et déchirure sont assez similaires à synthétiser. La découpe intervenant suivant un critère donné par l'utilisateur (par exemple la collision avec un objet coupant), la déchirure apparaissant suivant un critère plus automatique, souvent dépendant des contraintes internes. Celles-ci se modélisent principalement par une modification topologique de la structure représentant l'objet animé. Le modèle mécanique s'appuyant fortement sur cette représentation, les méthodes de découpe et les modèles mécaniques sont donc très fortement interconnectés. Le modèle mécanique doit très rapidement se réadapter à la nouvelle topologie à animer. En plus des problèmes concernant cette mise à jour rapide du modèle mécanique, une problématique complexe concerne l'affichage des zones découpées, car pour les objets généralement représentés par leur surface externe, il faut pouvoir créer et afficher les surfaces internes nouvellement visibles.

L'approche la plus simple pour faire de la découpe consiste à éliminer des éléments [Cot97], la mécanique et l'affichage pouvant trivialement s'adapter. Malheureusement, la perte de matière ainsi engendrée peut rendre la simulation peu réaliste, surtout dans le cas d'une simulation interactive utilisant des gros éléments. Il est possible de vraiment découper des éléments [BGTG04]. [MK00] fournit par exemple une liste de motifs de découpe de triangles et de tétraèdres. Dans ce cas, les modifications ne sont pas une tâche aisée, et de très nombreux petits, voire mal conditionnés, éléments vont être créés, pouvant rendre la simulation lente et instable. Il est possible de remailler après la phase de subdivision afin d'éviter les éléments dégénérés [GCMS00a, GO01], mais cette tâche reste complexe et coûteuse. Plutôt que de subdiviser les éléments découpés, il est possible de recréer un maillage aux bords respectant les passages de la coupure [OH99, OBH02], mais encore une fois, ceci est réalisé hors du contexte temps réel.

Une autre approche consiste à modifier les fonctions d'interpolation d'un élément découpé, de telle façon à ce qu'une particule à l'intérieur de l'élément découpé ne dépende plus que de certains nœuds, selon sa position, d'un côté ou de l'autre de la découpe [SMMB00]. C'est une approche très élégante permettant des résultats d'une bonne qualité sans gros ralentissement ni instabilité [VVW04, JJCK07].

Le méthode qui nous paraît la plus intéressante et prometteuse s'appelle les "virtual nodes" [MBF04, MTG04, SDF07]. L'idée consiste à dédoubler virtuellement des nœuds, qui deviennent indépendants, et permettent ainsi à deux éléments contigus de se séparer. Il est même possible de dédoubler des éléments, qui seront ainsi superposés dans l'espace et entraîneront avec eux, chacun une part de la matière. Une limitation semble être le dédoublement de masse, rendant l'objet plus lourd à chaque découpe, même si l'impact est très faible pour un gros objet ne subissant que quelque petites découpes. Nous envisageons volontiers de rendre compatible cette méthode à notre modèle, car elle semble y être très bien adaptée. Pour éviter les problèmes de remaillage de la surface au niveau de la coupure, nous proposerions bien d'exploiter les données volumiques, pour lesquelles la représentation de l'intérieur de l'objet existe déjà.

Les méthodes "meshless" [MKN+04], qui n'utilisent pas de représentation volumique de l'objet, permettent plus directement des découpes d'une grande qualité. L'idée est de faire varier les critères de voisinage d'une particule, marquant ainsi la discontinuité de la matière. Malheureusement de telles méthodes ne sont encore pas adaptées au temps réel [PKA+05].

1.7 Discussion

De nombreux modèles existent déjà en mécanique, et les plus adaptés à la simulation informatique ont été implémentés avec toutes sortes de solutions élégantes pour optimiser les temps de calcul, la robustesse... Alors que parallèlement, dans les applications concrètes, tels que les jeux vidéo ou les simulateurs chirurgicaux, seules les méthodes les plus simples sont employées. Comme principales limitations qui font que certaines méthodes ne sont pas utilisées, on peut retenir leur difficulté à être implémentées efficacement, leur difficulté à être mises en place et paramétrées par un utilisateur non averti et leur comportement face à l'instabilité. Inversement, les méthodes simples employées le sont principalement grâce à leur fiabilité et leur bonne maîtrise par l'utilisateur. Il nous semble donc crucial de respecter les critères qui semblent être prioritaires, que sont la simplicité, la rapidité et la stabilité, afin que les méthodes mises en œuvre puissent être vraiment exploitées. On cherche donc à tirer profit des méthodes précises basée sur la méthode des éléments finis déjà proposées en les intégrant dans un système simple à mettre en place et à utiliser, rapide et robuste.

2 Tétraèdres rapides et robustes

Dans ce travail, nous avons cherché à implémenter les éléments finis tétraèdriques les plus rapides possible, tout en imposant le respect de trois conditions importantes : invariance rotationnelle, loi de Newton sur les forces, loi d'Euler sur les rotations. La première est nécessaire pour traiter les grandes rotations. Les forces internes subies par chaque tétraèdre doivent être invariantes par translation (ce qui est toujours le cas) et par rotation (ce qui ne l'est pas pour les éléments purement linéaires, provoquant l'artefact bien connu de gonflement). La loi de Newton f=ma a pour conséquence que la force interne nette doite être

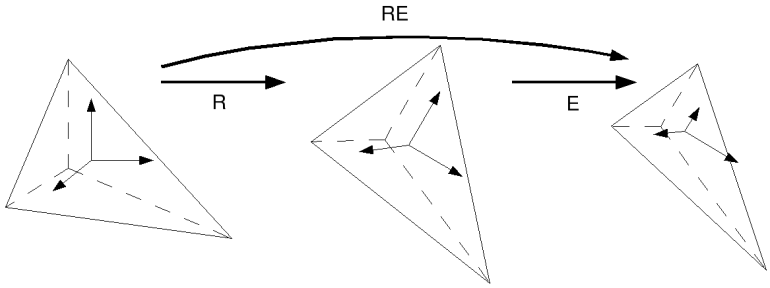


FIG. 2.3 – Décomposition corotationelle des déplacements d'un tétraèdre.

Le tenseur de déplacement est factorisé en un produit RE, où R est une rotation et E exprime la déformation dans le repère tourné.

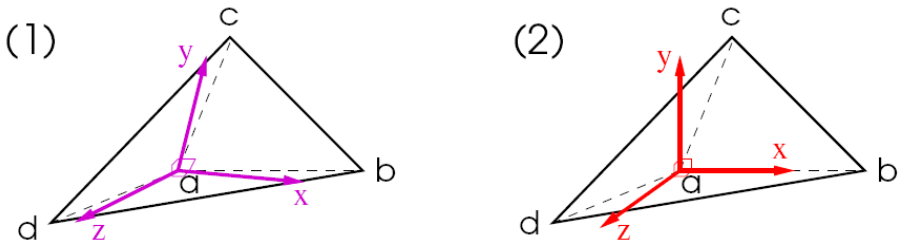


FIG. 2.4 – Comparaison dans le cas d'un tétraèdre à angles droits.

(1) : le repère local obtenu par décomposition polaire est le plus proche des arètes. (2) : le repère local obtenu par décomposition QR.

nulle. Ce n'était pas le cas dans les premières implémentations de méthodes corotationnelles [MDM+02]. La loi d'Euler, généralisation de celle de Newton aux rotations, a pour conséquence que le moment total dû aux forces intérieurs doit être nul. Nous avons montré que ce n'est pas le cas des méthodes qui utilisent un précalcul des matrices de déformation-déplacement des éléments (presque toutes les méthodes corotationnelles). La solution à laquelle nous sommes parvenus [NPF06] satisfait ces critères. De plus, sans que nous l'ayons cherché au départ, la méthode est automatiquement capable de répondre aux inversions, ce qui qui la rend robuste aux déformations extrêmes comme la méthode [ITF04], tout en étant beaucoup plus rapide. Nous avons obtenu l'invariance rotationnelle en développant une nouvelle variante de la méthode corotationnelle, illustrée en figure 2.3, et qui permet de traiter les déformations indépendamment des rotations.

Nous procédons élément par élément plutôt que sommet par sommet, ce qui nous permet de respecter automatiquement la loi de Newton. Formellement, nous effectuons une factorisation QR du tenseur de déplacements. En pratique, cela revient à aligner le premier axe du repère sur la première arête du tétraèdre, le deuxième orthogonal au premier et dans le plan des deux premières arêtes, et le troisième orthogonal aux deux premiers. La différence avec la décomposition polaire, plus souvent utilisée, est illustrée sur la figure 2.4.

La décomposition QR, moins exacte que la polaire, a néanmoins de précieux avantages. Premièrement, elle se calcule plus vite. Deuxièmement, les matrices déformations-déplacements écrites dans ce repère comportent de nombreux termes nuls et leur calcul est beaucoup plus rapide que dans le repère polaire. Il

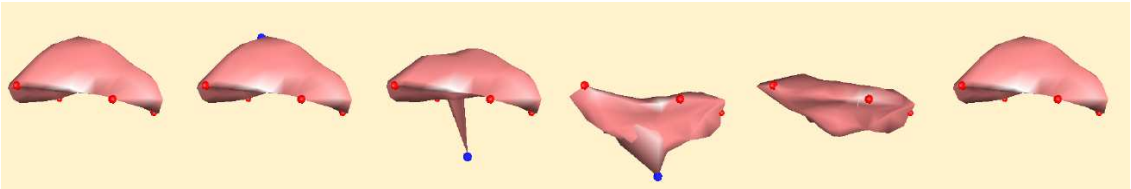


FIG. 2.5 – Simulation interactive robuste.
Le modèle de foie subit une inversion provoquée par une manipulation de l'utilisateur, et retrouve sans difficulté sa forme initiale.

est ainsi possible, pour un surcoût calculatoire réduit, de mettre à jour ces matrices à chaque pas de temps, ce qui permet d'éliminer les couples fantômes et donc de satisfaire la loi d'Euler.

Le repère étant orthonormé direct par construction, une inversion se traduit par une grande déformation, plutôt qu'avec la méthode polaire qui aboutirait à un repère non direct. La robustesse aux inversions est donc obtenue sans aucun traitement supplémentaire. Ceci est un atout intéressant pour les simulations interactives, comme illustré sur la figure 2.5. Ce travail a été effectué par Matthieu Nesme en stage de M2R, ce qui lui a permis de continuer en thèse.

3 Grilles

De gros efforts ont déjà été réalisés pour implémenter efficacement la méthode des éléments finis afin d'utiliser la théorie de la mécanique des milieux continus dans des simulations interactives d'objets déformables. Malheureusement leur efficacité est pleinement effective dans des cas assez rares en utilisation concrète. Il faut par exemple des maillages avec de très bons conditionnements et cette modalité est très dure à respecter sur des objets complexes, spécialement contenant des parties fines et étroites. En effet, la plupart des modèles physiques ont des variations d'échelles que les méthodes de résolutions numériques devraient prendre en compte pour optimiser les calculs. C'est pourquoi nous sommes persuadés que la multirésolution adaptative est une notion essentielle en animation physique interactive. Cette idée n'est pas nouvelle, elle est largement décrite en mécanique où l'on ne désire fréquemment une grande précision qu'à certains endroits. Elle est beaucoup exploitée en informatique graphique pour toutes sortes d'applications interactives, mais elle n'a pas été massivement étudiée en animation physique interactive ; la difficulté majeure résidant dans le fait de respecter les lois physiques malgré l'utilisation de différentes couches. Certaines méthodes adaptatives acceptent de grosses concessions pour passer certains calculs au niveau grossier entraînant une perte de la validation du sens physique des calculs [DDCB01, DMG05]. Un autre manquement actuel que nous avons souligné concerne la propagation dynamique des matériaux mal conditionnés. Une simulation hiérarchique sera mieux adaptée, car le problème pourra être traité à la fois dans les hautes et dans les basses fréquences.

Adapter localement la résolution induit des complications :

- **maillages :** il faut être capable de créer des maillages à différentes résolutions tout en garantissant leur bon conditionnement. Le tout avec des opérateurs qui permettent de passer facilement et rapidement d'un niveau à l'autre (pour enlever/ajouter des degrés de liberté), sans perdre l'avantage de la multirésolution en gaspillant du temps dans les transitions entre niveaux.
- **résolution efficace :** il faut être capable de résoudre un système à différentes échelles, tout en garantissant une bonne convergence, aussi bien au niveau grossier qu'au niveau fin où la complexité explose.
- **respect de la théorie :** il faut rester le plus rigoureux concernant les lois physiques et ne pas s'éloigner de la théorie mécanique malgré l'approche adaptative.

Actuellement, les problèmes sont bien séparés entre la génération de maillages, les calculs de forces et l'intégration du temps. L'approche générale regroupe en effet les étapes suivantes : acquisition des données, création du maillage mécanique, puis calcul de la mécanique. Chaque étape prenant ce qu'on lui donne en entrée pour ressortir ce qui lui semble le mieux par rapport au besoin de l'étape suivante. Nous pensons cependant qu'il est possible de faire mieux en traitant le problème dans son ensemble.

Afin de répondre au problème des maillages, dans ces travaux, nous faisons le choix des grilles régulières. Nous faisons donc un sacrifice au niveau des bords des objets qui ne sont plus décrits avec précision. Mais ce choix permet la gestion facile du bon conditionnement du maillage mécanique, qui est un point crucial des calculs interactifs. De plus, construire une hiérarchie de grilles régulières est trivial, et une représentation multirésolution peut aisément prendre forme. D'autre part, les niveaux de la hiérarchie peuvent facilement être emboîtés sous la forme d'un arbre octal afin d'optimiser les transitions entre niveaux en ayant des structures de données bien connues, pratiques à créer et à manipuler.

Pour répondre à un fort besoin théorique, une approche est proposée pour limiter les problèmes liés aux grilles régulières concernant les bords de l'objet. En effet, les éléments frontières d'une grille régulière ne représentent pas directement l'objet, et il faut être capable de prendre en compte la répartition non homogène de la matière à l'intérieur pour simuler le bon comportement de ces éléments frontières et pourvoir traiter les conditions aux limites avec finesse.

Une fois cette brique de base bien en place, nous avons étudié diverses solutions pour améliorer le dynamisme d'une animation, dans l'idée de bien rendre la raideur des matériaux interactivement. Gardons en tête que notre but n'est pas d'avoir la méthode qui donne la solution exacte dans le laps de temps le plus court, mais d'obtenir la meilleure solution approchée possible dans un temps donné.

Pour finir, une grosse partie de notre travail a concerné la mise en situation de notre modèle pour tester sa validité. En effet, suivre les lois théoriques est une première étape nécessaire, mais il faut ensuite s'assurer que notre interprétation fournit des résultats cohérents. Cette étape est d'autant plus importante dans le cas de simulations médicales où la précision est nécessaire. Très peu de validations mécaniques ont malheureusement été effectuées sur les modèles interactifs déjà présentés dans la littérature.

Pour résumer, l'utilisation d'hexaèdres en animation nous paraît intéressante car elle nous permet de proposer un modèle répondant à plusieurs besoins difficiles à mettre en œuvre, à savoir :
- la génération automatique et rapide d'une représentation hiérarchique avec un bon conditionnement,
- une mécanique adaptative simple, rapide et stable qui suit les lois de la mécanique des milieux continus,
- un compromis précision-temps de calcul,
- la simulation interactive de matériaux mal conditionnés.

Nous verrons même qu'elle est un bon support pour un rendu volumique efficace.

4 Animation d'objets par grilles déformables

Le but de ce travail est d'automatiser la construction de modèles mécaniques d'après des données d'entrée, qu'elles soient volumiques (images médicales) ou surfaciques (maillages graphiques). Une voxelisation est utilisée comme représentation volumique, sur laquelle appliquer la mécanique, comme illustré sur la figure 2.6.

Fabriquer les maillages mécaniques proposés à partir d'une représentation volumique de type scanner est trivial, puisque l'on dispose d'une voxelisation fine qu'il suffit de regrouper pour obtenir des niveaux plus grossiers, comme illustré en figure 2.7. Si on avait assez de puissance de calcul, on pourrait animer chaque voxel par un élément fini hexaédrique ; mais étant donné leur trop grand nombre, on est amené à les regrouper en hexaèdres plus gros.

Lorsque les éléments se déforment, l'objet doit suivre ces déformations. Pour cela, à la manière d'une FFD, l'objet est attaché aux éléments englobants : les points de sa surface $\boldsymbol{u}(p)$ sont tri-linéairement interpolés en utilisant les coordonnées barycentriques par rapport aux huit nœuds $\boldsymbol{u}(q)$ comme illustré en 2D dans la figure 2.8.

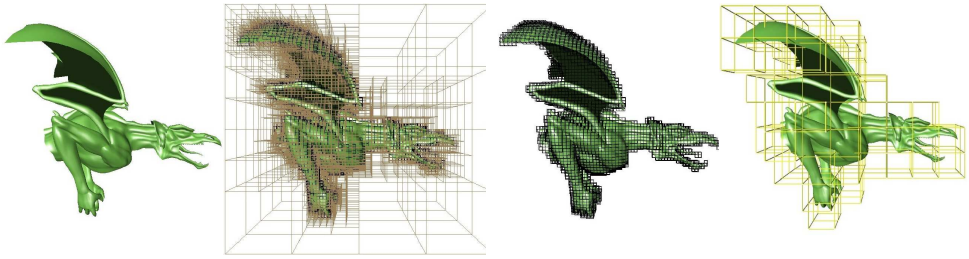


FIG. 2.6 – Maillage d'un dragon.

de gauche à droite : la forme de départ. L'arbre octal contenant le volume. Les éléments les plus fins. Une résolution intermédiaire.

FIG. 2.7 – Plusieurs résolutions de voxelisations à partir de données volumiques segmentées issues d'un scanner médical (ici pour un foie)

Données volumiques segmentées (source : IRCAD)

$$\boldsymbol{u}(p) = \boldsymbol{H}\boldsymbol{u}(q) \tag{2.2}$$

où la matrice d'interpolation $\boldsymbol{H}$ (3×24) décrit les influences des sommets de l'élément sur un point de l'objet donné.

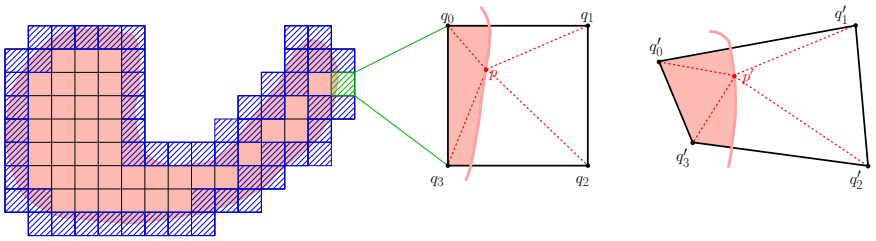


FIG. 2.8 – Interpolation

Un objet à animer est interpolé à l'intérieur d'éléments l'englobant. Lorsque les éléments se déforment, l'objet suit les déformations.

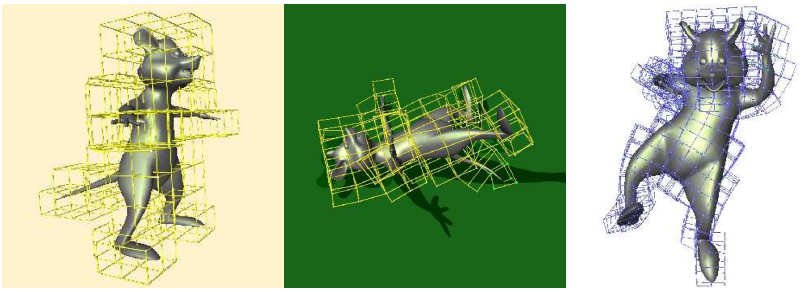


FIG. 2.9 – Exemples de maillages classiquement difficiles à animer. Les parties fines sont bien gérées (oreilles, queue).

Une contrainte en position s'écrit $\boldsymbol{u}(p) = \boldsymbol{c} \Leftrightarrow \boldsymbol{H}\boldsymbol{u}(q) = \boldsymbol{c}$ où $\boldsymbol{c}$ est la valeur de la contrainte et $\boldsymbol{H}$ est la matrice d'interpolation du point $\boldsymbol{p}$. Un ensemble de contraintes peut être résolu en utilisant une pseudo-inverse de la matrice assemblée $\boldsymbol{H}$:

$$\boldsymbol{u}(\boldsymbol{q}) = \boldsymbol{H}^{+}\boldsymbol{c}$$

Une force $\boldsymbol{f}(p)$ appliquée au point p de la surface doit être répartie sur les nœuds de l'élément l'englobant. Le principe des puissances virtuelles implique

$$\boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{H}^{T}\boldsymbol{f}(\boldsymbol{p})$$

comme expliqué chapitre 4, section 1.2.

Le maillage d'hexaèdres englobants est très facile à obtenir à une résolution désirée sur laquelle n'importe quelle méthode mécanique peut être appliquée pour animer l'objet. Comme la surface est interpolée dans les éléments, les résolutions des maillages utilisés pour la mécanique et l'affichage sont totalement indépendantes, et il est possible d'avoir un maillage mécanique grossier tout en conservant un objet très finement détaillé pour le rendu, comme illustré sur la figure 2.9 ;

Une contribution intéressante de cette approche est de pouvoir gérer des surfaces non-fermées (comme les ailes du dragon ou le T-shirt de la figure 2.10), et même des éléments linéiques très fins (queue de la souris) de la même façon qu'un volume. Une forme au repos en 3D peut être assignée à ce type d'objets, comme celle qui est la plus naturelle. Par exemple, le T-shirt tend à retrouver la forme d'un vêtement porté. Avec les approches de maillages classiques, animer ces objets est bien plus compliqué voire impossible, le faire en temps réel est un vrai défi.

Cette représentation volumique semble selon nous très adaptée à l'animation temps-réel d'organes spécifiques à un patient. En effet, son aspect régulier lui confère un bon conditionnement et facilitera les résolutions numériques pour plus de rapidité et de stabilité, et sa possibilité d'adopter une approche multirésolution permettra d'économiser en temps de calcul. Sa construction directement depuis des données volumiques médicales facilite l'adaptation au cas particulier, comme illustré sur la figure 2.11.

La plupart des modèles mécaniques s'adaptent bien aux grilles. Dans la section suivante, nous proposons un calcul des déformations qui profitera des caractéristiques de ce maillage pour fournir une méthode multirésolution rapide, stable et qui respectera au mieux le comportement de l'objet à toutes résolutions dans un souci de précision.

L'utilisation d'éléments non pleins (aux bords de l'objet) n'est pas habituelle en simulation mécanique. Nous allons montrer que leur usage est possible et avantageux, de telle manière que la formulation des éléments finis prenne en compte la répartition de la matière à l'intérieur de ceux-ci.

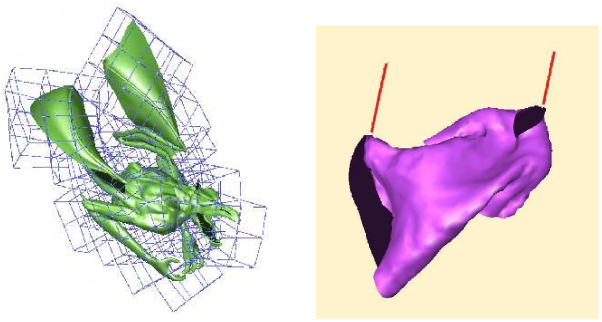


FIG. 2.10 – Exemples de maillages classiquement difficiles à animer. Les surfaces peuvent être animées comme des volumes.

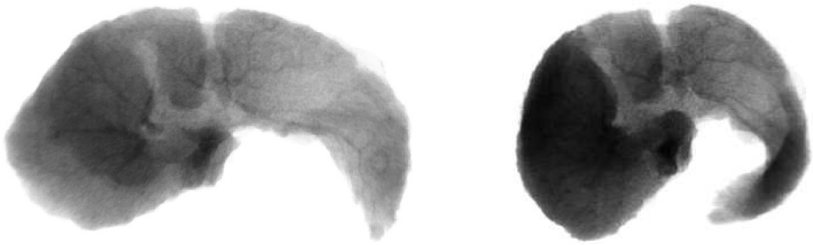


FIG. 2.11 – Foie modélisé à partir de données patient, avec rendu volumique interactif.

5 Mécanique des hexaèdres englobants

Nous avons adapté aux hexaèdres l'approche de simulation rapide de tétraèdres développée précédemment 2, en munissant chaque case de la grille d'un repère local comme illustré sur la figure 2.12.

Le principal inconvénient des grilles est de mal représenter les frontières des objets. Pour prendre en compte la répartition de matière dans les cellules, nous avons procédé par regroupements à partir de cellules fines. Chaque cellule est mécaniquement caractérisée par sa matrice de raideur et sa matrice de masse. Nous calculons les matrices des "grosses" cellules par sommes pondérées de celles des "petites" cellules. On peut utilise huit matrices $\boldsymbol{L}_{fils}$ qui représentent l'interpolation des nœuds fils $\boldsymbol{u}_{fils}$ en fonction des nœuds parents $\boldsymbol{u}_{\text{père}}$: $\boldsymbol{u}_{fils} = \boldsymbol{L}_{fils}\boldsymbol{u}_{\text{père}}$. Réciproquement, les forces appliquées aux nœuds fils doivent

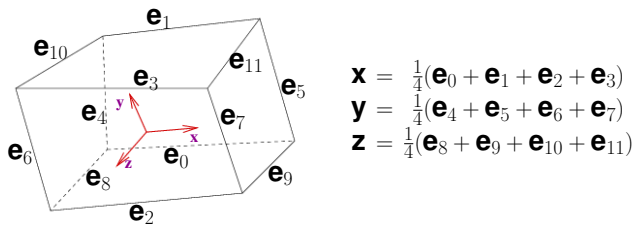


FIG. 2.12 – Repère local sur un hexaèdre déformé.

Pour former les arêtes d'un tétraèdre local, les moyennes des arêtes dans les trois directions sont considérées, avec lesquelles un repère orthogonal peut facilement être déduit comme dans [NPF05].

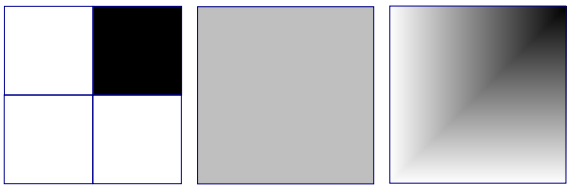


FIG. 2.13 – Schématisation en niveau de gris de la raideur en 2D

de gauche à droite : a) quatre petits éléments, trois vides, un plein. b) leur plus gros élément père regroupant ces éléments en utilisant un moyennage uniforme, l'élément est uniformément très mou. c) le même regroupement en utilisant la non-uniformité des propriétés va faire varier la raideur le long des arêtes.

être remontées aux pères en utilisant la transposée de l'interpolation : $\boldsymbol{f}_{\text{père}} = \boldsymbol{L}_{fils}^T \boldsymbol{f}_{fils}$. En sommant l'influence de tous les enfants d'un élément père, on obtient

$$\boldsymbol{K}_{\text{père}} = \sum_{i=0}^{7} \boldsymbol{L}_i^T \boldsymbol{K}_i \boldsymbol{L}_i$$

La même technique est utilisée pour les masses : $\boldsymbol{M}_{\text{père}} = \sum_{i=0}^{7} \boldsymbol{L}_i^T \boldsymbol{M}_i \boldsymbol{L}_i$,où $\boldsymbol{M}$ est la matrice de masse d'un élément. Nous obtenons des matrices reflétant la répartion de la matière dans les cellules, comme imparfaitement illustré sur la figure 2.13.

L'objectif de ce travail est essentiellement atteint. Nous arrivons désormais à créer des modèles mécaniques personnalisés de manière automatique. À résolution équivalente, ils sont aussi rapides, voire plus, que des tétraèdres et se créent automatiquement sans risque de maillage dégénéré. Une brêve comparaison est montrée sur la figure 2.14. Une attention importante a été portée à la validation, comme présenté dans la thèse de Matthieu.

6 Multirésolution

Les grilles régulières présentent l'intérêt de bien se prêter à des approches multirésolution. Celles-ci sont souhaitables pour pricipalement deux raisons : concentrer les calculs là où ils sont le plus nécessaire (section 6.1), et accélérer les résolutions d'équations (section 6.2).

6.1 Adaptativité

La principale difficulté dans la résolution numérique d'une approche multirésolution est le traitement des nœuds en T, qui se forment à l'interface de plusieurs niveaux de résolution (*cf.* figure 2.15). Ces nœuds doivent rester contraints sur l'arête (ou la face) sur laquelle ils se trouvent. Les travaux précédents utilisent des multiplicateurs de Lagrange, qui alourdissent considérablement les calculs, ou projettent simplement le point sur l'arête [DDCB01, WDGT01, DMG05] sans respecter tous les principes de la mécanique. CHARMS [GKS02] résoud le problème en reformulant les modèles comme des combinaisons de points indépendants, au prix d'une structure de données complexe.

Pour gérer ce problème, notre approche profite de l'application simple des contraintes dans l'intégration implicite proposée par [BW98] par filtrage des vecteurs dans le gradient conjugué employé. Nous étendons ce principe en proposant un filtre plus complexe permettant de traiter les nœuds en T en se basant sur le principe des travaux virtuels déjà exploité en section 5. L'idée est de ne pas considérer le nœud en T comme un degré de liberté à part entière. Au contraire, toutes les contraintes calculées sur ce nœud devront être remontées à ses parents (par la transposée de son interpolation). Ses variations d'états (déplacement, position, vitesse) seront alors déduites par interpolation de ses parents. Ces deux actions sont illustrées en figure 2.16. L'algorithme du gradient conjugué résultant est présenté ci-dessous (algorithme 1).

Cette idée semble équivalente aux *hard bindings* récemment publiés [SSIF07].

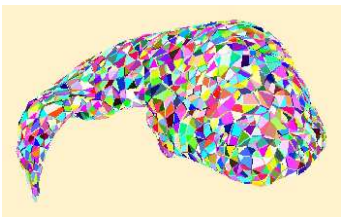
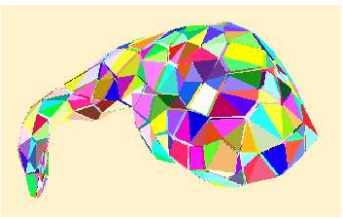
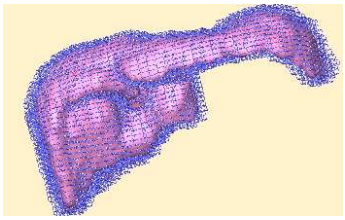
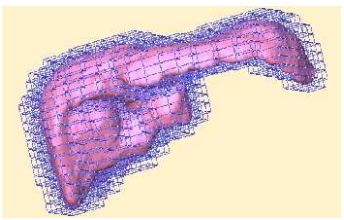
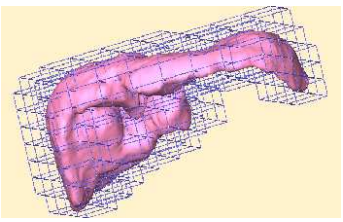
marching cube original 5541 sommets de la surface 11078 triangles	≈ 70% décimation 1455 sommets de surface 2904 triangles	≈ 95% décimation 229 sommets de surface, 452 triangles
7655 nœuds 25204 tétraèdres 1 Hz	2032 nœuds 6725 tétraèdres 5 Hz	309 nœuds 929 tétraèdres 40 Hz
		
11378 nœuds 8913 hexaèdres <1 Hz	2003 nœuds 1346 hexaèdres 13 Hz	436 nœuds 242 hexaèdres 70 Hz
		

FIG. 2.14 – Comparaison de temps de calculs entre des maillages tétraédriques classiques (haut) et nos hexaèdres englobants (bas) pour la même méthode corotationnelle (QR)

La vitesse d'animation est donnée pour plusieurs résolutions mécaniques à peu près équivalentes. Le rendu non réaliste des tétraèdres est présenté ici afin de bien les distinguer.

Adaptativité : Lors d'un raffinement ou d'une simplification adaptative, les mêmes opérateurs de remontée et de propagation devront être utilisés. Lorsqu'une particule "inutile" disparaît, ses contributions doivent être réparties sur ses parents. Et inversement, quand une particule apparaît pour ajouter du détail, ses états en cours (position, vitesse) sont déduits par interpolation.

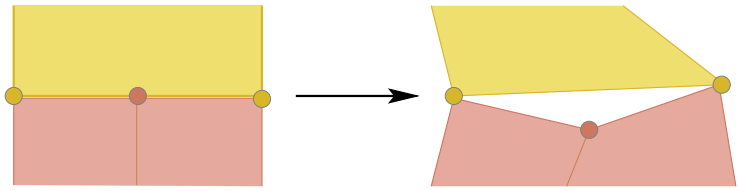


FIG. 2.15 – Nœud en T à l'interface de divers niveaux de multirésolution (2 dimensions)

Si aucune contrainte spécifique n'est gérée, après déformations le maillage devient non-conforme

Algorithme 1 GRADIENT CONJUGUÉ MULTIRÉSOLUTION $\boldsymbol{Ax} = \boldsymbol{b}$

$\boldsymbol{x} = \boldsymbol{0}$ // *inconnues initialisées à 0*
$\boldsymbol{r} = \boldsymbol{b}$
REMONTERFORCES($\boldsymbol{r}$) // *remonter les contributions des nœuds en T*
FILTRER($\boldsymbol{r}$) // *filtrage classique [BW98]*

$i = 0$ // *itération*

loop

$\rho = \boldsymbol{r} \cdot \boldsymbol{r}$

if $i == 0$ **then**
$\boldsymbol{p} = \boldsymbol{r}$
else
$\boldsymbol{p} *= \rho / \rho_{i-1}$
$\boldsymbol{p} += \boldsymbol{r}$
end if

PROPAGERDÉPLACEMENTS($\boldsymbol{p}$) // *interpoler dx aux nœuds en T*
$\boldsymbol{q} = \boldsymbol{Ap}$
REMONTERFORCES($\boldsymbol{q}$)
FILTRER($\boldsymbol{q}$)

$\alpha = \rho / (\boldsymbol{p} \cdot \boldsymbol{q})$
$\boldsymbol{x} += \alpha \boldsymbol{p}$
$\boldsymbol{r} -= \alpha \boldsymbol{q}$

$\rho_{i-1} = \rho$
$i += 1$

end loop

PROPAGERDÉPLACEMENTS($\boldsymbol{x}$)

Pour tester en pratique notre gestion des nœuds en T, nous avons évalué le comportement d'une poutre complètement compressible (coefficient de Poisson à 0) soumise à des forces externes l'allongeant, en comparant une approche non multirésolution et une approche multirésolution, tel que les deux modèles doivent donner les mêmes résultats (figure 2.17).

La gestion des nœuds en T que l'on propose permet bien de garder la cohérence des résultats, et tous les maillages, fin comme grossier et avec des nœuds en T donnent bien le même résultat.

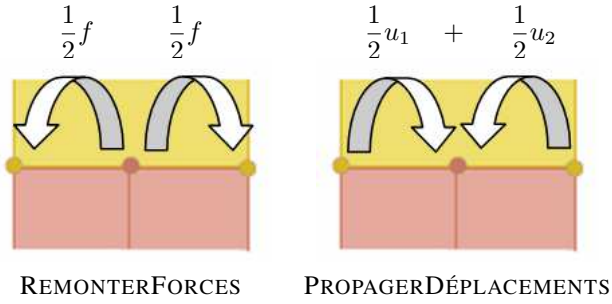


FIG. 2.16 – Illustrations des étapes du filtre gérants les nœuds en T

Toute force calculée sur un nœud en T est remontée vers les degrés de libertés indépendants. La solution peut être déduite sur un nœud en T par interpolation depuis les degrés de libertés indépendants.

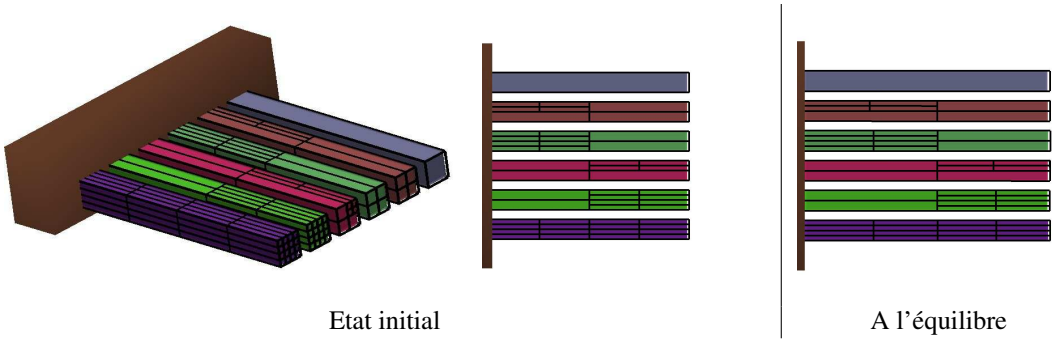


FIG. 2.17 – Vérification sur la gestion des nœuds en T

Plusieurs poutres identiques maillées de différentes manières, dont certaines avec des nœuds en T, sont soumises à une traction. Avec des matériaux identiques et un coefficient de Poisson nul, toutes donnent bien la même position finale.

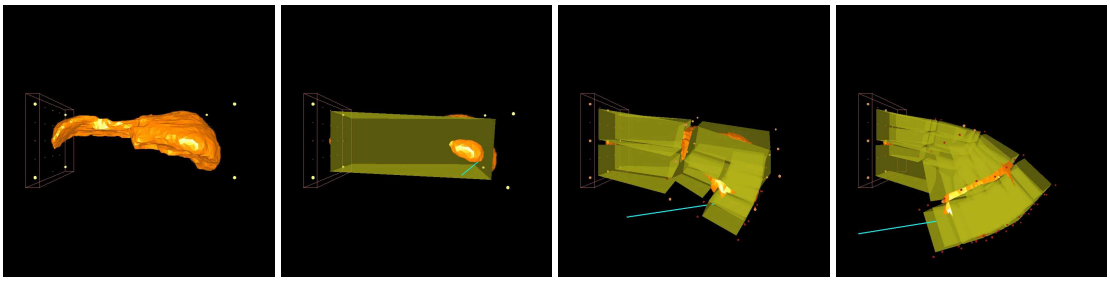


FIG. 2.18 – Raffinements automatiques

Un foie est animé au départ par un seul hexaèdre englobant, en appliquant une force externe il se déforme, les endroits de fortes courbures se raffinent automatiquement.

Cet exemple est simple, et sûrement pas suffisant mais il n'est pas évident de trouver des expériences plus complexes permettant de comparer les résultats donnés par différentes résolutions.

La figure 2.18 montre des résultats obtenus avec une stratégie adaptative en adoptant le critère de raffinement automatique décrit dans [DDCB01].

6.2 Accélération de la convergence

Les algorithmes de résolution des équations dynamiques procèdent par itérations, chaque particule propageant des informations à ses voisines comme illustré sur la figure 2.19.

Nous avons donc appliqué une grille hiérarchique, qui propage plus rapidement les informations, comme illustré sur la figure 2.20.

Chaque sommet, à part les huit premiers, mémorise la différence entre sa position effective et sa position interpolée d'après ses parents. Afin de gérer les grands déplacements, comme énoncé précédemment, nous employons une méthode corotationnelle. A cause de la représentation hiérarchique, chaque détail (c'est-à-dire chaque valeur hiérarchique) doit être indépendant des rotations. Pour cela les détails sont représentés dans un repère local suivant les rotations comme illustré en figure 2.21. Pour chaque repère local, la méthode corotationnelle sur les hexaèdres est employée.

Comme espéré, la convergence de la résolution de l'intégration implicite est plus rapide dans les premières itérations, comme présenté sur la figure 2.22. Dans un contexte de simulation temps-réel, seules les

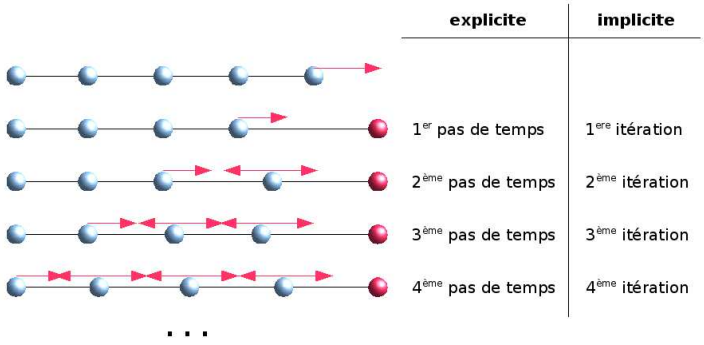


FIG. 2.19 – Propagation des déformations le long d'un objet discrétisé 1D.

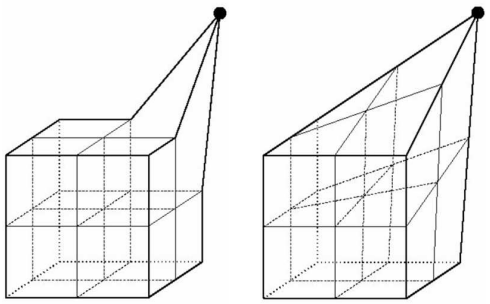


FIG. 2.20 – Conséquences géométriques d'une représentation hiérarchique

En nodal, quand un sommet est déplacé, il n'a aucune influence géométrique, tandis qu'en hiérarchique il va influencer toute sa "descendance" qu'il interpole. Instantanément, le hiérarchique va se rapprocher plus près de la solution finale.

premières itérations sont appliquées faute de temps, et une telle convergence donne donc de bien meilleurs résultats.

Toutefois, la lourdeur de la structure de données est un réelle difficulté, et la présence de termes croisés entre différents niveaux dans les matrices de masse et de raideur introduit des problèmes de conditionnement numérique. Peut-être aurait-il fallu creuser la question des préconditionneurs, mais nous avons préféré exploré une autre piste, la résolution multigrid [Bra77, GW06, OGRG07].

La méthode multigrid est une méthode hiérarchique qui exploite des représentations à différentes échelles du problème. Cette approche se base sur le fait que les méthodes de relaxation classiques (telle que Gauss-Seidel) réduisent rapidement les hautes fréquences, alors que les basses fréquences vont demander beaucoup d'itérations. Or les basses fréquences d'un problème discrétisé finement correspondent aux hautes fréquences du même problème discrétisé grossièrement. L'idée qui en découle est de résoudre le système à différentes échelles pour lisser toutes les fréquences rapidement et ainsi obtenir une très bonne convergence. Son fonctionnement est très bien détaillé dans [BHM00]. Cette méthode peut être très utile en animation physique d'objets déformables afin d'améliorer la convergence de l'intégration implicite qui correspond à un système linéaire.

Le principe de base est de corriger la solution approximée au niveau fin en effectuant des relaxations aux niveaux plus grossiers. Une itération au niveau fin étant coûteuse, leur nombre est limité. Dans le multigrid, la solution approchée au niveau fin peut être améliorée à faible coût par les niveaux grossiers pour lesquels

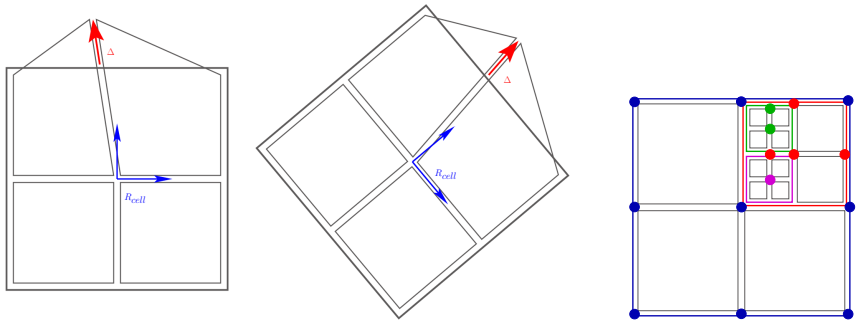
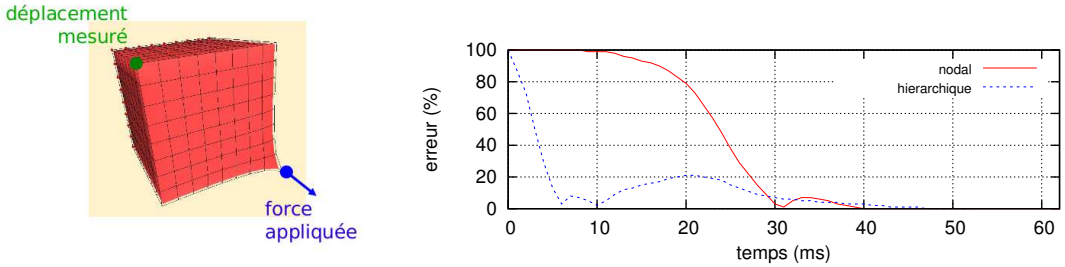


FIG. 2.21 – Prise en compte des grands déplacements hiérarchiques

(gauche) Un repère local corotationnel pour chaque élément à tous les niveaux de hiérarchie. Les détails exprimés dans ces repères locaux deviennent invariants en rotation.
(droite) Repères dans lesquels sont exprimés les valeurs des nœuds sur un exemple multirésolution. La couleur du nœud correspond à la couleur de l'élément dans lequel il est exprimé.

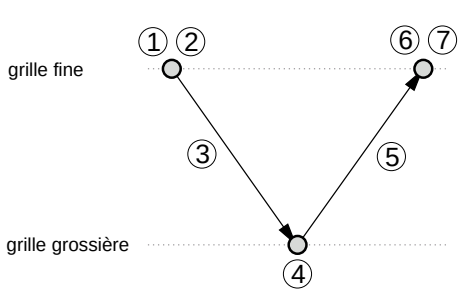


exemple 1 exemple 2

FIG. 2.22 – Un test de rapidité de convergence de la méthode hiérarchique.

une itération est beaucoup moins coûteuse. Pour cela, l'idée est de transférer le résidu restant du niveau fin vers le niveau grossier (opérateur de restriction), de calculer l'erreur corrigeant ce résidu au niveau grossier, et d'interpoler la correction au niveau fin pour s'en servir de correction de la solution approchée initialement calculée. Les opérateurs de transferts sont ceux déjà évoqués pour la multirésolution. Leurs coefficients sont simples à évaluer dans nos grilles emboîtées, et les opérateurs peuvent être implémentés de façon très efficace en ajoutant un coût négligeable à la résolution par rapport aux relaxations.

Ce principe est schématisé étape par étape sur deux niveaux dans la figure 2.23.



① pré-relaxation	$\boldsymbol{x}_h = \boldsymbol{A}_h^{-1}\boldsymbol{b}_h$
② résidu	$\boldsymbol{r}_h = \boldsymbol{b}_h - \boldsymbol{A}_h\boldsymbol{x}_h$
③ restriction	$\boldsymbol{r}_H = \boldsymbol{R}_h^H\boldsymbol{r}_h$
④ solution grossière	$\boldsymbol{e}_H = \boldsymbol{A}_H^{-1}\boldsymbol{r}_H$
⑤ interpolation	$\boldsymbol{e}_h = \boldsymbol{I}_H^h\boldsymbol{e}_H = \boldsymbol{R}_H^h\boldsymbol{e}_H$
⑥ correction	$\boldsymbol{x}_h = \boldsymbol{x}_h + \boldsymbol{e}_h$
⑦ post-relaxation	$\boldsymbol{x}_h = \boldsymbol{A}_h^{-1}(\boldsymbol{A}_h\boldsymbol{x}_h - \boldsymbol{b}_h)$

FIG. 2.23 – Cycle de multigrid à 2 niveaux

On essaie de trouver $\boldsymbol{x}_h$ tel que $\boldsymbol{A}_h\boldsymbol{x}_h = \boldsymbol{b}_h$. h représente le niveau fin et H le niveau grossier. $\boldsymbol{r}$ est le résidu à un certain niveau, $\boldsymbol{e}$ représente l'erreur au niveau grossier et $\boldsymbol{A}_H$ est la matrice équivalente à $\boldsymbol{A}_h$ au niveau grossier (construite par la condensation des propriétés non-uniforme).

Ce principe peut être utilisé sur plus de deux niveaux comme illustré dans la figure 2.24. Le cycle en V correspond à un passage par chaque niveau, chacun des niveaux grossiers corrigeant son prédécesseur. Dans le cycle complet, une solution est d'abord approchée à chaque niveau (à l'aide d'un cycle multigrid en V), afin de débuter la résolution fine avec une solution déjà bien approchée.

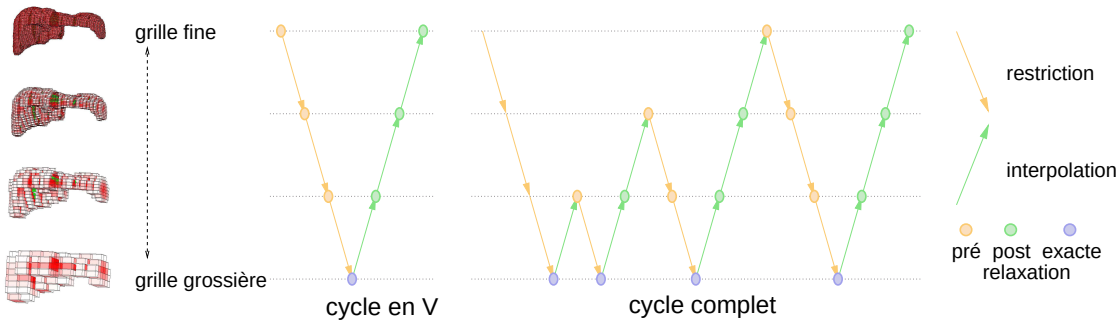


FIG. 2.24 – Les cycles de multigrid

Notre implémentation comporte quelques points originaux qui méritent d'être détaillés.

Relaxation

D'habitude, Gauss-Seidel (GS) est la méthode de relaxation préférée dans l'approche multigrid car elle lisse très bien les hautes fréquences. Sans avoir la justification théorique de cette même propriété concernant le Gradient Conjugué (GC), nous avons préféré GC à GS, car il offre des avantages non négligeables décrits dans [BW98]. Tout d'abord, sa convergence est bien meilleure pour des systèmes symétriques définis positifs comme c'est le cas pour l'intégration implicite employée. De plus, sa gestion des contraintes par filtrage est simple et ne rajoute pas de surcoût. Enfin GC va être encore plus intéressant à nos yeux dans une approche multigrid grâce à ses critères d'arrêt facilement implémentables. Alors qu'avec une relaxation de GS, un certain nombre d'itérations fixe est effectué, GC peut, à moindre coût, s'arrêter si le résidu est assez petit ou si numériquement la solution ne peut être améliorée. Ainsi, à la descente les pré-relaxations peuvent être très rapides si le système est facile à résoudre. De même les post-relaxations (qui ne sont pas obligatoires) GC ne seront effectuées que si le résidu est encore trop grand, même après correction. Le GC offre donc un contrôle automatique entre la qualité de la solution et le nombre d'itérations à employer.

Opérateurs de transferts

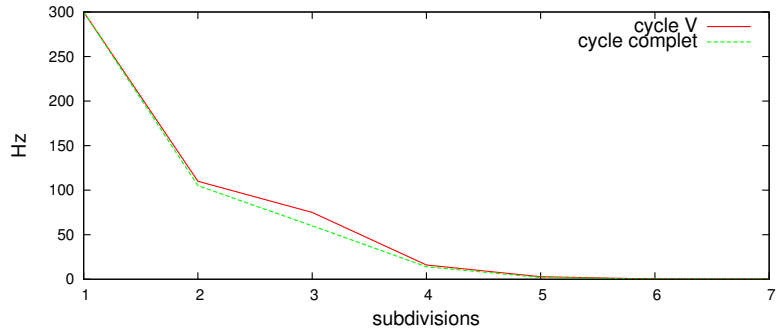


FIG. 2.25 – Comparaison de vitesses entre des résolutions multigrid basées sur un cycle en V et un cycle complet

Dans le cas d'une poutre encastrée soumise à la gravité en flexion, $E = 40000$ Pa, $\nu = 0.3$, pour diverses tailles de maillages mécaniques données par le nombre de subdivisions.

Grâce à la structure emboîtée des grilles utilisées, les opérateurs de transferts d'un étage à l'autre sont triviaux et très rapides. Les matrices correspondant aux discrétisations grossières sont calculées à l'aide des propriétés non-uniformes présentées en section 5. Elles ont le gros avantage d'être une bonne approximation du système fin à tous les niveaux en respectant la propriété de Galerkin [BHM00] afin d'assurer une convergence optimale.

Cycle

Nous avons comparé une implémentation utilisant un cycle en V face à un cycle complet (figure 2.25). Le cycle en V est celui qui semble offrir les meilleurs résultats pour le temps réel. Le cycle complet converge généralement un tout petit peu plus rapidement, mais quelque fois, sa descente est beaucoup plus lente. Ceci s'explique peut-être par le fait que nos systèmes convergent déjà assez vite dans les premières itérations au niveau fin, et que débuter par une bonne solution initiale prend du temps et n'améliore pas assez la convergence. L'utilisation de préconditionneurs pourrait aussi améliorer les choses.

La figure 2.25 présente des vitesses de calcul dans un contexte interactif, avec peu d'itérations (7 à chaque niveau, et 150 au niveau grossier). Les cycles en V et complet offrent significativement les mêmes résultats, avec un léger avantage pour le cycle en V. Lorsque le cycle complet semble prendre l'avantage (>6 subdivisions), de très gros maillages mécaniques sont utilisés, très loin de ceux qui peuvent être animés en temps réel.

Nombre de niveaux

Avec les grilles emboîtées, il est très facile de faire varier le nombre de résolutions. Au dessus d'un certain nombre de résolutions grossières, il semble que la convergence ne soit plus significativement améliorée. Dans nos études, le choix de trois/quatre niveaux de raffinement semblent un bon compromis simplicité/-précision/vitesse (*cf.* figure 2.26).

À cause de difficultés d'implémentation, nous n'avons pas encore réussi à développer une méthode adaptative accélérée par la technique du multigrid. Ces difficultés proviennent essentiellement des discontinuités de résolution. La continuité des champs de déplacements aux frontières des éléments est une contrainte lourde. Il se peut que les approches "meshless" puissent apporter une souplesse appréciable. Ce serait une piste intéressante à explorer.

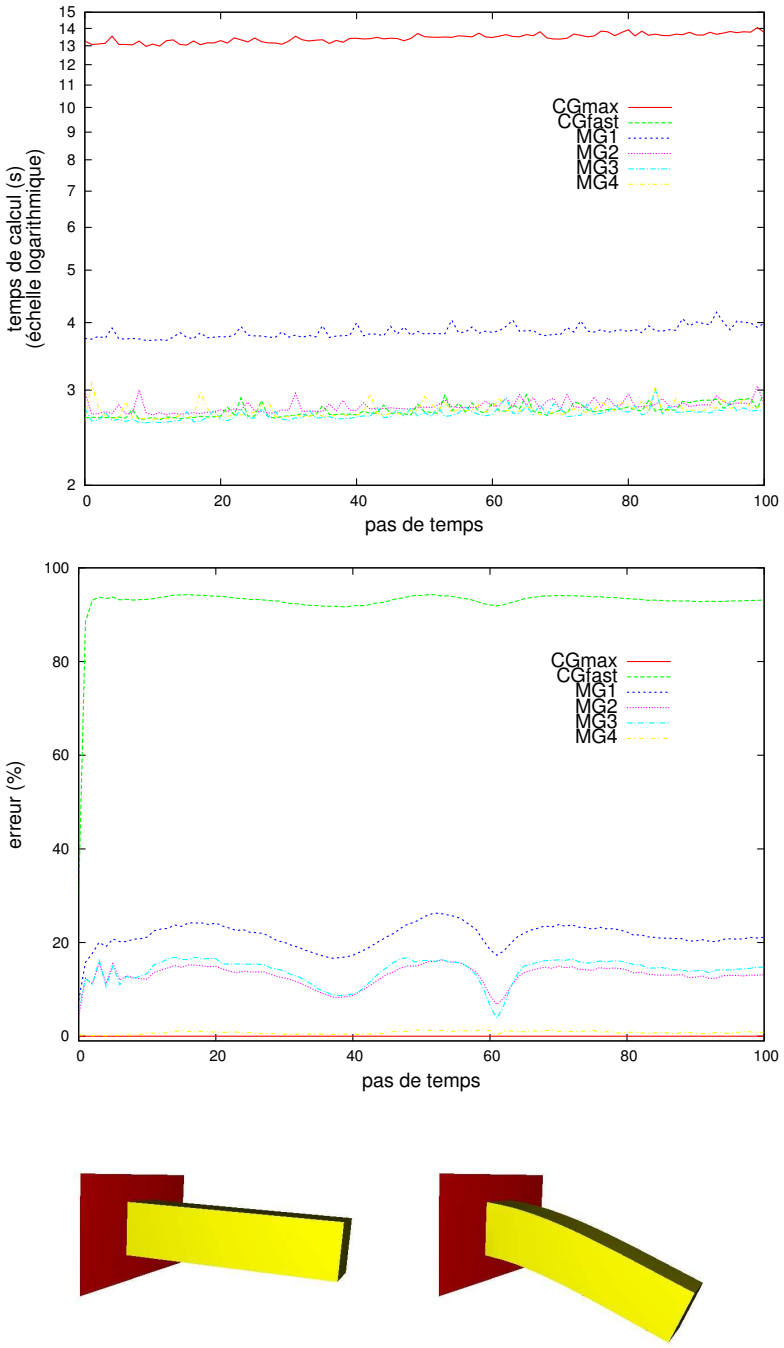


FIG. 2.26 – Comparaison de la qualité (vitesse et erreur) des différentes approches pour la poutre en flexion

chapitre 3

Collisions

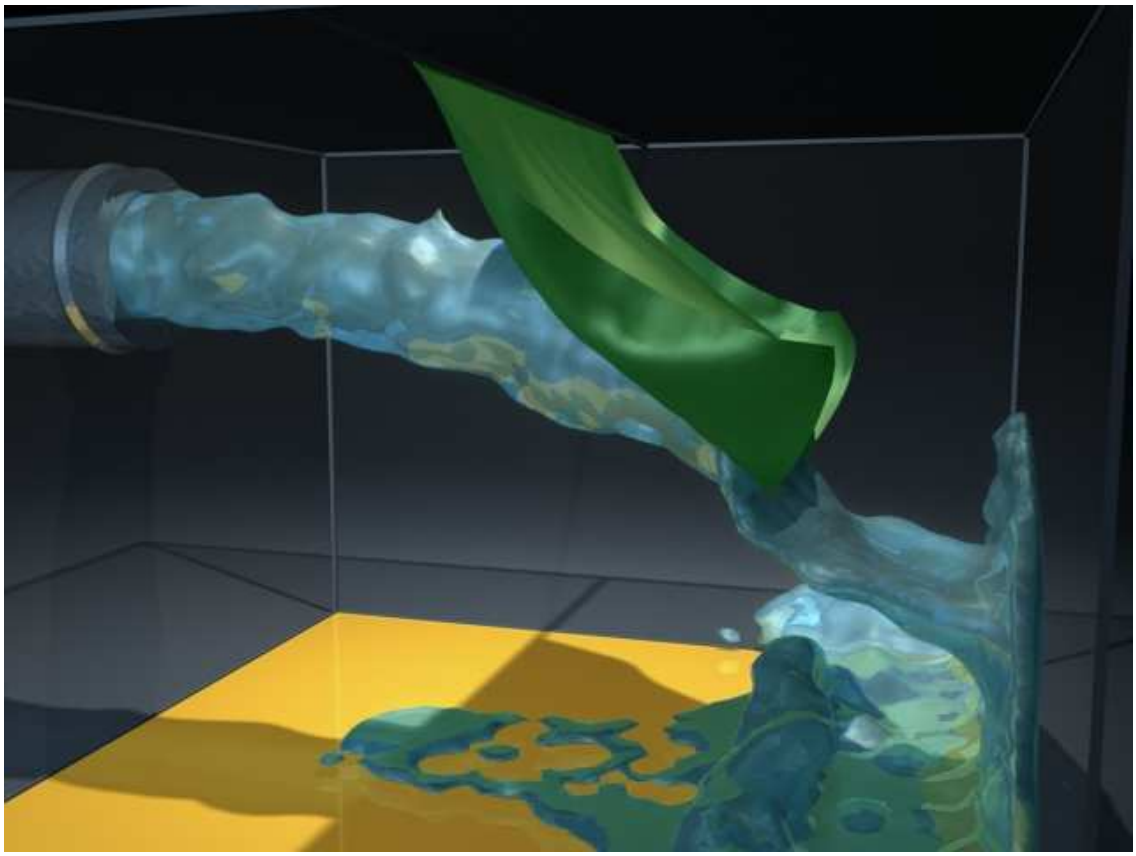


FIG. 3.1 – Objets en contact.

La détection et le traitement des contacts sont des sujets d'importance majeure pour notre domaine. Par défaut, les objets simulés se traversent comme des fantômes, et ce n'est qu'au prix de calculs complexes qu'une réaction peut être obtenue. Ils peuvent ralentir la simulation au point de rendre impossible son application en temps interactif. Ces sujets sont étudiés depuis longtemps et d'excellents états de l'art sont disponibles [JTT01, LM03, TKH+05]. J'ai eu le plaisir d'animer l'AS Collisions du CNRS, en 2002 et 2003, qui a permis de dégager une vue synthétique du domaine. La recherche est toujours active dans ce domaine car on est loin d'avoir trouvé des réponses générales au problème.

Mes travaux visent à trouver des méthodes permettant les simulations en temps réel. Dans ce domaine, il n'est pas simple de transiger entre précision (détection et traitement de tous les contacts) et rapidité (rapport entre temps simulé et temps réel). En effet, les erreurs mécaniques s'accumulant avec le temps, la simuation peut avoir tendance à diverger (explosion de la scène).

Mes travaux se sont d'abord inscrits dans le cadre d'une collaboration avec des industriels du jeu vidéo, dans le but d'animer des vêtements virtuels. J'ai ainsi encadré la thèse de Laks Raghupathi [Rag06] sur la détection et la réponse aux collisions entre surfaces déformables. Suite au retrait des industriels, nous nous sommes réorientés vers des applications médicales, en simulant le mésentère, une membrane complexe reliant l'intestin au tronc. Dans ce travail, nous avons étudié les différentes phases du traitement de contact entre les surfaces déformables, avec pour l'essentiel :

- détection de proximité en temps constant
- tests robustes entre primitives, au moyen de détection continue
- réaction au contact, par la résolution d'un système global plutôt que des réactions idépendantes à chaque contact.

Par la suite, j'ai eu la chance de participer au développement de SOFA, librairie de simulation physique prioritairement dédiée à la simulation médicale. La variété des objets et modèles intervenant dans un corps humain pose un défi rarement relevé dans le domaine de la simulation : l'interaction temps-réel d'objets de natures très diverses : solides rigides, solides déformables, liquides. Cela m'a donné l'occasion d'aborder les problèmes suivants :

- empêcher l'explosion combinatoire des méthodes spécialisées (ce travail est présenté plus précisément dans la section 1 du chapitre 4)
- permettre l'interaction de tout objet avec tout autre
- allonger les pas de temps

1 Introduction

Lorsque l'on désire faire interagir différents objets entre eux, ou un outil virtuel sur un objet, voire carrément un objet se repliant sur lui-même, il faut être en mesure de répondre aux possibles pénétrations entre les objets. Cette étape est appelée la gestion des collisions qui peut se décomposer en trois étapes : la détection des intersections, la modélisation des pénétrations (point de contact, normale au contact, profondeur de pénétration, instant du contact...) et la réaction à la collision. Ces trois étapes sont très complexes, spécialement concernant les objets déformables, car ceux-ci évoluant constamment au cours du temps, il est très compliqué de faire des précalculs puisque tout doit être remis à jour à chaque mouvement. De plus, les objets déformables peuvent s'auto-collisionner, compliquant grandement la tâche de détection.

C'est un domaine de recherche très actif, beaucoup de solutions très diverses ont été proposées ; un état de l'art peut être trouvé dans [TKH+05]. Malheureusement, elles sont fréquemment spécifiques, et aucune méthode vraiment universelle n'a encore été proposée. Par exemple, certaines méthodes sont seulement efficaces entre objets rigides [CLMP95, GLM96, GBF03], d'autres entre corps déformables et fluides [GSLF05], mais aucune ne s'adapte efficacement à tous les modèles, obligeant leurs utilisations conjointes pour un simulateur animant différents types d'objets. On peut quand même noter l'effort d'unification dans [FAN07], malgré le manque de performance. Beaucoup de méthodes efficaces sont trop dépendantes des modèles mécaniques sous-jacents et ne sont donc pas exploitables sur d'autres modèles [JP04]. La détection reste encore un sujet ouvert, et ne peut pas toujours se traiter indépendamment de la réponse.

Au niveau de la réaction, on peut distinguer deux grandes familles : le traitement par forces de pénalité ou le traitement par contraintes dures (sur les positions et/ou les vitesses). La grande difficulté à ce niveau est de proposer une méthode proche de la physique bien que la pénétration de matière n'existe pas dans la réalité. Notons que les méthodes dites "continues" tentent de détecter le moment et l'endroit exacts du contact, à l'intérieur d'un pas de temps, afin de ne pas avoir à gérer de pénétration de matière [BFA03]. À un niveau plus global, il faut aussi être capable de répondre à des collisions sans créer d'énergie ou créer d'autres intersections ailleurs, phénomène complexe à obtenir en temps réel.

Ce qui nous semble important à ce niveau, et qui est très rarement soulevé est l'importance d'être en mesure de choisir la résolution à laquelle vont être gérées les collisions. L'approche classique consiste à considérer en dernière phase précise, le modèle surfacique utilisé pour l'affichage. Or celui-ci peut être très détaillé et donc énormément alourdir la recherche de collisions, pour laquelle un modèle plus grossier pourrait faire l'affaire. Dans ce cas, il faut être cependant en mesure de relier différents maillages et de faire passer des contraintes de l'un à l'autre tout en respectant la physique, comme proposé dans [ACF+07].

2 Contacts entre objets surfaciques

2.1 Détection de proximité

La détection de proximité consiste à effectuer des tests simples permettant d'éliminer des tests précis plus coûteux. Plutôt que de tester toutes les paires de primitives, on commence par chercher les régions entre lesquelles la probabilité de contact est non nulle. Les test précis sont alors limités à ces régions. La plupart des méthodes reposent sur des grilles ou des volumes englobants, éventuellement hiérarchiques.

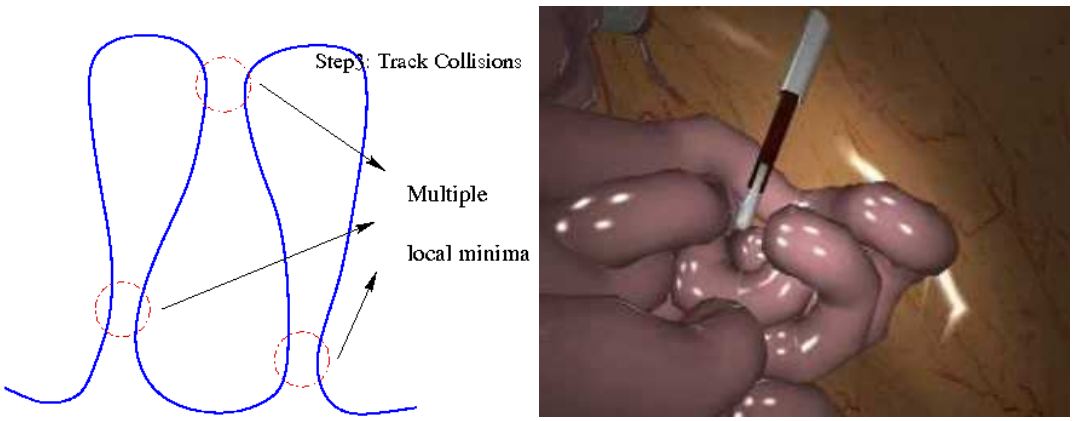


FIG. 3.2 – Détection de proximité en temps limité.

Gauche : minima locaux. Droite : application.

La complexité est alors linéaire dans les bons cas, mais sans garantie en général. Or dans une application temps-réel, il est dangereux de ne pas maîtriser le temps de calcul. Nous avons donc développé une nouvelle approche permettant de limiter facilement le temps de calcul, au prix d'une possible perte d'informations. Pour cela, nous nous sommes inspirés de travaux de Gilles Debunne [GD04] utilisant des tirages aléatoires de paires de primitives, suivis de convergences vers des minima locaux. La convergence s'effectue de manière gloutonne, chaque primitive de la surface étant successivement remplacée par sa voisine la plus proche de l'autre primitive. On arrête ce processus dès que le temps de calcul autorisé est atteint. La cohérence temporelle est exploitée en conservant les minima locaux d'un instant à l'autre. Cette technique nous a permis d'effectuer la simulation en temps réel d'un intestion et son mésentère, comme illustré sur la figure 3.2. Elle s'applique aux objets surfaciques et volumiques, et a donné lieu à plusieurs publications [FAM+02, RCFC03, RGF+04, FLA+05].

L'inconvénient de cette approche est qu'elle ne garantit pas de trouver toutes les collisions. Pour améliorer sa fiabilité, nous l'avons couplée à une méthode utilisant des hiérarchies de volumes englobants. Nous effectuons une première passe de détection de proximité entre volumes englobants, sans raffiner les volumes pour assurer la rapidité. Puis, nous choisissons les paires dans les volumes en intersection. Nous obtenons ainsi des accélérations importantes par rapport à une détection hiérarchique classique [KNF04]. La figure 3.3 présente des résultats. Nous voyons tout de même que des artefacts apparaissent quand le temps de calcul est trop réduit. Pour obtenir des accélérations de calcul intéressantes par rapport à la méthode des volumes hiérarchiques effectuée jusqu'au niveau le plus fin, il est nécessaire d'ajuster les paramètres : profondeur de recherche dans les arbres, nombre de paires tirées. Cet ajustement s'effectuant au coup par coup, on peut en déduire que la méthode est mal adaptée à la simulation facile de scènes arbitraires. Notons que le tirage aléatoire n'est pas la panacée. Par exemple, la zone sous le bras du personnage collisionne presque toujours avec la même partie du corps. A l'avenir, il serait utile de tenir compte de ce genre de connaissance a priori.

2.2 Détection continue

L'intersection d'objets surfaciques est un problème particulièrement difficile, car ces objets ne possédant pas d'intérieur et d'extérieur, il est difficile de savoir dans quelle direction appliquer des forces de répulsions une fois l'intersection détectée. C'est pourquoi la plupart des méthodes considèrent que la configuration à l'instant considéré est sans intersection, et cherchent un état futur lui aussi sans intersection. Si, après intégration des positions, le nouvel état comporte des intersections, alors on retourne en arrière avant la collision, on traite cette collision généralement par une impulsion annulant les vitesses relatives, puis on poursuit l'intégration du temps. Ces interruptions peuvent ralentir considérablement la simulation. Nous

FIG. 3.3 – Méthode hybride

Gauche : volumes englobants hiérarchiques fins. Centre : méthode hybride. Droite : méthode hybride, trop peu de paires.

nous sommes donc tournés vers la détection continue, qui permet en principe de prévoir les collisions. Notre idée était de prévenir les collisions futures en installant des interactions raides entre les primitives proches de la collision. Nous espérions ainsi pouvoir appliquer des pas de temps constants, nécessaires pour les simulations temps-réel. Malheureusement, la détection continue souffre de nombreux problèmes numériques et cas particuliers. En pratique, il semble impossible d'empêcher toutes les intersections à l'aide de cette technique. Même si la détection continue est très utile, elle ne suffit pas et il reste à développer des méthodes robustes et rapides pour séparer les intersections. La solution proposée par Baraff [BWK03] est lente et ne convient qu'aux simulations non interactives. L'approche de Volino [VMT06], par minimisation des longueurs d'intersections, me paraît très prometteuse.

2.3 Réaction au contact

La plupart des méthodes de réaction entre objets déformables sont locales : les positions ou les vitesses des primitives en contact sont mises à jour, indépendamment des autres éléments des objets. Par conséquent, l'influence d'un contact ne peut se transmettre qu'à l'instant suivant aux particules voisines, et des déformations locales se créent même si les objets sont très raides. Ce manque de réalisme n'empêche pas de réaliser de très belles simulations de vêtements, mais une réponse globale aux contacts est nécessaire en général. Répondre globalement nécesite de prendre en compte toute la dynamique interne des objets. Dans le cas des objets déformables, celle-ci s'exprime par l'intégration implicite des forces internes et externes. Par conséquent, la réponse au contact doit être fusionnée à l'intégration implicite. Celle-ci nécessitant la résolution d'un système d'équations, elle n'est généralement envisageable en temps réel que par des méthodes itératives de type gradient conjugué, permettant de contrôler le temps de calcul. Nous avons donc cherché à exprimer le contact comme des équations supplémentaires dans le système résolu par gradient conjugué. En exprimant les contact par des équations sur les positions ou les vitesses, on ajoute des multiplicateurs de Lagrange au système d'équations. Malheureusement, les équations peuvent être fortement redondantes et le gradient conjugué et ses variantes échouent à résoudre efficacement le système. Nous nous avons donc développé une méthode utilisant des ressorts entre primitives, ce qui garantit d'aboutir à des matrices symétriques définies positives, compatibles avec le gradient conjugué [Rag06]. La prise en compte des interactions futures modifiant les vitesses, de nouvelles collisions peuvent apparaître et il faut donc itérer. Cette méthode, illustrée sur la figure fonctionne assez bien mais nous nous sommes heurtés au problème des détections évoqué en section 2.1.

Quand les surfaces s'intersectent en fin de pas de temps à cause des imperfections de détection, les répulsions entre particules tendent à maintenir l'intersection. Il serait donc utile d'appliquer, après intégration du temps, une méthode robuste pour séparer les objets, comme évoqué précédemment. D'autre part,

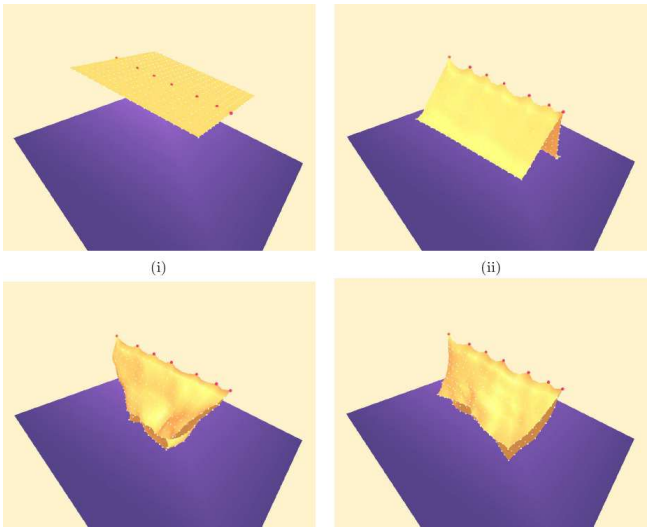


FIG. 3.4 – Simulation d'un drap

Autocollisions gérées par des forces élastiques.

les équations de contact étant non linéaires (répulsion seulement, frottement de Coulomb), chaque système d'équations correspond à des hypothèses sur les contacts (état=décollement ou adhérence ou glissement) qui ne sont pas forcément vérifiées à l'issue du calcul. Il conviendrait donc de réitérer la résolution après mises à jour des hypothèses, mais le temps de calcul peut alors devenir prohibitif. Des méthodes à base de solveurs de linéarité complémentaire (LCP) ont été proposées [Dur04], mais le temps de calcul semble incompatible avec la simulation en temps réel de scènes complexes. Les résolutions locales du contact ont donc encore de beaux jours devant elles, d'autant que ce problème ne semble pas passionner les chercheurs actuellement. Je crois que la résolution efficace des contacts reste un des enjeux majeurs de la recherche en simulation.

3 Contact entre objets volumiques

3.1 Grille eulérienne

La plupart des méthodes de contact entre objets déformables ont été étudiées dans le contexte de l'animation de vêtements. Comme expliqué précédemment, il est important dans ce cadre d'empêcher toute intersection, car les répulsions tendent à la maintenir. On est donc parfois amené à réduire le pas de temps, ce qui rend le temps de calcul imprévisible, et qui empêche donc de développer des applications temps-réel fiables. Les scènes chirurgicales comportent de nombreux objets volumiques déformables, qui posent un problème différent. D'une part, on ne visualise que les surfaces libres des objets, donc les intersections peuvent être visuellement acceptables. D'autre part, ces objets possédant un intérieur et un extérieur, il est en principe beaucoup plus facile de les séparer, en repoussant les intrus vers l'extérieur. On peut donc espérer des gains importants par rapport aux méthodes issues de l'animation de vêtements. La difficulté est de trouver dans quelle direction repousser les points. Il semble naturel de repousser les points d'une surface pénétrante vers les points les plus proches de la surface pénétrée. Malheureusement, trouver ces points est complexe en général. Dans le cas d'objets rigides, on peut précalculer une carte de distance dont le gradient donne une bonne indication de la surface la plus proche. Pour les objets déformables, le recalcul de la carte de distance à chaque pas de temps est prohibitif. Avec Matthieu Nesme et Jérémie Allard, nous avons donc reformulé le problème de manière à se passer de carte de distance. Chaque objet s'échantillonne volumiquement dans une grille régulière commune, qui contient dans chaque case une liste d'objets pré-

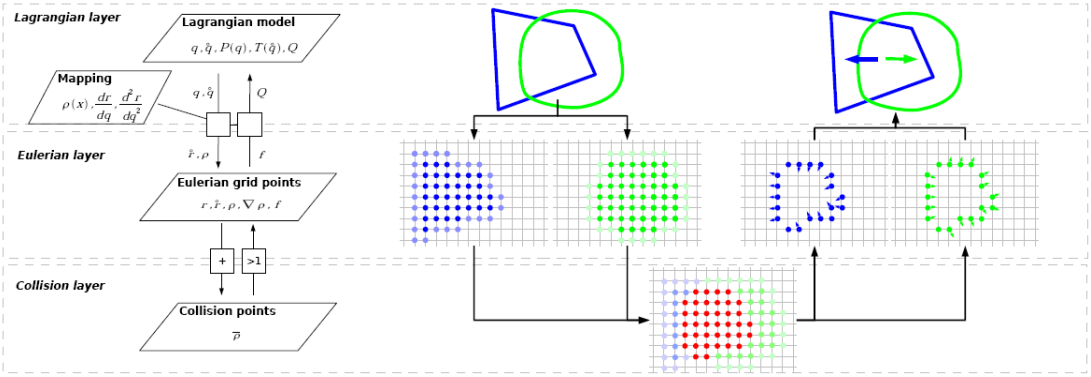


FIG. 3.5 – Contacts eulériens.

La couche supérieure contient les modèles mécaniques. les volumes sont échantillonnés dans la couche eulérienne, au milieu, et des mappings sont utilisées pour répercuter vitesses et forces. La couche inférieure représente les sommes de densités.

sents et le taux d'occupation spatiale (densité) de chaque objet dans la case. Une collision est détectée là où la somme des taux excède 100%. Dans les cases en collision, des forces de répulsion sont appliquées selon les gradients de densité, puis reportées aux degrés de liberté de objets par la technique du mapping présentée au chapitre 1. La méthode complète est illustrée sur la figure 3.5.

Cette méthode permet d'appliquer des répulsions entre volumes sans aucun calcul de distance. Elle est applicable à tout corps capable de se décrire comme un champ de densité spatiale, et de répercuter des forces. Nous l'avons appliquée avec succès à des solides rigides, déformables, ainsi qu'à des fluides [FAN07], comme illustré sur les figures 3.1 et 3.6.

Cette méthode présente un avantage important par rapport aux méthodes appliquant des répulsions entre polygones : elle est robuste aux intersections, même profondes. On peut ainsi appliquer des pas de temps constants, caractéristique nécessaire aux applications en temps réel. Elle est toutefois lente, car chaque corps s'échantillonne dans les cases d'une grille volumique, et y calcule la formule de répartition des forces vers ses degrés de liberté. Par la suite, j'ai donc cherché à conserver l'avantage de la robustesse, tout en éliminant la représentation volumique.

3.2 Lancer de rayons

Les objets sont généralement décrits par des surfaces, qui représentent des quantités de données beaucoup plus faibles que des volumes. De plus, les répulsions basées sur la densité ne s'appliquent que sur les surfaces des objets, là où les gradients de densité ne sont pas nuls, comme on le voit au milieu à droite de la figure 3.5. Il doit donc être possible d'expoiter seulement les modèles surfaciques, en appliquant les forces selon les normales. C'est ce que nous avons essayé avec Everton Hermann [HFR08]. Là encore, nous évitons de calculer des distances, en effectuant des lancers de rayons. Un rayon est tiré à partir de chaque sommet de la surface, parallèlement à la normale. Les orientations des surfaces intersectées par le rayon permettent de déduire si le sommet est à l'intérieur d'un objet, comme le point 1 sur la figure 3.7 percutant l'intérieur d'un objet au point $1'$. La force de répulsion s'applique alors dans la direction $11'$. Cette orientation peut être fausse, mais l'erreur diminue quand les objets sont lisses, comme illustré à droite de la figure 3.7.

Cette méthode remplace donc le calcul de plus proche primitive, qui est une recherche en trois dimensions, par un lancer de rayon, qui est une recherche en une dimension. Implémentée à l'aide d'un octree, elle obtient des temps de calcul comparables à notre méthode de référence, appliquant des répulsions entre plus proches primitives, elle aussi accélérée par des grilles hiérarchiques. L'avantage est qu'elle est beaucoup plus robuste aux intersections profondes, comme illustré sur la figure 3.8.

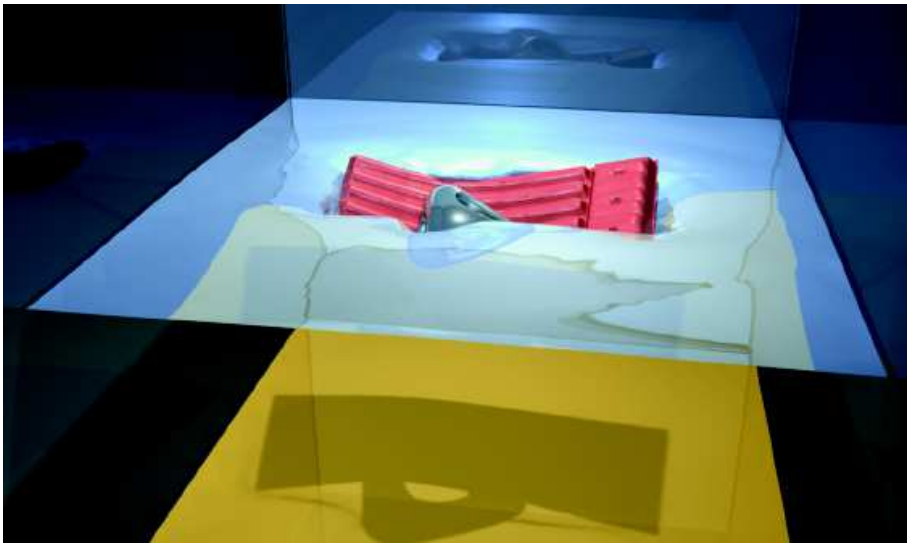


FIG. 3.6 – Applications des contacts eulériens.

Un solide rigide tombe sur un solide déformable (éléments finis tétraèdriques) flottant sur un liquide eulérien.

Un inconvénient de cette méthode, qu'on retrouve dans bien d'autres et qui est rarement discuté, est que l'intensité des forces est délicate à régler. Si on se contente d'appliquer des forces proportionnelles aux distances entre les points correspondants (commme 1 et $1'$ sur la figure 3.7) alors l'intensité des forces dépend de la finesse du maillage : plus il y a de sommets immergés, plus la force est importante. Une solution raisonnable est de pondérer ces forces par l'aire des triangles adjacents aux sommets. Toutefois, seule la partie immergée des triangles est pertinente. Quand les triangles sont grands, des erreurs importantes peuvent se produire aux triangles partiellement immergés. Calculer la surface immergée nécessiterait des calculs complexes. Notons que la méthode à base de grille eulérienne présentée en section 3.1 ne présente pas ce problème, car la quantité immergée est approchée par un nombre de cases.

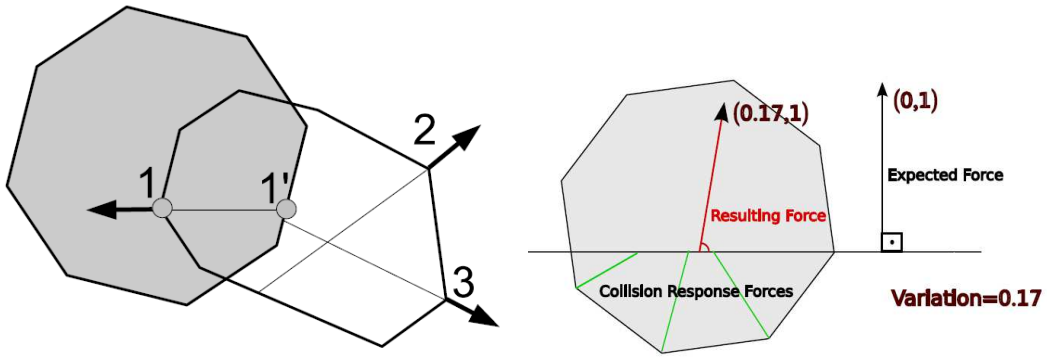


FIG. 3.7 – Détection de collision par lancer de rayons.

À gauche, un point 1 *est détecté en collision, et une force appliquée entre lui et son correspondant* $1'$. *À droite, la force nette appliquée par un cylindre discrétisé n'a pas exactement la bonne direction.*

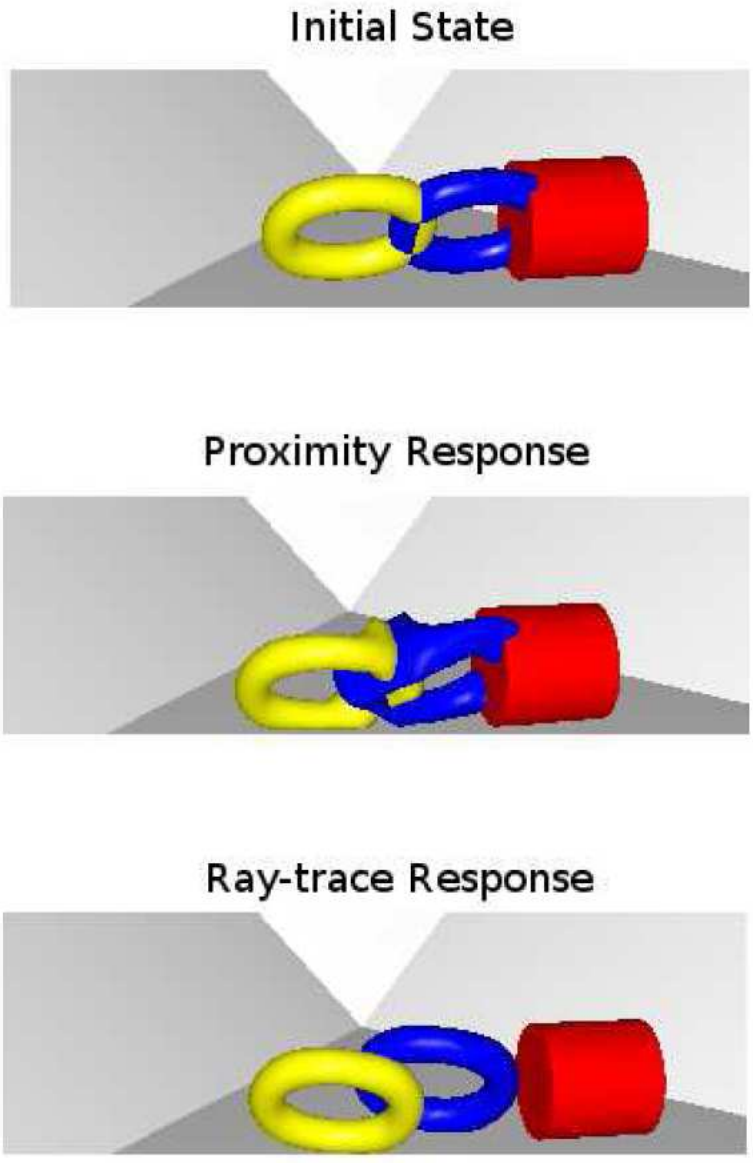


FIG. 3.8 – Essai de robustesse de la méthode par lancer de rayons.

En haut : état de départ. Au milieu : une méthode à base de proximités échoue. En bas : notre méthode par lancer de rayons parvient facilement à séparer les objets.

3.3 Méthode à base d'images

La méthode présentée ici regroupe les avantages des deux précédentes et en élimine les inconvénients : elle est robuste aux intersections profondes, ne calcule pas de distances, n'utilise que les surfaces, et est très peu sensible à la discrétisation des objets. De plus, elle délègue la plupart des calculs au matériel graphique (GPU), et un certain compromis entre précision et temps de calcul peut être facilement réglé.

Les surfaces sont échantillonnées sous forme de textures par des opérations de rendu sur le GPU. L'ensemble des surfaces projetées, et pas seulement la surface au premier plan, sont stockées dans un ensemble d'images appelé LDI (Layered Depth Images). Chaque pixel contient entre autres sa profondeur et l'indice du triangle auquel il appartient. Les collisions correspondent à de simples intersections d'intervalle, et les auto-collisions peuvent être détectées d'après les orientations des normales [HTG03, HTG04b]. Cette technique de détection, présentée par Heidelberger *et al.*, ne donnait que des réponses binaires.

Avec Florent Falipou, Sébastien Barbier et Jérémie Allard, nous l'avons généralisée pour déduire des forces de répulsion [FBAF08]. Nous avons d'abord remarqué que le volume d'intersection, en jaune sur la figure 3.9, se calcule simplement par somme des intervalles multipliée par surface d'un pixel. Notre principale contribution a été de montrer que la dérivée de ce volume par rapport aux coordonnées des sommets se calcule très facilement d'après les coefficients de Gouraud utilisés pour interpoler pendant le rendu. La force de répulsion, qui tend à réduire le volume d'intersection, est parallèle à cette direction, et nous l'avons choisie proportionnelle au volume d'intersection. On peut montrer qu'elle est équivalente à des forces de pression s'appliquant à la surface du volume d'intersection.

Cette méthode exerce les mêmes répulsions que la grille eulérienne, mais sans avoir à échantillonner volumiquement les objets. Nous obtenons donc des temps de calcul considérablement meilleurs. Le poulpe à gauche de la figure 3.10 est animé en temps réel. Les formes anguleuses sont aussi bien simulables que les formes lisses, et les intersections d'arêtes sont automatiquement traitées, comme illustré sur la figure 3.11.

Nous pensons que cette méthode est très bien adaptée au traitement des contacts entre objets volumiques quelconques. En effet, aucun précalcul n'est nécessaire, elle est donc compatible avec les objets déformables aussi bien que rigides, y compris avec auto-collisions, et des modifications de topologie comme des découpes semblent possibles. Elle est robuste aux intersections profondes, ce qui permet d'apliquer les pas de temps constants nécessaires aux applications temps-réel. Les forces de contact peuvent être entièrement calculées sur GPU, ce qui libère le CPU pour d'autres tâches. Un compromis entre précision et temps de calcul peut être obtenu en choisissant la résolution appropriée dans les LDIs, comme illustré sur la figure 3.12.

Des défauts subsistent, comme la possibilité que des objets fins se croisent au cours d'un pas de temps. Pour une meilleure fiabilité, il serait donc nécessaire d'étendre la méthode à la détection continue, ou de la combiner à une technique existante.

L'application de forces de pénalités peut générer d'importantes discontinuités de forces au moment où les objets commencent à s'interpénétrer. Il conviendra d'appliquer un traitement moins naïf.

La limitation à des objets volumiques devra être assouplie. Il semble possible de traiter le contact entre un objet non volumique (ligne, surface déformable) et un objet volumique. Toutefois, la collision entre objets non volumiques devra se faire, soit en les enrobant de volumes, soit par d'autres méthodes.

Le temps de calcul est essentiellement passé à effectuer les rendus, et cette partie reste proportionnelle à la complexité géométrique, quelle que soit la résolution des LDIs. Des surfaces multi-resolutions pourraient aider à accélérer les calculs.

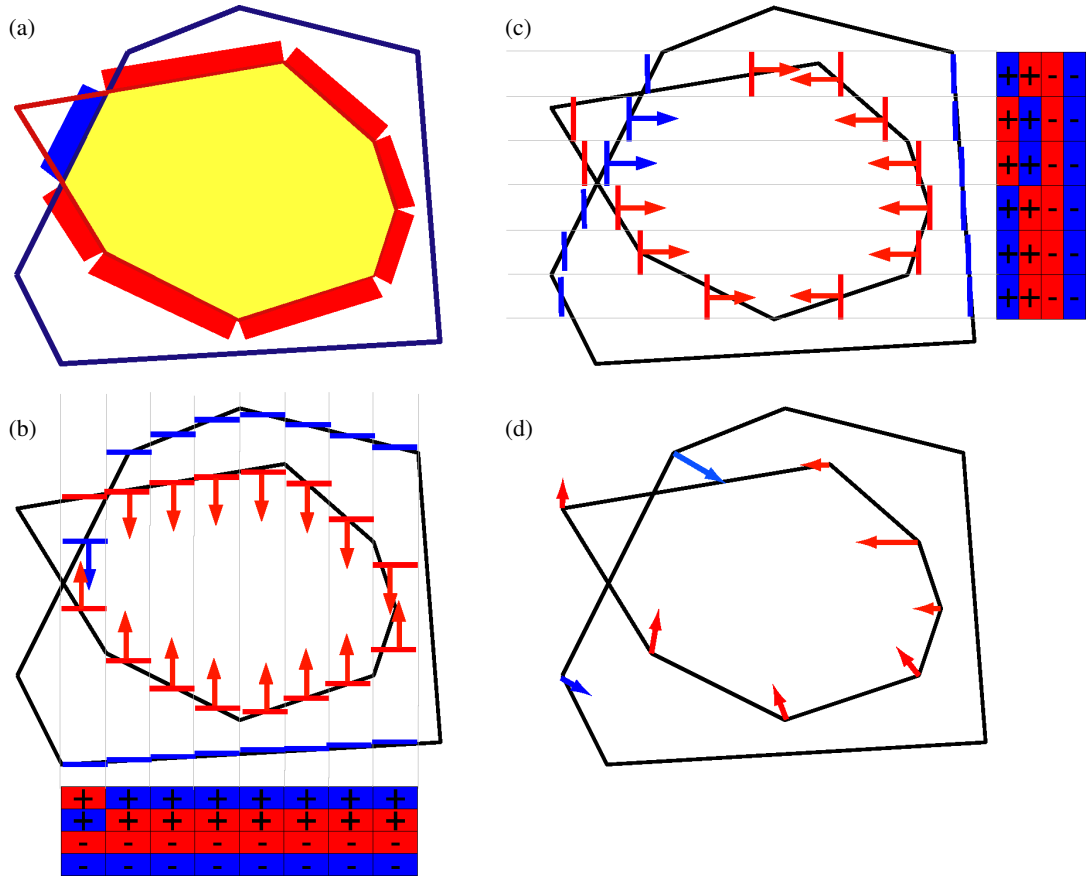


FIG. 3.9 – Vue d'ensemble de la méthode à base d'images.

(a) : deux objets en contact, et leur volume d'intersection en jaune, qui subit des forces de pression à sa surface (rectangles rouge et bleu). (b) : pixélisation des surfaces dans la direction verticale, et intervalles utilisés pour détecter la collision. Les flèches représentent les forces de répulsion. les couleurs représentent les objets, et les signes représentent l'orientation des normales. (c) : pixelisation dans la direction horizontale, fournissant une composante supplémentaire des forces de contacts. (d) : les forces transmises aux sommets.

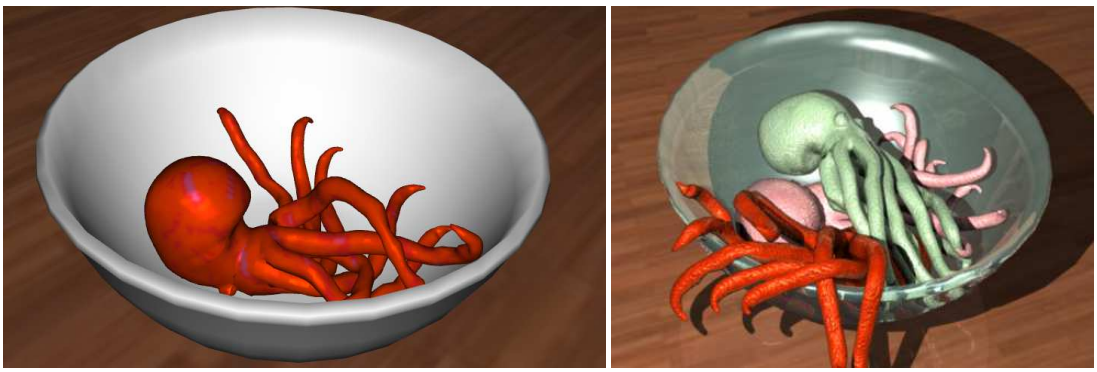


FIG. 3.10 – Objets très déformables avec collisions et auto-collisions.

Gauche : poulpe enrobé par skinning dans un bol rigide, 20000 triangles, 25 fps. Droite : Trois instances du même, non interactif.

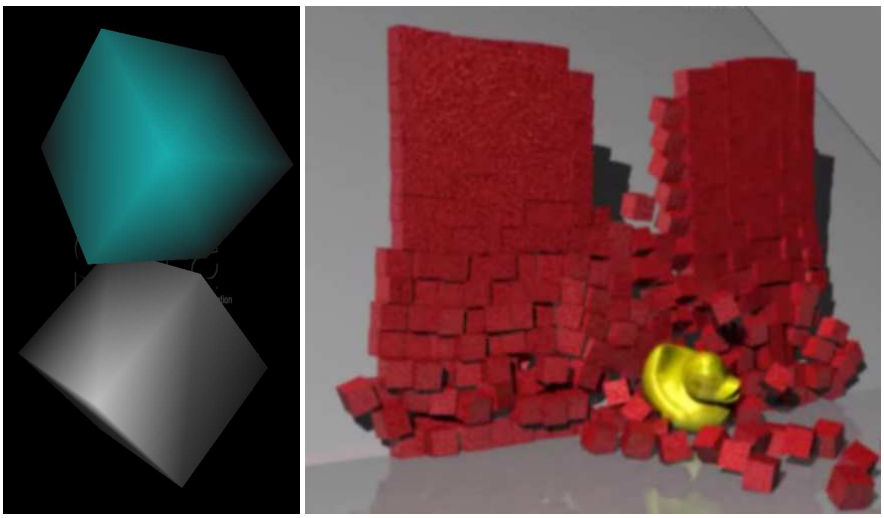


FIG. 3.11 – Géométries anguleuses.

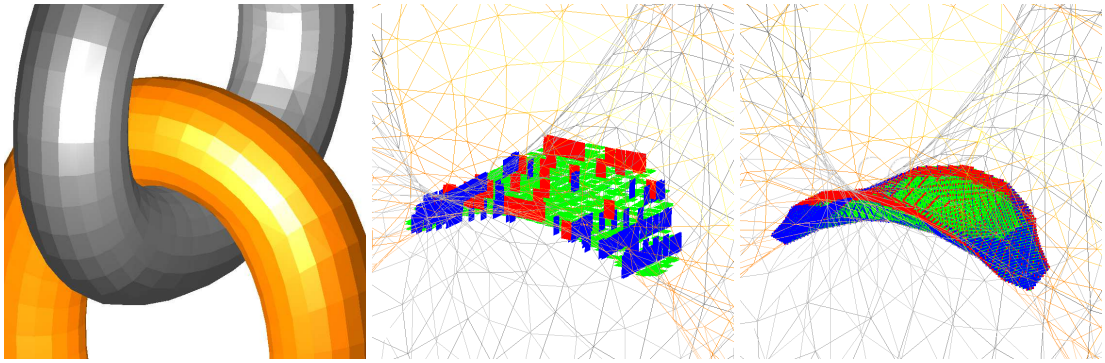


FIG. 3.12 – Volume d'intersection à différentes résolutions.

Gauche : les surfaces de contact. Milieu : le volume d'intersection échantillonné dans des images de 16×16. Les pixels rouges, verts et bleus montrent les limites du volume dans les directions x, y, et z, respectivement. Droite : à la résolution 256×256.

chapitre 4

Architecture logicielle

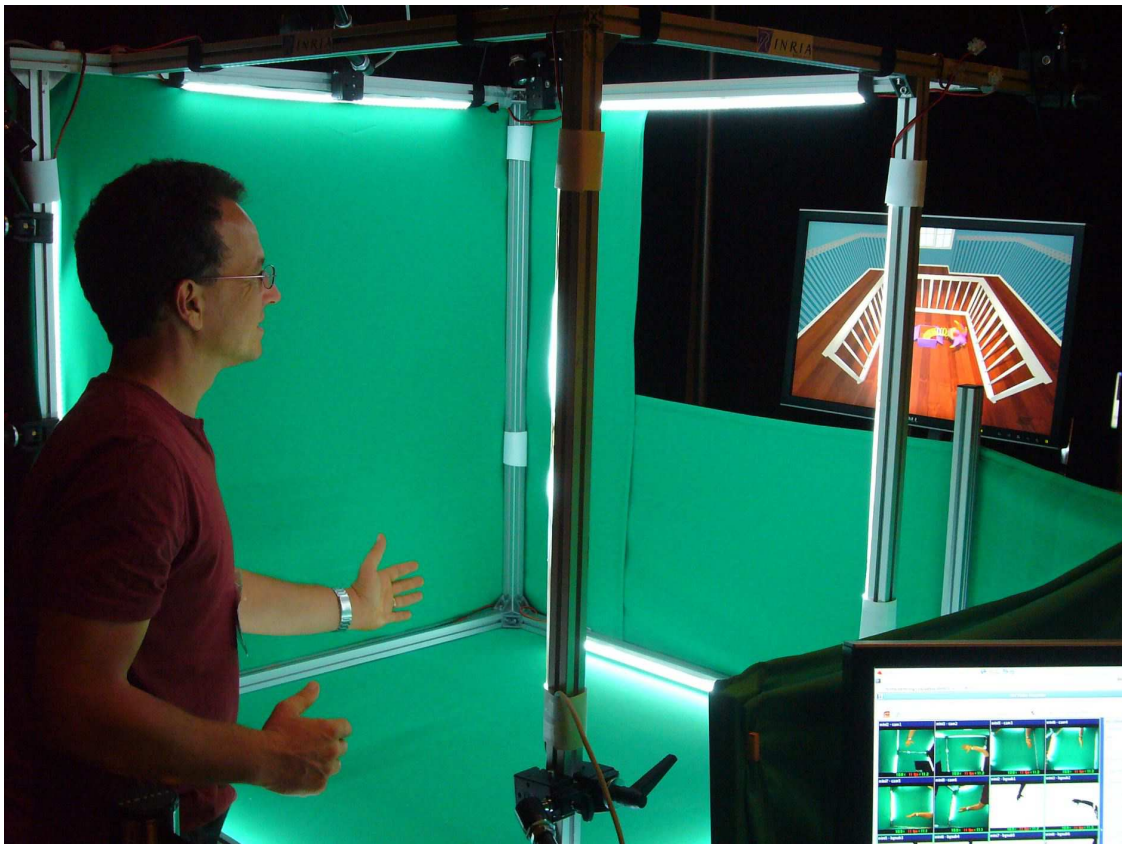


FIG. 4.1 – Interactions en temps réel.

Un cluster de PC reconstruit en temps réel les objets plongés dans le volume de capture et les fait interagir avec des objets virtuels.

Mes travaux visent à rendre la simulation physique robuste, rapide, et simple à mettre en oeuvre. Les simulations complexes mettent en jeu une grande quantité de modèles et d'algorithmes et leur programmation fait appel à des compétences diverses (mécanique, géométrie, algorithmique, rendu, etc.) rarement réunies chez une seule personne, en admettant qu'elle ait le temps de les implémenter. En pratique, le développement d'une plate-forme de simulation versatile et efficace ne peut s'envisager que par un travail collaboratif de longue haleine entre spécialistes de différentes disciplines. Pour cela, il est nécessaire de créer une architecture logicielle qui permette de combiner aisément des modules indépendants. De nombreuses tentatives ont été effectuées, mais à ma connaissance toutes échouent jusqu'ici sur au moins un des points suivants :

- pouvoir inclure tous les modèles géométriques et physiques
- pouvoir leur appliquer tous les algorithmes d'animation
- obtenir des temps de calcul proches de l'optimal

La recherche dans ce domaine est difficile à valoriser d'un point de vue académique, mais je suis persuadé qu'elle est essentielle et j'y ai consacré une grande part de mes travaux. Je cherche la généricité en explorant des extensions du concept de graphe de scène, très utilisé en rendu mais peu en simulation. Inspiré par les travaux pionniers de Jean-Dominique Gascuel sur la plate-forme Fabule, j'ai développé avec David Bourguignon et Laure Heïgeas une librairie, AnimAL, capable de représenter les scènes mécaniques par des modules hiérarchiques offrant non seulement une certaine variété de modèles, mais aussi un certain choix d'algorithmes d'animation. Puis j'ai eu la chance d'être invité par Stéphane Cotin à participer au développement de SOFA [SOF], librairie de simulation physique principalement dédiée à la simulation

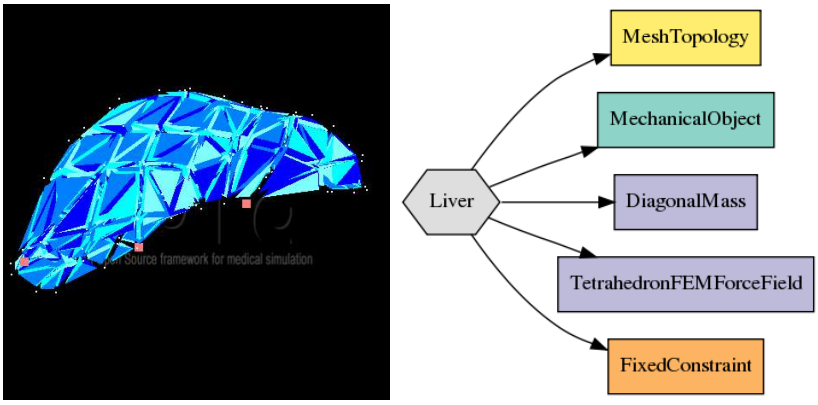


FIG. 4.2 – Modèle simple de foie.

Gauche : les degrés de liberté (points blancs), contraintes (points rouges) et éléments finis (tétraèdres). Droite : représentation en plusieurs composants rassemblés dans un groupe.

médicale. Sa vision était que les objets doivent être représentés par plusieurs modèles, chacun spécialisé dans une sous-tâche de simulation comme la mécanique interne, la détection de collision, le rendu visuel et haptique. Avec Jérémie Allard et Hervé Delingette, nous y avons appliqué certains concepts introduits dans AnimAL, et créé d'autres répondant à son cahier des charges. Grâce à la participation de nombreuses personnes dont la liste serait trop longue à énuérer ici, SOFA s'est beaucoup étoffé et suscite aujourd'hui un intérêt croissant dans la communauté médicale [ACF+07, FAC+07]. On ne trouvera pas ici de description complète de SOFA , mais la section 1 présente ce que je pense être mes principales contributions son architecture générale.

Obtenir de bons temps de calculs nécessite un effort important de génie logiciel pour organiser les données de manière efficace. Toutefois, la puissance des processeurs étant limitée, même la meilleure implémentation ne peut permettre de simuler interactivement des scènes au-delà d'un certain niveau de complexité. L'autre manière de gagner du temps est de paralléliser les calculs. Co-encadrer la thèse de Florence Zara avec Jean-Marc Vincent sur la simulation de textiles sur grappes d'ordinateurs m'a permis de m'initier au parallélisme et de découvrir les problématiques spécifiques à la simulation physique en parallèle. J'ai poursuivi l'expérience en co-encadrant depuis deux ans la thèse d'Everton Hermann avec Bruno Raffin, sur la parallélisation de SOFA . Ces travaux sont présentés en section 2.

1 SOFA

1.1 Modèles mécaniques composites

Pour programmer simplement la simulation d'objets complexes, nous avons décidé de fractionner les modèles en entités que nous nommons *composants*. Une version préliminaire de modèles composites avait été testée dans AnimAL, une librairie de simulation développée précédemment à EVASION. Chaque composant représente un aspect du modèle, et est interchangeable avec des composants du même type. La figure 4.2 en présente un exemple. La géométrie de l'objet est rerpésentée par une topologie (`MeshTopology`) et des degrés de liberté (`MechanicalObject`). Le matériau est représenté par la masse (`DiagonalMass`) et un champ de forces (`TetrahedronFEMForceField`). La fixation de trois points dans l'espace est assurée par une contrainte simple (`FixedConstraint`). La boucle principale d'un simulateur dédié à une telle scène pourrait ressembler à :

$f = \text{weight}(m, g)$
$f+ = \text{FemForce}(x, v)$
$a = f/M$
$a = \text{constraint}(a)$
$v+ = a * dt$

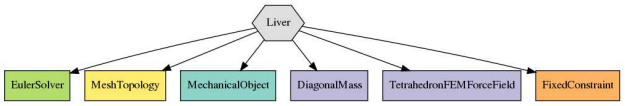


FIG. 4.3 – Modèle simple de foie avec un algorithme d'animation (EulerSolver).

$x+ = v * dt$
display()

Dans notre architecture composite, elle ressemble plutôt à :

$f = 0$
for all F in ForceFields **do**
$f+ = F(x, v)$
end for
$a = f/M$
for all C in Constraints **do**
$a = C(a)$
end for
$v+ = a * dt$
$x+ = v * dt$
display()

Cette approche apporte beaucoup de souplesse. Le calcul des forces peut être modifié sans changer le reste du programme. On peut rajouter des champs de forces, par exemple, des tensions superficielles sur les faces externes des tétraèdres. On peut aussi en créer de nouveaux, par exemple, remplacer les éléments finis par des ressorts sur les arêtes. En programmation objet, cela utilise un méthode virtuelle :

```
1 void BaseForceField::addForce(); // f += F(x,v)
```

De même, on peut définir et appliquer un nombre quelconque de contraintes, pourvu qu'elles implémentent les méthodes virtuelles :

```
1 void BaseConstraint:: projectPosition (); // x=C(x)
2 void BaseConstraint:: projectVelocity (); // v=C'(v)
```

Chaque composant peut accèder à la topologie et aux degrés de liberté. C'est ainsi que les forces et contraintes accèdent aux vecteurs d'état. La topologie n'est pas explicitement appelée pendant la boucle de simulation, mais elle sert lors de l'initialisation. Par exemple, le `TetrahedronFEMForceField` s'initialise d'après la liste des tétraèdres fournie par la topologie.

Il n'y a qu'une seule masse par groupe. Elle dérive d'un champ de force, mais doit en plus implémenter :

```
1 void BaseMass::accFromF(); // a=f/M
```

Notons que l'algorithme de simulation, ici une simple intération explicite du temps, reste "en dur" et n'est pas modulaire. Si nous voulons le remplacer, par exemple, par un schéma de Runge-Kutta, il faut récrire le programme. Pour résoudre ce problème, nous allons naturellement encapsuler l'algorithme dans une nouvelle classe de composant implémentant :

```
1 void OdeSolver::solve( double dt ); // remplace x(t) et v(t) par x(t+dt) and v(t+dt)
```

Un composant est ajouté à l'objet comme illustré sur la figure 4.3, et l'algorithme d'animation devient :

odeSolver.solve(dt)
display()

La méthode `solve` du `EulerSolver` s'écrit alors :

computeForce(f)
accFromF(a, f)
projectResponse(a)

$v+ = a * dt$
$x+ = v * dt$

et celle du `RungeKutta2Solver` :

computeAcc(a, x, v) // computeForce, accFromF, projectResponse
$x1 = x + v * dt/2$
$v1 = v + a * dt/2$
computeAcc($a, x1, v1$)
$v+ = a * dt$
$x+ = v * dt$

Avec un telle architecture, il est possible de comparer différents algorithmes de simulation sur un même modèle, et d'en développer de nouveaux sans interférer avec d'autres développements. Cette caractéristique est très importante pour le développement collaboratif. Pour plus de souplesse, les classes de SOFA qui manipulent des vecteurs d'état sont paramétrées par les types qu'elles manipulent (*template* en C++).

1.2 Modèles hiérarchiques

Dans une simulation mécanique, les objets interagissent par l'intermédiaires d'actions mécaniques : forces ou contraintes. Cela permet par exemple de fixer des parties d'objets, de les attacher à d'autres, ou d'appliquer des répulsions. Sans elles, les objets ne peuvent pas entrer en contact. Les actions les plus faciles à appliquer sont les forces, puisqu'elles s'ajoutent simplement au bilan des forces appliquées aux objets, sans modification des algorithmes de simulation. Toutefois, quand ces forces varient fortement en fontion des mouvements (grandes raideurs), la stabilité de la simulation peut être remise en cause, imposant alors de plus courts pas de temps. Au contraire, les déplacements imposés nécessitent généralement de résoudre des équations, dans lesquelles ces déplacements forment les contraintes.

Les forces et les équations portent sur les degrés de liberté des objets, mais souvent indirectement, et il est alors nécessaire de les transformer. Par exemple, la force appliquée en un point d'un triangle est transformée en trois forces appliquées aux sommets. Et si c'est un déplacement qui y est imposé, alors il est nécessaire de le transformer en une équation sur le déplacement des sommets. Parfois, les sommets ne sont pas des degrés de liberté indépendants, mais des points attachés à deux-ci. Par exemple, une surface attachée à un solide rigide, ou plongée dans des tétraèdres déformables. Il peut encore y avoir encore des niveaux supplémentaires, comme les coordonnées articulaires d'un modèle à squelette hiérarchique. De nouvelles reformulations sont alors nécessaires pour les exprimer enfin les actions mécaniques sur les degrés de liberté indépendants.

Dans SOFA , nous avons organisé ces niveaux en couches reliées par des relations que nous appelons *MechanicalMappings*. Cela définit une hiérarchie cinématique, avec au sommet les degrés de liberté indépendants. Par assemblage de couches, on peut modéliser simplement des objets complexes. L'important pour la modularité est qu'à chaque couche, le modèle local puisse être défini indépendemment des autres, en appliquant les mappings adéquats. La figure 4.4 montre l'exemple d'un foie. Il est affiché à l'aide d'un maillage triangulaire très fin, en rouge, afin d'assurer une bonne qualité visuelle. Ce même maillage peut servir à représenter la surface de contact du foie avec d'autres objets. Ce foie est modélisé mécaniquement comme un milieu continu viscoélastique maillé en éléments finis tétraèdriques. Ceux-ci pourraient être basés sur les triangles de la surface, mais cela aboutirait à un nombre très élevé de primitives mécaniques qui ne permettrait pas la simulation en temps réel.

Ma contribution a été de définir les opérations génériques permettant de réaliser ces mappings mécaniques. Elles sont au nombre de trois :

```
1 apply( VecCoord& x_fils, const VecCoord& x_pere ); /// Calcule les coordonnées du niveau fils d'après celles de son père
2 applyJ( VecDeriv& v_fils, const VecDeriv& v_pere ); /// Calcule les vitesses du fils d'après les vitesses du père
3 applyJT( VecDeriv& f_pere, const VecDeriv& f_fils ); /// Accumule sur le père les forces appliquées au fils
```

Voyons à quoi elles correspondent sur la figure 4.4 qui illustre la modélisation hiérarchique du foie. Les degrés de liberté indépendants sont les sommets des tétraèdres, ils sont donc au sommet de la hiérarchie. D'après leurs positions x_0, on peut déduire celles des sommets de la surface (x_1), ce qui est représenté par l'opérateur $\mathcal{J}$. Cette opération est implémentée par la méthode `apply`. D'après leurs vitesses v_0, on peut aussi déduire celles des sommets de la surface (v_1), par une relation linéaire représentée par la matrice

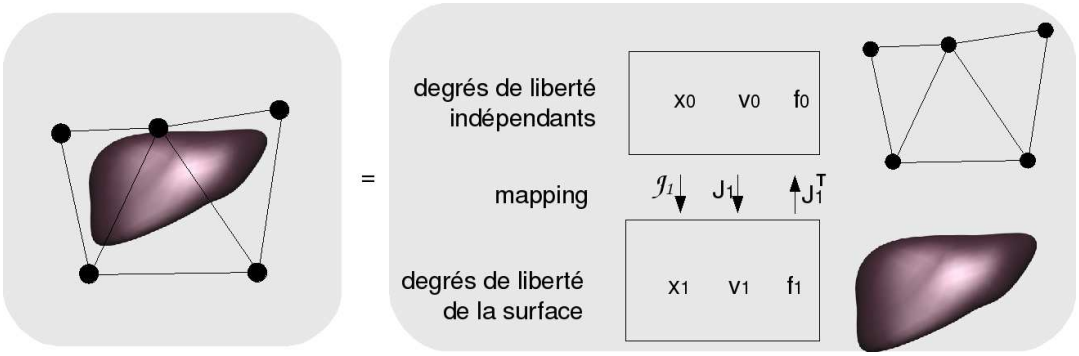


FIG. 4.4 – Hiérarchie de modèles.
Le mapping assure la transmission des positions et déplacements de haut en bas, ainsi que la transmission des forces de bas en haut.

J. Cette opération est implémentée par la méthode `applyJ`. Quand $\mathcal{J}$ est linéaire, ces deux opérations sont identiques. Connaissant les positions et les vitesses de la surface, on peut en déduire des forces f_1 appliquées sur celle-ci. Pour les prendre en compte, il est nécessaire de les "remonter" sur les degrés de liberté indépendants, par l'opération $f_0 += J^T f_1$. Cette opération est implémentée par la méthode `applyJT`.

Quand les objets entrent en contact, les points de contact sur chaque objet ne coïncident pas forcément avec les degrés de liberté du modèle géométrique de contact, par exemple à l'intérieur d'un triangle ou sur une arête. Un niveau supplémentaire est donc inséré sous le modèle de contact pour contenir ces points, comme illustré sur la figure 4.5. Les calculs correspondants s'insèrent automatiquement dans les propagations hiérarchiques de mouvements et de forces.

Il est intéressant de noter que la même matrice J est utilisée dans les opérations de propagation des vitesses et de remontée des forces. En effet, la puissance $f^T v$ des actions mécaniques est indépendante de la base dans laquelle on exprime les vitesses et les forces. On a donc $f_O^T v_0 = f_1^T v_1$. En y injectant la relation cinématique $v_1 = J v_0$, on obtient $f_O^T v_0 = f_1^T J v_0$. Cette relation étant vraie pour toutes les vitesses v_0 possibles, on peut simplifier par ce terme et obtenir $f_0 = J^T f_1$. Cette force "remontée" s'ajoute aux autres forces appliquées aux degrés de liberté indépendants.

La hiérarchie cinématique présentée sépare nettement les modèles en couches indépendantes reliées par des mappings. Elle offre une grande souplesse de modélisation, en permettant par exemple d'accrocher différents modèles mécaniques à une même modèle de collision, comme illustré sur la figure 4.6.

La structuration d'objets en couches reliées par des mappings permet une grande souplesse de modélisation, comme illustré sur la figure 4.7. Les composants de l'objet sont représentés sur la figure 4.8. Les centres des spères de collisions et les sommets des triangles de visualisation sont attachés au modèle mécanique par des `BarycentricMapping`. Ces mappings positionnent les points fils par combinaisons linéaires des points pères. Les coefficients sont les coordonnés barycentriques, calculés à l'initialisation.

Notons que nos modèles hiérarchiques peuvent avoir un nombre quelconque de niveaux. L'algorithme d'animation ne peut donc pas accéder aux composants, par exemple pour calculer la somme des forces, par un parcours de liste trivial comme dans l'exemple présenté en section 1.1. De plus, les forces doivent être transformées et accumulées de bas en haut comme expliqué en section 1.2. Nous avons donc recours à un mécanisme de *visiteur*. Les algorithmes d'animation, au lieu d'accéder directement aux vecteurs d'état et aux composants, envoient un visiteur traverser le graphe et appliquer les opérations requises à chaque noeud traversé. Dans notre graphe, les noeuds sont les groupes organisés hiérarchiquement (Liver, Visu et Surf dans l'exemple de la figure 4.8). Chaque visiteur implémente une méhode virtuelle `enter`, qu'il applique quand il entre dans un noeud, et une méthode virtuelle `leave`, qu'il applique en sortant. On peut ainsi implémenter des calculs récursifs en surchargeant simplement ces méthodes, sans se préoccuper de la structure d'arbre ni de son implémentation. Par exemple, pour propager les mises à jour du modèle mécanique vers les modèles auxiliaires, on utilise un visiteur qui, dans sa méthode `enter`, applique le mapping (`apply`, `applyJ`) puis les contraintes (`projectPosition`, `projectVelocity`), et ne

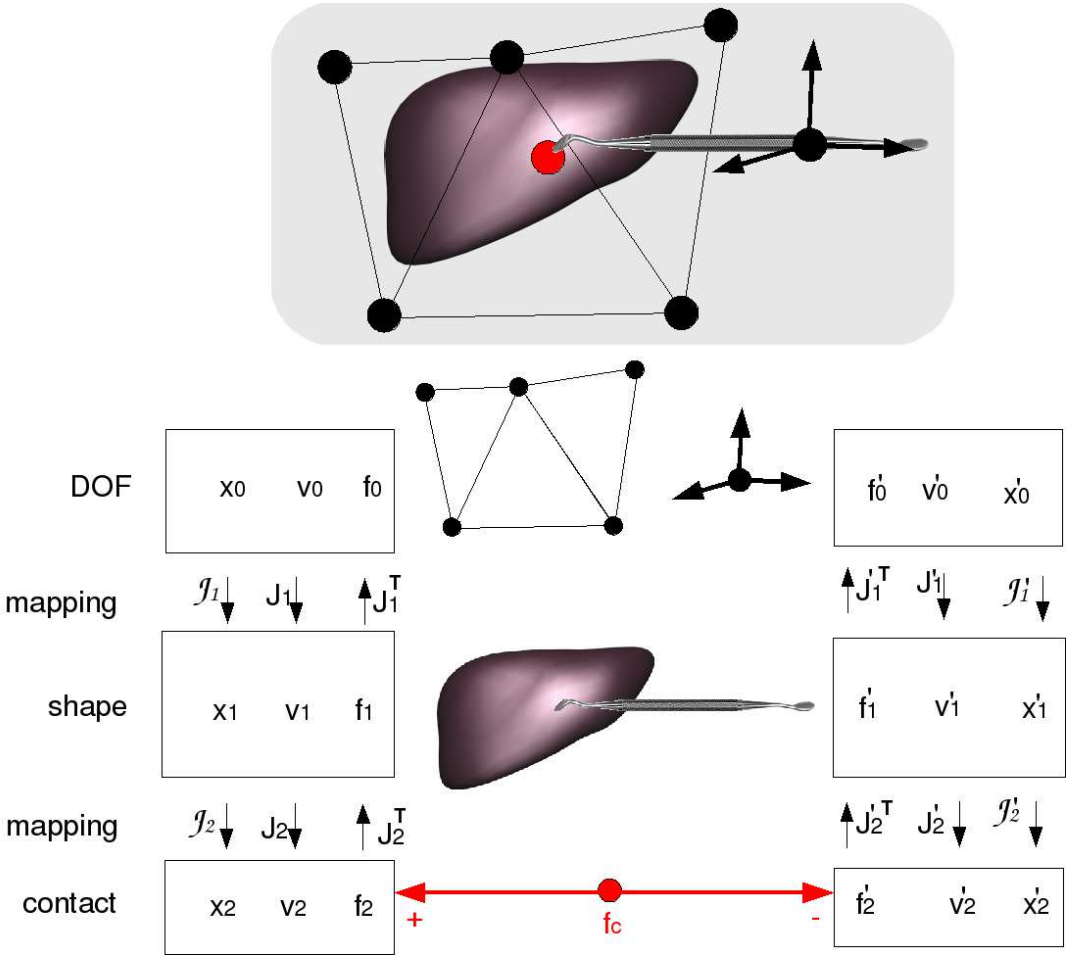


FIG. 4.5 – Modèle hiérarchiques en contact.

fait rien dans sa méthode `leave`. Pour accumuler les forces, le visiteur applique les forces (`addForce`) dans enter, et les mappings (`applyJT`) dans leave pour accumuler les forces aux parents. Ainsi, à la fin du parcours, les degrés de liberté indépendants, situés à la racine, contiennent les forces totales, y compris celles qui leur sont indirectement appliquées sur des modèles auxiliaires.

Une telle architecture logicielle est bien connue depuis longtemps dans la communauté graphique sous le nom de *graphes de scènes* [sce]. Elle permet de combiner de multiples objets d'implémentations variées, et d'appliquer des propriétés communes à des groupes d'objets en les plaçant dans un ancêtre commun. Ils servent généralement à l'affichage de scènes complexes. Dans SOFA , nous avons étendu cette approche à la modélisation mécanique en définissant les composants adéquats, et surtout en développant des algorithmes d'animation adaptés aux graphes de scène. Les algorithmes sont représentés par des composants capables d'animer les sous-graphes dont ils sont à la racine. Ils n'ont aucune connexion directe avec les modèles qu'ils animent, ce qui leur permet d'animer aussi bien des objets simples que des scènes complexes, sans modification. La figure 4.9 présente une scène composée de deux objets animés par le même algorithme (`CGImplicitSolver`). Les instructions de l'algorithme sont implémentées par des visiteurs. Durant les traversées par les visiteurs, chaque opération est réalisée localement sur les vecteurs contenus dans le `MechanicalObject` du noeud. Les vecteurs d'état sont représentés par des valeurs symboliques contenues dans les visiteurs. Par exemple, le symbole f manipulé par l'algorithme de simulation représente l'union des vecteurs f, un par `MechanicalObject`. Le même composant d'animation peut ainsi est appliqué à n'importe quelle scène.

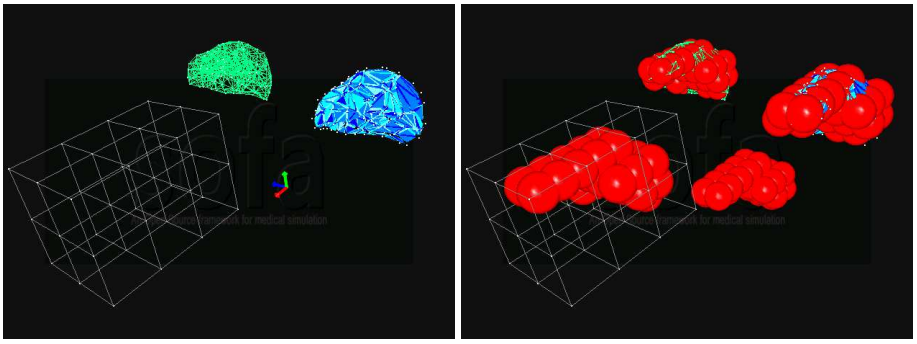


FIG. 4.6 – Différents modèles mécaniques et un même modèle de contact.

Gauche : quatre modèles mécaniques (de gauche à droite : grille déformable, ressorts, solide rigide, éléments finis tétraèdriques). Droite : modèle de contact composé de spères, attaché à ces modèles mécaniques par des mappings, respectivement, barycentrique (grille), barycentrique (tétraèdres), rigide, barycentrique (tétraèdres).

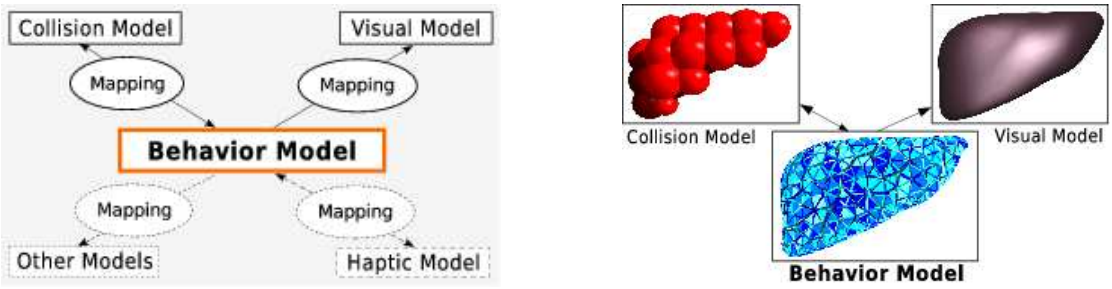


FIG. 4.7 – Modèle hiérarchique.

Gauche : conceptuellement, chaque objet est composé d'un modèle mécanique auquel se rattachent différents aspects de l'objet. La synchronisation s'effectue par des mappings. Droite : exemple d'un foie dont les modèles mécanique, de contact et visuel sont respectivement composés de gros tétraèdres, de sphères et de petits triangles.

1.3 Couplages

Les objets peuvent interagir par des forces, réalisant ainsi des couplages. La force d'interaction ne fait partie ni d'un objet ni de l'autre, elle doit donc être placée dans un ancêtre commun dans la structure hiérarchique. Il y a deux manières possibles d'organiser le couplage. Soit chaque objet a son propre composant d'animation, soit un composant commun anime les deux en même temps. Dans le premier cas, dont

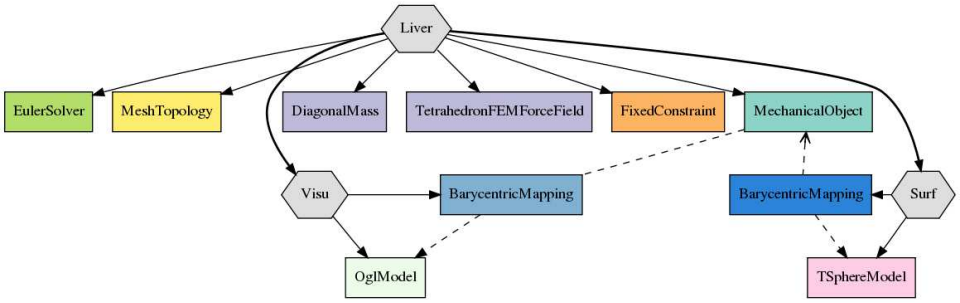


FIG. 4.8 – Structure du foie schématisé sur la figure 4.7.

Le modèle mécnique a deux fils, pour la collision et le rendu. Les pointillés illustrent la transmission des mouvements.

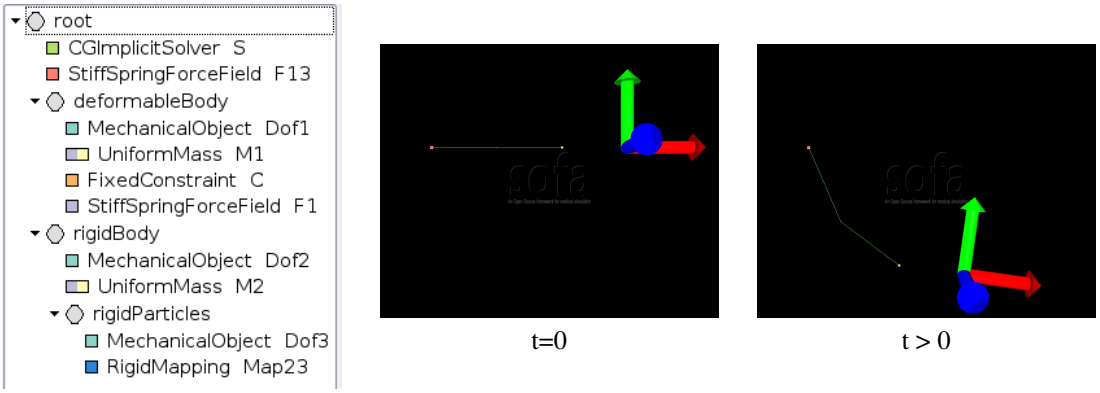


FIG. 4.9 – Scène composée de deux objets.

Un système masses-ressorts, et un solide rigide. La force d'interaction est placée dans l'ancêtre commun des objets. Le solveur implicite commun réalise un couplage fort. À gauche, le graphe de scène tel qu'il apparaît dans l'interface de SOFA .

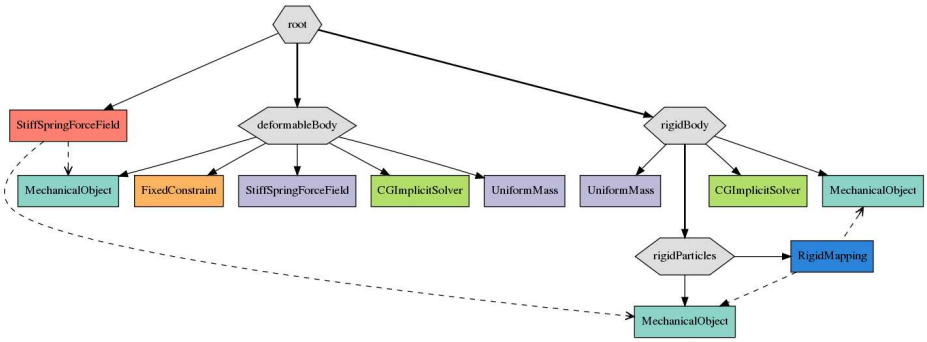


FIG. 4.10 – Couplage faible.

Chaque objet possède son propre algorithme d'animation (case verte) et la force d'interaction (StiffSpringForceField en haut à gauche) est considérée constante au cours de chaque pas de temps. Graphe de scène exporté et affiché par GraphViz.

le graphe est illustré sur la figure 4.10, la force d'interaction est obligatoirement considérée par chaque objet comme une force externe constante pendant la durée du pas de temps. La variation de cette force ne peut pas être anticipée car l'autre objet est inconnu du premier. Il est donc impossible pour un intégrateur implicite de prendre en compte la raideur de la force d'interaction. Nous appelons cette configuration un *couplage faible*. Celle-ci peut donc générer des instabilités, nécessitant de réduire le pas de temps.

Au contraire, un même algorithme d'animation peut prendre en charge le système formé des deux corps et de leur force d'interaction, comme illustré sur la figure 4.11. Dans ce cas, la raideur de cette dernière peut être prise en charge par le solveur, ce qui permet une plus grande stabilité et donc de plus longs pas de temps. Nous appelons cette configuration un *couplage fort*. Notons que si la force d'interaction est peu raide, elle n'introduit pas d'instabilités, et il est alors plus intéressant numériquement d'appliquer un couplage faible. En effet l'intégration implicite se traduira par la résolution de deux petits systèmes d'équations indépendants, plus efficaces qu'un gros système commun.

Nous avons validé les couplages de systèmes à l'aide de l'expérience présentée sur la figure 4.12, dans laquelle une barre déformable est décomposée en sous-systèmes reliés par des forces d'interactions. Le résultat est le même qu'en modélisant la barre sous forme d'un unique objet, et le temps de calcul est quasiment identique.

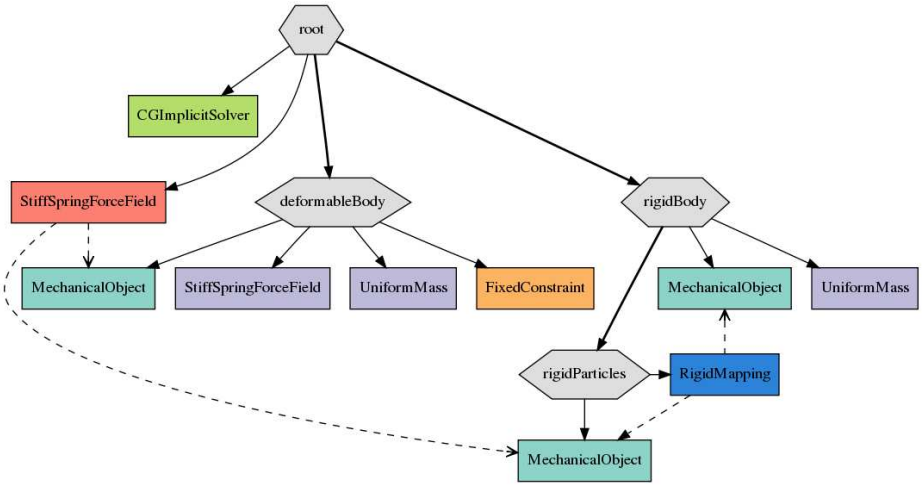


FIG. 4.11 – Couplage fort.

L'algorithme d'animation placé au sommet de la hiérarchie prend en charge le système formé de deux corps en interaction, et la raideur de la force d'interaction est prise en compte dans l'intégration implicite.

Le même solveur peut être appliqué à des scènes composées d'objets divers, comme illustré sur la figure 4.13, dans laquelle certains objets déformables de la figure 4.12 ont été remplacés par des objets rigides.

1.4 Solveurs génériques

Les solveurs sont les implémentations des algorithmes d'animation. Leur capacité de gérer de manière transparente un nombre arbitraire d'objets de types divers, dont un exemple est donné en figure 4.13, est obtenue par l'utilisation de constantes symboliques représentant les vecteurs d'état, récursivement véhiculées par les visiteurs à travers les noeuds de la hiérarchie. Les vecteurs d'état sont alloués et stockés par des MechanicalObject, au maximum un par noeud, qui ont aussi pour tâche d'implémenter l'algèbre linéaire de ces vecteurs. Les opérations plus spécifiques, comme l'accumulation de forces, sont réalisées par les autres composants du groupe.

La plupart du temps, aucune information n'est renvoyée au solveur. Seule exception, le produit scalaire est accumulé dans le visiteur et remonté au solveur sous formes d'un nombre réel, quel que soit le type d'objet simulé. Ainsi, chaque opération déclenchée par le solveur se traduit par un ensemble d'opérations locales effectuées indépendamment par les objets. La communication entre solveur et objets simulés est réduite à l'envoi de constantes symboliques et la récupération éventuelle d'un réel.

Dans une scène de SOFA , les opérations symboliques suivantes sont automatiquement implémentées, quels que soit le nombre, le type et la structure des objets simulés :

- algèbre linéaire des vecteurs d'état
- $f(x, v)$
- $M^{-1}x$
- Mx
- $df(dx)$

Les trois premières permettent d'implémenter les méthodes d'intégration explicite du temps. Les deux premières et les deux dernières permettent d'implémenter des intégrateurs explicites, en résolvant l'équation par gradient conjugué. En effet, cet algorithme n'accède pas directement à la matrice du système d'équations, mais se contente de la multiplier par un vecteur d'état. Les objets structurés en hiérarchies de couches sont automatiquement gérés au moyen des opérations de mapping :

- $\mathcal{J}(x)$
- Jx

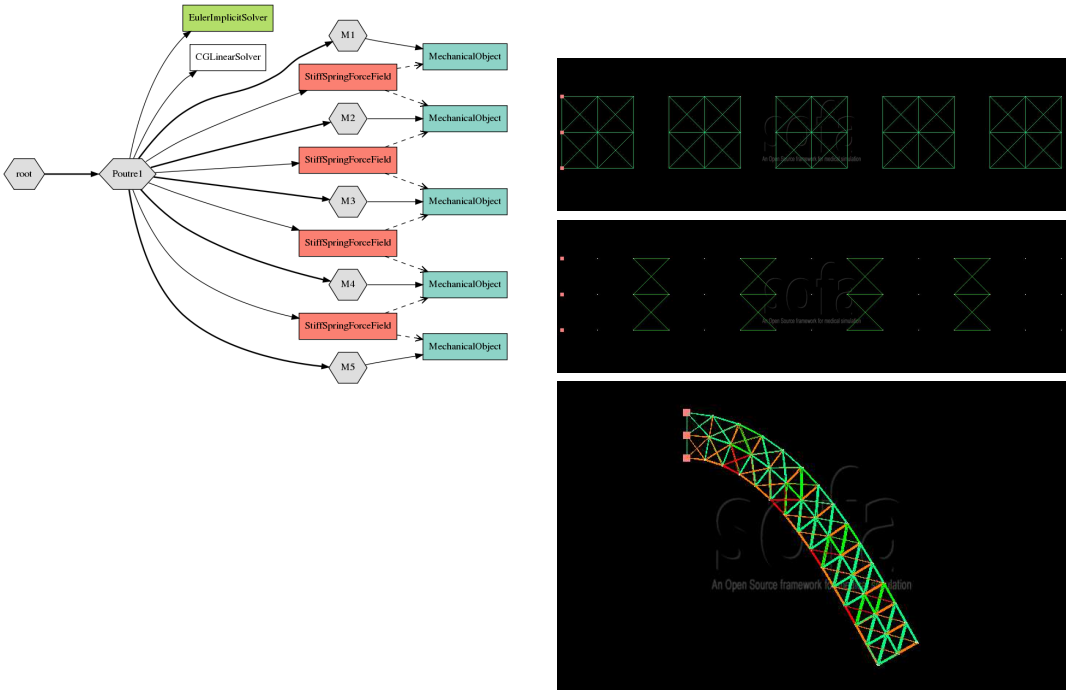


FIG. 4.12 – Validation des couplages avec une barre composée de 5 objets reliés par 4 interactions.

Gauche : graphe de scène (certains composants omis). Droite : haut : les objets. milieu : les interations. bas : simulation.

– $J^T x$

L'opération $M^{-1}x$ utilisée dans les intégrateurs explicites ne fonctionne que si la masse est définie pour les degrés de liberté indépendants. Ainsi, dans le cas de la grille déformable illustré sur la figure 4.6, la masse doit être exprimée aux sommets de la grille. Pour un solveur implicite, la masse peut être exprimée à n'importe quel niveau de la hiérarchie.

Les matrices M et df/dx du système d'objets en interaction ne sont pas assemblées. Les produits de ces matrices par des vecteurs sont calculés indépendamment par les composants et accumulés dans le résultat. Ne pas calculer la matrice assemblée a des avantages. Les objets peuvent être de type différents (particules, solides rigides, etc.) et l'assemblage nécessiterait de coûteuses conversions vers un type commun (`float` ou `double`). De plus, cette matrice peut être arbitrairement grande et ne pas tenir sur une seul ordinateur. Ne pas effectuer l'assemblage de la matrice a toutefois des inconvénients. Les méthodes de résolution utilisant une factorisation de la matrice ne peuvent pas être utilisées, de même les méthodes de Jacobi et Gauss-Seidel qui traitent les lignes de la matrice.

Un autre inconvénient est que nous ne disposons pas de préconditionneur pour le gradient conjugué, ce qui peut nuire à son efficacité. Il serait toutefois possible d'assembler la diagonale de la matrice pour un coût raisonnable, ce qui permettrait l'implémentation d'un préconditionneur diagonal.

Une dernière limitation concerne les méthodes à base de contraintes sur les déplacements. On les rencontre dans les simulations incluant des frottements de Coulomb, indispensables pour le réalisme de nombreuses applications. Ce frottement se traduit par des équations non linéaires qui, comme discuté en sectin 2.3, ne peuvent pas être efficacement traitées par des solveurs à base de gradient conjugué. Deux sortes de solveurs sont couramment appliqués à ce genre d'équations. Les premiers utilisent des solveurs numériques de programmation linéaire ou quadratique qui utilisent explicitement la matrice des équations. Ils résolvent globalement le système d'équations et sont généralement très lents et peu adaptés aux simulations interactives de scènes complexes [Dur04]. Les autres effectuent répétitivement des résolutions locales d'équations, à la manière de l'algorithme de Gauss-Seidel. Leur programmation est simple et le

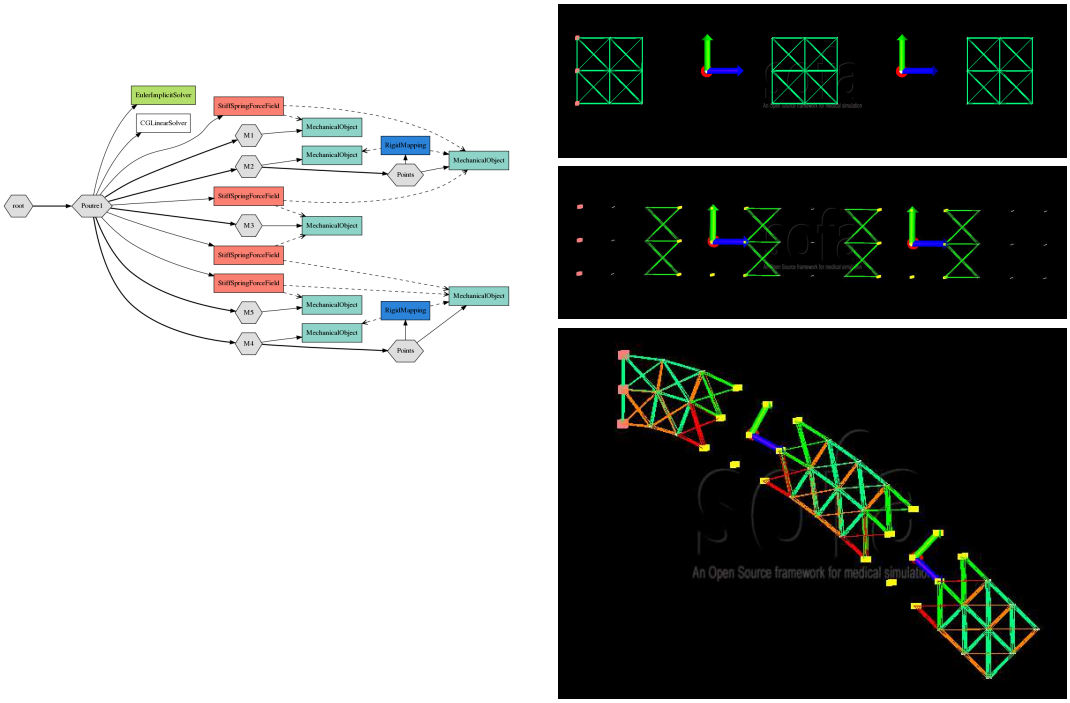


FIG. 4.13 – Expérience similaire à celle de la figure 4.12, avec une barre composée d'objets d'objets hétérogènes (particules et rigides).

temps de calcul est facilement adaptable aux besoins d'interactivité, au détriment de la précision. On les rencontre beaucoup dans les jeux video [MHHR06]. En général, ils nécessitent des calculs de compliance aux points de contact, qui font intervenir des inverses de matrices qui ne sont actuellement pas disponibles dans SOFA .

1.5 Conclusion

L'architecture logicielle de SOFA est à la fois modulaire et efficace. Les objets sont décomposés hiérarchiquement en couches implémentant des modèles spécialisés pour différents traitements, et reliées par des mappings. Chaque modèle peut être conçu indépendamment des autres. Les couches sont des groupes de composants dont chacun représente un aspect du modèle, et est interchangeable avec un autre composant de même type. De nombreuses opérations utilisées par les algorithmes d'animation sont automatiquement implémentées, quels que soit le nombre, le type et la structure des objets simulés. Cette architecture a permis de modéliser et simuler des objets complexes en interaction à l'aide de composants logiciels génériques et réutilisables. À l'heure actuelle, les systèmes élastiques fonctionnent bien, le principal progrès à réaliser dans ce domaine étant l'application de préconditionneurs dans le gradient conjugué. Le prochain pas important à réaliser est l'application de contraintes de déplacements, nécessaires pour simuler le frottement de Coulomb de manière satisfaisante.

2 Parallélisme

La montée en fréquence des processeurs s'est arrêtée ces dernières années aux alentours de 3 GHz pour des raisons physiques de surchauffe. Désormais, et jusqu'à ce qu'une nouvelle technologie remplace les actuels circuits en silicium, la course à la puissance sera basée sur le parallélisme des calculs. Il devient courant qu'un ordinateur possède un ou plusieurs processeurs à 4 coeurs, ainsi qu'un coprocesseur graphique

(GPU) contenant des centaines d'unités de calcul parallèle. De plus, les grappes et grilles d'ordinateurs continuent de se développer. La loi de Moore continue de s'appliquer, mais les progrès sont désormais obtenus par parallélisation. Par conséquent, pour continuer à bénéficier d'un doublement des performances tous les deux ans, il est désormais indispensable de développer des applications parallèles.

Je m'intéresse à la parallélisation de simulation physique depuis que j'ai eu le plaisir de co-encadrer avec Jean-Marc Vincent la thèse de Florence Zara. Elle a porté sur la simulation physique de textiles sur grappes de processeurs, avec pour application visée la simulation de vêtements. Elle repose sur la décomposition de domaine, chaque partie d'un tissu étant traitée sur un processeur différent. Ces travaux sont présentés en section 2.2. L'architecture logicielle de SOFA , conçue avec Jérémie Allard, offre de larges opportunités de parallélisation à plusieurs niveaux. Jérémie a commencé à les exploiter en portant des domaines décomposés sur GPU, avec des lois de comportement à bases de ressorts ou d'éléments finis. Il a également exploré un autre étage de parallélisme, la possibilité de traiter différents objets sur différents CPU. Je poursuis actuellement dans cette direction en co-encadrant avec Bruno Raffin la thèse d'Everton Hermann. Dans ce travail, le but n'est pas de paralléliser les parties d'un même objet, ce que le GPU fait maintenant très bien, mais de paralléliser tous les objets d'une scène avec leurs interactions. Ces travaux sont présentés en section 2.3. Cette section reprend de larges extraits de la thèse de Florence Zara [Zar03].

2.1 Rappels

Le parallélisme de données est appelé ainsi car il exploite la concurrence qui résulte de l'exécution de la même opération sur des éléments différents d'une structure de données. Dans ce mode d'expression du parallélisme, le travail des processeurs est guidé par la distribution des données. Les communications sont définies comme des phases de redistribution des données sur les processeurs. Le programme parallèle est une succession de phases de calculs et de phases de communications pour la redistribution des données. Cette source de parallélisme est souvent utilisée lors d'opérations sur des vecteurs, fréquentes dans les problèmes d'algèbre linéaire dense. Elle apparaît dans nos simulations quand les mêmes calculs sont effectués sur des données différentes, comme par exemple dans les combinaisons linéaires des vecteurs d'état. Des librairies spécialisées existent pour répartir les calculs sur différents processeurs, ou sur différentes unités de calcul d'un GPU. Ce parallélisme très efficace ne traite malheureusement pas trivialement les opérations plus complexes nécessitant partages de données et contrôle logique.

Le parallélisme de contrôle consiste à décrire un algorithme parallèle sous la forme d'un graphe orienté sans cycle. Les noeuds du graphe sont des suites d'opérations élémentaires exécutées séquentiellement, ce sont les tâches du graphe. Les arcs du graphe indiquent des contraintes de précédence entre les taches. C'est-à-dire l'utilisation par une tâche d'une donnée calculée par une ou plusieurs tâche(s) précédente(s). Ce graphe est appelé graphe de précédence ou graphe de tâches, il est déduit d'un ordre partiel sur les tâches. Les tâches non ordonnées par cet ordre partiel peuvent être exécutées en parallèle. On peut voir le parallélisme de données comme un cas particulier de parallélisme de contrôle, mais celui-ci peut donner lieu à des graphes plus complexes comme illustré sur la figure 4.14. C'est ce paradigme que nous avons utilisé.

2.2 Simulation de textiles sur grappes de processeurs

Schématiquement, la boucle de simulation de textiles comporte les phases suivantes :

1. calcul des forces
2. calcul des accélérations
3. intégration
4. réactions aux collisions

Le calcul des forces s'effectue en itérant sur un liste d'interactions, des ressorts dans notre cas. Il implique des données partagées en lecture (positions, vitesses) et en écriture (forces). Le calcul des accélérations peut être très simple, comme dans Euler explicite où les forces sont simplement divisées par les masses, indépendamment sur chaque particule, ce qui correspond à un parallélisme de données. Il peut aussi être beaucoup plus complexe, comme dans Euler implicite où un système d'équations doit être résolu au moyen

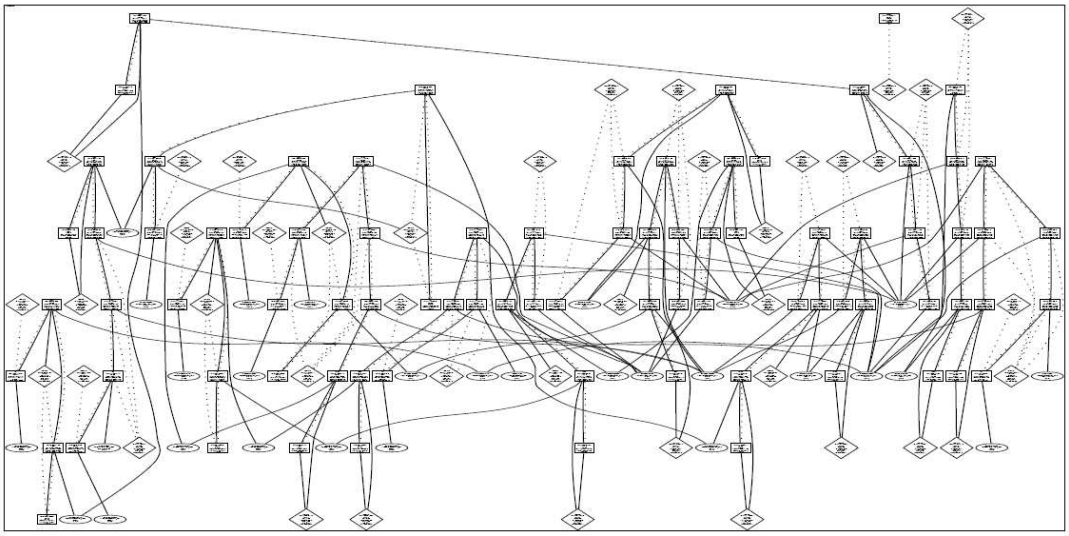


FIG. 4.14 – Un graphe de parallélisme de contrôle.

Les rectangles, les losanges et les cercles représentent respectivement les accès aux don- nées partagées, les données partagées et les tâches de calculs. Les lignes représentent les dépendances.

de multiples produits matrice-vecteur, dans un processus itératif à terminaison non prévisible. Nous avons particulièrement travaillé sur ce dernier point. L'intégration se limite à des combinaisons linéaires de vecteurs. La détection et la réaction aux collisions est un processus complexe que nous avons gardé pour des travaux ultérieurs. Une gande partie du travail de Florence a donc été de créer et résoudre à chaque instant simulé un sytème d'équations linéaires à l'aide de l'algorithme du gradient conjugué, et en s'appuyant sur la librairie ATHAPASCAN [ath] développée dans l'équipe de Jean-Marc Vincent.

Il y a cinq étapes illustrées par la figure 4.15, pour passer de la spécification du problème initial à l'élaboration de l'algorithme parallèle :

1. Partitionnement. Le problème initial est décomposé en tâches de calculs plus petites. Cette décomposition est effectuée sans se soucier du nombre de processeurs sur lequel le calcul sera ensuite exécuté, mais l'attention est plutôt orientée sur les opportunités apportées par une exécution parallèle.
2. Communication. Les communications nécessaires entre les tâches sont mises en évidence permettant de définir les structures algorithmiques et de communication appropriées.
3. Agglomération. Les tâches et les structures de communication définies précédemment sont évaluées et si nécessaire des tâches sont combinées en tâches plus grosses afin d'améliorer les performances et de réduire les coûts.
4. Ordonnancement. Chaque tâche est assignée à un processeur de façon à optimiser l'utilisation des processeurs et à minimiser les coûts des communications. Le placement peut être spécifié de manière statique ou déterminé lors de l'exécution en utilisant des algorithmes d'équilibrage de charge.
5. Interaction. Les interactions entre la simulation et la visualisation doivent être gérées dans le cadre d'une simulation graphique.

On peut partitionner par particules, mais l'étude des dépendances et des communications nous a suggéré une agglomération par zones connexes, comme illustré sur la figure 4.16. Nous avons mis en évidence le fait que la découpe de la simulation en sous- ensembles de particules doit faire l'objet d'un compromis. Un nombre élevé de petites tâches permet en théorie une meilleure répartition de charge entre les processeurs, mais engendre un coût de communication et de synchronisation important.

Une difficulté imprévue est apparue, dûe au fait que les simulations interactives reposent sur une boucle infinie contenant un pas de simulation et les entrées-sorties. ATHAPASCAN est conçu pour développer le graphe des tâches menant à un résultat, mais pas pour répéter indéfiniment l'exploitation de ce graphe.

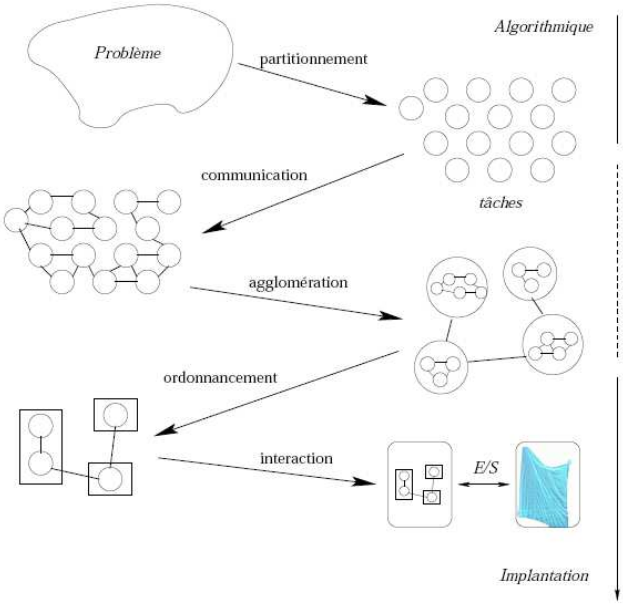


FIG. 4.15 – Elaboration d'un algorithme parallèle.

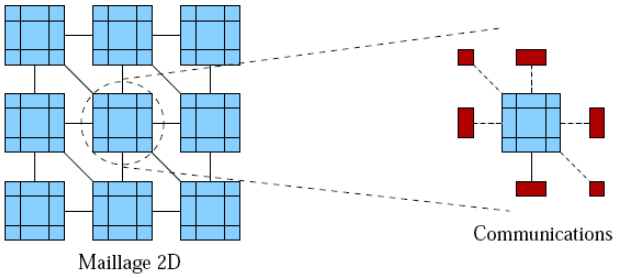


FIG. 4.16 – Agglomération des tâches.

Il a donc fallu, à chaque nouvel instant de la simulation, recréer le graphe. Nos travaux ont porté sur la création et l'exploitation du graphe des tâches. Pour l'intégration explicite, celui-ci est relativement simple à construire. La figure 4.17 en montre un extrait concernant une particule avec une seule voisine.

Pour l'intégration implicite, les multiples itérations du gradient conjugué compliquent les choses. La librairie ATHAPASCAN se prêtant mal aux itérations non prévisibles (while), nous avons dû "dérouler" les itérations. Le placement des tâches sur les processeurs a été expérimenté à l'aide des librairies Scotch et Cyclic, comme illustré sur la figure 4.18.

La figure 4.19 présente un exemple de performances obtenues par notre parallélisation. Le temps de calcul décroit très rapidement jusqu'à l'emploi d'une dizaine de processeurs pour ensuite rester stable, signifiant que le temps atteint ne peut actuellement être diminué, trahissant le surcoût de la machine ATHAPASCAN. Plus généralement, cette étude a permis de mettre en évidence des points faibles de l'environnement ATHAPASCAN et ainsi de le faire évoluer tout au long de ce travail et par la suite. Ces travaux ont fait l'objet de plusieurs publications scientifiques [ZFV02, ZFV04, ZG03].

Par la suite, nous avons étudié le couplage entre une simulation sur grappes et un dispositif parallèle d'affichage sur plusieurs écrans, comme illustré sur la figure 4.20, afin d'obtenir des résultats visuels comme ceux présentés sur la figure 4.21. Le problème de récupération de données dans une grappe d'or-

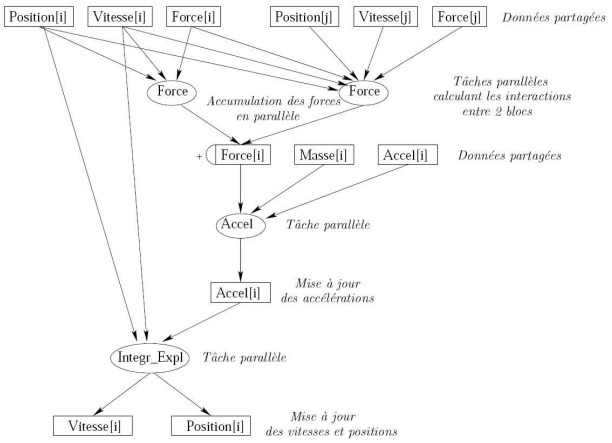


FIG. 4.17 – Extrait du graphe des tâches pour l'intégration explicite.

dinateurs et de leur répartition pertinente sur une autre grappe est loin d'être trivial, et ces travaux ont fait l'objet de publications scientifiques [ARZ04, ZVF03].

Pour conclure, Florence a montré qu'il était possible de déployer automatiquement une simulation physique massive sur une grappe d'ordinateurs à l'aide d'ATHAPASCAN, et de connecter cette simulation à des dispositifs de visualisation et d'interaction. Elle a surmonté avec brio de nombreuses limitations dûes à l'état de développement d'ATHAPASCAN et au fait que cette librairie n'avait pas été pensée dès le départ pour traiter la simulation physique interactive, basée sur une boucle infinie contenant un nombre imprévisible de sous-boucles. Les résultats montrent qu'il est difficile d'exploiter efficacement plus d'une dizaine d'ordinateurs en réseau pour simuler un objet déformable, à cause du coût des communications et synchronisations. Notons que depuis ces travaux, les GPU ont fait d'énormes progrès et de nos jours, la simulation d'un objet déformable semble se paralléliser plus efficacement sur un GPU.

2.3 Parallélisation de scènes complexes

La thèse d'Everton Hermann s'inscrit dans la suite des travaux de Florence Zara, en tenant compte des évolutions technologiques et scientifiques. Le matériel a évolué récemment vers des ordinateurs multicoeurs très performants permettant d'implanter sur une seule machine ce qui nécessitait auparavant une grappe de quelques dizaines de PC. Le problème de communication entre processeurs y est moins crucial que sur les grappes car la mémoire est partagée, comme illustré sur la figure 4.22. De plus, les processeurs centraux sont maintenant épaulés par de puissants GPU reprogrammables, auxquels on peut confier la simulation d'objets déformables. Le nouveau défi est d'arriver à exploiter toute la puissance d'un PC de haut de gamme, qui comportera très bientôt plusieurs dizaines de processeurs centraux et plusieurs milliers d'unités de calcul sur GPU.

La nature des scènes à simuler a aussi évolué. Nous ciblons désormais des simulations médicales impliquant de nombreux objets rigides et déformables, ainsi que la détection et le traitement des contacts entre eux et avec des outils manipulés par un utilisateur, comme illustré sur la figure 1.1 page 3. L'application concrète consiste donc à déployer automatiquement SOFA sur les ressources matérielles disponibles.

Considérons le processus de parallélisation présenté en figure 4.15. Désormais, le partitionnement correspond à l'exécution par un composant d'une procédure élémentaire comme celles présentées en section 1.1 de ce chapitre. Nous n'étudions pas la décomposition de domaines, qui peut être effectuée préalablement comme illustré sur la figure 4.12. D'autre part, les tâches peuvent elles-mêmes être parallélisées, par exemple sur GPU et nos partenaires dans SOFA continuent d'explorer cette voie. La souplesse de SOFA permet de travailler indépendamment sur les deux niveaux de parallélisme, CPU et GPU, en combinant les accélérations à tous les niveaux. De plus nous utilisons la librairie KAAPI [kaa], évolution d'ATHAPASCAN, qui nous permettra de déployer les applications sur grappes de PC, afin d'expoiter en-

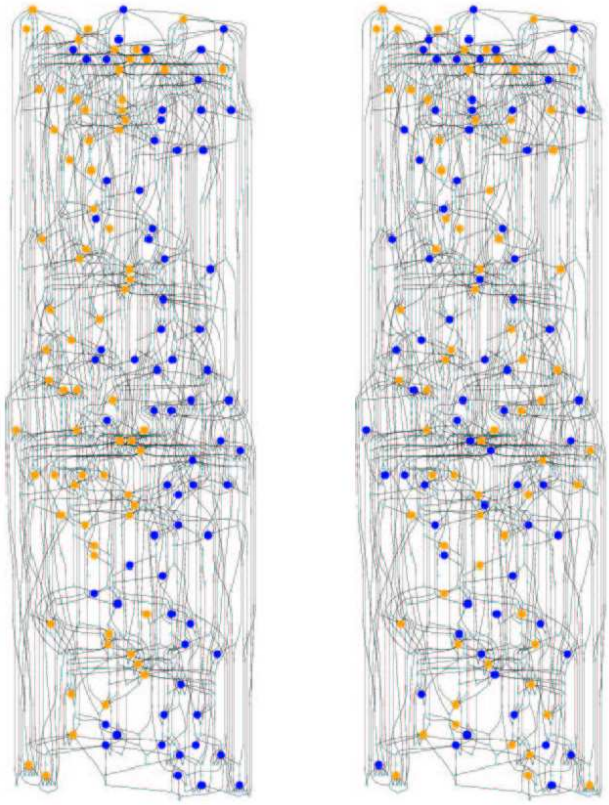


FIG. 4.18 – Extrait du graphe des tâches pour l'intégration implicite.

Placement Scotch (à gauche) et Cyclic (à droite) du raphe de tâches sur 2 itérations avec 1 itération du gradient conjugué pour un partitionnement en 4 sous-ensembles de particules sur 2 processeurs.

core un niveau supplémentaire de parallélisme. La communication entre les tâches est analysée par KAAPI au moyen des variables `shared` déjà utilisées dans ATHAPASCAN. Les tâches sont agglomérées par séquences. Le placement s'effectue autant que possible par objet, correspondant à une branche de la hiérarchie de modèles, afin de respecter la localité des données. Les interactions et couplages avec d'autres processus sont pris en charge par FlowVR [flo], en encapsulant le simulateur dans un module de cette librairie. Ceci nous a permis le couplage avec la plate-forme GrImage [gri], plate-forme parallèle de vision et visualisation en temps réel. Nous avons ainsi pu réaliser la simulation en temps réel d'un objet virtuel animé par SOFA avec un objet réel reconstruit par caméras. La figure 4.1 en montre une présentation dans l'exposition *Emerging Technologies* à SIGGRAPH 2007.

SOFA étant destiné à un large public non spécialiste du parallélisme, il est important que la parallélisation puisse être transparente aux programmeurs des composants. C'est une des raisons pour lesquelles nous avons choisi une granularité relativement épaisse, afin que chaque composant puisse être programmé séquentiellement. D'autre part, l'utilisation de vecteurs d'états abstraits dans les solveurs permet l'analyse et le déploiement du parallélisme indépendamment de l'algorithme d'animation.

Ce travail nécessite une excellente compréhension des mécanismes de SOFA et de KAAPI. Everton s'est initié à SOFA en y développant et appliquant le nouvel algorithme de détection et modélisation des contacts présenté en section 3.2. Il a ensuite pu développer une version parallèle de SOFA dans laquelle il parvient à générer les graphes de tâches pour toutes scènes et tous solveurs, au prix de légères modifications (quasiment syntaxiques) des solveurs. Il travaille maintenant à lever les limitiations de KAAPI pour la simulation interactive. Le graphe des tâches est désormais réutilisable à chaque itération de la boucle infinie, ce qui fait gagner un temps important. Le problème des boucles conditionnelles, impossibles à dérouler en

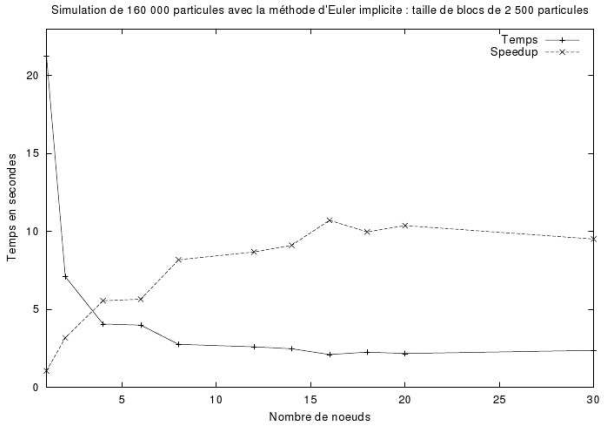


FIG. 4.19 – Exemple de performances obtenues.

Performances pour une itération d'un problème comportant 490000 particules avec une taille de blocs de 2 500 particules en employant la méthode d'intégration d'Euler implicite sur le I-Cluster.

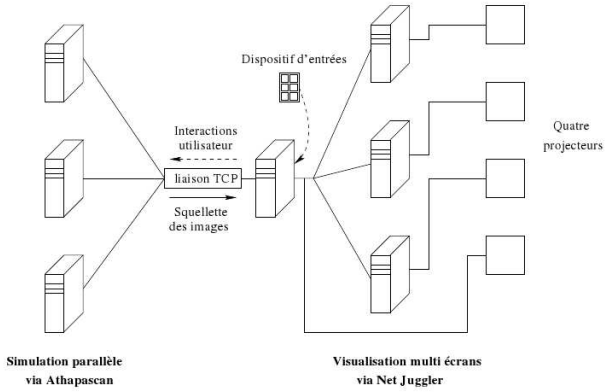


FIG. 4.20 – Couplage d'applications parallèles.

Simulation parallèle (ATHAPASCAN) avec une visualisation multi écrans (Net Juggler). Les deux parties de l'application s'exécutent sur des grappes de machines différentes.

pré-analyse, est en passe d'être résolu. D'autre part, un prototype de plate-forme basée sur FlowVR pour visualiser et interagir avec les scènes simulées est en cours de développement.

La partie détection et traitement des contacts reste à améliorer. Une des difficultés est que des tâches se créent ou disparaissent au gré des collisions et séparations. Une partie du graphe des tâches devrait donc être dynamique, en complément du graphe statique calculé en pré-analyse. Pour l'instant, nous considérons des contacts pré-établis, actifs ou passifs, entre chaque paire d'objets en interaction potentielle. Cette approche ne passera pas à l'échelle de scènes contenant des grands nombres d'objets. De plus, la distribution des tâches d'interaction sur les différents processeurs devra être gérée dynamiquement.

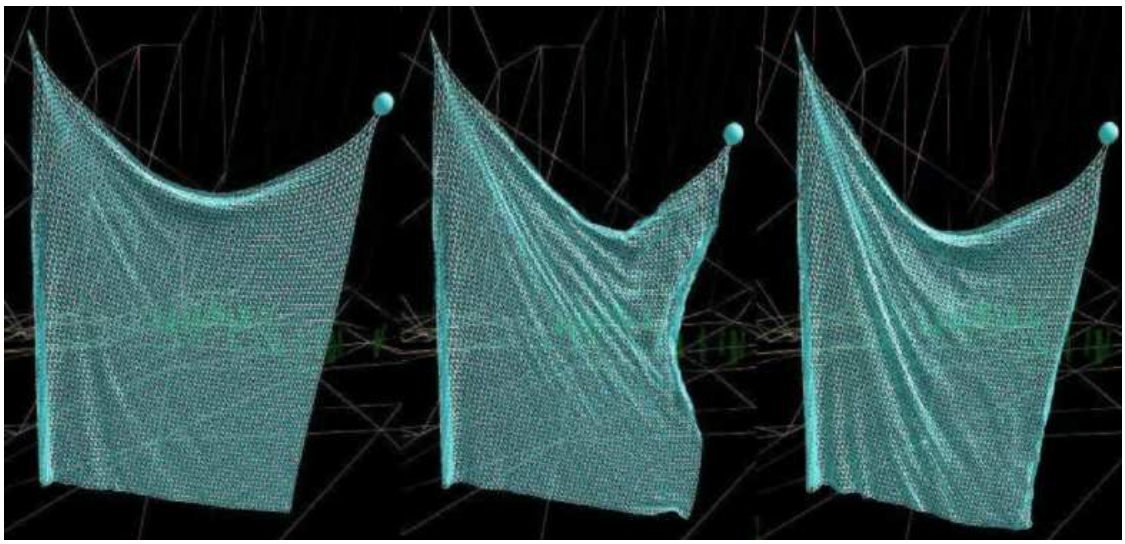


FIG. 4.21 – Visualisation sur mur d'écrans et interaction avec l'utilisateur.

Le coin en haut à gauche est fixé, tandis que le coin en haut à droite est dirigé par l'utilisateur via la souris.

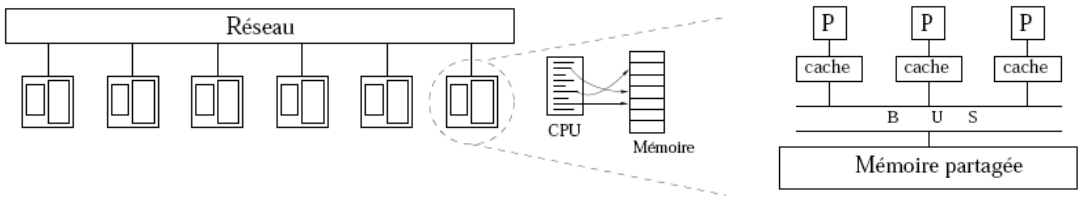


FIG. 4.22 – Comparaison d'architectures.

Gauche : dans une grappe de PC, les données circulent entre processeurs sur un réseau relativement lent. Droite : sur une machine multicoeurs, la mémoire est partagée entre les processeurs et accessible par un bus relativement rapide.

chapitre 5

Conclusion

Ô que c'est dur, ô que c'est long, ô que c'est bon !
d'écrire une habilitation !

La recherche en animation par modèles physiques a encore un long chemin à parcourir avant d'offrir la simplicité et la performance requises pour une utilisation quotidienne. Les recherches présentées dans ce mémoire sont un modeste pas dans cette direction.

Les grilles régulières semblent un outil intéressant pour la simulation d'objets visco-élastiques. Elles permettent d'automatiser la création du modèle d'après une description surfacique ou volumique de l'objet, sans recours à des méthodes complexes de maillage tétraédriques, et sans risque d'obtenir des éléments dégénérés. L'application à la simulation chirurgicale personnalisée semble une application-phare, mais les graphistes devraient aussi pouvoir y trouver leur bonheur. De plus, l'indépendance entre modèles mécanique et de collision permet d'envisager l'extension aux découpes interactives. Les travaux de post-doctorat de Lenka Jerabkova vont dans ce sens. Les approches hiérarchiques semblent très prometteuses pour mieux retranscrire la raideur des matériaux interactivement, et ainsi pouvoir simuler plausiblement des objets en temps réel. Nous sommes persuadés que l'utilisation d'un multigrid adaptatif est une bonne direction à prendre pour animer les hiérarchies de voxelisations. Toutefois, les développements récents en méthodes "meshless" laissent entrevoir des alternatives intéressantes aux éléments finis.

Les progrès dans le traitement temps-réel des collisions nécessiteront d'améliorer la robustesse des méthodes de détection et de réponse. Notre méthode de modélisation des contacts à base d'images semble une piste intéressante. Elle permet de calculer le volume d'interpénétration de deux objets, et surtout d'exprimer sa dérivée par rapport aux sommets des objets en contact, ce qui permet de facilement appliquer des forces de répulsion. Elle est géométriquement très robuste puisqu'elle calcule des forces de répulsions correctement orientées même en cas d'interpénétration profonde. Toutefois, nous avons constaté que la robustesse géométrique doit aller de pair avec la robustesse mécanique, afin déviter que les forces ne provoquent des instabilités. Le traitement robuste des interpénétrations, ainsi que des contraintes liées au frottement sec, reste une question à creuser.

Les architectures logicielles devront encore se perfectionner. S'il l'on peut aujourd'hui assembler des scènes composées de multiple objets de toute natures physiques dans SOFA , et les faire interagir suivant des lois élastiques, il n'est pas encore possible de leur appliquer des algorithmes exprimant des contraintes de position relative, et des lois de contact perfectionnées. Un important travail de parallélisation est en cours, mais la route est encore longue jusqu'à l'exploitation complète de toutes les recettes du parallélisme : décomposition de domaine, implémentation sur GPU, multicore, cluster.

Bibliographie

[ACF+07] Jérémie Allard, Stéphane Cotin, François Faure, Pierre-Jean Bensoussan, François Poyer, Christian Duriez, Hervé Delingette, and Laurent Grisoni. Sofa an open source framework for medical simulation. In *Medicine Meets Virtual Reality (MMVR)*, 2007.

[ARZ04] Jérémie Allard, Bruno Raffin, and Florence Zara. Coupling parallel simulation and multi-display visualization on a PC cluster. In Harald Kosch, László Böszörményi, and Herrmann Hellwagner, editors, *International Conference on Parallel and Distributed Computation, Euro-Par 2003, August, 2003*, number 2790 in Lecture Notes in Computer Science, pages 1287–1290, Klagenfurt, Autriche, June 2004. Springer.

[Asc97] Uri M. Ascher. Stabilization of invariants of discretized differential systems. *Numerical Algorithms*, 14, 1997.

[ath] Athapascan. http ://www-id.imag.fr/Logiciels/ath1/.

[Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. 28(Annual Conference Series) :23–34, July 1994.

[Bar96] David Baraff. Linear-time dynamics using lagrange multipliers. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146, August 1996.

[BC94] J. M. Brown and J. E. Colgate. Physics-based approach to haptic display. In *Proc of ISMCR 94, Topical Workshop on Virtual Reality*, 1994.

[BC00] David Bourguignon and Marie-Paule Cani. Controlling anisotropy in mass-spring systems. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, Springer Computer Science, pages 113–123, 2000.

[Bel76] T. Belytschko. A survey of numerical methods and computer programs for dynamic structural analysis. *Nuclear Engineering and Design*, 37 :23–24, 1976.

[BFA03] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *ACM Transactions on Graphics (Proc of SIGGRAPH)*, volume 21(3), pages 594–603, 2003.

[BGTG04] D. Bielser, P. Glardon, M. Teschner, and M. Gross. A state machine for real-time cutting of tetrahedral meshes. *Graph. Models*, 66(6) :398–417, 2004.

[BHG92] D. Breen, D. House, and P. Getto. A physically-based particle model of woven cloth. *The Visual Computer*, 8(5–6) :264–277, 1992.

[BHM00] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial (second edition)*. SIAM. 2000.

[BJ71] J. J. Batter and F. P. Brooks Jr. Grope-1 : A computer display to the sense of feel. In *Information Processing (Proc of IFIP Congress)*, pages 759–763, 1971.

[BJ05] Jernej Barbič and Doug L. James. Real-time subspace integration for St. Venant-Kirchhoff deformable models. In *ACM Transactions on Graphics (SIGGRAPH)*, volume 24 (3), pages 982–990, 2005.

[Bra77] A. Brandt. *Multi-level adaptive solutions to boundary value problems*, volume 31 of *Mathematics of Computation*. 1977.

[BW98] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH '98*, 1998.

[BWK03] David Baraff, Andrew Witkin, and Michael Kass. Untangling cloth. *ACM Trans. Graph.*, 22(3) :862–870, 2003.

[Cad99] C. Cadoz. sound, music, and image creation with physical model- the cordis and genesis tools from acroe. 1999.

[Can93] Marie-Paule Cani. An implicit formulation for precise contact modeling between flexible solids. In *Computer Graphics Proceedings, Proc of SIGGRAPH*, pages 313–320, 1993.

[CBP05] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Symposium on Computer Animation*, pages 219–228, 2005.

[CDA00] S. Cotin, H. Delingette, and N. Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. *The Visual Computer*, 16(8), 2000.

[CDC+96] S. Cotin, H. Delingette, J.-M. Clement, V. Tassetti, J. Marescaux, and N. Ayache. Volumetric deformable models for simulation of laparoscopic surgery. In *Computer Assisted Radiology*, 1996.

[CEO+93] Steven Cover, Norberto Ezquerra, James O'Brien, Richard Rowe, Thomas Gadacz, and Ellen Palm. Interactively deformable models for surgery simulation. *IEEE Comput. Graph. Appl.*, 13(6) :68–75, 1993.

[CF97] Stephen Chenney and David A. Forsyth. View-dependent culling of dynamic systems in virtual environments. In *Symposium on Interactive 3D Graphics*, pages 55–58, 1997.

[CGC+02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. A multiresolution framework for dynamic deformations. In *SCA '02*, 2002.

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

[Cot97] S. Cotin. *Modèles anatomiques déformables en temps réel : Application à la simulation de chirurgie avec retour d'effort*. PhD thesis, Université de Nice Sophia-Antipolis, 1997.

[DC95] Mathieu Desbrun and Marie-Paule Cani. Animating soft substances with implicit surfaces. In *Computer Graphics Proceedings, Proc of SIGGRAPH*, pages 287–290, 1995.

[DCA99] H. Delingette, S. Cotin, and N. Ayache. Efficient linear elastic models of soft tissues for real-time surgery simulation. *Proc. MMVR (Medicine Meets Virtual Reality)*, (7) :139–151, 1999.

[DDCB01] G. Debunne, M. Desbrun, M-P. Cani, and A. H. Barr. Dynamic real-time deformations using space and time adaptive sampling. In *SIGGRAPH '01*, 2001.

[Di 01] Thomas Di Giacomo. Méthode multi-résolutions pour l'animation d'une forêt interactive. Master's thesis, Institut National Polytechnique de Grenoble, June 2001.

[DMG05] J. Dequidt, D. Marchal, and L. Grisoni. Time critical animation of deformable solids. *Journal of Computer Animation and Virtual Worlds*, 16 :177–187, 2005.

[DO00] J. Dingliana and C. O'Sullivan. Graceful degradation of collision handling in physically based animation. In *Computer Graphics Forum*, volume 19(3), pages 239–248, 2000.

[DSB99] Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects. In *Graphics Interface*, pages 1–8, 1999.

[Dur04] Christian Duriez. *Contact frottant entre objets déformables dans des simulations temps-réel avec retour haptique*. PhD thesis, Université d'Evry, 2004.

[EEH00] B. Eberhardt, O. Etzmuss, and M. Hauth. Implicit-explicit schemes for fast animation with particles systems. In *Proceedings of the Eurographics workshop on Computer Animation and Simulation*, pages 137–151, 2000.

[FAC+07] François Faure, Jérémie Allard, Stéphane Cotin, Paul Neumann, Pierre-Jean Bensoussan, Christian Duriez, Hervé Delingette, and Laurent Grisoni. SOFA : A modular yet efficient simulation framework. In Jocelyne Troccaz and Philippe Merloz, editors, *Surgetica 2007 : Computer-Aided Medical Interventions : tools and applications, September, 2007*, pages 101–108, Chambéry, France, November 2007. Sauramps Medical.

[FAM+02] Laure France, Alexis Angelidis, Philippe Meseure, Marie-Paule Cani, Julien Lenoir, François Faure, and Christophe Chaillou. Implicit representations of the human intestines for surgery simulation. In Marc Thiriet, editor, *Modelling and Simulation for Computer-aided Medicine and Surgery, MS4CMS 2002, November, 2002*, volume 12 of *ESAIM : Proceedings*, pages 42–47, Rocquencourt, France, 2002. INRIA.

[FAN07] François Faure, Jérémie Allard, and Matthieu Nesme. Eulerian contact for versatile collision processing. Technical Report RR-6203, INRIA, 2007.

[Fau95] François Faure. Inverting the reflectance map with binary search. In Václav Hlavác and Radim Sára, editors, *6th International Conference on Computer Analysis of Images and Patterns, CAIP'95, September, 1995*, volume 970 of *Lecture Notes in Computer Science*, pages 814–819, Prague, Tchéquie, September 1995. Springer.

[Fau96a] François Faure. An energy-based approach for contact force computation. *Comput. Graph. Forum*, 15(3) :357–366, 1996. Special Issue : Proceedings of Eurographics, Poitiers, France.

[Fau96b] François Faure. Modélisation cinématique du contact pour la dynamique inverse. *Rev. Int. CFAO Inform. Graph.*, 1996.

[Fau97] François Faure. *Deux problèmes physiques pour la synthèse d'images*. PhD thesis, Université Joseph Fourier, September 1997.

[Fau98] François Faure. Interactive solid animation using linearized displacement constraints. In *9th Eurographics Workshop on Computer Animation and Simulation, EGCAS, September, 1998*, pages 61–72, 1998.

[Fau99] François Faure. Fast iterative refinement of articulated solid dynamics. *IEEE Trans. Vis. Comput. Graph.*, 5(3) :268–276, July 1999.

[FBAF08] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou. Image-based collision detection and response between arbitrary volumetric objects. In *ACM Siggraph/Eurographics Symposium on Computer Animation, SCA 2008, July, 2008*, Dublin, Irlande, July 2008.

[FDCM97] François Faure, Gilles Debunne, Marie-Paule Cani, and Franck Multon. Dynamic analysis of human walking. In Daniel Thalmann and Michiel Van De Panne, editors, *8th Eurographics Workshop on Computer Animation and Simulation, September, 1997*, pages 53–66, Budapest, September 1997. Published under the name Marie-Paule Cani-Gascuel.

[Fea87] Roy Featherstone. *Robot Dynamics Algorithm*. Kluwer Academic Publishers, Norwell, MA, USA, 1987. Manufactured By-Kluwer Academic Publishers.

[Fel00] C.A. Felippa. Nonlinear finite element methods. *Lecture Notes of The University of Colorado*, 2000.

[FFH+99] François Faure, Chris Faisstnauer, Gerd Hesina, Amaury Aubel, Marc Escher, Frédéric Labrosse, Jean-Christophe Nebel, and Jean-Dominique Gascuel. Collaborative animation over the network. In *Computer Animation 1999, May, 1999*, pages 107–117, Geneva, Suisse, 1999. IEEE.

[FHFG99] Anton Fuhrmann, Gerd Hesina, François Faure, and Michael Gervautz. Occlusion in collaborative augmented environments. *Comput. Graph.-UK*, 23(6) :809–819, December 1999.

[FLA+05] Laure France, Julien Lenoir, Alexis Angelidis, Philippe Meseure, Marie-Paule Cani, François Faure, and Christophe Chaillou. A layered model of a virtual human intestine for surgery simulation. *Med. Images Anal.*, 9(2) :123–132, 2005.

[flo] Flowvr. http ://flowvr.sourceforge.net/.

[FvdPT97] P. Faloutsos, M. van de Panne, and D. Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3) :201–214, 1997.

[GBF03] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. In *ACM TOG, Proc. of SIGGRAPH*, volume 22, pages 871–878, 2003.

[GCMS00a] F. Ganovelli, P. Cignoni, C Montani, and R. Scopigno. Enabling cuts on multiresolution representation. In *CGI '00 : Proceedings of the International Conference on Computer Graphics*, page 183, 2000.

[GCMS00b] Fabio Ganovelli, Paolo Cignoni, Claudio Montani, and Roberto Scopigno. A multiresolution model for soft objects supporting interactive cuts and lacerations. *Computer Graphics Forum (Proc. of EG)*, 19(3), 2000.

[GCS99] F. Ganovelli, P. Cignoni, and R. Scopigno. Introducing multiresolution representation in deformable object modeling. In *ACM Spring Conference on Computer Graphics*, pages 149–158, 1999.

[GD01] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. *IEEE Transactions on Visualization and Computer Graphics*, 7(4) :289–298, 2001.

[GD04] Stéphane Guy and Gilles Debunne. Monte-carlo collision detection. Technical Report RR-5136, INRIA, March 2004.

[GEW05] Joachim Georgii, Florian Echtler, and Rüdiger Westermann. Interactive simulation of deformable bodies on gpus. In *Proceedings of Simulation and Visualisation 2005*, pages 247–258, 2005.

[GKLM07] Naga K. Govindaraju, Ilknur Kabul, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection among deformable models using graphics processors. In *Computer Graphics*, volume 31(1), 2007.

[GKS02] Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms : a simple framework for adaptive simulation. In *SIGGRAPH '02*, 2002.

[GLM96] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree : a hierarchical structure for rapid interference detection. In *SIGGRAPH : Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180, 1996.

[GO01] F. Ganovelli and C. O'Sullivan. Animating cuts with on-the-fly re-meshing, 2001.

[GPR+03] Sylvain Guerraz, Frank Perbet, David Raulo, François Faure, and Marie-Paule Cani. A procedural approach to animate interactive natural sceneries. In *16th International Conference on Computer Animation and Social Agents, CASA 2003, May, 2003*, pages 73–78, Rutgers University, New Brunswick, NJ, Etats-Unis, 2003. IEEE.

[gri] Grimage. http ://www.inrialpes.fr/grimage.

[GSLF05] E. Guendelman, A. Selle, F. Losasso, and R. Fedkiw. Coupling water and smoke to thin deformable and rigid shells. In *SIGGRAPH*, pages 973–981, 2005.

[GW06] Joachim Georgii and Rüdiger Westermann. A multigrid framework for real-time simulation of deformable bodies. *Computer & Graphics*, 30 :408–415, 2006.

[HE01] M. Hauth and O. Etzmuss. A high performance solver for the animation of deformable objects using advanced numerical methods. In *Proceedings of Eurographics*, 2001.

[HFR08] Everton Hermann, François Faure, and Bruno Raffin. Ray-traced collision detection for deformable bodies. In *3rd International Conference on Computer Graphics Theory and Applications, GRAPP 2008, January, 2008*, Funchal, Madeira, Portugal, January 2008.

[HH98] P. Howlett and W. T. Hewitt. Mass-spring simulation using adaptive non-active points. *Computer Graphics Forum (Proc. of EG)*, 17(3), 1998.

[HPH96] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulations. In *Eurographics Workshop on Computer Animation and Simulation*, pages 31–45, 1996.

[HSO03] Kris K. Hauser, Chen Shen, and James F. O'Brien. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, pages 247–256, 2003.

[HTG03] Bruno Heidelberger, Matthias Teschner, and Markus Gross. Real-time volumetric intersections of deforming objects. In *Proc. of Vision, Modeling, Visualization (VMV)*, 2003.

[HTG04a] Bruno Heidelberger, Matthias Teschner, and Markus Gross. Detection of collisions and self-collisions using image-space techniques. In *WSCG*, 2004.

[HTG04b] Bruno Heidelberger, Matthias Teschner, and Markus Gross. Detection of collisions and self-collisions using image-space techniques. In *Proc. of WSCG'04*, pages 145–152, 2004.

[ITF04] G. Irving, J. Teran, and R. Fedkiw. Invertible finite elements for robust simulation of large deformation. In *Proc SCA*, 2004.

[JJCK07] Lenka Jerabkova, Jakub Jerabek, Rostislav Chudoba, and Torsten Kuhlen. A stable cutting method for finite elements based virtual surgery simulation. In *Proc of Medicine Meets Virtual Reality (MMVR)*, 2007.

[JP04] D. L. James and D. K. Pai. Bd-tree : Output-sensitive collision detection for reduced deformable models. *Proc of ACM SIGGRAPH*, 2004.

[JTT01] P. Jimenez, F. Thomas, and C. Torras. 3d collision detection : A survey. *Computer and Graphics*, 25(2) :269–285, 2001.

[kaa] Kaapi. http ://kaapi.gforge.inria.fr/.

[KNF04] Stefan Kimmerle, Matthieu Nesme, and François Faure. Hierarchy accelerated stochastic collision detection. In Bernd Girod, Marcus Magnor, and Hans-Peter Seidel, editors, *9th International Workshop on Vision, Modeling, and Visualization, VMV 2004, November, 2004*, pages 307–314, Stanford, California, Etats-Unis, November 2004. Aka.

[KSS01] S. W. Kirkpatrick, M. Schroeder, and J. W. Simons. Evaluation of passenger rail vehicle crashworthiness. *International Journal of Crashworthiness*, 6 :95–106, 2001.

[KZ03] J. Klein and G. Zachmann. Adb-trees : Controlling the error of time-critical collision detection. In *8th International Fall Workshop Vision, Modeling, and Visualization (VMV)*, pages 19–21, 2003.

[LC84] A. Luciani and C. Cadoz. Modélisation et animation gestuelle dóbjets - le système anima. Technical report, ACROE, 1984.

[LJC+91] A. Luciani, S. Jimenez, C. Cadoza, J.-L. Florens, and O. Raoult. An unified view of multitude behaviour, flexibility, plasticity and fractures balls, bubbles and agglomerates. *Modeling in Computer Graphics (Proc of IFIP WG 5.10 Working Conference)*, 1991.

[LM03] M. Lin and D. Manocha. Collision and proximity queries. In *Handbook of Discrete and Computational Geometry*, 2003.

[LSH07] B. Lloyd, G. Szekely, and M. Harders. Identification of spring parameters for deformable object simulation. *IEEE Transactions on Visualization and Computer Graphics*, 13, 2007.

[MBF04] Neil Molino, Zhaosheng Bao, and Ron Fedkiw. A virtual node algorithm for changing mesh topology during simulation. In *SIGGRAPH*, volume 23, pages 385–392, 2004.

[MCG03] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Symposium on Computer Animation*, pages 154–159, 2003.

[MDM+02] Matthias M&#252 ;ller, Julie Dorsey, Leonard McMillan, Robert Jagnow, and Barbara Cutler. Stable real-time deformations. In *SCA '02 : Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54, New York, NY, USA, 2002. ACM Press.

[MHHR06] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. In *Proceedings of Virtual Reality Interactions and Physical Simulations (VRIPhys)*, pages 71–80, 2006.

[MHTG05] M. Müller, B. Heidelberger, M. Teschner, and M. Gross. Meshless deformations based on shape matching. In *Proc SIGGRAPH'05*, pages 471–478, july 2005.

[Mil05] Karol Miller. Method of testing soft biological tissues in compression. *J. Biomechanics*, 38, 2005.

[MK00] Andrew B. Mor and Takeo Kanade. Modifying soft tissue models : Progressive cutting with minimal new element creation. In *MICCAI '00 : Proceedings of the Third International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 598–607, 2000.

[MKN+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *Proc SCA*, pages 141–151, 2004.

[MO03] M. Matyka and M Ollila. A pressure model for soft body simulation. In *Sigrad*, 2003.

[MO05] C. Mendoza and C. O'Sullivan. Towards time-critical collision detection for deformable objects based on reduced models. In *CASA*, 2005.

[MO06] Cesar Mendoza and Carol O'Sullivan. Interruptible collision detection for deformable objects. *Computer and Graphics Journal*, 30(2), 2006.

[MOK95] K. Myszkowski, O. Okunev, and T. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. In *The Visual Computer*, volume 11(9), pages 497–512, 1995.

[MP89] Gavin Miller and Andrew Pearce. Globular dynamics : A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3) :305–309, 1989.

[MTG04] M. Müller, M. Teschner, and M. Gross. Physically based simulation of objects represented by surface meshes. In *Proc SCA*, 2004.

[Nes08] Matthieu Nesme. *Milieu mécanique déformable multirésolution pour la simulation interactive*. PhD thesis, Université Joseph Fourier, juin 2008.

[NFP06] Matthieu Nesme, François Faure, and Yohan Payan. Hierarchical multi-resolution finite element model for soft body simulation. *Lecture Notes in Computer Science (Proc. of Symposium on Biomedical Simulation)*, 2006.

[NMP+05] Matthieu Nesme, Maud Marchal, Emmanuel Promayon, Matthieu Chabanas, Yohan Payan, and François Faure. Physically realistic interactive simulation for biological soft tissues. *Recent Research Developments in Biomechanics*, 2, 2005.

[NPF05] M. Nesme, Y. Payan, and F. Faure. Efficient, physically plausible finite elements. In *Eurographics (short papers)*, pages 77–80, august 2005.

[NPF06] Matthieu Nesme, Yohan Payan, and François Faure. Animating shapes at arbitrary resolution with non-uniform stiffness. In *Eurographics Workshop in Virtual Reality Interaction and Physical Simulation (VRIPHYS)*, 2006.

[OBH02] James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins. Graphical modeling and animation of ductile fracture. *ACM Trans. Graph.*, 21(3) :291–294, 2002.

[OGRG07] Miguel A. Otaduy, Daniel Germann, Stephane Redon, and Markus Gross. Adaptive deformations with fast tight bounds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 181–190, 2007.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99 : Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 137–146, 1999.

[PBP96] E. Promayon, P. Baconnier, and C. Puech. Physically-based deformations constrained in displacements and volume. *Computer Graphics Forum, Eurographics96*, 15(3) :155–164, August 1996.

[PC01] Frank Perbet and Marie-Paule Cani. Animating prairies in real-time. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)*, USA, Mar 2001.

[PDA00] Guillaume Picinbono, Hervé Delingette, and Nicholas Ayache. Real-time large displacement elasticity for surgery simulation : Non-linear tensor-mass model. *MICCAI*, pages 643–652, 2000.

[PKA+05] Mark Pauly, Richard Keiser, Bart Adams, Philip Dutré ;, Markus Gross, and Leonidas J. Guibas. Meshless animation of fracturing solids. *ACM Trans. Graph.*, 24(3) :957–964, 2005.

[Pro95] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–154. Canadian Human-Computer Communications Society, 1995.

[PW89] A. Pentland and J. Williams. Good vibrations : modal dynamics for graphics and animation. In *ACM SIGGRAPH*, volume 23 (3), pages 215–222, 1989.

[Rag06] Laks Raghupathi. *Vers Une Animation Robuste d' Objets Complexes en Interaction*. PhD thesis, Institut National Polytechniqe de Grenoble, 46 Avenue Felix Viallet, 38031 Grenoble Cedex, France, November 2006.

[RCFC03] Laks Raghupathi, Vincent Cantin, François Faure, and Marie-Paule Cani. Real-time simulation of self-collisions for virtual intestinal surgery. In Nicholas Ayache and Hervé Delingette, editors, *International Symposium on Surgery Simulation and Soft Tissue Modeling*, number 2673 in Lecture Notes in Computer Science, pages 15–26. Springer-Verlag, 2003.

[RGF+04] Laks Raghupathi, Laurent Grisoni, François Faure, Damien Marchall, Marie-Paule Cani, and Christophe Chaillou. An intestinal surgery simulator : Real-time collision processing and visualization. *IEEE Trans. Vis. Comput. Graph.*, 10(6) :708–718, November 2004.

[RJ07] Alec R. Rivers and Doug L. James. Fastlsm : Fast lattice shape matching for robust real-time deformation. In *ACM Transactions on Graphics (Proc of SIGGRAPH)*, 2007.

[sce] Scenegraphs : past, present and future. http ://www.scribd.com/doc/4103/scenegraphs-past-present-and-future.

[SDF07] E. Sifakis, K. Der, and R. Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *Symposium on Computer Animation*, pages 73–80, 2007.

[SGHS98] Jonathan W. Shade, Steven J. Gortler, Li-Wei He, and Richard Szeliski. Layered depth images. *Computer Graphics*, 32 :231–242, 1998.

[SML02] K. Sundaraj, C. Mendoza, and C. Laugier. A fast method to simulate virtual deformable objects with force feedback. In *Control, Automation, Robotics and Vision (ICARCV)*, volume 1, pages 413–418, 2002.

[SMMB00] N. Sukumar, N. Moës, B. Moran, and T. Belytschko. Extended finite element method for three-dimensional crack modeling. In *International Journal for Numerical Methods in Engineering*, volume 48 (11), pages 1549–1570, 2000.

[SOF] SOFASimulation open framework architecture. http ://www.sofa-framework.org.

[SSIF07] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. Hybrid simulation of deformable solids. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 81–90, 2007.

[TDLD07] Nicolas Tsingos, Carsten Dachsbacher, Sylvain Lefebvre, and Matteo Dellepiane. Instant sound scattering. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, 2007.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, Laks Raghupathi, A. Fuhrmann, Marie-Paule Cani, François Faure, N. Magnetat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1) :61–81, 2005.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *SIGGRAPH '87*, 1987.

[TPF91] Demetri Terzopoulos, John Platt, and Kurt Fleischer. Heating and melting deformable models. *The Journal of Visualization and Computer Animation*, 2(2) :68–73, 1991.

[vdDKP01] Kees van den Doel, Paul G. Kry, and Dinesh K. Pai. Foleyautomatic : physically-based sound effects for interactive simulation and animation. In *SIGGRAPH '01 : Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 537–544, 2001.

[VMT01] P. Volino and N. Magnenat-Thalmann. Comparing efficiency of integration methods for cloth animation. In *Proceedings of IEEE Computer Graphics International (CGI)*, pages 265–274, 2001.

[VMT05] P. Volino and N. Magnenat-Thalmann. Implicit midpoint integration and adaptive damping for efficient cloth simulation. In *Computer Animation and Virtual Worlds*, volume 16 (3-4), pages 163–175, 2005.

[VMT06] P. Volino and N. Magnenat-Thalmann. Resolving surface collisions through intersection contour minimization. *ACM Transactions on Graphics (SIGGRAPH)*, 25(3) :1154–1159, July 2006.

[VVW04] L.M. Vigneron, J. G. Verly, and S. K. Warfield. Modelling surgical cuts, retractions, and resections via extended finite element method. In *Lecture Notes in Computer Science*, volume 3217, pages 311–318, 2004.

[WDGT01] Xunlei Wu, Michael S. Downes, Tolga Goktekin, and Frank Tendick. Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshe. In *Computer Graphics Forum (Proc. of EG)*, volume 20(3), pages 349–358, 2001.

[WW90] A. Witkin and W. Welch. Fast animation and control for non-rigid structures. In *ACM SIGGRAPH*, volume 24 (4), pages 243–252, 1990.

[Zar03] Florence Zara. *Algorithmes parallèles de simulation physique pour la synthèse d'images : application à l'animation de textiles*. PhD thesis, Institut National Polytechnique de Grenoble, December 2003.

[ZFV02] Florence Zara, François Faure, and Jean-Marc Vincent. Physical cloth simulation on a PC cluster. In *Parallel Graphics and Visualisation, September, 2002*, EG Workshop Proceedings, Blaubeuren, Allemagne, 2002.

[ZFV04] Florence Zara, François Faure, and Jean-Marc Vincent. Parallel simulation of large dynamic system on a pcs cluster : Application to cloth simulation. *International Journal of Computers and Applications*, 2004.

[ZG03] Florence Zara and Thierry Gauthier. Efficient and easy parallel implementation of large numerical simulations. In Jack J. Dongarra, Domenico Laforenza, and Salvatore Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, ParSim, Special Session of EuroPVM/MPI, September, 2003*, number 2840 in Lecture Notes in Computer Science, pages 663–666, Venise, Italie, September 2003. Springer.

[ZKGB82] O.C. Zienkiewicz, D.W. Kelly, J. Gago, and I. Babuska. Hierarchical finite element approaches, error estimates and adaptive refinement. *Mathematics of Finite Elements and Applications*, 4 :313–346, 1982.

[ZVF03] Florence Zara, Jean-Marc Vincent, and François Faure. Coupling parallel simulation and parallel visualization on pc clusters. In *Commodity Clusters For Virtual Reality (VR)*, 2003.

chapitre A

Travaux anciens

FIG. A.1 – Solides en contact.
Un compromis entre précision et temps de calcul est obtenu en limitant les itérations du gradient conjugué dans les phases de correction des accélérations, des vitesses et des positions.

1 Premiers travaux

Au cours de mon DEA, j'ai effectué une étude théorique sur la résolution de problèmes de planification robotique avec partages de ressources. Puis j'ai eu la chance d'être accepté par Claude Puech dans son équipe de synthèse d'images, iMAGIS. Mes premiers travaux de thèse [Fau97] concernaient la reconstruction d'objets réels, en utilisant les nuances de gris sur plusieurs images : stéréovision photométrique, comme illustré sur la figure A.2. En chaque point, on calcule la normale en résolvant un système de trois équations non linéaires à trois inconnues. Puis on intègre le champ de normales pour obtenir la surface. En utilisant une particularité des équations, j'ai proposé une nouvelle méthode de résolution du système d'équations qui ramène le problème à une recherche dichotomique en une dimension [Fau95]. Ces travaux m'ont sensibilisé au problème de la robustesse : l'intégration des normales était très sensible aux erreurs, et il m'a fallu réaliser des passes de correction après le calcul des normales. J'ai rendu le champ intégrable par des corrections itératives qui m'ont fait découvrir la possibilité de compromis entre précision et temps de calcul.

2 Dynamique des solides rigides

Mon intérêt s'est ensuite porté sur la synthèse d'animation, où j'ai pu exploiter mes connaissances de mécanique. Toutefois, contrairement aux mécanismes que j'avais l'habitude d'étudier, les simulations concernent souvent des systèmes gouvernés par des contacts entre objets qui s'établissent et se défont au cours du temps, comme la marche d'un personnage ou l'empilement et l'écroulement d'objets. Les contacts

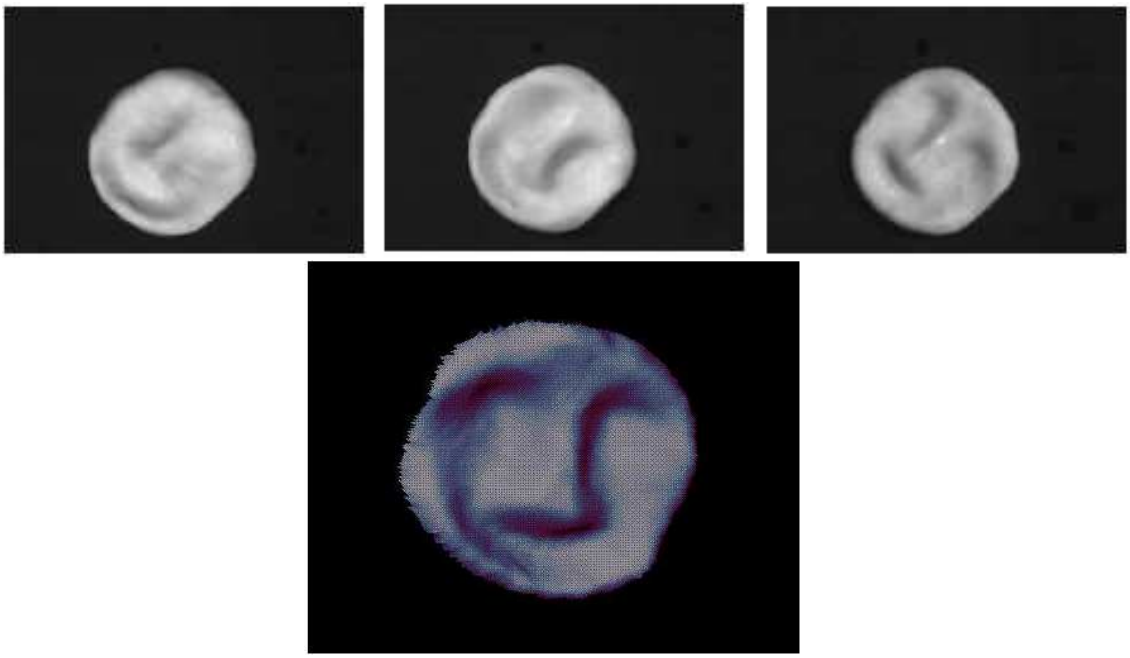


FIG. A.2 – Stéréovision photométrique.

Haut : Images d'un objet éclairé dans trois directions différentes.
Bas : Vue de la surface 3D reconstruite.

non permanents existent en mécanique de l'ingénieur, comme dans les cames ou les butées, mais leur influence est beaucoup plus simple que dans des scènes de synthèse d'animation. Habitué à des formulations en coordonnées relatives dans des hiérarchies cinématiques, j'ai donc développé un modèle articulaire du contact, dans lequel un repère intermédiaire glisse et roule sur les objets [Fau96b]. Ces travaux ont plus tard servi à analyser la dynamique de la marche d'un personnage [FDCM97].

La problématique temps-réel a très vite attiré mon attention. La simulation rigoureuse des collisions implique de traiter chacune d'entre elles au moment précis où elle se produit *(event-driven simulation)*, ce qui peut impliquer un nombre arbitrairement grand d'intervalles de temps entre deux instants. Le même constat peut être effectué quand on traite la dynamique des contacts établis avec des méthodes de programmation linéaire ou quadratique [Bar94]. Le problème est que si la quantité de calculs nécessaire pour avancer le temps simulé d'un intervalle donné (par exemple $1/25s$) n'est pas bornée, alors la simulation en temps réel est tout simplement impossible. Depuis que j'ai constaté ce problème, j'ai toujours cherché des méthodes de calcul à durée bornée, quitte à sacrifier une part de précision. J'applique systématiquement des méthodes qui avancent d'un pas de temps fixe *(time-stepping simulation)*, quitte à corriger itérativement le résultat obtenu. Pour la dynamique des contacts (calcul des accélérations d'objets soumis à des contraintes de répulsion, assorties ou non de frottement de Coulomb), j'ai proposé une méthode qui traite tous les contacts simultanément et fournit une suite de résultats dont chacun porte sur l'ensemble des objets, et qui convergent vers la solution globale [Fau96a]. La convergence n'est malheureusement pas garantie en cas de frottement de Coulomb.

La complexité algorithmique est un élément important pour le passage à l'échelle des méthodes de simulation. Mes premières implémentations formaient des matrices denses, qui ne peuvent se factoriser en temps inférieur à $O(n^3)$, où n est le nombre de degrés de liberté indépendants dans la scène. Les ordinateurs de l'époque permettaient alors simuler en temps réel un assemblage d'une quinzaine d'objets. De nos jours, à cause de la complexité cubique, il reste difficile de dépasser quelques dizaines. C'est une leçon que j'ai retenue : proscrire les matrices denses, et chercher les complexités linéaires. Baraff [Bar96] a montré que les formulations de la dynamique des solides rigides en coordonnées absolues, assortis de contraintes exprimant les liaisons, s'exprime par des matrices toujours creuses. L'algorithme du gradient

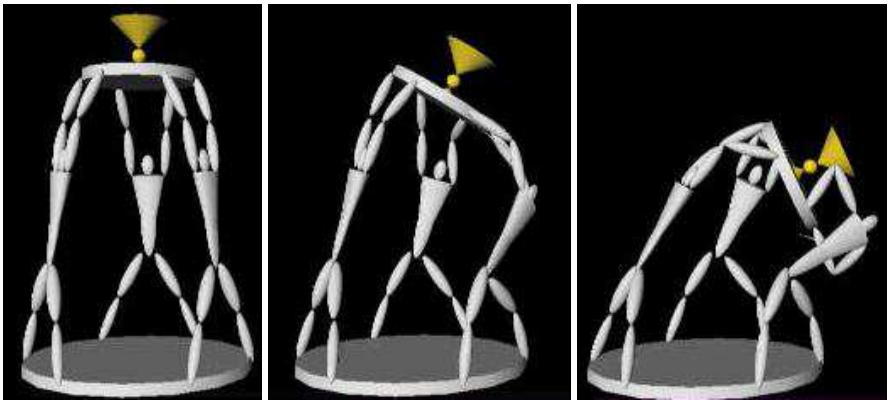


FIG. A.3 – Simulation physique interactive.

Les personnages ont les pieds fixés au sol et tiennent debout grâce à une gravité ascendante. Un utilisateur déplace interactivement les objets.

(bi)conjugé se prête bien à la résolution de tels systèmes, car la complexité théorique y est alors $O(n^2)$. En pratique, dans la plupart des cas, la solution exacte n'est jamais trouvée mais l'algorithme fournit une suite de résultats qui convergent, au moins au début, vers la solution. En limitant le nombre d'itérations, on obtient donc un algorithme en temps linéaire. Toutefois, un résultat imprécis n'est exploitable que si les erreurs ont un effet limité et ne provoquent pas de divergence. J'ai proposé une méthode formulée sur des variations de positions qui absorbe ces erreurs et permet ainsi de d'exploiter l'efficacité algorithmique obtenue [Fau98]. Une exemple d'application interactive est présenté sur la figure A.3.

Featherstone [Fea87] et Baraff [Bar96] ont montré comment réduire la partie cubique de la complexité au nombre de cycles dans le graphe formé par les liaisons entre objets, et obtenir des complexité linéaires quand ce graphe est un arbre. Dans le cas d'une structure comportant peu de chaînes fermées, cette approche est nettement plus efficace que le gradient conjugué. C'est à mon avis une autre leçon à retenir : l'analyse des graphes donne de précieuses indications pour optimiser les calculs. J'ai proposé une méthode hybride qui exploite la structure de graphe afin d'obtenir une solution en temps linéaire quand c'est possible, mais qui résoud les fermetures de chaînes par gradient conjugué. On combine ainsi efficacité et maîtrise du temps de calcul [Fau99]. Ces travaux ont trouvé des applications en réalité virtuelle [FHFG99] pendant mon séjour post-doctoral à l'université de Vienne. La résolution rapide d'équations a notamment permis d'effectuer un suivi de mouvement permettant de calculer des occultations d'objets virtuels, comme illustré sur la figure A.4.

Mon post-doc a été financé par un projet européen, Platform for Animation and Virtual Reality, dont l'esprit était de rassembler les compétences de multiples équipes scientifiques dans un "pot commun" mal défini. Certaines étaient spécialistes de contrôle cinématique, d'autres de dynamique, d'autres encore de traitement d'images ou d'installations de réalité virtuelle. Aucune application particulière n'était identifiée. En pratique, la collaboration a été très difficile non seulement à cause des objectifs différents de chaque équipe, mais aussi parce que la plupart d'entre elles refusaient de diffuser leur code. J'ai proposé de contourner l'obstacle en collaborant à travers une plate-forme distribuée dans laquelle les objets échangeaient des messages dans un langage VRML enrichi par des instructions de mises à jour. Nous avons ainsi pu réaliser des animations impliquant des objets animés par différentes méthodes sur différents ordinateurs [FFH+99]. Des résultats sont présentés sur la figure A.5. De cette expérience, j'ai conclu que la collaboration scientifique gagnait beaucoup par le développement de logiciel open-source et que d'autre part, l'interfaçage de méthodes d'animation est un problème difficile qui m'occupe encore aujourd'hui.

Par la suite, j'ai continué à chercher l'efficacité maximale dans la simulation de solides en collaboration avec Frédéric Donzé, du laboratoire de mécanique L3S, sur la simulation d'éboulements rocheux. Les équations de la mécanique peuvent être exprimées de manière classique sur les accélérations, mais aussi sur les vitesses comme c'est courant dans les simulations de collisions *(impulse-based physics)*, et enfin sur les positions pour obtenir une stabilité maximale *(position-based physics)*. Le même type d'équations

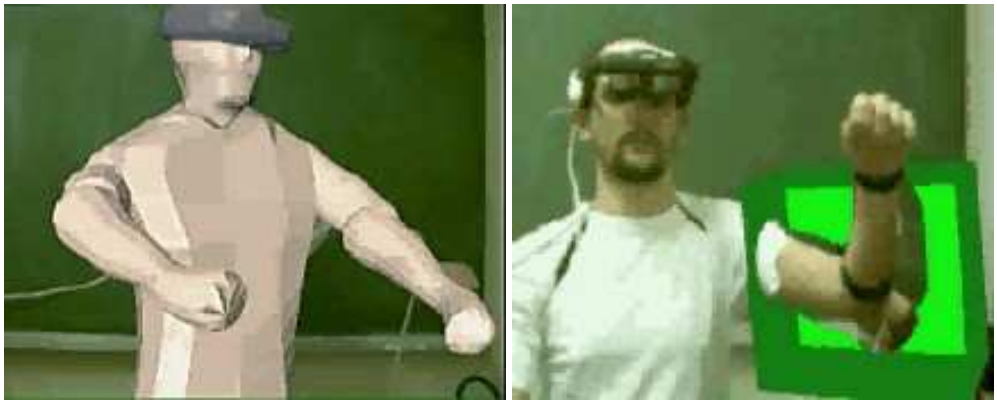


FIG. A.4 – Occultation d'objets virtuels par un personnage réel.

Des capteurs sur le personnage réel permettent de construire la trajectoire d'un personnage virtuel (à gauche) utilisé pour calculer les occultations (à droite).

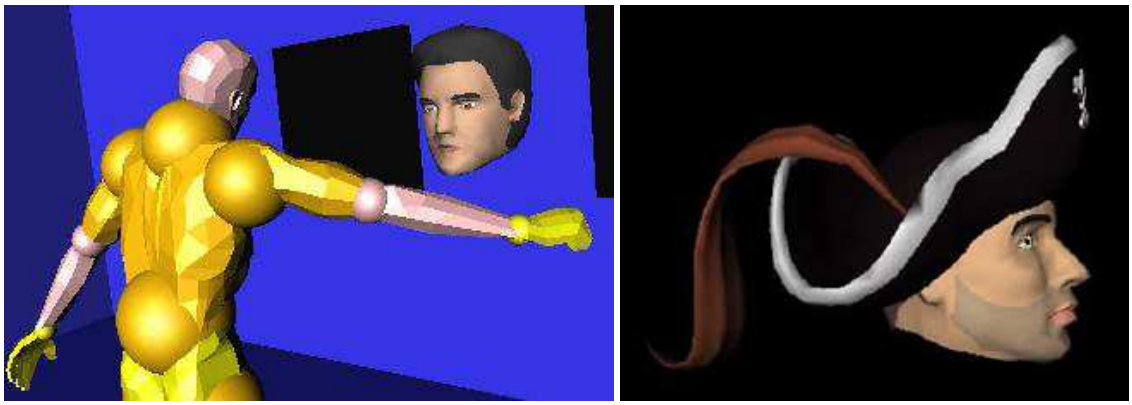


FIG. A.5 – Animation collaborative en réseau.

Gauche : le personnage jaune est animé à Glasgow et Elvis, à Lausanne.
Droite : le pirate est animé par cinématique à Lausanne et sa plume par dynamique à Vienne.

intervient à chaque niveau, qu'on peut donc résoudre itérativement avec une des méthodes discutées précédemment. Nous nous sommes posés la question de la meilleure répartition des calculs parmi ces trois niveaux afin d'obtenir le meilleur compromis entre précision et temps de calcul. Nous avons essayé différentes répartitions du temps de calcul et mesuré le temps de calcul et l'erreur géométrique moyenne (pénétrations entre objets) au cours d'une simulation. Pour un temps de calcul donné, nous avons trouvé que les corrections de vitesses sont à privilégier, suivi des corrections de positions, et que les corrections d'accélérations peuvent être quasiment négligées, comme illustré sur la figure A.6. J'en ai déduit qu'un certain flou sur les contraintes de positions est bénéfique, ce qui renforce ma préférence pour les méthodes de *time-stepping* au détriment des méthodes *event-driven*. Par ailleurs, nous avons noté que la correction indépendante des positions, vitesses et accélérations est considérablement plus simple à règler et plus rapide à simuler que le traitement simultané des trois niveaux en utilisant des stabilisateurs de Baumgarte. Ces résultats sont confirmés par des travaux de mathématiques appliquées et robotique [Asc97] dans lesquels l'approche indépendante est appelée post-stabilisation. C'est la leçon que j'en retiens : préférer la post-stabilisation aux stabilisateurs de Baumgarte. Cette approche a été utilisée pour générer l'animation présentée sur la figure A.1 calculée en 1998.

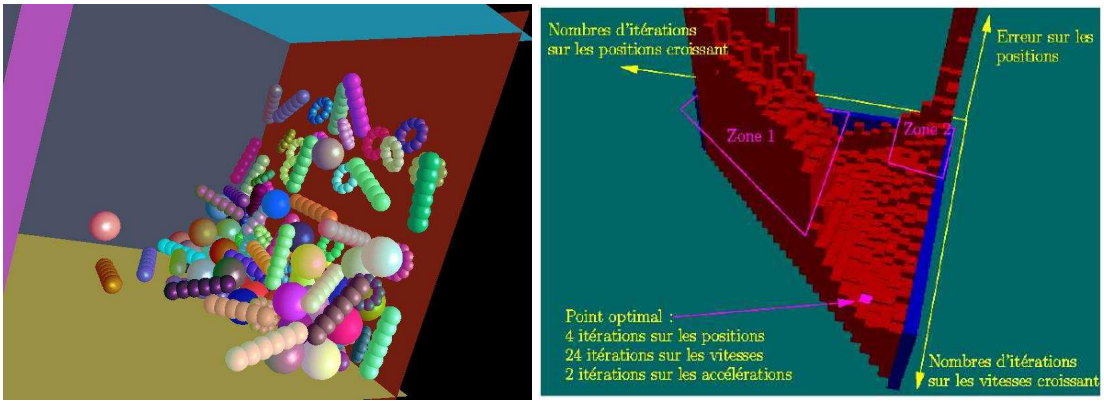


FIG. A.6 – Répartition optimale entre corrections d'accélérations, de vitesses et de positions.

Gauche : la scène de test. Droite : erreur géométrique en fonction des itérations sur les vitesses et les positions.

L'incertitude sur l'état des contraintes de position autorise toutefois des interpénétrations entre objets. Les méthodes géométriques de modélisation du contact doivent donc être robustes à des intersections, ce qui n'est souvent pas le cas pour les surfaces polygonales. Le *time-stepping* n'est donc possible qu'avec des méthodes robustes de modélisation de contact.

3 Modèles adaptatifs

Une façon d'accélérer les calculs est d'en faire moins, en les concentrant là où ils sont les plus nécessaires, et en les espaçant là où la précision importe moins. Les objets déformables se prêtent relativement bien à cette approche car on peut jouer sur la densité d'échantillonnage du milieu continu. En animation, quand seul le résultat visuel compte, on peut aussi choisir un moteur plus simple, comme par exemple rejouer ou combiner des trajectoires précalculées ou procédurales. Au cours du stage de DEA de Thomas Di Giacomo, nous avons exploré ces pistes pour l'animation d'arbres dans le jeu vidéo. Les arbres devaient non seulement réagir au vent, mais aussi des branches devaient être manipulables interactivement. Nous avons modélisé les arbres comme des hiérarchies de solides articulés. Pour la réaction au vent, nous avons appliqué des variations d'angles proportionnelles à sa vitesse. L'effet dynamique, matérialisé par des oscillations, était directement codé dans la vitesse du vent. Pour régler l'échantillonnage du modèle, nous avons figé les articulations à partir d'une certaine distance donnée de la racine. Pour interagir avec une branche, nous avons utilisé un modèle dynamique, nettement plus complexe, restreint à une portion du chemin entre la branche et la racine. Nous avons ainsi pu animer une grande quantité d'arbres par une méthode simple et réglable, tout en autorisant leur interaction physique avec des sollicitations externes [Di 01], comme illustré sur la figure A.7.

Animer une prairie est un encore plus grand défi en raison du nombre élevé de brins d'herbe. De plus, pour des raisons de place mémoire, il est impossible de mémoriser l'état de chaque brin. Dans le contexte du jeu vidéo, une difficulté supplémentaire est que les personnages traversant la prairie doivent y laisser des traces. En nous appuyant sur des travaux antérieurs [PC01], nous avons proposé de représenter la prairie comme un milieu continu échantillonné en grille, chaque brin étant interpolé d'après des valeurs exprimées aux coins de la grille [GPR+03]. Ainsi, seuls les coins sont animés explicitement, et régler la densité de la grille permet d'obtenir un compromis entre précision et temps de calcul. Les traces sont représentées par un aplatissement de l'herbe que nous avons modélisé comme un champ de déformation de l'espace. Ce champ est basé sur la ligne de trajectoire des objets, représentée vectoriellement. Ainsi, la représentation des traces est totalement indépendante de l'herbe, et ne dépend ni du type ni de la quantité d'herbe déformée, comme illustré sur la figure A.8.

La densité d'échantillonnage de la prairie pourrait être adaptative, mais au prix d'une plus grande difficulté d'implémentation pour assurer la continuité entre les cellules. Ce problème ne se pose pas en

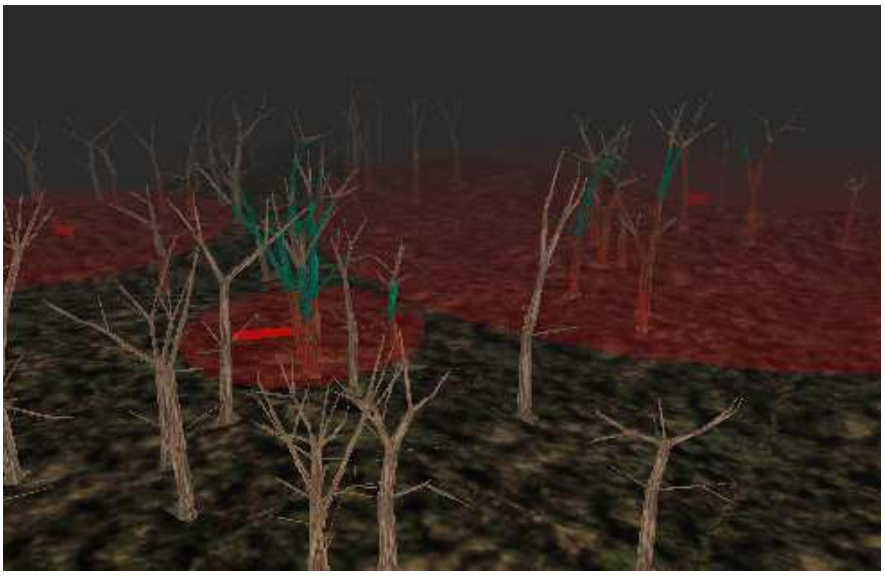


FIG. A.7 – Animation d'arbres pour le jeu vidéo.

Les disques et les traits rouges représentent le vent. Seuls les segments colorés des arbres sont animés, par une méthode indiquée par la couleur.

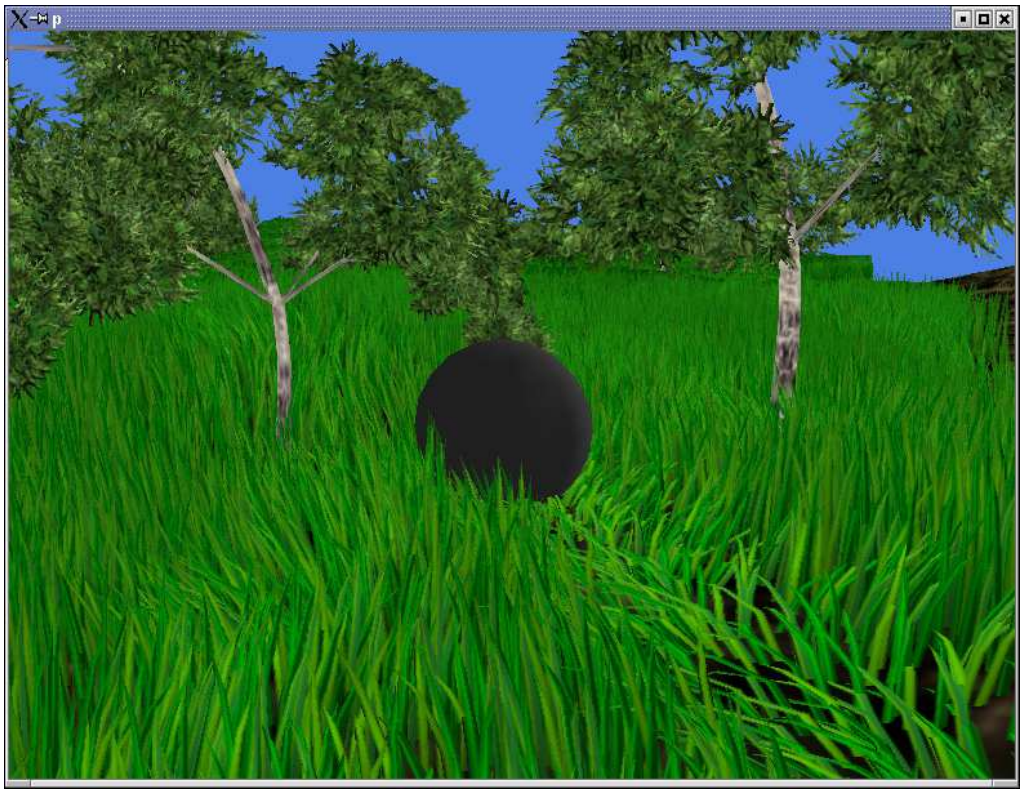


FIG. A.8 – Prairie animée avec traces de passage.

Les traces sont modélisées par déformation de l'espace au voisinage des trajectoires.

une dimension, ce qui permet de rendre facilement les arbres adaptatifs. Il s'aggrave par contre en trois dimensions, comme discuté en section 6 chapitre 2.

Nous nous intéressons aux simulations mécaniques pour des applications de synthèse d'images, particulièrement en temps réel, comme les jeux vidéos ou les simulateurs d'apprentissage de gestes techniques. La simulation physique interactive a fait des progrès remarquables ces dernières années, mais n'a pas encore atteint un niveau de maturité suffisant pour être applicable par des non-spécialistes à une grande variété de modèles. Nous présentons ici des contributions permettant de lever un certain nombre de verrous technologiques.

La modélisation d'objets souples repose généralement sur une décomposition en cellules déformables. Nous avons proposé des éléments finis tétraédriques rapides et robustes, et généralisé à des grilles hexaédriques simplifiant grandement les problèmes de maillage et de contrôle de la résolution.

La détection et la réponse aux collisions posent des problèmes géométriquement complexes et mécaniquement discontinus, qui causent bien souvent les principaux goulots d'étranglement des simulations. Nous avons proposé des solutions au problème du contrôle du temps de calcul, de la diversité des modèles géométriques des surfaces de contact, ainsi que de la robustesse des réponses et de leur calcul accéléré sur GPU.

Une grande attention a été portée aux architectures logicielles permettant d'accroître la modularité et les performances des simulateurs. Nous avons proposé un graphe de scène mécanique permettant de hiérarchiser les modèles et décomposer les algorithmes en modules indépendants. Noua avons également exploré la distribution des simulations sur architectures parallèles, rendue nécessaire par l'évolution des matériels.

We consider mechanical simulations in moving image synthesis applications, especially in real time, such as video games or simulators. Interactive physical simulation has made large progress in the recent years, but it has not yet reached enough maturity to be usable by non-specialist over a large variety of models. We present contributions toward this goal.

Soft object models are generally based on deformable cells. We have proposes fast and robust tetrahedral elements, and generalized the approach to regular hexahedral grids which greatly simplify the problems of meshing and controling the resolution.

Collision detection and response are geometrically complex and mechanically discontinuous problems, which often create bottlenecks in the simulations. We have proposed solutions to control the computation time devoted to detection, to alleviate the problems due to multiple geometric models of the contact surfaces, as well as the robustness of the reaction and its computation on GPU.

Much attention has been paid to software architecture in order to allow both modularity and efficiency. We propose a mechanical scene graph which allows to build model hierarchies and to decompose the simulation algorithms in independent modules. We have also investigated the distribution of the computations in parallel architectures, made necessary by the recent evolution of hardware.