Real-Time Dynamic Simulation and 3D Interaction of Biological Tissue: Application to Medical Simulators

Kenneth Sundaraj

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, FRANCE

No. attribué par la bibliothèque
|_|_|_|_|_|_|_|_|_|_|

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Imagerie, Vision, Robotique

Préparée au laboratoire GRAVIR à l'INRIA Rhône-Alpes,
dans le cadre de l'Ecole Doctorale
Mathématiques, Sciences et Technologie de l'Information

présentée et soutenue publiquement par

Kenneth Sundaraj

le 23/01/2004

Titre

Real-Time Dynamic Simulation and 3D Interaction of Biological Tissue : Application to Medical Simulators

Directeur de Thèse

Christian Laugier

Composition du Jury

M. Augustin Lux	Président
M. Kamal Gupta	Rapporteur
M. Philippe Meseure	Rapporteur
M. François Faure	Examinateur
M. François Leitner	Examinateur
M. Christian Laugier	Directeur de thèse

To my mother,
To my father.

Acknowledgements

It has been a long journey for me. As such, allow me to pen down *more* than just a few words ...

Being a believer, let me begin by thanking God for everything. Nothing is possible without Him.

That said and done, let me get over with the formalities ...

I would like to firstly thank my thesis director, Christian Laugier, for his constant support and encouragement throughout this research. Besides supervising me over these years, he has also given me enough freedom to pursue my research independently.

Next, I am grateful to my thesis committee members; Professor Augustin Lux (Institut National Polytechnique of Grenoble, France), for assuming the role as president of the committee. Professor Kamal Gupta (University of Simon Fraser, Canada) and Associate Professor Philippe Meseure (University of Poitiers, France) for their steadfast enthusiasm for the thesis, and for reading it meticulously. Associate Professor François Faure (Institut National Polytechnique of Grenoble, France) and François Leitner (Aesculap-BBraun, France) for reviewing my manuscript and their constructive comments, and finally Christian Laugier (Research Director at INRIA, France), who deserve commendation for his patience with the many drafts he perused conscientiously.

Many thanks also to all the numerous people in the Sharp/CyberMove/e-Motion team, whom I had pleasure working with. They have provided me with a great research environment and a relaxed atmosphere. In all these years, many have come and have gone. It is almost impossible of me to thank each one of them individually, with all of whom I developed warm friendships.

For financial support during my PhD, I would like to thank the French Ministry of Foreign Affairs (MAE) for the research scholarship.

I am also grateful to everybody at INRIA, especially the staff who have been both efficient and friendly. Special thanks to the project assistants; Anne Pasteur and Veronique Roux who have guided me on more than one occasion through the often obscure bureaucratic procedures that perplexed me. The librarians also de-

serve no less; they have on numerous occasions helped me to find an article or book on which I was not able to lay my hands upon.

I guess that about sums up all that has to be said, formally speaking. Now (inevitably the most interesting part), I would like acknowledge the people (in order of appearance) who have made this journey very special and memorable ...

Some friends have been with me right from the beginning. That has been the case of Diego, François and César. Their presence, friendship and cheerful good humor have enabled me to just enjoy life at work and even after work.

My deepest gratitude to Carla and Kamel who have been like an elder sibling to me. Their eternal and personal kindness has translated into numerous pleasurable moments over these years.

I also appreciate the efforts of my *soul-buddies*, Anne and Laks, for putting up with me during the sometimes very stressful last few months of my PhD. I can't even begin to imagine the pain and agony that I've put them through. Their patience and serenity has helped me to keep calm and relax.

Aside from acknowledging the many friends that have made my time at Grenoble pleasurable, I would especially like to thank Christophe, Cedric, Veronika, Marta and Frank for the many lovely times that we spent together.

Many thanks to all of those that have proof-read the early versions of my thesis manuscript. They have allowed me to correct several errors in this document. Furthermore, their comments and suggestions have helped me to present my work in a more comprehensible manner. Any remaining mistakes are, of course, entirely their responsibility.

For giving me a home away from the research lab, I thank my residence housemates through the years; Blandine, Pietro and Natalia.

And finally, I would like to thank my parents, all my brothers and my sister for their love and encouragement during my absence amongst my family.

Contents

Extended Abstract (in French)

Chapitre 1 : Introduction

1.1 Contexte et Motivations

L'avènement de l'imagerie médicale a bouleversé les méthodes de travail des médecins. Les investigations de l'informatique pour faire avancer la médecine continuent, et un champs de travail, très actif actuellement, est celui du développement de simulateurs médicaux.

Un des objectifs de ces travaux est de proposer aux médecins des simulateurs pour l'entraînement à des procédures chirurgicales, de la même manière qu'il existe des simulateurs de vol pour former les pilotes. Afin d'être réalistes, ces simulateurs doivent évidemment intégrer des modèles mécaniques de déformation des organes. Le défi est d'envergure, car la mécanique du *vivant* est loin d'être complètement connue, du fait même de la complexité des systèmes vivants.

1.2 Description du problème

Aujourd'hui plusieurs simulateurs médicaux existent déjà. Par exemple on peut s'initier à la laparoscopie en utilisant un simulateur doté d'un dispositif retour d'efforts ou interagir avec un modèle mécanique de l'oeil basé sur les éléments finis. Aussi des modèles construits à partir de donnés du patient ont été proposés.

Par contre, il y a encore beaucoup de problèmes á résoudre pour avoir un simulateur de chirurgie qui soit réaliste et interactif. Tout d'abord, il faut simuler les déformations des organes dues aux forces des outils chirurgicaux. Ces déformations sont á la base non linéaires et sa simulation en temps réel constitue un des domaines de recherche les plus importants dans la simulation physique des organes. Dans ce cadre, nous nous sommes intéressés au problème de la modélisation des phénomènes

de déformation de tissu biologique et à la détection des collisions dans un environment virtuel.

1.3 Buts et Contributions

L'objectif de ce travail porte sur la création d'un cadre pour le développement d'un simulateur de chirurgie en réalité virtuelle. Plus particulièrement, les objectifs sont de:

- Proposer un modèle numérique efficace et réaliste pour la simulation de tissu mou.
- Implémenter une librairie de détection de collision pour les objets déformables.
- Explorer sa faisabilité dans un simulateur médical.

Les contributions apportées par ce travail sont les suivantes:

- Un nouveau modèle déformable pour la simulation de tissus mous - Volume Distribution Method (VDM).
- Une librairie de détection de collision pour des simulateurs médicaux - ColDetect.
- Développement de simulateurs médicaux prototypes - Simulateur de chirurgie arthroscopique du ligament du genou.
- Un ensemble d'études comparatives des différents modèles physiques et des méthodes de détection de collision.

1.4 Organisation de la thèse

Le document est constitué de la manière suivante : le chapitre 2 présente l'état de l'art sur les modèles physiques de tissu mou et leur résolution. Le chapitre 3 présente ensuite un modèle développé pour la simulation de tissu biologique, en présentant successivement les aspects liés à la formulation du modèle, à la résolution du modèle, et au traitement des interactions physiques. Ce modèle, basé sur l'utilisation du principe de Pascal, permet de modéliser de manière relativement satisfaisante des corps biologiques, tout en permettant une simulation interactive. Dans le chapitre 4, nous présentons les différents algorithmes existants pour la détection de collision, ainsi que la difficulté d'adapter ces algorithmes aux simulateurs médicaux où les objets déformables complexes forment la base du modèle. Le chapitre 5 expose

les algorithmes développés pour traiter ce problème dans le cadre des simulateurs médicaux. Ces algorithmes présentent des caractéristiques de robustesse numérique et d'efficacité supérieures á l'existant, et permettent de traiter des corps déformables. Dans le chapitre 6, nous présenterons les résultats dans le cadre d'un simulateur échographique de la cuisse humaine et d'une simulateur de chirurgie arthroscopique du LCA (ligament croisé antérieur du genou). Enfin, le chapitre 7 présente le bilan des études menées et expose quelques perspectives.

Partie I : Modèles Physiques de Tissu Mou

Chapitre 2 : État de l'Art

2.1 Introduction

La majorité de l'anatomie humaine est composée de tissus mous. Pour décrire le phénomène de la déformation, le domaine de la mécanique a mis en place une base théorique pour les milieux continus; mais pour modéliser les objets de géométrie complexe on devra néanmoins les discrétiser en éléments pour appliquer cette théorie de la déformation.

2.2 Modèles Numériques

2.2.1 Modèle Masses-Ressorts (MSN)

C'est le modèle le plus simple. Il consiste á modéliser l'objet par un ensemble de points reliés entre eux par des segments. Les points représentent des masses considérées comme ponctuelles, et les segments sont considérés comme des ressorts. Dans le processus de simulation, á chaque itération, on calcule les forces exercées par chaque ressort sur les deux masses situées á ses extrémités. Ces forces s'expriment en fonction de la variation de longueur du ressort et de sa raideur. Chaque masse ou point présente un comportement dynamique en suivant la loi de Newton:

$$M_i\ddot{\mathbf{U}}_i + D_i\dot{\mathbf{U}}_i + \sum_j \mathbf{F}_i^{int} = \sum \mathbf{R}_i^{ext} \tag{2.1}$$

2.2.2 Modèle Élasticité (ETM)

ETM est basé sur la loi de Hooke, qui relie le tenseur d'effort au tenseur de contrainte; il emploie une approximation discrète des opérateurs dérivés sur les points irréguliers. Il permet l'adaptation d'espace, l'adaptation de temps et la distribution de ressources de calcul d'une manière efficace pour assurer la stabilité numérique.

Dans cette méthode, pour un maillage géométrique donné, nous pulvérisons arbitrairement quelques points de prélèvement à l'intérieur. Selon la théorie d'élasticité, pour chaque point de prélèvement, nous avons:

$$\rho\, \ddot{\mathbf{U}}_i = \mu\, (\nabla^2 \mathbf{U}_i) + (\lambda + \mu)\Big\{\nabla(\nabla \cdot \mathbf{U}_i)\Big\} \tag{2.2}$$

où λ et μ sont des coefficients de Lamé caractérisant la rigidité du matériel, ρ est la densité matérielle, $\ddot{\boldsymbol{U}}_i$ est l'accélération d'un point et $\boldsymbol{U}$ est son déplacement.

2.2.3 Modèle Éléments Finis (FEM)

La méthode des élément finis utilise les bases de la mécanique pour la simulation de déformations. La méthode analyse l'énergie potentielle de déformation de l'objet. Le but est de chercher la position d'équilibre qui minimise cette énergie. Pour cela, on découpe l'objet en sous-domaines (tétrahèdres ou hexahèdres). Sur chaque sous-domaine, on définit un ensemble de points (les noeuds) de contrôle sur lesquels le problème sera évalué. Ensuite on utilise des fonctions de forme qui définissent le champ local continu en fonction des valeurs aux points de contrôle. Ce champ local continu dépend de déformations et contraintes de l'objet. L'équation d'équilibre du système doit être vérifiée sur chacun des éléments, ce qui va se traduire par des équations sur chacun des noeuds. En chaque noeud on peut écrire une équation linéaire faisant intervenir la valeur de la fonction de forme dans cette zone. En regroupant toutes ces équations linéaires dans une matrice, on obtient un système matriciel de type statique:

$$\mathbf{KU} = \mathbf{R} \tag{2.3}$$

où $\mathbf{K}$ est appelée matrice de rigidité du système, et $\mathbf{U}$ et $\mathbf{R}$ sont les vecteurs rassemblant respectivement les déplacements et les forces externes appliquées aux noeuds. Si on ajoute l'influence de la masse $\mathbf{M}$ et des amortissement $\mathbf{D}$ on aura un système matriciel de type dynamique:

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{D}\dot{\mathbf{U}} + \mathbf{KU} = \mathbf{R} \tag{2.4}$$

2.2.4 Modèle Masse-Tenseur (TMM)

Le modèle de masse-tenseur discrétise l'organe virtuel avec des tétraèdres. Comme le modèle de masse-ressort, il distribue également la masse dans l'objet sur les noeuds i. L'équation régissant le mouvement des noeuds de maillage est également basée sur la loi newtonienne:

$$M_i \frac{d^2 \boldsymbol{U}_i}{dt^2} + D_i \frac{d\boldsymbol{U}_i}{dt} + \boldsymbol{F}_i^{int} = 0 \tag{2.5}$$

La différence avec MSN est qu'ici $\boldsymbol{F}_i^{int}$ est obtenu par la méthode d'éléments finis basée sur l'énergie. Par rapport au modèle de masse-ressort, le modèle de masse-tenseur calcule la force par la mécanique des milieux continus et est donc indépendant de la topologie de maillage. En revanche, nous savons que le modèle de masse-ressort est sensible à la topologie de maillage. Par conséquent, quand il y a un procédé de modification de topologie, le modèle de masse-tenseur peut donner des résultats plus précis. Ce modèle au debut a été seulement adapté pour des petits déplacements, mais récemment des modifications ont été faites au modèle pour de grands déplacements en employant les tenseurs de contrainte nonlinéaires et les lois matérielles anisotropes.

2.2.5 Modèle Élasticité Hybride (HEM)

Dans le modèle élastique hybride, il y a combinaison d'un modèle élastique quasi-statique linéaire pré calculé avec un modèle de masse-tenseur. Ainsi, ce modèle béneficie des bonnes propriétés des modèles combinés. Mais la précomputation est seulement appropriée aux situations où il n'y a aucun changement de topologie. D'autre part, nous savons également qu'en employant la précomputation, nous pouvons réaliser l'exécution en temps-interactif et obtenir des résultats précis de déformation.

2.2.6 Modèle Sphères Finis (MFS)

La méthode de sphères finies est une méthode sans maillage. Elle a été développée pour surmonter le problème de remaillage dans les méthodes telles FEM. Cette méthode emploie un ensemble de points pour résoudre les équations régissant l'équilibre. Cette méthode a été récemment étendue aux objets déformables. Dans un simulateur médical par exemple, quand un outil chirurgical touche le tissu mou virtuel, un ensemble de points est mis localement autour de la pointe et une sphère avec

un rayon fini est construite à chacun de ces points. Comme dans la méthode classique de FEM, des fonctions de forme sont employées pour rapprocher les champs de déformation. La différence ici est que nous devons donc employer des fonctions raisonnables á la place des polynômes. Bien que les fonctions raisonnables soient soigneusement choisies pour augmenter l'efficacité de calcul, elles mènent toujours à un calcul plus étendu dans la partie d'intégration. Ceci est dû à l'augmentation des points d'interpolation utilisés dans MFS par rapport cas du polynôme normal. C'est évidemment un inconvénient pour des applications d'interactif-temps.

2.2.7 Modèle Éléments Frontières (BEM)

Dans ce modèle, on divise la frontière de l'objet (surface) en n éléments qui représentent les déplacements et les contraintes. Ensuite, on utilise la formulation intégrale de frontière de Navier pour générer un système de $3n$ équations. Enfin, on applique des conditions de frontière en fixant les noeuds. Pour trouver les déformations de l'objet $\mathbf{V}$, il faut inverser la matrice $\mathbf{K}$ du système suivant:

$$\mathbf{KV} = \mathbf{Z}. \tag{2.6}$$

2.2.8 Modèle Éléments Longs (LEM)

Ce modèle considère les tissus comme des objets incompressibles composés en majeure partie de liquide (ce qui est le cas en pratique). L'objet est caractérisé par le principe de Pascal, qui établit que, dans un fluide incompressible en équilibre, les pressions se transmettent intégralement. L'incompressibilité du fluide implique que la conservation du volume doit être garantie lorsqu'il y a interaction externe et déformation de l'objet. Le comportament de l'objet est exprimé par:

$$\mathbf{KX} = \mathbf{B} \tag{2.7}$$

où les déformations sont données par les changements d'état, $\mathbf{X}$, sous l'influence des forces externes (gravité, pression externe, etc.), $\mathbf{B}$. $\mathbf{K}$ est la matrice de rigidité du système.

2.3 Méthodes de Résolutions

Dans la section précédente on a examiné différents modèles physiques qui fournissent les forces intérieures à l'objet. Cela nous permettra de calculer la déformation

due à une force extérieure. On pourra choisir entre deux méthodes de résolution: statique ou dynamique. La résolution statique cherche à trouver directement une position d'équilibre, pendant que la résolution dynamique s'intéresse à l'évolution de la déformation dans le temps.

2.3.1 Système Statique

Étant donné que le déplacement de tous les points de l'objet modélisé est petit une résolution linéaire est suffisante. Pour cela on doit résoudre un système d'équations pour trouver une configuration où les forces intérieures à l'objet s'équilibrent parfaitement avec les forces extérieures (principe des travaux virtuels).

2.3.1.1 Statique Linéaire

Ces approches utilisent les équations d'équilibre statique et ignorent les effets de l'inertie et amortissement. Dans le cas linéaire, le problème est:

$$\mathbf{KU} = \mathbf{R} \tag{2.8}$$

où $\mathbf{K}$ est la matrice de rigidité, $\mathbf{u}$, $\mathbf{F}$ sont les déplacements et forces appliquées á l'objet.

2.3.1.2 Statique Nonliéaire

Si l'objet subit une grande déformation ou une rotation, on doit prendre en compte les non-linéarités géométriques et matérieles. Cela nous oblige à résoudre un système non-linéaire:

$$\mathbf{K}(\mathbf{U})\mathbf{U} = \mathbf{R} \tag{2.9}$$

par exemple, à l'aide de la méthode de Newton-Raphson.

La méthode de Newton-Raphson nous permet de résoudre un système non-linéaire par la résolution *itérative* d'un système d'équations. Cela implique que l'on doit recalculer les forces internes et la matrice de rigidité à chaque itération. Cette méthode est donc beaucoup plus coûteuse qu'une résolution linéaire.

2.3.2 Système Dynamique

La résolution statique cherche directement une configuration d'équilibre si elle existe. Par contre s'il n'y a pas de configuration unique d'équilibre ou que l'on

s'intéresse à l'évolution de la déformation dans le temps une analyse dynamique s'avère nécessaire.

2.3.2.1 Intégration Newton-Euler Explicite

L'intégration explicite utilise les dérivées à la configuration courante. Malheureusement ces dérivées ne sont pas constantes ce qui provoquera une solution erronée. Il est possible de prendre en compte des dérivées d'ordre supérieur pour obtenir une solution plus exacte, mais le pas de temps sera toujours limité par les paramètres de rigidité, viscosité, et masse pour une intégration stable.

2.3.2.2 Intégration Newton-Euler Implicite

A la différence de l'intégration explicite, l'intégration implicite cherche à trouver la configuration dans laquelle les dérivées vont nous ramener à l'état courant pour un pas de temps négatif. Cela veut dire qu'on pourra toujours retourner à l'état précédent, ce qui garantit que cette méthode d'intégration est absolument stable (mais pas forcément exacte) indifféremment des paramètres physiques du système. Malheureusement cette méthode nécessite la résolution d'un système non-linéaire à chaque pas de temps, ce qui la fait très coûteuse en termes de complexité de calcul.

2.4 Bilan

Les modèles décrits présentent tous des avantages et des limitationes. Nous présentons une analyse comparative sur les plans de la rapidité de calcul (aspect important pour les simulations en temps réel), du réalisme physique et de la facilité d'implantation.

Il est possible de trouver la déformation soit par une résolution statique où dynamique. Pour une résolution statique il faut qu'une configuration d'équilibre existe; pour des petites déformations on peut choisir un résolution linéaire, mais sinon une résolution non-linéaire plus coûteuse est exigée. Par contre si l'évolution de la déformation dans le temps nous intéresse, nous devrons procéder à une analyse dynamique. La stabilité de l'intégration explicite est dépendante des paramètres physiques du système. Cela n'est pas le cas avec une intégration implicite, mais on doit résoudre un système non-linéaire à chaque pas de temps.

Chapitre 3 : Méthode de Distribution du Volume (VDM)

La méthode VDM a été conçu pour la simulation en temps réel de tissus biologiques mous (tel que le ligament du genou) dans le cadre de gestes chirurgicaux assistés par ordinateur. Une bonne approximation des tissus biologiques consiste à considérer ces derniers comme des objets déformables remplis d'un fluide incompressible. Notre approche est basée sur la recherche d'une solution statique (i.e. à force résultante nulle) pour des déformations élastiques nonlinéaires. La surface de l'objet est discrétisé par un ensemble de facettes ce qui permet de réduire la complexité du modèle par rapport à une méthode classique tétraédrique (volumique).

3.1 Formulation Mathématique

Pour chaque noeud on définit une équation à l'équilibre sur les variables globales; l'ensemble des équations statiques, associé au Principe de Pascal et à la loi de conservation des volumes, permet alors de définir un système d'équations dont les solutions caractéristiques les déformations et les forces appliqués à l'objet. L'approche consiste alors à:

- Construire une discrétisation surfacique de l'objet à modéliser.
- Définir les équations à l'équilibre pour chaque un des éléments en utilisant le principe de Pascal et la loi de la conservation du volume.
- Prendre en compte les contraintes globales afin d'obtenir un comportement globale de l'objet coherent avec les lois de la physique.

En effet le volume de l'objet est considéré comme rempli d'un fluide incompressible d'une densité donnée; le comportement mécanique de l'objet est alors caracterisé par celui des VDM de son modèle discrétisé, en applicant les deux conditions aux limites suivants:

- Le principe de Pascal, qui établit que dans un fluide incompressible en équilibre, les pressions se transmettent intégralement.
- L'incompressiblité du fluide implique que la conservation du volume doit être garantie lorsqu'il y a interaction externe et déformation de l'objet, ce qui signifie que toute déformation locale se représente sur l'ensemble de l'object. En d'autre termes, la variation de volume des éléments modifiés par l'intérieur doit être égale à la somme des variations de volume des autres éléments.

3.2 Résolution du Système

Autant la résolution statique comme l'intégration implicite nécessitent la résolution d'un ou plusieurs systèmes d'équations linéaires. Une multitude d'approches existent pour ce problème et nous avons étudié:

- Méthode de décomposition Lower-Upper pour un système linéaire.
- Méthode de Sherman-Morrison pour un système quasi-linéaire.
- Méthode de Bi-Conjugate-Gradient pour un système nonlinéaire.

3.3 Résultats Expérimentaux

On présente un ensemble de tests qui valident notre modèle pour la simulation de tissu mou. Pour cela, les expériences essaient différentes valeurs de paramètres physique du modèle et les différentes types de grande déformation. Enfin, nous comparons VDM avec le modèle classique FEM.

3.4 Bilan

Nous présentons un nouveau modèle pour la simulation de tissu mous basé sur les données comme pression et volume. Le principe de Pascal et la conservation du volume sont utilisé comme conditions aux limites. Dans le passé une résolution statique linéaire avait été proposé pour la simulation en temps-réel. Par contre ce type de résolution se limite à des petits déplacements. Pour cette raison on a ajouter une résolution statique quasi-linéaire et une résolution statique nonlinéaire qui peut prendre en compte les grands déplacements. Les résultats expérimentaux montrent que VDM produit des résultats satisfaisants.

Partie II : Collision Détection

Chapitre 4 : État de l'Art

4.1 Introduction

La détection de collision entre deux objets dans un environnement virtuel dynamique est un problème essentiel en infographie, en robotique et en géométrie

algorithmique. Il est important d'éviter l'interpénétration des objets et de simuler précisément différents phénomènes physiques afin d'améliorer le réalisme du monde virtuel. Ainsi, il est nécessaire de détecter les collisions entre les objets polyédriques. Il est également primordial de déterminer le plus précisément possible les éventuelles zones de l'objet en collision pour pouvoir fournir une simulation fidèle d'un phénomène physique. Malheureusement les algorithmes de détection de collision représentent toujours un goulot d'étranglement dans différents domaines de simulation dynamique. Ce problème a été largement étudié dans la littérature. Cependant peu d'algorithmes robustes et rapides sont connus pour répondre à des requêtes d'interaction sur objets polyédriques quelconques.

Au cours du processus de détection de collision, on effectue des tests d'intersection statique à différents instants. L'intervalle de temps entre deux tests est supposé suffisamment petit pour que l'on ne manque pas de collisions. Ces tests d'interférence statique (SIT) nous permettent de savoir si, à un instant t donné, deux polyèdres ou bien deux primitives élémentaires s'intersectent. Dans cette section, on présente la détection de collision en trois étapes:

- La phase grossière. Cette phase nous permet de sélectioner des paires d'objets pour le test de collision.
- La phase étroit. C'est une étape d'optimisation où on ne trouve pas l'entité exacte de la collision mais où on trouve la région où la collision peut avoir lieu.
- La phase exacte. Cette phase nous permet de connaître les entités exactes qui sont en collision.

4.1.1 Détection grossière

Le but de cette étape dans l'algorithme de détection de collision est d'éliminer rapidement les objets dont on a la certitude qu'elles n'intersectent aucun autre objet. Les algorithmes utilisés dans cette phase utilisent généralement une approche basée sur une décomposition spatiale.

4.1.1.1 Approche Voxels

Pour les grilles, l'espace est découpé en N cellules sur chacune des dimensions dites *voxels*. Ces grilles peuvent être régulières ou adaptatives. Pour les octrees, l'espace est découpé en huit voxels contigus et de même taille ; chacun de ces voxels sera à son tour découpé en huit sous-voxels, et ainsi de suite, les voxels vides n'étant pas divisés. La structure de données est donc un arbre dont les noeuds sont les

voxels contenant des surfaces, et les feuilles les voxels vides ou les voxels de précision suffisante. L'octree permet un gain de mémoire par rapport aux grilles grâce à la moindre occupation des zones vides, qui sont en grande majorité. Cependant la durée d'exploration est plus importante pour les octrees que pour les grilles.

4.1.1.2 Approche Arbres

Un arbre BSP est une division récursive de l'espace qui considère chaque polygone comme un plan de coupe. Ce plan est utilisé pour classer tous les objets restant comme étant soit devant, soit derrière ce plan. Autrement dit, quand on insère un polygone dans l'arbre, on le classe relativement à chaque noeud fils approprié.

4.1.1.3 Approche Sweep & Prune

Cette technique semble être la plus efficace pour les environnements dynamiques et multi-objets. Dans cet algorithme, des boîtes englobantes fixes (objets rigides) ou des boîtes englobantes dynamiques (objets déformables) sont employées pour répresenter les objets dans l'environnement virtuel. Ces boîtes englobantes sont alignées. Puis une technique de balayage est appliquée sur l'axe de x,y et z pour trouver les intervalles qui s'intersectent. La présence d'une intersection dans tous les axes indique la collision. Ces intersections sont stockées dans une liste et elles sont assorties à chaque pas de temps pour exploiter la cohérence temporelle.

4.1.2 Détection étroite

Le but de cette étape dans l'algorithme de détection de collision est de localiser rapidement le zone de collision. Pour cela, nous allons étudier les algorithmes de calcul de distances et les intersection des hiérarchies des volumes englobants.

4.1.2.1 Calcul de distances

Les algorithmes de calcul de distances commencent généralement par déterminer les plus proches **éléments caractéristiques** (sommet, arête, facette) de deux polyèdres. On peut alors calculer la distance euclidienne qui les séparent. Si cette distance est négative, les objets s'interpénètrent. Trois méthodes efficaces ont été proposées pour parvenir à ce résultat. Deux d'entre elles procèdent par un calcul incrémental de la distance entre objets polygonaux convexes obtenue à partir de la distance à

l'instant précédent, alors que la troisième méthode explore la frontière des polyèdres afin de trouver la distance minimale.

4.1.2.2 Intersection des hiérarchies des volumes englobants

Un volume englobant permet de délimiter une zone d'intérêt autour de l'objet virtuel. Par exemple, une sphère ou une boîte peuvent approximer une primitive polygonale, une soupe de polygones ou un objet complet. Le problème de détection de collision est réduit au test de superposition des deux volumes englobants, plutôt qu'aux objets directement. Quand la complexité d'un objet augmente, une hiérarchie de volumes englobants est utilisée. Il s'agit généralement d'un arbre de sphères ou bien de boîtes. Chaque volume délimite une ou plusieurs primitives géométriques. Un ensemble de ces volumes, dit *parent*, délimite l'espace de toutes les primitives géométriques de ses feuilles. Généralement, chaque volume englobant est optimisé en termes de volume, d'aire de surface, de diamètre, *etc.* dans le but d'avoir la meilleure compacité autour de la primitive encadrée. Selon les choix de conception, les feuilles d'un arbre peuvent contenir une unique primitive géométrique ou une collection de primitives géométriques.

4.1.3 Détection exacte

Dans la phase exacte, nous cherchons a connaître les entités exactes qui sont en collision. L'interaction peut être détectée si au moins deux primitives s'intersectent. Nous allons considérer différents types de paires de primitives : sphère/sphère, boîte/boîte et triangle/triangle.

4.2 Bilan

La détection de la collision entre deux enveloppes polyédriques peut demander un temps de calcul assez élevé. Il est donc important d'optimiser cette procédure pour pouvoir envisager une simulation en temps-réel. Il existent plusieurs approches pour détecter la collision ou calculer la distance entre deux enveloppes polyédriques convexes. On comparera ces approches en termes de complexité de calcul. Malheureusement pas toute l'anatomie humaine est composée d'organes convexes. Dans certains cas il est possible de diviser un objet concave en plusieurs objets convexes et appliquer les algorithmes présentes dans la section précédente. Sinon il faudra trouver explicitement les facettes en interaction; pour éviter une complexité quadratique

différentes méthodes d'optimisation existent. Pour cela on a besoin d'adapter ces algorithmes aux simulateurs médicaux où les objets déformables complexes forment la base du modèle.

Chapitre 5 : Détection de Collision pour les Simulateurs Medicaux

5.1 Introduction

Les algorithmes étudiés dans les parties précédentes s'attachent à résoudre efficacement un problème précis lié à une application donnée. Ainsi, beaucoup de méthodes requièrent des propriétés géométriques (concavité, convexité) ou physiques (rigide, déformable) sur les objets. Certaines ont besoin de pré-calculs pour être efficaces et s'appuient donc sur des structures de données intermédiaires comme par exemple un diagramme de Voronoi, une hiérarchie de boîtes (AABB ou OBB) ou de sphères englobantes. D'autres méthodes nécessitent une discrétisation de l'espace en grilles de voxels, en arbre BSP ou en octree.

Dans de nombreuses applications, un même objet peut être soumis à plusieurs types de tests d'interférence. Par exemple, considérons un polyèdre représentant un personnage évoluant dans un environnement virtuel. Si l'on s'intéresse à son déplacement dans le décor statique, alors nous nous ramenons à un problème de planification de mouvement où seul des tests d'interférence statique comme des calculs de distance seront effectués. Mais, des éléments dynamiques peuvent apparaître dans l'environnement virtuel, notre personnage sera alors soumis à des tests d'intersection avec ces éléments. En cas de collision, on peut vouloir connaître toutes les informations de contact pour être à même de répondre à cette collision. Ainsi, on a besoin de mettre en place un modèle générique de collision avec des structures de données intermédiaires adaptées aux différents tests d'interférence. Pour cela, nous allons travaillé sur les algorithmes suivantes:

- Calcul de distance entre polyèdres convexes.
- Calcul de distance entre polyèdres concaves.
- Détection de collision entre polyèdres déformables.
- Détection du volume d'interpénétration.

5.1.1 Calcul de distance entre polyèdres convexes

Afin de calculer la distance entre deux polyèdres convexes, nous avons choisi d'utiliser l'algorithme décrit dans [Sundaraj et al., 2000]. Cette implémentation repose sur l'algorithme GJK en apportant des améliorations afin d'éliminer le problème lié à la convergence dans les cas dégénérés ainsi que l'utilisation de la cohérence de frame améliorant la vitesse de la méthode.

5.1.2 Calcul de distance entre polyèdres concaves

Les algorithmes efficaces de calcul de distance entre deux objets se bornent bien souvent aux objets convexes. Gilbert [Gilbert et al., 1988] et Lin et Canny [Lin and Canny, 1991] ont présenté des algorithmes qui calculent la distance entre deux polyèdres convexes. Chacun de ces trois algorithmes itératifs trouve des paires de points, un sur chaque objet, tels que la distance entre ces points converge vers le minimum. Ces algorithmes exploitent les propriétés convexes des objets et il semble difficile de les étendre directement au cas des objets non convexes.

L'algorithme utilisé est une adaptation de l'algorithme décrit dans [Quilan, 1994] à la structure de données décrite précédemment. De plus, cette méthode a été améliorée puisque les conditions de descente dans la hiérarchie sont différentes. L'idée de l'algorithme exposé est d'utiliser les méthodes de calcul existant entre les polyèdres convexes. Pour ce faire, nous utilisons une description du polyèdre concave par un ensemble de polyèdres convexes. Nous construisons donc ici, une représentation de l'objet grâce à une hiérarchie de sphères englobantes. Il convient de rappeler que dans la structure hiérarchique de boîtes (AABB) englobantes expliquée précédemment, les boîtes étaient décrites par un centre et un vecteur représentant une diagonale de l'objet. Nous pouvons donc parfaitement mettre à profit nos résultats et notre implémentation du problème de détection de collision dans ce cas. En effet, il nous suffit de décrire nos sphères par un centre et un rayon. L'intérêt d'une hiérarchie de sphères et non de boîtes est de minimiser le nombre d'opérations et donc le temps de calcul de la distance entre deux éléments de la structure hiérarchique englobant chacun des polyèdres concaves.

5.1.3 Détection de collision entre polyèdres déformables

Nous avons choisi d'utiliser une hiérarchie de boîtes AABB englobantes. Sa construction se fait par une méthode bas-haut, dans laquelle, on regroupe dans une paire deux boîtes minimisant le volume de leur boîte mère. Il en résulte une

hiérarchie compacte autour de la surface de l'objet.

A chaque pas de temps, on reconstruit les boîtes de toutes les feuilles de l'arbre dont au moins un des points de la facette associée subi une déformation. Puis, on propage ces changements vers les boîtes ascendantes en effectuant une remontée niveau par niveau dans la hiérarchie, on met à jour toute boîte dont au moins l'une des feuilles a été mise à jour. Notons qu'une boîte ne peut être reconstruite que si ses deux boîtes filles sont traitées.

Une fois que les mises à jour éventuelles dans la hiérarchie sont effectuées pour les deux objets, le processus de détection de collision commence. Lors de la descente dans la hiérarchie, on utilise le test de l'axe séparateur [Gottschalk et al., 1996] afin de déterminer les boîtes en intersection. Si deux boîtes sont en collision, on vérifie si leurs boîtes filles respectives sont deux à deux en collision. Si on est arrivé jusqu'aux feuilles de l'arbre, on teste la possible collision entre les faces du polyèdres qu'elles englobent. On utilise pour cela l'algorithme [Moller, 1997]. A l'issue de ce procédé, on obtient dans le cas d'intersection entre les objets, le contour de collision de chaque polyèdre, *i.e.* la liste des paires de facettes deux à deux en collision.

5.1.4 Détection du volume d'interpénétration

Il peut résulter de la collision des deux polyèdres, une interpénétration fictive des deux objets. Ainsi, on doit déterminer le volume de contact de chacun des objets afin de pouvoir calculer la force résultant de la collision, et en déduire les déformations locales des objets. Afin de déterminer le volume d'interpénétration, nous construisons deux contours qui sont à l'extérieur et à l'intérieur du contour de collision. Enfin, nous cherchons tous les éléments en contact. Nous utilisons pour cela l'algorithme [Sundaraj and Laugier, 2000].

5.2 Bilan

Nous avons présenté des aspects importants de la modélisation d'interactions. Nous avons proposé un modèle de représentation hiérarchique de l'objet permettant de répondre rapidement et précisément aux différentes requêtes d'interaction. Nous avons proposé un algorithme de calcul de distance entre objets concaves ainsi qu'un algorithme de détection de collision et de localisation de contact dans le cas d'objets déformables. Ces deux méthodes sont à la fois rapide et numériquement stable.

Ces algorithmes ont été implémentés dans une librairie de détection de collision

permettant :

- de calculer les distances entre des objets convexes;
- de calculer les distances entre des objets concaves;
- de détecter les collisions entre des objets déformables;
- de fournir les éventuels volumes d'interpénétration.

Cette librairie est décrite et téléchargeable à l'url suivante: http://www.inrialpes.fr/sharp/coldetection. Notre approche permet donc une détection de collision plus rapide et plus stable. Elle s'applique à des objets polyédriques sans nécessité de propriétés géométriques ou physiques particulières puisqu'ils peuvent être convexes, concaves, rigides ou déformables.

Partie III : Applications en Réalité Virtuelle

Chapitre 6 : Simulateurs Médicaux

Dans ce chapitre, nous appliquons nos résultats dans le cadre d'un simulateur échographique de la cuisse humaine et d'une simulateur de chirurgie arthroscopique du LCA (ligament croisé antérieur du genou).

6.1 Simulateur Échographique

L'échographie est largement utilisée dans le milieu médical comme un moyen pour diagnostiquer différentes pathologies de façon non-invasive et peu coûteuse. Un exemple est la détection de thromboses veineuses. Le désavantage de cet examen est qu'il est difficile d'apprendre à diagnostiquer correctement les pathologies, car cela demande l'expérience acquise sur un nombre considérable de patients. Pour cette raison nous proposons le développent d'un simulateur échographique.

Nous avons décrire les étapes suivantes pour ce simulateur:

- Modélisation de la cuisse humaine avec VDM.
- Résolution quasi-linéaire du systeme.
- Interaction - détection de collision et retour de force.
- Résultats expérimentaux.

6.2 Simulateur Arthroscopique

Nous avons commencé à développer un simulateur d'anthroscopie en collaboration avec la societé Aesculap. Le but du simulateur est d'aider le chirugien pendant l'opération de remplacement du ligament LCA. Le problème consiste à trouver le placement pour le greffon en respectant des contraintes géométriques et physiques. Afin de résoudre le problème physique du placement du ligament, nous avons modélisé le greffon du LCA en utilisant le modèle VDM. Cela nous permet de trouver, interactivement, la déformation du greffon en donnant des informations, *a priori*, sur le placement du greffon.

Nous avons décrire les étapes suivantes pour ce simulateur:

- Modélisation du ligament LCA avec VDM.
- Résolution nonlinéaire du systeme.
- Interaction - détection de collision et distribution de contrainte.
- Résultats expérimentaux.

Chapitre 7 : Conclusion

Nous avons examiné différentes approches de la modélisation de tissus mous. En particulier nous avons comparé l'utilisation des différents modèles physiques existants et les différentes méthodes de résolution numérique associées aux objets déformables. Nous avons constaté que les comportements comme l'incompressibilité ne sont pas faciles à modéliser. C'est pourquoi nous proposons ensuite un modèle développé pour la simulation de tissu biologique, en présentant successivement les aspects liés à la formulation du modèle, à la résolution du modèle, et au traitement des interactions physiques. Ce modèle, basé sur l'utilisation du principe de Pascal, permet de modéliser de manière relativement satisfaisante des corps biologiques, tout en permettant une simulation interactive.

Pour la détection de collision, nous avons examiné différents algorithmes existants, ainsi que la difficulté d'adapter ces algorithmes aux simulateurs médicaux où les objets déformables complexes forment la base du modèle. Nous avons ensuite proposé les algorithmes développés pour traiter ce problème dans le cadre des simulateurs médicaux. Ces algorithmes présentent des caractéristiques de robustesse numérique et d'efficacité supérieures à l'existant, et permettent de traiter des corps déformables.

Enfin, nous avons appliqué ces résultats dans le cadre d'un simulateur échographique de la cuisse humaine et d'une simulateur de chirurgie arthroscopique du LCA (ligament croisé antérieur du genou).

Chapter 1

Introduction

1.1 Context and Motivations

Context. The scope of this thesis is within the framework of developing medical simulators. Medical simulators like flight simulators are meant for training and assisting the operator. But the current form of learning among novice physicians is far from this. History claims that the apprenticeship of basic practices are taught using books describing surgical procedures and techniques. This is enhanced by using simple training equipment and if lucky, by watching or participating in operations. This type of exchange of knowledge is commonly referred to as master apprentice.

Today, the above scenario hasn't changed much. In many countries, graduates from medical schools, having obtained preliminary knowledge using textbooks, are obliged to follow a three year in-house residency at a reputable medical institution to earn their title as a certified physician. During this residency period, there are many disciplines where procedures are learned by actually watching certified doctors performing on patients. They then repeat, by doing what they have seen, on the patient that eventually walks through their office with the risk of making mistakes owing to inexperience.

At this point we might be tempted to ask ourselves why have we not done anything to avoid this situation. Why don't we have a medical simulator that has its purpose like a flying simulator; to train novice surgeons adequately so that no mistakes happen during actual diagnosis. The answer to this question lies in the crucial difference between medical simulators and flight simulators: medical simulation requires that the user **interact** with the simulated environment, while flying simulation requires that the pilot **react** to the simulated environment. It is this crucial difference, along with the computational power and algorithmic complexity that

entails it, that has created a significant lag in the development of medical simulators.

Motivations. When we talk about developing medical simulators, the main question that is posed to us is *"Why do we need medical simulation?"* The first and foremost reason is quiet obvious; we as patients want to be cared for, not to be used as a tool for teaching or learning. Formally speaking, it becomes a question of ethics. The patient who has paid to be cared for, is very often unaware that he or she is being used for other purposes. Currently the law allows this, because all forms of teaching requires supervision under certified physicians. But the hard fact remains the same, damage to the human body is normally irreversible or not fully recoverable. Simulation provides a way of avoiding this undesirable situation. Novice physicians or surgeons can train adequately to learn a specialty before coming in contact with patients.

In another question of ethics, we realize that in some countries, experimenting with animals and cadavers are prohibited or will be so in the future. For example, the European Parliament voted on the 11th of June 2002 to ban cosmetic testing on animals. Such a move could as well extend to cadaveric experiments. On another note, in some cultures, the local religion and belief, does not allow any form of intervention on a cadaver. In both these cases, medical simulators have the potential of becoming a platform for such experiments.

The costs incurred during the training of novice physicians is another important factor. According to the American Association of Medical Colleges, the median annual cost for training a resident is around 76,470 USD. The interesting part of this report is that, in most developed countries, these expenses are siphoned off to the taxpayers by respective governments in the form of some special health program that is tabulated in the annual federal budget. In other words, the common layperson is going to be burdened with higher taxes in the future. Yet, we cannot ignore our responsibility to train these residents, because for all we know, we might be one of those patients that require such care one day.

1.2 Description of the Problem

There are several key problems in the development of a medical simulator. Within this context, we are interested in physical models for soft tissue simulation and algorithms for collision detection in virtual environments. These require numerical models in which all computation can be performed fast enough to sustain interactive-time performance.

Interactive-time performance becomes a critical aspect in the framework of surgical simulation, surgical training and surgical assistance. Interactive-time means that visual and haptic feedback can be produced at the correct frequency such that the user does not feel any discomfort. Normally, this means that the visual feedback needs to be at least 20Hz-30Hz while the haptic feedback around 400Hz-1KHz.

The rate at which a simulation can take place could be said to be the core problem. This is because the state of the art for interactive soft tissue simulation is not sufficiently advanced. Models that are used for simulation must be physically realistic and possess a relatively low computational complexity. It has been shown that physically based models have an advantage over previous computation animation techniques. Modeling soft tissue using physically based models has been carried out by Delingette [Delingette et al., 1999], Bro-Nielsen [Bro-Nielsen and Cotin, 1996], Aulignac [Aulignac et al., 1999], Meseure [Meseure and Chaillou, 2000], [Meseure et al., 2003] and Laugier [Laugier et al., 2003]. A description of some of these methods are presented later in this thesis.

Figure 1.1: *Computer assisted surgery and medical robotics for MIS procedures. Here, a training session for surgeons is conducted on a pig cadaver.*

Another major problem of medical simulators is interaction within a virtual environment. It is well known that collision detection is a major bottleneck in many applications that require simulation. The problem becomes more complex with virtual organs as they are concave and deformable. Although the current performance of computers have significantly improved, there is much to be done to achieve large-scale interactive-time collision detection; the development of new algorithms is still mandatory. Some important work based on collision detection of rigid convex and rigid concave polyhedra can be found in Gilbert [Gilbert et al., 1988], Moore [Moore and Wilhelms, 1988], Lin [Lin and Canny, 1991], Cameron [Cameron, 1997], Mirtich [Mirtich and Canny, 1995], [Mirtich, 1998], Derek

[Derek and Gupta, 1996] and Lombardo [Lombardo et al., 1999]. We will treat this aspect in more detail in a chapter of this thesis.

1.3 Goals and Contribution

- **Efficient and Realistic Computational Models.** We mentioned in the previous section that the state of the art for interactive soft tissue simulation is not adequately advanced. Numerical models are still topology dependent and computationally expensive. Hence, the first aim of this thesis is to explore the possibilities of finding an alternative numerical model that is physically realistic and computationally cheap. This alternative model must at least fulfill the following requirements:

  - **Physically Realistic.** The model should be derived from a physical observation. Assumptions which lead to simplification of the model will be allowed but must be justified. This proposed model must produce a behavior that is similar to soft tissue.
  - **Efficient.** The proposed model must have a low computational complexity such that global deformation due to an externally applied load can be obtained in real-time.

- **Efficient and Robust Collision Detection.** Another important goal of this thesis is the implementation of various algorithms for collision detection in medical simulators. The aim here is to use a single underlying data structure such that all intersecting primitives can be obtained in real-time. Our requirements for these algorithms would be the following:

  - **Efficient.** The proposed methods must have low algorithmic complexity. This will allow the use of such methods even in large-scale environments.
  - **Robust and Optimized.** The proposed combination of algorithms must support rigid, deformable, convex and concave polyhedral objects. Furthermore, these algorithms must use an underlying data structure that is invariant to the type of interference query. This will optimize memory usage.

- **Application to Medical Simulators.** A third goal of this thesis is to implement and test our findings in a medical simulator. This application will be a prototype for future research. The aim here is to try and integrate our findings in a single application such that interactivity and realism is not compromised.

Contribution. In this thesis, we have contributed the following:

- We have explored the possibility of finding an alternative physical model for interactive-time applications. We propose a physical model that is based on bulk variables. This physical model is in general one order of magnitude lower in complexity with respect to the classical finite element method. Hence we believe that this model is an interesting alternative.
- We have also written a collision detection library for deformable objects. This library contains the proposed algorithms for medical simulators. It also has the aim of being flexible and efficient. Optimal algorithms and data structures have been used to ensure the interactive-time compliance is satisfied at all times. This library is also independent of the 3D rendering module and operating system platform.
- We have also contributed in terms of software development in this thesis. Two prototypes medical simulators have been used as case studies; an echographic thigh exam simulator for the human thigh and an arthroscopy knee reconstruction simulator for the replacement of torn ligaments. In these developments, we have contributed in terms of physical models and collision detection for interactive-time applications.
- A through state of the art in various fields have been carried out. We have summarized them in two chapters; soft tissue modeling and collision detection.

1.4 Thesis Outline

The organization of this document is as follows; in the first part, we focus on soft tissue modeling. We begin in chapter 2 by presenting the basic ideas and formulations of various physical models used for soft tissue simulation. Numerical resolution methods used to solve the governing differential equations of these models are also presented. The complexity of the resolution method depends on whether the system is dynamic or static, linear or nonlinear. We end this chapter with a summary on some important aspects regarding the implementation of these models for interactive-time applications.

In chapter 3 of the first part, we present a new physical model which we call the **Volume Distribution Method (VDM)** for soft tissue simulation where volume conservation is of paramount importance. This physical model is derived using bulk variables like pressure and volume. Pascal's principle and volume conservation is used as boundary conditions in this model. The fact that this model is of one order

of magnitude lower than a classical model like the finite element method, makes it an interesting alternative for soft tissue simulation in interactive-time applications. We present the characteristics of this model and also discuss some possible numerical resolution methods.

The second part of this thesis is devoted to collision detection in virtual environments. Chapter 4 gives the relevant state of the art in this field. Most of these algorithms have been tailored for rigid objects. Since we do not assume any *a priori* information regarding the geometry or the dynamics of the object, the algorithms presented in this chapter fall into the static interference test (SIT) group. We end this chapter with a summary on some important aspects regarding the implementation of these algorithms for deformable objects.

Following this, in chapter 5, we present the required algorithms for collision detection in medical simulators. Our main concern is deformable objects. To improve efficiency and optimize hardware resources, a single underlying data structure for various interference queries is preferred. Within this context, we present our results. Where possible, we have improved efficiency, robustness and optimization. These algorithms have been integrated in a library for collision detection called **ColDetect**. ColDetect has been coded such that it is independent of the 3D rendering engine and the operating system platform. To the best of our knowledge, this library is the first of its kind for deformable objects.

Part three of this thesis is dedicated to the application of our findings in virtual reality. We are particularly interested in interactive-time medical simulators used for training purposes or assisting surgeons during operations. We present two case studies in chapter 6; an echographic thigh exam simulator and an arthroscopy knee reconstruction simulator. The thigh and the knee ligaments contain a high concentration of blood. Thus volume conservation can be assumed for these soft tissue. Hence, the VDM model is found suitable for these applications. Within this context, we are interested in global deformations, collision detection and force feedback. These must be computed and executed in interactive-time.

Finally in chapter 7, we summarize the findings of this thesis and discuss the future directions of our research.

Part I

Modeling Soft Tissue

Chapter 2

State of the Art

2.1 Introduction

Human tissue is deformable. Hence, modeling this behavior is essential for applications that involve soft tissue simulation. In a medical simulator for example, human tissue is generally represented by a geometrical model and a physical model. Often, a combination of the two is referred to as a numerical model. Surgical simulation requires that the numerical model be subjected to real-time interactive deformation and haptic feedback under different gestures such as palpitate, twist, drag, cut, suture, stapling, *etc.* So accurate models need to be designed to realize the consistency between them. In other words, we need to model the physical nature of soft tissue by using mathematical equations so that consistent deformation and haptic feedback can be provided interactively to the user.

Soft tissue, being complex physically and geometrically, is often divided into a set of smaller *elements* to facilitate analysis. Over the past fifteen years or so, many models that are based on the concept of discrete elements have been proposed for soft tissue; the more prominent ones being mass-spring networks (MSN), finite element method (FEM), method of finite spheres (MFS), elasticity theory method (ETM), tensor-mass model (TMM), hybrid elasticity model (HEM), boundary element method (BEM) and recently the long element method (LEM). The main concerns regarding these physical models are:

- **Interactive-Time.** The first concern is the effective modeling of soft tissue to achieve an interactive-time surgical simulation. As mentioned above, up to now, several models have been suggested, but none of them has been satisfactory from the simulation point of view as yet. Since the simulated object itself is very complicated and computation resources are limited, it is natural

to consider a trade-off between physical accuracy and computation efficiency. A compromise can be done by laying more emphasis on the areas of interest, for example, linear elasticity, local deformation or volume conservation. So this idea has to be implemented adaptively. This adaptation scheme has been studied for each kind of model and still a lot of research needs to be done.

- **Numerical Stability.** A second concern regarding soft tissue models is the numerical resolution scheme applied to solve these systems. Currently we find several methods; linear static, nonlinear static, linear dynamic and nonlinear dynamic, each being applied depending on the application and interactive-time requirements. For example, simulating cutting and tearing generally requires a dynamic model to accurately capture the viscoelastic properties of soft tissue when topology changes. On the other hand, simulating large deformations or stress-relaxation may only require at most a nonlinear static model owing to the well-damped nature of soft tissue. In either case, the main issue of interest is always stability and rapidity of the chosen scheme, during the entire history of load application.
- **Realism.** Another concern regarding soft tissue models is realism. This corresponds to identifying the physical parameters of a model such that the behavior of the model is close to reality. This part is the most difficult as it requires a lot of expertise and experimentation. Expertise, usually from medical professionals, is required because soft tissue needs to be *alive* during experimentation so that the results obtained from them are valid and accurate.

This chapter aims to present the essential concepts of these physical models, highlighting advantages and disadvantages. We begin with the description of these models and we give some important examples of applications of these models. We also look into the numerical resolution methods that have been used to solve these systems. We end this chapter by stressing the main drawbacks that we aim to solve which will be our main motivation to propose an alternative model for soft tissue simulation.

2.2 Computation Models

2.2.1 Mass-Spring Network (MSN)

The method of using mass-spring networks is a physically based technique that has been used widely and effectively to model soft tissue. This physical model consists of a lattice structure of point masses connected by elastic links. The mass-

spring network is mapped onto the geometrical mesh, *i.e.* the masses are the vertices and the springs are the edges of the mesh, to obtain a simulation model.

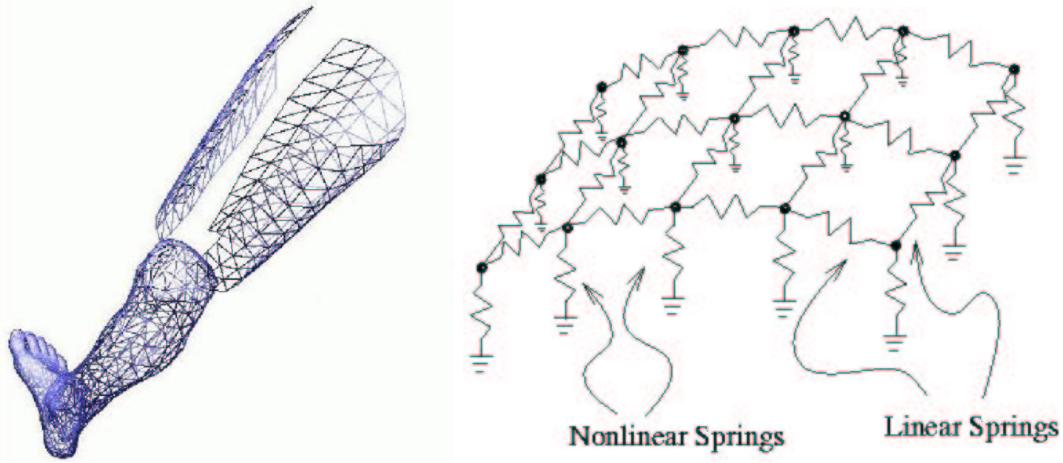


Figure 2.1: *A virtual human thigh mesh and the mapping of the mass-spring physical model onto the geometrical model. Taken from [*Aulignac, *2001].*

This mass-spring network is then used to discretize the equations of motion. For example, the link connecting pairs of nodes, allows the local description of the elastic properties and consequently the displacement of the tissue. Any change of length relative to the resting length of the spring produces an internal force between the two connecting nodes. These internal forces are often linear but nonlinear springs can also be used to model tissues of human organs that exhibit inelastic behavior.

In a dynamic system, Newton's second law governs the motion of each point mass in the lattice:

$$M_i\ddot{\boldsymbol{U}}_i + D_i\dot{\boldsymbol{U}}_i + \sum_j \boldsymbol{F}_i^{int} = \sum \boldsymbol{R}_i^{ext} \tag{2.1}$$

where for each node i, $\boldsymbol{U}_i$ is the position, $\dot{\boldsymbol{U}}_i$ and $\ddot{\boldsymbol{U}}_i$ are it's velocity and acceleration, M_i is the lumped mass, D_i is the damping coefficient to model viscosity, $\boldsymbol{R}_i^{ext}$ is the total externally applied load vector (*e.g.* gravity or user exerted forces) and $\boldsymbol{F}_i^{int}$ is the internal force exerted by a neighboring node j to which node i is connected to by a link (the sum $\sum$ is taken over all such nodes j). Generally, this internal force is the viscoelastic response of the spring connectors and is given by :

$$\boldsymbol{F}_i^{int} = (\ \lambda\Delta d + \mu\dot{d}\ )\boldsymbol{k} \tag{2.2}$$

where λ is the coefficient of rigidity of a spring connector, μ is its damping coefficient, Δd and $\dot{d}$ are the relative variation of distance and speed between the two connected nodes and $\boldsymbol{k}$ is the unit vector joining these two nodes. Note that if we let the first

two terms on the left-hand side of equation 2.1 equal to zero, the dynamic equation is transformed into a static equation and the corresponding system becomes static.

Based on the equation of motion for each point, we can obtain the motion for the entire lattice structure in matrix form:

$$\boldsymbol{M}\ddot{\boldsymbol{U}} + \boldsymbol{D}\dot{\boldsymbol{U}} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{R} \tag{2.3}$$

where $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{K}$ are the $3\aleph \times 3\aleph$ mass matrix ($\aleph$ is the number of nodes), damping matrix and state matrix respectively. Note that $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{K}$ are diagonal and banded due to mass being lumped at the nodes. The above second-order system can be converted to a first-order system for the convenience of analysis or integration. Using $\boldsymbol{V} = \dot{\boldsymbol{U}}$, we have:

$$\dot{\boldsymbol{V}} = -\boldsymbol{M}^{-1}\boldsymbol{D}\boldsymbol{V} - \boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{U} - \boldsymbol{M}^{-1}\boldsymbol{R} \tag{2.4}$$

The mass-spring network is a simple physical model with a solid mathematical foundation and well-understood dynamics. Its computational burden is relatively small [Aulignac, 2001] and is thus suitable for interactive-time applications. Since the mass-spring network has a simple structure, many operations like large deformations and topology modifications can be simulated easily. Furthermore, as interactions in this model are local between nodes, parallel computations are possible. Hence, it is common that we find this model in many applications involving soft tissue.

Mass-spring networks has been widely used in 2D and 3D facial static and dynamic animation [Waters, 1987], [Terzopoulos and Waters, 1990]. It also has been used for cloth simulation, video games and animation movies. Several methods have been suggested to avoid numerical instability [Baraff and Witkin, 1992], [Baraff and Witkin, 1998], [Desbrun et al., 1999]. A lot of research work has also been done on the mass-spring network to improve various aspects like adaptive refinement of the parameters [Hutchinson et al., 1996] and controlling the isotropy or anisotropy of the material being simulated [Bourguignon and Cani, 2000]. Recently, Brown and Montgomery [Brown and Montgomery, 2001] developed a simple but efficient algorithm based on the mass-spring model for microsurgery simulation. This algorithm took advantage of the locality of the deformations to reduce calculations by using a *wave-propagation* technique that automatically halts computation when deformations become insignificant. Using this algorithm, they achieved an updating frequency of 30Hz for the deformations in a suturing vessel surgery, which is compatible with interactive-time graphic animation.

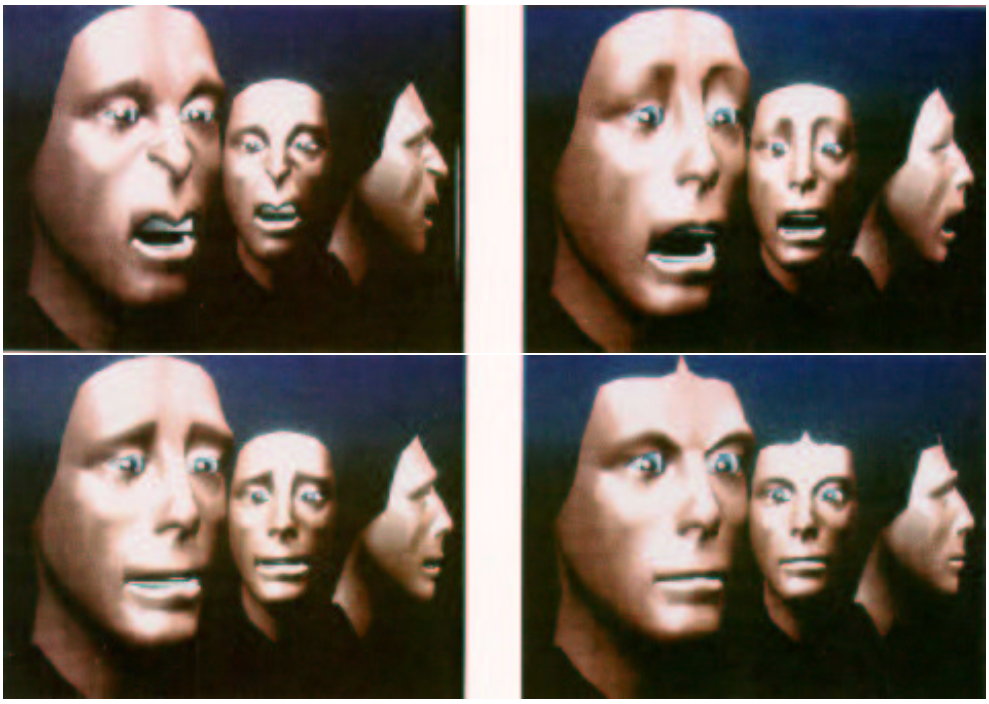


Figure 2.2: *Facial (skin) modeling and animation by K. Waters using mass-spring models. Taken from [*Terzopoulos and Waters, 1990*].*

Figure 2.3: *Real-time 2D topology modification of a mass-spring network used to model the membrane of a virtual human liver. Taken from [*Boux-de-Casson, *2000*].

Unfortunately, this physical model has some drawbacks. When representing a volume using binary connectors, the model can lead to several problems. Certain constraints like volume conservation are not easily expressed in the model. Of course, more springs will improve connectivity and thus produce a better approximation of the volume. Thus, a volumetric object could perhaps be accurately modeled by an infinite amount of particles and springs, but this is clearly not an option computationally speaking. To remedy this problem, it has been proposed to add cross springs, thereby connecting opposing corners. However, this implies that the physical behavior of the object is intrinsically dependent on the connectivity of the springs. When aiming for physical realism, this is clearly a handicap. Alternatively [Deguet et al., 1998b] proposed the use of angular and torsion springs, but this again is another form of topological dependency. Also, proper values for the constants of the mass-spring network are not easy to specify and the user's choice remains a black art.

2.2.2 Elasticity Theory Method (ETM)

ETM is based on Hooke's law, which relates the stress tensor and the strain tensor, and it uses a discrete approximation of derivative operators on irregular sample points [Debunne et al., 1999]. It allows space-time adaptation to distribute computation resources in an efficient way and ensures numerical stability.

In this method, given a geometrical mesh, we spray arbitrarily some sampling points inside. According to the theory of elasticity, for every sampling point, we have:

$$\rho\,\ddot{\boldsymbol{U}}_i = \mu\,(\nabla^2 \boldsymbol{U}_i) + (\lambda+\mu)\Big\{\nabla(\nabla\cdot\boldsymbol{U}_i)\Big\} \tag{2.5}$$

where λ and μ are Lamé coefficients characterizing the stiffness of the material, ρ is the material density, $\ddot{\boldsymbol{U}}_i$ is the acceleration of a point and $\boldsymbol{U}$ is its displacement. In this method, a scale-dependent umbrella operator is used to approximate the *Laplacian* operator:

$$\nabla^2 \boldsymbol{U}_i = \frac{2}{\sum_j |\boldsymbol{L}_{ij}|}\sum_j \frac{\boldsymbol{U}_j - \boldsymbol{U}_i}{|\boldsymbol{L}_{ij}|} \tag{2.6}$$

where $|\boldsymbol{L}_{ij}|$ is the distance between sample point i and corresponding neighboring points j. To provide a stable pair of operators for simulation, the gradient-of-divergence operator is approximated in the following way:

$$\nabla(\nabla\cdot\boldsymbol{U}_i) = \frac{2}{\sum_j |\boldsymbol{L}_{ij}|}\sum_j \frac{\Big\{(\boldsymbol{U}_j - \boldsymbol{U}_i)\cdot\widetilde{\boldsymbol{L}}_{ij}\Big\}\widetilde{\boldsymbol{L}}_{ij}}{|\boldsymbol{L}_{ij}|} \tag{2.7}$$

where $\widetilde{\boldsymbol{L}}_{ij}$ is the normalized vector of $\boldsymbol{L}_{ij}$.

The implementation of this method is straight forward just as in the mass-spring model. The recursive process for simulation is listed as follows:

- Calculate the Laplacian and gradient-of-divergence operator;
- Calculate the acceleration of each sample point using equation 2.5;
- Integrate the acceleration over a time step Δt to update positions and velocities.

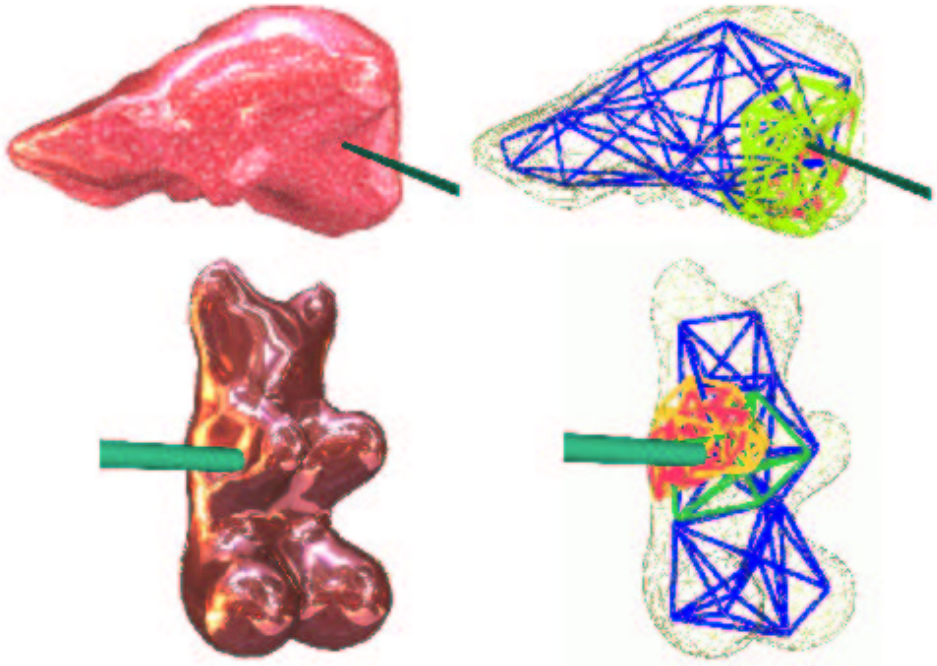


Figure 2.4: *A wide range of applications using ETM has been proposed by [Debunne et al., 1999]. Virtual surgery simulators like laparoscopic operations and immersive simulation of toys.*

This method uses both space and time adaptation to concentrate computation where and when required. The space refinement criterion is:

$$h^2|\nabla^2 \boldsymbol{U}| > \epsilon_{max} \tag{2.8}$$

And the simplification criterion is:

$$h^2|\nabla^2 \boldsymbol{U}| < \epsilon_{min} \tag{2.9}$$

where h represents the shortest distance between a particle and its neighbors. ϵ_{max} and ϵ_{max} are the refinement threshold and simplification threshold respectively.

The time adaptation is constrained by the following two inequalities:

$$\Delta t < \sqrt{\frac{^0\rho\ h^2}{\lambda + 2\mu}} \tag{2.10}$$

$$|\ddot{\boldsymbol{U}}\Delta t| < v_{max} \tag{2.11}$$

where $^0\rho$ is the material's rest density and v_{max} is the threshold for velocity changes. This method also uses internal damping to add realism and ensure numerical stability like in the mass-spring model. In [Debunne et al., 1999], the authors have implemented a system and achieved simulation at a rate of 30Hz without haptic feedback. But in [Debunne et al., 2001], force feedback was added to the simulation system. They showed that for a system of a few hundred sampling points, real-time interaction with visual and haptic feedback can be achieved.

2.2.3 Finite Element Method (FEM)

Finite element method [Bathe, 1996] is the most accurate method for solving a deformation problem under certain boundary conditions. It decomposes the object of interest into small polygonal or polyhedral elements (typically triangles in 2D and tetrahedras in 3D). In each of these small elements, the field of deformation is expressed by a polynomial interpolation as a function of the deformation values at the nodes of the element. For nonlinear analysis, which is usually the case for soft tissue simulations, deformation for example can be measured using the *Green-Lagrange* strain tensor $\boldsymbol{E}$, that is invariant to rigid body transformations. In terms of displacements, it is given by:

$$E_{ij} = \frac{1}{2}\left\{\frac{\partial \boldsymbol{u}}{\partial x_i} + \frac{\partial \boldsymbol{u}}{\partial x_j} + \frac{\partial \boldsymbol{u}}{\partial x_i}\frac{\partial \boldsymbol{u}}{\partial x_j}\right\} \tag{2.12}$$

where $\boldsymbol{x}$ is the position of the node in the undeformed configuration and $\boldsymbol{u}$ is the displacement of the node. To obtain the internal forces, a corresponding stress measure is required. The *2nd Piola-Kirchoff* stress tensor is symmetric and energetically consistent with the Green-Lagrange strain tensor. It is derived using constitutive equations or material laws which describes the relationship between stress and strain in a material.

Once appropriate stress and strain measures have been defined, the governing equation of continuum mechanics is applied to each element to obtain a set of equations with deformation values and external forces as unknowns. This formulation will be in the following static form:

$$\boldsymbol{KU} = \boldsymbol{R} \tag{2.13}$$

where $\boldsymbol{U}$ is the nodal displacement vector, $\boldsymbol{K}$ is the state matrix of the element assemblage and $\boldsymbol{R}$ is the load vector that includes the effects of element body forces and the effects of the element surface traction. If inertia forces needs to be considered for dynamic analysis, we obtain the following dynamic form:

$$\boldsymbol{M}\ddot{\boldsymbol{U}} + \boldsymbol{D}\dot{\boldsymbol{U}} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{R} \tag{2.14}$$

where $\boldsymbol{M}$ is the mass matrix and $\boldsymbol{D}$ is the damping matrix of the assemblage.

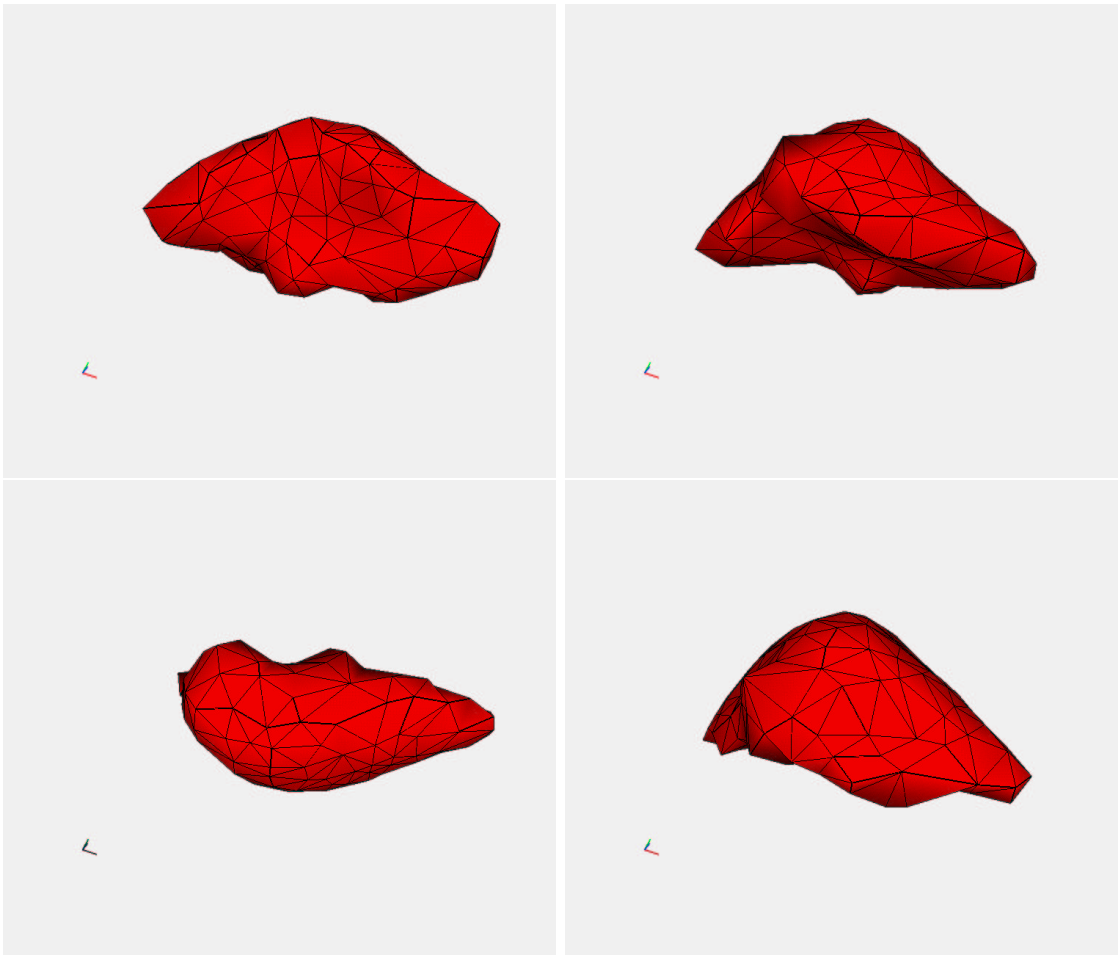


Figure 2.5: *Real-time deformation of a virtual human liver using a static FEM formulation. Taken from [*Aulignac*, 2001].*

In the finite element method, equations 2.13 and 2.14 are obtained by integration over the small polygons or polyhedra. These integrations are calculated using Gauss product rules to reduce computation. Since the interpolation functions (shape functions) are polynomials, we can obtain an accurate integration by using only a small number of integration points. Usually the integration is reduced to only a small number of multiplications and additions.

The accuracy of the results obtained through FEM depends on the type of polygon or polyhedron used and the size of it. We can increase the number of elements (h-refinement) and/or the number of nodes of each element (p-refinement) to improve accuracy. Note that the shape functions are chosen to ensure this property; deformation results will converge to the real values as we tend to use better h-refinement and/or p-refinement. To get a good trade-off between accuracy and computation speed, we need to choose meshes of proper size (corresponding to the number of elements) and a suitable number of nodes (corresponding to a type of polygon or polyhedron).

The elements used most frequently are triangles in 2D and tetrahedras in 3D. Usually several thousand of them are used to obtain a good result for a simple object under simple boundary conditions. For complicated objects, under linear analysis, we may be interested only on the parts that can be seen (the surface). It is then possible to condense the matrix equation and apply some precomputation techniques to reduce computation [Bro-Nielsen and Cotin, 1996].

On the whole, FEM has too heavy a computation burden to achieve accurate and interactive-time results. Generally, FEM is not suitable for interactive-time applications for the present CPU capacity. But if there is no topology changes, it is possible to obtain real-time deformations by using precomputation [Dimaio and Salcudean, 2002]. Nevertheless this is limited to small deformations which can be a handicap for soft tissue simulations. But recently, Mendoza and Laugier [Mendoza and Laugier, 2003] have proposed an implementation of an explicit formulation of FEM taking into account large deformations and topology changes. They managed to perform cutting on a virtual human liver by considering human behavior and limiting stress conditions.

2.2.4 Tensor-Mass Model (TMM)

The tensor-mass model (TMM) discretizes the virtual organ with conformal tetrahedras [Cotin, 1997]. Just like the mass-spring model, it also distributes the mass in the object to lumped masses on the mesh nodes i. The governing equation for the motion of the mesh nodes is also based on the Newtonian Law:

$$M_i \frac{d^2 \boldsymbol{U}_i}{dt^2} + D_i \frac{d\boldsymbol{U}_i}{dt} + \boldsymbol{F}_i^{int} = 0 \tag{2.15}$$

The difference here with respect to MSN is that $\boldsymbol{F}_i^{int}$ is obtained through the energy-based finite element method. The computation of this linear elastic force can be decomposed into four steps:

- Define the interpolation equations that give the displacement vector at a point inside tetrahedra $\mathcal{T}_k$ as a function of the four nodal displacement vectors $\boldsymbol{U}_i$.
- Express the elastic energy W_k, of a tetrahedra as a function of these four nodal displacement vectors.
- Compute the elastic force $\boldsymbol{F}^{int}_{i,\mathcal{T}_k}$ produced by tetrahedra $\mathcal{T}_k$ which is applied to node i using W_k.
- Add all the $\boldsymbol{F}^{int}_{i,\mathcal{T}_j}$ produced by the neighboring tetrahedras j connected to node i to obtain $\boldsymbol{F}^{int}_{i}$.

After the first three steps above, we can obtain the force $\boldsymbol{F}^{int}_{i,\mathcal{T}_k}$ applied on node i within each tetrahedra $\mathcal{T}_k$.

$$\boldsymbol{F}^{int}_{i,\mathcal{T}_k} = \sum_{l=0}^{3} \boldsymbol{K}^{\mathcal{T}_k}_{il}\, \boldsymbol{U}_{l,\mathcal{T}_k} \tag{2.16}$$

where l are the nodes within tetrahedra $\mathcal{T}_k$ and the set $\boldsymbol{K}^{\mathcal{T}_k}_{il}$ are the state matrices or tensors. $\boldsymbol{K}^{\mathcal{T}_k}_{il}$ can be computed as follows:

$$\boldsymbol{K}^{\mathcal{T}_k}_{il} = \frac{1}{36\mathcal{V}_{\mathcal{T}_i}}\left[\lambda_k\left(\boldsymbol{n}_{l,\mathcal{T}_k}\, \boldsymbol{n}^T_{i,\mathcal{T}_k}\right) + \mu_k\left(\boldsymbol{n}_{i,\mathcal{T}_k}\, \boldsymbol{n}^T_{l,\mathcal{T}_k}\right) + \mu_k\left(\boldsymbol{n}_{l,\mathcal{T}_k}\, \boldsymbol{n}_{i,\mathcal{T}_k}\right)\boldsymbol{I}_{3\times 3}\right] \tag{2.17}$$

where for tetrahedra $\mathcal{T}_k$, $\mathcal{V}_{\mathcal{T}_k}$ is its volume, $\boldsymbol{n}_{l,\mathcal{T}_k}$ and $\boldsymbol{n}_{i,\mathcal{T}_k}$ are the normal vectors of its surfaces and λ_k and μ_k are its Lamé coefficients.

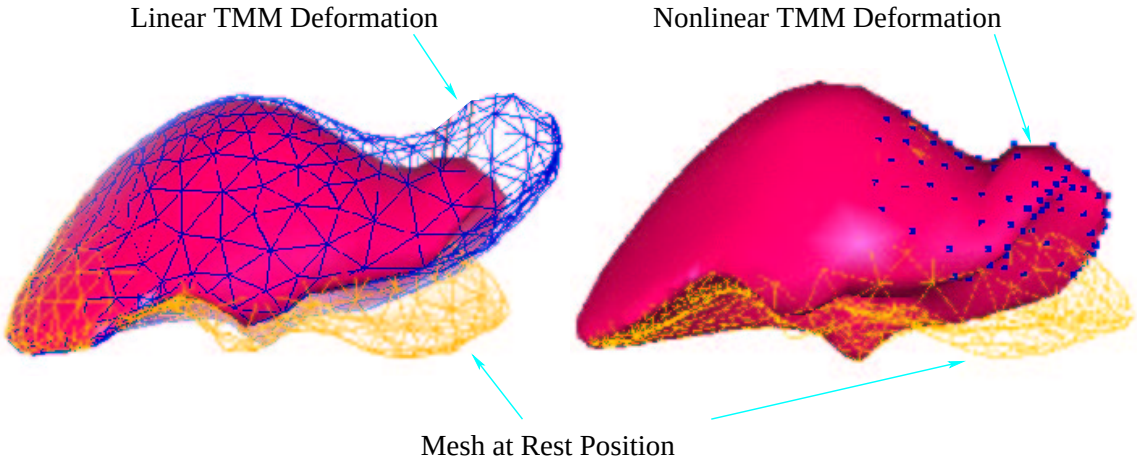


Figure 2.6: *Deformation obtained using the linear and nonlinear TMM model. The rest position of the mesh is indicated. Taken from [*Picinbono et al., 2002*].*

From these discussions, we can see that $\boldsymbol{F}^{int}_{i}$ is computed locally since it is only related to the tetrahedras connected to node i. When compared to the mass-spring model, the tensor-mass model computes force by continuum mechanics and therefore

is independent of the mesh topology. In contrast, we know that the mass-spring model is sensitive to mesh topology. Hence, when there is a topology modification procedure, the tensor-mass model can give more accurate results. The initially proposed tensor-mass model only accommodated small displacements, but recently Picinbono [Picinbono et al., 2002] made modifications to the model for large displacements by using nonlinear strain tensors and anisotropic material laws.

2.2.5 Hybrid Elasticity Model (HEM)

In the hybrid elastic model (HEM) [Cotin et al., 1999], there is a combination of a quasi-static precomputed linear elastic model and a tensor-mass model (TMM). Thus, HEM takes advantage of the good properties of the combined models. We have mentioned earlier that precomputation is only suitable for situations where there are no changes in topology. On the other hand, we also know that by using precomputation, we can achieve interactive-time performance and obtain accurate deformation results.

In this dual model coexistence structure, different models share some common boundaries. One model may provide boundary conditions for the other. Since different models are used, there may be some artifacts in the areas close to the common boundaries between different models. But if we choose two models with similar theoretical foundations, it is possible for us to reduce these artifacts such that they are negligible. As to the behavior of HEM, since the two models follow the same physical law; *Hooke's Law*, so the combination of these two models should behave exactly as a global linear elastic model. Here we note that the combination of other models are also possible for a hybrid model system [Tseng and Lin, 2000].

In [Cotin et al., 2000], the hybrid elastic model is used to simulate a *hepatectomy* surgical procedure. HEM is used as the global model in which TMM is used for the parts where there is a cut or tear operation, and a quasi-static precomputed linear elastic FEM model for the part that there is no cut or tear operations. Similar to TMM, the quasi-static precomputation model is only suitable for small deformations. The main difference between the quasi-static precomputation model and the tensor-mass model is that the displacement vector of the elements in the former is calculated by a continuous finite element method, which is more accurate, while in the latter, it is obtained through a discrete lumped mass method, which is less accurate. The artifacts close to the common boundaries of the two kinds of model are very small and are unperceivable to the human eye. So this model is a good model for specific surgical simulators meant for training cut or tear operations that are simulated by removing material. But one drawback of this model is that it

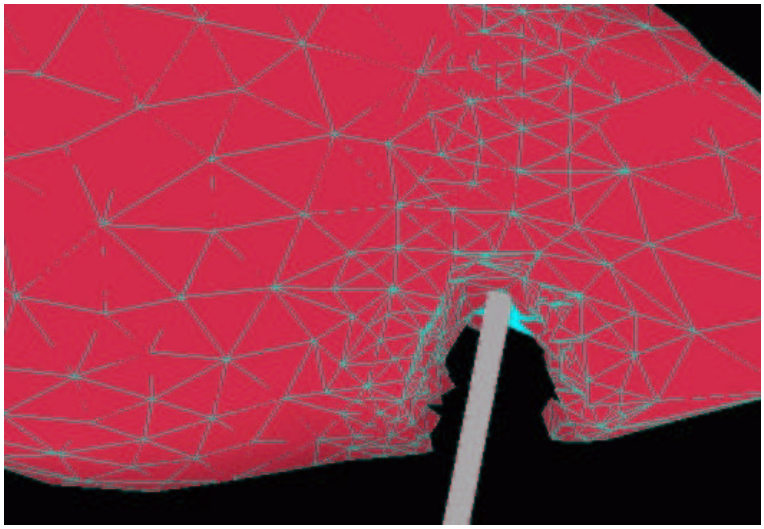


Figure 2.7: *Cotin in [Cotin et al., 1999] has applied the HEM to a virtual human liver. In the above application, a predefined cutting section is identified. Then this part is modeled using TMM and the remaining portions of the liver is modeled using linear elastic FEM. Condensation and precomputation are used to solve the linear elastic FEM parts while TMM allows resolution of systems with topology modifications.*

is only suitable for global small deformations.

2.2.6 Method of Finite Spheres (MFS)

Method of finite spheres (MFS) is a complete meshless method. It was developed by S. De and K. J. Bathe [De and Bathe, 2001] to overcome the remeshing burden in methods like FEM. This method uses a set of points instead of meshes to solve the governing equations of equilibrium.

This method has recently been extended for deformable objects. In a medical simulator for example, when a surgical tool touches the virtual soft tissue, a set of points is sprinkled locally around the tool tip and a sphere with a finite radius is built at each sprinkled point. Just as in the classical FEM method, shape functions are used to approximate the deformation fields. The difference here is that in the MFS method, we have to use rational functions instead of polynomials. Although the rational functions are carefully chosen to enhance the computation efficiency [De and Bathe, 2001], they still lead to more extensive computation in the integration part. This is due to the increase in interpolation points used in MFS as compared to the normal polynomial case. This is obviously a disadvantage for interactive-time applications. Although J. Kim *et al.* in [Kim et al., 2002a]

claimed that the method of finite spheres could produce reasonably good local deformation, comparable to FEM, it is still not so convincing that this method can give interactive-time and accurate global deformation results in complex meshes. This is because in their experiments, the number of the points used in their simulation is far too small compared to that in the finite element method (about 40 times less). For MFS to be really applicable in interactive-time applications, more progress needs to be made to reduce the computation burden.

2.2.7 Boundary Element Method (BEM)

One of the recent research work on simulating accurate deformation has been done by James and Pai using boundary elements for deformable objects [James and Pai, 1999]. They propose to use a quasi-static method to model deformations by describing how the object interacts with the environment at its boundary Γ. For example, in figure 2.8 the object Ω is subjected to two *displacement boundary conditions*; Γ_i, displacements due to user interaction and Γ_c, displacements conditioned by some fixed contact. The remaining boundary parts of the object Γ_f, are free to move.

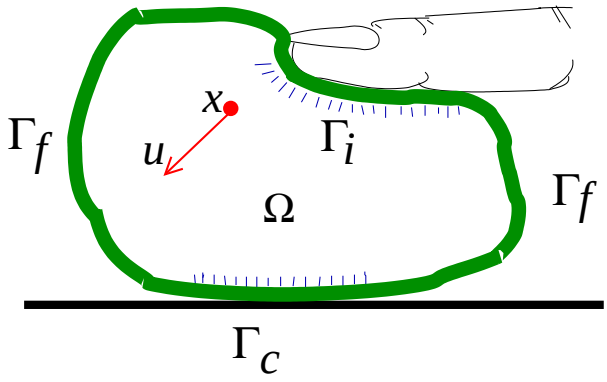


Figure 2.8: *Boundary element method notations.*

In this model, when an object is deformed, a displacement $\boldsymbol{U}$ of a point $x \in \Omega$ occurs. This deformation is based on linear elasticity governed locally by *Navier's* equation which is a generalization of Hooke's law and it can be written as:

$$\boldsymbol{N}\boldsymbol{U} + \boldsymbol{X} = 0 \tag{2.18}$$

where $\boldsymbol{N}$ is a linear second-order differential operator, and $\boldsymbol{X}$ is a term due to body forces, like gravity, acting everywhere in the object. The boundary conditions, along with Navier's equation, constitute the boundary value problem (BVP). To determine

the displacement vector $\boldsymbol{U}$, and the traction vector $\boldsymbol{T}$, on the boundary of the body, they use a boundary integral formulation of Navier's equation expressed as:

$$\int_{\Gamma} \boldsymbol{U}^{\star}(x,y)\, \boldsymbol{p}(y)\, d\Gamma(y) + \int_{\Omega} \boldsymbol{U}^{\star}(x,y)\, \boldsymbol{b}(y)\, d\Omega(y) \tag{2.19}$$

where the elements of the matrix function $\boldsymbol{U}^{\star}(x,y)$ are denoted as $[U^{\star}_{ij}]$ and they represent the displacement in direction j at a field point y. This displacement is due to a unit load applied in each of the i directions at point x.

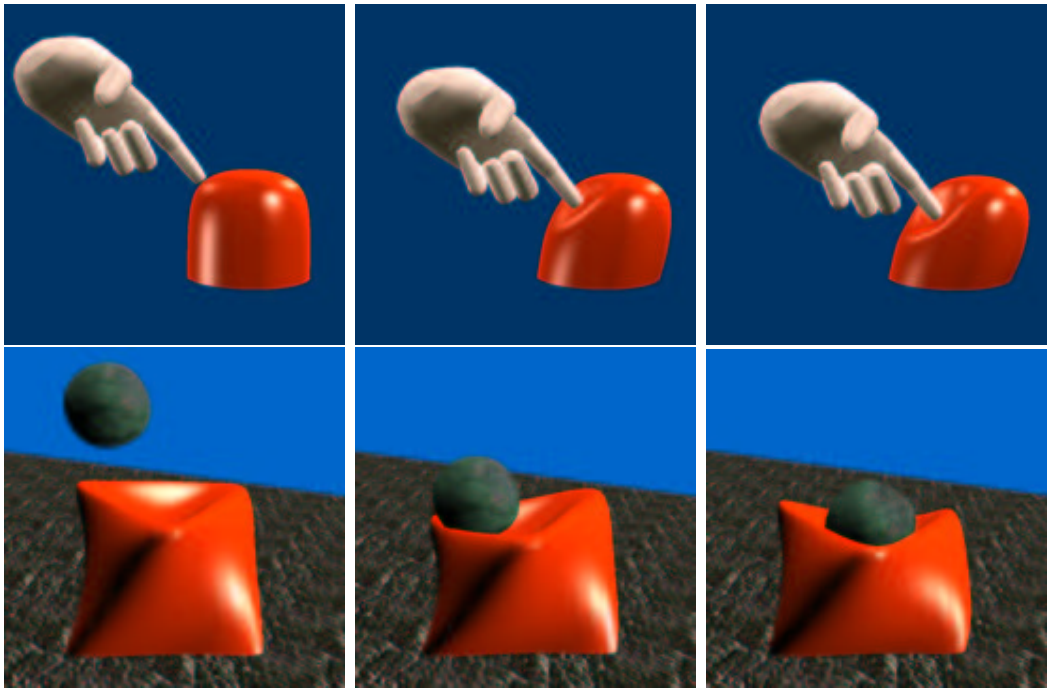


Figure 2.9: *Examples of deformations obtained using the boundary element method. Taken from [James and Pai, 1999].*

The boundary element method can be summarized in the following steps:

- Discretize the boundary Γ into a set of $\aleph$ non-overlapping elements whose degrees of freedom represent the displacements and traction that are piecewise interpolated between the element's nodal points.
- Apply the integral equation 2.19 at each of the $\aleph$ boundary nodes. This generates a system of $3\aleph$ equations involving $3\aleph$ nodal displacements and $3\aleph$ nodal traction of the form:

$$\boldsymbol{H}\boldsymbol{U} = \boldsymbol{G}\boldsymbol{T} \tag{2.20}$$

  where $\boldsymbol{H}$ and $\boldsymbol{G}$ are $3\aleph \times 3\aleph$ dense (non-sparse) matrices.

- Apply boundary conditions of the desired BVP by fixing nodal values (either displacements $\boldsymbol{U}$ or traction $\boldsymbol{T}$). The final linear system of $3\aleph$ equations is transformed to

$$\boldsymbol{KV} = \boldsymbol{Z} \tag{2.21}$$

  where the unknown boundary values are placed on the left-hand side.

The solution of this matrix system is executed using a quasi-static resolution approach. A drawback in this method is the loss of accuracy when modeling soft tissue since it is based on linear elasticity and no consideration is made to accommodate large displacements. The application of this method for calculating force feedback is explained in detail in [James and Pai, 2001] which is an advantage in this method for haptic simulation.

2.2.8 Long Element Method (LEM)

Long element method views the objects as two-dimensional distributed elements filled with an uncompressible fluid. The advantage of this method is that the number of the elements is one order of magnitude less than in a discretization based on tetrahedral or cubic elements [Costa and Balaniuk, 2001b]. In the static long element method, each element is assumed to be filled with fluid. But at the same time, each element is also assumed to obey Hooke's Law in the axial direction. Pascal's principle and the law of conservation of volume are used as boundary conditions to establish the state of equilibrium.

(a) (b) (c)

Figure 2.10: *(a) A long element. (b) Cylinderical mesh with discretized surface. (c) Cylinder represented by long elements in 1D.*

A problem with this method is the error in the discretization when the object undergoes large deformation resulting in inconsistent results and the absence of volume

conservation. A naive solution would be to discretize the object at each simulation interval but this is clearly not an efficient option, computationally speaking. So the validity of the long element method for interactive-time appliacations is doubtful although some of its assumptions are reasonable. Many aspects of this approach still remains to be explored.

2.3 Resolution Methods

In the preceding sections, we presented some of the common physical models used to represent biological tissue. They are formulated in two ways; linear and nonlinear, resulting in a matrix system of equations. In this section, we briefly look at techniques used to resolve these systems. Details about the convergence criteria and rate of convergence have been omitted because they are not the primary concern of this thesis. We recall that for both the linear and nonlinear models, a static and dynamic formulation is possible.

2.3.1 Static Systems

A static system is obtained when inertia and viscoelastic effects can be neglected. This happens when the frequency of excitation is lower than roughly one-third of the structure's natural frequency. The physical model is then characterized by the following matrix system:

$$\boldsymbol{K}\boldsymbol{U} = \boldsymbol{R} \tag{2.22}$$

where $\boldsymbol{K}$ is the constant state matrix, $\boldsymbol{U}$ is the displacement vector and $\boldsymbol{R}$ is the load vector.

2.3.1.1 Linear Static

A linear static system is obtained when the state matrix $\boldsymbol{K}$ is constant throughout the entire history of load application. Typically, this means small strains and displacements. Two methods can be used to solve this system; direct solution techniques and iterative solution methods [Press et al., 1992].

In the direct solution technique, the number of operations can be determined exactly. Preconditioning is often done to improve computational efficiency. Several techniques exist for such operations; Lower-Upper Decompositions (LUD), Cholesky Factorization (CF) *etc.* The efficiency of these algorithms depends on factors like nodal numbering and sparsity.

In large systems however, a direct solution can require a large storage space and computation time. Hence, an iterative method is more appropriate. Two important techniques for this purpose is the Gauss-Seidel method and the Conjugate-Gradient method. In both cases, the rate of convergence depends on the initial displacement vector $\boldsymbol{U}$ used and the condition number of the matrix $\boldsymbol{K}$.

2.3.1.2 Nonlinear Static

Linear analysis is only valid for small strains and displacements. Typically this excludes deformations of more then five percent or rotation. Beyond this we have to take into account nonlinearities. We can differentiate between:

- **Material Nonlinearity.** Real materials are nonlinear. The constitutive relationship between strain and stress must be taken into account.
- **Geometric Nonlinearity.** When the solid undergoes large variations in shape or rotation, the stiffness matrix will change.

Hence the state matrix, $\boldsymbol{K}$, is now a function of the displacements $\boldsymbol{U}$:

$$\boldsymbol{K}(\boldsymbol{U})\ \boldsymbol{U} = \boldsymbol{R} \tag{2.23}$$

This gives rise to a nonlinear problem since we can no longer just inverse the state matrix $\boldsymbol{K}$ to find a solution for an external force being applied. Therefore we must resort to a nonlinear solution technique.

A numerical scheme frequently used to solve this system is the Newton-Raphson iterative method [Bathe, 1996]. In this method, at each iteration, an out-of-balance load vector is calculated. This vector produces an increment in the displacement vector. Iteration is continued until either vectors becomes sufficiently small. Mathematically, the procedure is described as follows, where $i = 1, 2, 3, \ldots$ is the sequence of iteration.

$$\begin{aligned} \Delta\boldsymbol{R}^{i-1} &= \boldsymbol{R} - \boldsymbol{R}^{i-1} \\ \boldsymbol{K}^{i-1}\ \Delta\boldsymbol{U}^{i} &= \Delta\boldsymbol{R}^{i-1} \\ \boldsymbol{U}^{i} &= \boldsymbol{U}^{i-1} + \Delta\boldsymbol{U}^{i} \end{aligned} \tag{2.24}$$

An important observation in this method is the choice for updating the state matrix $\boldsymbol{K}$. Three schemes exist for doing this; the choice depending on the accuracy sought after and computation time constraint (interactiveness) [Aulignac, 2001].

2.3.2 Dynamic Systems

A dynamic system is obtained when inertia and viscoelastic effects are included. The equations of equilibrium of a dynamic system is governed by the following matrix system:

$$\boldsymbol{M}\ddot{\boldsymbol{U}} + \boldsymbol{D}\dot{\boldsymbol{U}} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{R} \tag{2.25}$$

where as usual, $\boldsymbol{M}$, $\boldsymbol{D}$, and $\boldsymbol{K}$ are the mass, damping and state matrix. $\ddot{\boldsymbol{U}}$, $\dot{\boldsymbol{U}}$, and $\boldsymbol{U}$ are the nodal acceleration, velocity and displacement matrix. $\boldsymbol{R}$ is the external load vector.

We can restate this problem as a second-order differential equation:

$$\ddot{\boldsymbol{U}} = \boldsymbol{M}^{-1}(-\boldsymbol{D}\dot{\boldsymbol{U}} - \boldsymbol{K}\boldsymbol{U} + \boldsymbol{R}\ ) \tag{2.26}$$

If the mass is lumped at the nodes, the mass matrix is diagonal and therefore easily invertible. Since matrices $\boldsymbol{D}$ and $\boldsymbol{K}$ are not constant, the differential equation is *nonlinear*. Hence, there is no analytical solution to this problem. However, a variety of numerical integration techniques exist to compute position and velocity as a function of time. The simplest method for finding the change in state with respect to time is the Newton-Euler method. It finds the next state by following the current derivative during one Δt. We can then forward integrate in time the system using an explicit or implicit method.

2.3.2.1 Explicit Newton-Euler Integration

Once the the external load vector $\boldsymbol{R}$ is known it becomes possible to evaluate the change in velocity and position of the system:

$$\begin{aligned} \dot{\boldsymbol{U}}^{t+\Delta t} &= \dot{\boldsymbol{U}}^{t} + \Delta t\ \ddot{\boldsymbol{U}}^{t} \\ \boldsymbol{U}^{t+\Delta t} &= \boldsymbol{U}^{t} + \Delta t\ \dot{\boldsymbol{U}}^{t} \end{aligned} \tag{2.27}$$

where the method is termed *explicit* because the derivative is evaluated for the *current* state. The explicit Newton-Euler integration method takes no notice of changing derivatives. It is actually a truncated Taylor series, discarding all but the first two terms. This means the Newton-Euler method is only correct if the first time derivative is constant. Otherwise, larger the time-step, the larger the error is as well. This accumulating error is observed in objects that are not highly elastic. For these objects, the ordinary differential equations are *stiff*, resulting in poor stability and requiring the numerical integrator to take very small time-steps.

2.3.2.2 Implicit Newton-Euler Integration

To get around the problem of *stiff* ordinary differential equations, an implicit Newton-Euler integration can be used. Implicit methods are different; the change in state is calculated by the derivative at the *next* state. Hence equation 2.27 may be rewritten as the two following equations:

$$\begin{aligned} \dot{\boldsymbol{U}}^{t+\Delta t} &= \dot{\boldsymbol{U}}^{t} + \Delta t\ \ddot{\boldsymbol{U}}^{t+\Delta t} \\ \boldsymbol{U}^{t+\Delta t} &= \boldsymbol{U}^{t} + \Delta t\ \dot{\boldsymbol{U}}^{t+\Delta t} \end{aligned} \tag{2.28}$$

where if we let y be any required state and $f(y)$ the derivative, then we have the following for a general linearized system:

$$f(y^t) = \lambda y^t \tag{2.29}$$

Then equation 2.28 can be restated as follows:

$$\begin{aligned} y^{t+\Delta t} &= y^t + \Delta t\ f(y^{t+\Delta t}) \\ &= y^t + \Delta t\ \lambda y^{t+\Delta t} \\ &= \frac{y^t}{1-\lambda\Delta t} \end{aligned} \tag{2.30}$$

Thus at the new state the derivative will take you back to where you came from. The system is therefore reversible in time and hence, completely stable. Hence by linearizing equation 2.25, using an iterative method for example, we can solve analytically for the position after a given time-step Δt. Other variations that concentrate on optimizing this technique are also available. A recent work in this aspect is [Hilde et al., 2001].

2.4 Summary

In this chapter, we have presented the important physical models used for soft tissue modeling. Amongst these models, FEM and TMM have the most realistic results due to their solid mathematical and physical foundation.

FEM is a continuous model, but is not a purely continuous model. The discrete component in this method lies in the meshing step, which causes the deformation field to be only continuous across the mesh boundaries. FEM is accurate only when the discontinuity of the derivative of the deformation field across the mesh boundaries is negligible.

TMM is a semi-continuous model. The interactions are calculated on a continuous base (elasticity theory). The material mass is lumped onto the vertices of

the mesh to establish the motion equations. For small deformations, it can replace mass-spring model. But it is not suitable for large deformations. Quasi-static pre-computed elastic method is actually an energy based finite element method. It is also only suitable for small deformations. Combined with the tensor-mass model, it forms a hybrid model, which can take advantage of pre-computations.

MSN is a complete discrete model in which the continuous material is discretized into lumped masses and the distributed interactions are discretized into springs. Due to the discretization of both the two aspects, mass-spring model is the simplest and easiest to implement, and it can handle all kinds of the interactions between the surgeon and the organs. However, a simple model is not necessary an efficient model. The constraints for a continuous system sometimes is not easy to be transformed into constraints for the corresponding discrete system, which means the mass-spring model is not efficient in handling some constraints, like volume conservation for example.

ETM is a complete continuous model. The key point in this elasticity theory method is the approximation of Laplacian operator and divergence of gradient operator by a umbrella operator. If this approximation were good for human organ deformations, the elasticity theory method would be an efficient model.

BEM and LEM has its advantage in haptic rendering applications because the traction vector is obtained directly from the solution of the system. Furthermore, LEM is interesting due to the presence of a reduced and sparse state matrix.

We also briefly presented MFS, a meshless method, which has recently made appearance as a soft tissue model. Essentially, MFS is a method very close to FEM. The main difference is that MFS is a meshless method and the shape functions are rational functions instead of polynomials. Unfortunately, the complexity of its resolution technique has restricted its application to non-complex simulations.

Besides MFS, all the models generally map a physical model onto a volumetric geometrical model to achieve maximum realistic simulation. Thus the size of the state matrix $\boldsymbol{K}$ is dependent on the number of nodes but sparse to some extent (depending on topology). By condensation, the size of $\boldsymbol{K}$ is reduced but becomes dense. A reduced and sparse $\boldsymbol{K}$ is of course the ideal outcome. Hence, at this point, we are motivated to try and find a new model that satisfies this criteria.

We also presented various resolution methods for the physical models. We showed that dynamic implicit nonlinear analysis requires the update and factorization of the state matrix $\boldsymbol{K}$ which can be computationally very expensive. An exception is in the dynamic explicit nonlinear analysis, and hence we find that this technique is

most favorable. But as we mentioned before, the problem here is the choice of the time-step which has to be adaptive so that numerical stability is ensured throughout the entire history of load application. This compounds to recalculating the $\boldsymbol{K}$ of the system. Thus, a static resolution seems favorable because we avoid stability problems. Furthermore, since soft tissue is well-damped, viscoelastic effects can be neglected. Thus a static nonlinear analysis should suffice. But static nonlinear analysis still requires the evaluation of $\boldsymbol{K}$ at each interval. Our immediate response to this question is that if an ideal $\boldsymbol{K}$ can be found (small and sparse), an iterative method can be used for a fast evaluation of the inverse of $\boldsymbol{K}$.

	MSN	ETM	FEM	MFS	HEM	TMM	BEM	LEM
I	★★★	★★★	★	★	★★	★	★★	★★★★
R	★	★★	★★★★	★★★	★★	★★★★	★★	★★

Table 2.1: *Comparison of various soft tissue physical models. Interactivity (I) against Realism (R).*

We end of this chapter by tabulating a comparison between these models. They are summarized in table 2.1. The main drawback in most of these models is the complexity of the models which is significant for interactive-time applications. For static simulations, this could be due to the absence of a reduced and sparse state matrix $\boldsymbol{K}$. This has been our main motivation to explore another possible soft tissue model. Within this context, we take the LEM model as our starting point. In the next chapter, we present the formulation and results of this new physical model. This model must be suitable for soft-tissue simulation in interactive-time applications.

Chapter 3

The Volume Distribution Method (VDM)

3.1 Introduction

In the previous chapter, we saw some common physical models for deformable objects. These models were carefully adapted to model deformable objects and their physical parameters tuned for a specific application. This is normally the case, because there is no absolute model that can possibly represent every real phenomenon associated with deformable objects. These models were briefly presented and their qualities analyzed and compared. FEM has been known to be most realistic, but nevertheless, it is computationally expensive and unsuitable for interactive-time applications. We finally concluded by emphasizing that for interactive-time applications, a deformable physical model that is physically realistic, numerically stable and computationally cheap is desirable.

The objective of this chapter is to present the work done on a new physical model initially developed in our research group for deformable objects that we hope will satisfy the above mentioned goals. This new model is called the **Volume Distribution Method** (VDM) which has been inspired from the Long Element Method (LEM) [Costa and Balaniuk, 2001b], [Costa and Balaniuk, 2001a], [Sundaraj et al., 2001] and [Sundaraj and Laugier, 2002]. We are interested in *deformable objects which have an elastic skin as surface and are filled with an incompressible fluid*. Soft tissue can be considered as such an object. Let us take the liver for example, it is composed of three major parts; an elastic skin called *Capsule of Glisson* as the surface, the *Parenchyma* which is the interior and is full of liquid ($\approx$ 95% blood) and a complex vascular network designated to irrigate the liver. These observations allow us to make some soft assumptions about the

behavior of the liver. For example, due to the high content of blood in the liver, it is possible to deduce that the liver is incompressible *i.e.* the liver conforms to a change in shape but not to a change in volume. The elastic capsule indicates a behavior that obeys some form of *Hook's Law*.

This chapter begins with the basic formulation of the VDM model. First, we will explain how a deformable object is transformed from a continuum to a discretized VDM object. Then, we shall formulate the equations of equilibrium for VDM. We then consider the resolution of these equations for linear (small deformation) and nonlinear (large deformation) analysis. Finally, before we conclude, we present some experimental results.

3.2 Mathematical Formulation

We begin by giving a formal description of this model. VDM is a surface based method that allows the computation of a **global** deformation vector produced by an external load vector. It only requires the surface to be discretized with the inside being transparent to the model. The interior of the object is assumed to be filled by some incompressible fluid. This fluid acts as the medium that transfers the change in energy experienced by the deformable object due to a change in state from equilibrium.

3.2.1 Notations

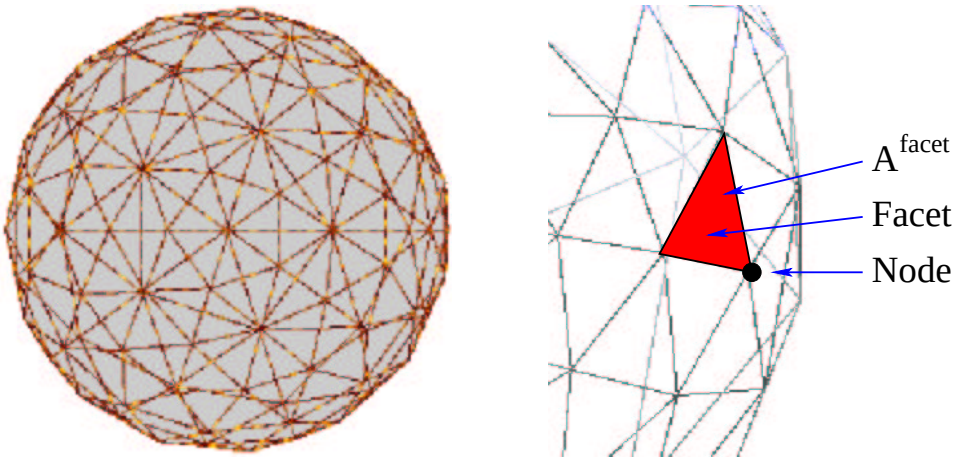


Figure 3.1: *A deformable object with the surface discretized and a zoomed view of the surface. The interior of this object is filled with some incompressible fluid.*

In this section, we describe the mathematical formulation of the VDM model [Sundaraj et al., 2003]. Consider the triangulaire surface mesh of a deformable object as shown in figure 3.2 filled by some incompressible fluid. Let us now suppose that this surface is composed of ℵ vertices, which we will from now on refer to as nodes.

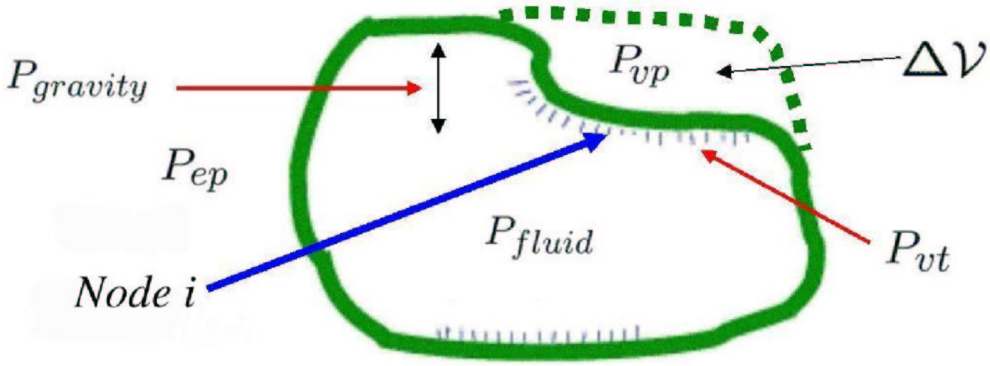


Figure 3.2: *Volume Distribution Method (VDM) notations.*

Let us now define the following notations for our deformable object. These notations (see figure 3.2) are for a node i, on the surface of our deformable object:

- Volumic Pressure, P_{vp} - The pressure experienced by a node due to the displacement of the node.
- Volumic Tension, P_{vt} - The pressure experienced by a node due to the displacement of neighboring nodes.
- Contact Pressure, P_{cp} - The pressure experienced by a node due to contact.
- Fluid Pressure, P_{fluid} - The pressure exerted by the incompressible fluid.
- Environment Pressure, P_{ep} - The pressure exerted by the surroundings of the deformable object.
- Bulk Modulus, B_i - The ratio of pressure of node i, to the fractional volume compression of node i (equivalent to normal stress).
- Connectivity Bulk Modulus, B_{ij} - The ratio of pressure of node i, to the fractional volume compression between node i and j (equivalent to shear stress).
- Area, $\mathcal{A}$ - The total surface area of the deformable object.
- Volume, $\mathcal{V}$ - The total volume of the deformable object.
- Distributed Area, $\boldsymbol{\mathcal{A}}_i$ - The area assigned to a node i.
- Distributed Volume, $\mathcal{V}_i$ - The volume assigned to a node i.

- Facet Area, $\boldsymbol{\mathcal{A}}^{facet}$ - The area of a facet.
- Displacement Vector, $\boldsymbol{\Delta L}_i$ - The displacement vector of a node i.

3.2.2 Distributed Area and Volume

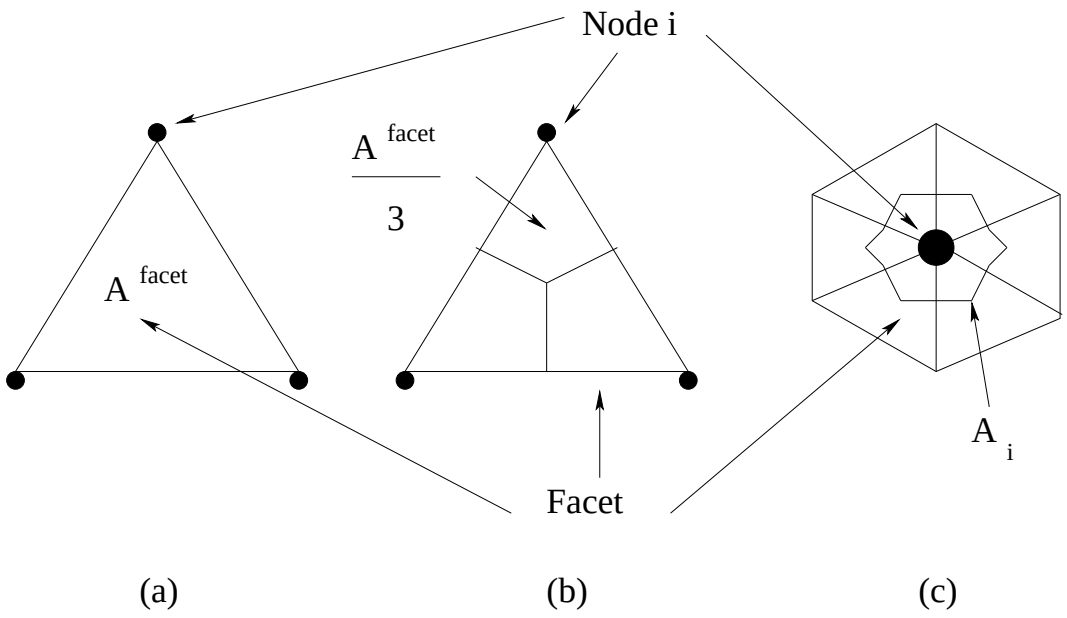


Figure 3.3: *(a) The above facet has 3 nodes and area $\boldsymbol{\mathcal{A}}^{facet}$. (b) This area is distributed equally to each node, $\boldsymbol{\mathcal{A}}^{facet}/3$. (c) In the presence of neighboring facets, the sum is taken over all neighbors to obtain the distributed area of a node i, $\boldsymbol{\mathcal{A}}_i$.*

Let us first derive $\boldsymbol{\mathcal{A}}_i$ and $\mathcal{V}_i$. Consider a deformable object with a discretized surface. Each node i is connected to j neighboring facets. These neighboring facets each have surface area $\boldsymbol{\mathcal{A}}_j^{facet}$. Within each facet, $\boldsymbol{\mathcal{A}}^{facet}$ is distributed to each of the it's nodes equally. This is graphically shown in figure 3.3. The distributed area for each node i is then obtained as follows:

$$\boldsymbol{\mathcal{A}}_i = \sum_j \left(\frac{\boldsymbol{\mathcal{A}}_j^{facet}}{3} \right) \tag{3.1}$$

$$\sum_i |\boldsymbol{\mathcal{A}}_i| = \mathcal{A} \tag{3.2}$$

Once the area has been distributed, the volume can be distributed as well. Given the total volume of the geometrical mesh model as $\mathcal{V}$ and the total surface area $\mathcal{A}$, we chose to distribute the volume as a function of the distributed area $\boldsymbol{\mathcal{A}}_i$. Then, the distributed volume for each node i is obtained as follows:

$$\mathcal{V}_i = \frac{|\boldsymbol{\mathcal{A}}_i|\mathcal{V}}{\mathcal{A}} \tag{3.3}$$

$$\sum_i \mathcal{V}_i = \mathcal{V} \tag{3.4}$$

3.2.3 Bulk Modulus

The bulk modulus for a node B_i and the connectivity bulk modulus B_{ij} are the physical parameters of the object being modeled. These values are generally obtained from experiments or from the literature. The physical definition of bulk

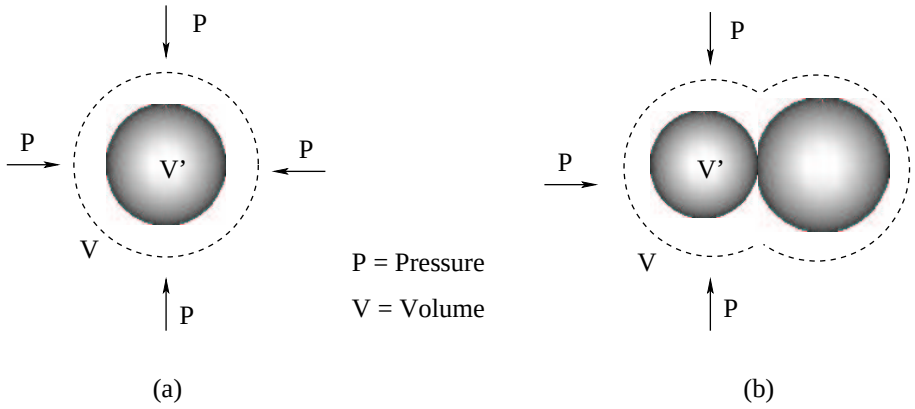


Figure 3.4: *Physical definition of (a) bulk modulus and (b) connectivity bulk modulus.*

modulus can be understood by considering a piece of material of volume $\mathcal{V}$ such as the one shown in figure 3.4. Given a change in pressure acting on this piece of material, a change of volume will be observed in this material. Then, the bulk modulus B can be defined as follows:

$$B = \frac{\Delta P}{\frac{\Delta \mathcal{V}}{\mathcal{V}}} \tag{3.5}$$

where $\Delta\mathcal{V} = \mathcal{V} - \mathcal{V}'$. The physical meaning of connectivity bulk modulus can then be defined as the same influence to a change in volume, but this time in the presence of a neighboring piece of material.

3.2.4 Volumic Pressure

Consider a force per unit area applied to a node i on the surface of our deformable object. This force produces deformation. However, deformation of a node induces volumic change. Now, by introducing bulk modulus B_i for this node, we have:

$$P_{vp} = \frac{B_i}{\mathcal{V}_i}\Delta\mathcal{V}_i \tag{3.6}$$

where $\Delta\mathcal{V}_i$, the volumic change, is our measure of strain and $\mathcal{V}_i$ is the volume associated to a node. Now volumic pressure can alternatively be written in the following form:

$$P_{vp} = K_i \Delta\mathcal{V}_i \tag{3.7}$$

where $K_i = \frac{B_i}{\mathcal{V}_i}$. Note that K_i is dependent on $\mathcal{V}_i$. We have derived the volumic pressure for a node.

3.2.5 Volumic Tension

Consider now a group of neighboring nodes. These nodes are linked topologically by the surface of our deformable object. This surface is elastic and to represent this in 3D, a difference in volumic change can be used (similar to a difference in length for the 1D case).

In this case, volumic tension P_{vt} can be written as:

$$P_{vt} = \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\Delta\mathcal{V}_i - \Delta\mathcal{V}_j\Big) \tag{3.8}$$

for all neighboring node j of node i. B_{ij} is the connectivity bulk modulus constant between node i and j. Note that we have not made any assumptions on the nature of elasticity between nodes, they can in general be of linear elasticity as in this case or of nonlinear elasticity.

3.2.6 Equilibrium State

The equilibrium state within a node is obtained by considering the following:

$$P_{ext} = P_{int} \tag{3.9}$$

where P_{ext} is the external pressure and P_{int} is the internal pressure.

The external pressure associated to a node is affected by the surrounding environmental pressure P_{ep} and by the stress due to volumic change:

$$P_{ext} = P_{ep} + P_{vp} \tag{3.10}$$

while the internal pressure is due to pressure of the incompressible fluid P_{fluid} and the effects of gravity $P_{gravity}$:

$$\begin{aligned} P_{int} &= P_{fluid} + P_{gravity} \\ &= P_{fluid} + \rho g \delta \end{aligned} \tag{3.11}$$

where ρ is the density of the incompressible fluid and δ is the measured hydrostatic distance of the node due to the contained fluid in our deformable object.

By considering neighboring nodes, equilibrium is attained by including effects from the volumic tension. Hence equation 3.10 becomes:

$$P_{ext} = P_{ep} + P_{vp} + P_{vt} \tag{3.12}$$

3.2.7 Model Assemblage

To obtain the VDM model assemblage, we formulate and group the equations for the state of equilibrium of each node on the surface:

$$P_{ext} = P_{int} \tag{3.13}$$

where by substituting equation 3.11 and equation 3.12, we get:

$$P_{ep} + P_{vp} + P_{vt} = P_{fluid} + P_{gravity} \tag{3.14}$$

Applying this equation to a group of $\aleph$ nodes, the following can be written using index notations:

$$\frac{B_i}{\mathcal{V}_i}\Delta\mathcal{V}_i + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\Delta\mathcal{V}_i - \Delta\mathcal{V}_j\Big) - \Delta P_i = \rho_i g \delta_i \quad \forall \quad i = 1 \ldots \aleph \tag{3.15}$$

where:

$$\Delta P_i = P_{fluid_i} - P_{ep_i} \tag{3.16}$$

and j indicates all neighboring nodes for each node i.

We will now use the following theorem as a boundary condition:

THEOREM 1. Pascal's Principle *states that a change in pressure ΔP, exerted on an enclosed static fluid, is transmitted undiminished throughout this medium and acts perpendicularly on the surface of the container.*

By applying Pascal's Principle which gives constant change in pressure throughout the deformable object, the index i can be removed from ΔP_i. By doing this, the following set of equations is obtained:

$$\frac{B_i}{\mathcal{V}_i}\Delta\mathcal{V}_i + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\Delta\mathcal{V}_i - \Delta\mathcal{V}_j\Big) - \Delta P = \rho_i g \delta_i \tag{3.17}$$

Since the fluid is incompressible, we can add another boundary condition to our set of equations. The incompressibility of the fluid imposes the constraint that the

volume of the deformable object is maintained at all times. By applying the principle of conservation of volume:

$$\sum_{i}^{\aleph} \Delta\mathcal{V}_i = 0 \tag{3.18}$$

Equation 3.17 and equation 3.18 are then rewritten such that ΔL_i appears as the variable instead of $\Delta\mathcal{V}_i$. This is done by using the following equation:

$$\Delta\mathcal{V}_i = \boldsymbol{\mathcal{A}}_i \, \boldsymbol{\Delta L}_i + \boldsymbol{L}_i \, \boldsymbol{\Delta \mathcal{A}}_i \tag{3.19}$$

If $\boldsymbol{\mathcal{A}}_i$ is assumed to be known throughout the history of load application, then $\boldsymbol{\Delta \mathcal{A}}_i$ is null and equation 3.19 is reduced to:

$$\Delta\mathcal{V}_i = \boldsymbol{\mathcal{A}}_i \, \boldsymbol{\Delta L}_i \tag{3.20}$$

We now have $\aleph + 1$ equations and $\aleph + 1$ unknowns; $\boldsymbol{\Delta L}_i$ for $i = 1 \ldots \aleph$ and ΔP. These $\aleph + 1$ equations can be written in the following matrix form:

$$\boldsymbol{K \Delta L} = \boldsymbol{R} \tag{3.21}$$

The matrix $\boldsymbol{K}$ is the state matrix of the VDM assemblage, $\boldsymbol{\Delta L}$ is the deformation vector matrix and the load vector assemblage $\boldsymbol{R}$ consists the hydrostatic pressure terms, $P_{gravity}$.

Model Illustration Example. To illustrate this assemblage, let us take a deformable tetrahedra with 4 nodes as an example. Given the volume $\mathcal{V}$ and surface area $\mathcal{A}$ of this tetrahedra, we can distribute them to the nodes as previously defined.

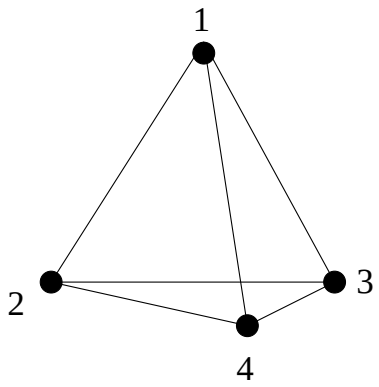


Figure 3.5: *A deformable tetrahedra with 4 nodes. Each node is connected to every other node. Each node also has a distributed area $\boldsymbol{\mathcal{A}}_i$ and volume $\mathcal{V}_i$.*

Now, the following is obtained, for example, for node 1:

Volumic Pressure:

$$P_{ip_1} = \frac{B_1}{\mathcal{V}_1}\Big(\boldsymbol{\mathcal{A}}_1 \boldsymbol{\Delta L}_1\Big) \tag{3.22}$$

Volumic Tension:

$$\begin{aligned} P_{vt_1} &= \frac{B_{12}}{\mathcal{V}_1}\Big(\boldsymbol{\mathcal{A}}_1 \boldsymbol{\Delta L}_1 - \boldsymbol{\mathcal{A}}_2 \boldsymbol{\Delta L}_2\Big) + \frac{B_{13}}{\mathcal{V}_1}\Big(\boldsymbol{\mathcal{A}}_1 \boldsymbol{\Delta L}_1 - \boldsymbol{\mathcal{A}}_3 \boldsymbol{\Delta L}_3\Big) \\ &+ \frac{B_{14}}{\mathcal{V}_1}\Big(\boldsymbol{\mathcal{A}}_1 \boldsymbol{\Delta L}_1 - \boldsymbol{\mathcal{A}}_4 \boldsymbol{\Delta L}_4\Big) \end{aligned} \tag{3.23}$$

Now by formulating the equilibrium state for node 1, the following is obtained:

$$\begin{aligned} \frac{B_1 \boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1}\boldsymbol{\Delta L}_1 + \frac{B_{12}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1}\boldsymbol{\Delta L}_1 + \frac{B_{13}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1}\boldsymbol{\Delta L}_1 + \frac{B_{14}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1}\boldsymbol{\Delta L}_1 \\ - \frac{B_{12}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_1}\boldsymbol{\Delta L}_2 - \frac{B_{13}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_1}\boldsymbol{\Delta L}_3 - \frac{B_{14}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_1}\boldsymbol{\Delta L}_4 \\ -\Delta P = \rho_1 g \delta_1 \end{aligned} \tag{3.24}$$

Similarly, the equation for node 2, 3 and 4 can be derived. Then the constraint on volume for this deformable tetrahedra can be written as follows:

$$\boldsymbol{\mathcal{A}}_1\ \boldsymbol{\Delta L}_1 + \boldsymbol{\mathcal{A}}_2\ \boldsymbol{\Delta L}_2 + \boldsymbol{\mathcal{A}}_3\ \boldsymbol{\Delta L}_3 + \boldsymbol{\mathcal{A}}_4\ \boldsymbol{\Delta L}_4 = 0 \tag{3.25}$$

By grouping all these equations, the state matrix $\boldsymbol{K}$, the displacement matrix $\boldsymbol{\Delta L}$ and the load matrix $\boldsymbol{R}$ can be written as follows, where the subscript j is used to indicate summation over all neighboring nodes:

$$\boldsymbol{K} = \begin{pmatrix} \frac{B_1\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} + \frac{B_{1j}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_2} & \frac{B_2\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} + \frac{B_{2j}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_3} & \frac{B_3\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} + \frac{B_{3j}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_3} & -1 \\ \frac{-B_{41}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_4} & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1 & \boldsymbol{\mathcal{A}}_2 & \boldsymbol{\mathcal{A}}_3 & \boldsymbol{\mathcal{A}}_4 & 0 \end{pmatrix}$$

$$\boldsymbol{\Delta L} = \begin{pmatrix} \boldsymbol{\Delta L}_1 \\ \boldsymbol{\Delta L}_2 \\ \boldsymbol{\Delta L}_3 \\ \boldsymbol{\Delta L}_4 \\ \Delta P \end{pmatrix} \qquad \boldsymbol{R} = \begin{pmatrix} \rho_1 g \delta_1 \\ \rho_2 g \delta_2 \\ \rho_3 g \delta_3 \\ \rho_4 g \delta_4 \\ 0 \end{pmatrix} \tag{3.26}$$

Each of this matrix is then decomposed in the respective x, y and z components. The resulting state matrix $\boldsymbol{K}$ is sparse and diagonally banded.

3.2.8 Anisotropic Behavior

In the previous chapter, we mentioned that the TMM model has been used to model anisotropic behavior. Anisotropy refers to the fact that deformation is experienced in a preferred direction. Soft tissue that are made up of nonhomogeneous material, for example fibers, exhibit this behavior. Some examples of soft tissue that fall into this category are muscles, ligaments and tendons.

The VDM model can also be used to characterize anisotropic behavior. By varying the bulk modulus B_i and the connectivity bulk modulus B_{ij} associated to each node, deformation can be preferred in a particular direction as compared to another. This can be explained by noting that in the presence of an external load, a change in volume is experienced. This change in volume is distributed uniformly to the other parts of the surface by the pressure experienced by the incompressible fluid. Since, by definition, the bulk modulus relates pressure to the fractional change in volume, it can be used to control the amount of volume distributed, and hence deformation.

In practice, the bulk modulus B_i and the connectivity bulk modulus B_{ij} are tuned based on the characteristics of the simulated material, for example, fibers are only allowed to deform in their respective fibrous orientation.

Model Illustration Example. To demonstrate the changes in the formulation, let us once again take the deformable tetrahedra for example. The state matrix $\boldsymbol{K}$ for the 4 nodes is given as follows:

$$\boldsymbol{K} = \begin{pmatrix} \frac{B_1\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} + \frac{B_{1j}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_2} & \frac{B_2\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} + \frac{B_{2j}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_3} & \frac{B_3\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} + \frac{B_{3j}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_3} & -1 \\ \frac{-B_{41}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_4} & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1 & \boldsymbol{\mathcal{A}}_2 & \boldsymbol{\mathcal{A}}_3 & \boldsymbol{\mathcal{A}}_4 & 0 \end{pmatrix} \tag{3.27}$$

If we do not want node 2 to move due to the internal pressure and node 4 to move due to volumic tension from node 1, we set $B_2 = \infty$ and $B_{41} = \infty$. Then the

state matrix $\boldsymbol{K}$ becomes:

$$\boldsymbol{K} = \begin{pmatrix} \frac{B_1\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} + \frac{B_{1j}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_2} & \infty & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_3} & \frac{B_3\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} + \frac{B_{3j}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_3} & -1 \\ \infty & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1 & \boldsymbol{\mathcal{A}}_2 & \boldsymbol{\mathcal{A}}_3 & \boldsymbol{\mathcal{A}}_4 & 0 \end{pmatrix} \tag{3.28}$$

3.2.9 Stress Distribution

Stress $\boldsymbol{\sigma}$, is a function of strain, and strain in the VDM model is a function of volumic change $\Delta\mathcal{V}$. Hence for a node i, stress can be measured by calculating the net volumic change of the node with respect to its neighbors j:

$$\begin{aligned} \boldsymbol{\sigma}_i &= \frac{B_i}{\mathcal{V}_i}\Big(\Delta\mathcal{V}_i - \sum_j \Delta\mathcal{V}_j\Big) \\ &= \frac{B_i}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\ \boldsymbol{\Delta L}_i - \sum_j \boldsymbol{\mathcal{A}}_j\ \boldsymbol{\Delta L}_j\Big) \end{aligned} \tag{3.29}$$

3.2.10 Imposing Constraints

Once the state matrix $\boldsymbol{K}$ has been obtained, in the presence of constraints, the respective nodes in $\boldsymbol{K}$ has to be modified. We identify four types of constraints as shown in figure 3.6, that can be subjected to our deformable object.

FREE : Nodes that belong to this group are free to move. These nodes are not subjected to an external force and hence have $P_{cp} = 0$. Thus, they are governed by the following equation:

$$\frac{B_i}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\ \boldsymbol{\Delta L}_i\Big) + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\ \boldsymbol{\Delta L}_i - \boldsymbol{\mathcal{A}}_j\ \boldsymbol{\Delta L}_j\Big) - \Delta P = \rho_i g \delta_i \tag{3.30}$$

CONTACT : Nodes that belong to this group are subjected to an external load vector which can be modeled as a contact pressure P_{cp}. Hence, these nodes are

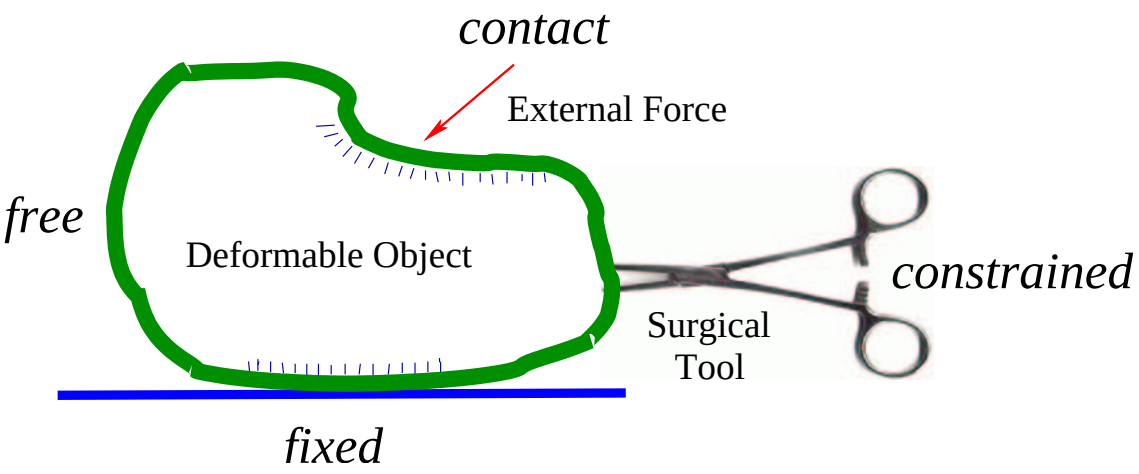


Figure 3.6: *A resting deformable object that is touched by a finger and manipulated by a tool. The following constraints manifest in this deformable object; FIXED: nodes that do not move, FREE: nodes that are free to move, CONTACT: nodes that are subjected to an external force, CONSTRAINED: nodes that are subjected to a displacement vector.*

governed by the following equation:

$$\frac{B_i}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\ \boldsymbol{\Delta L}_i\Big) + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\ \boldsymbol{\Delta L}_i - \boldsymbol{\mathcal{A}}_j\ \boldsymbol{\Delta L}_j\Big) - \Delta P = \rho_i g \delta_i + P_{cp_i} \tag{3.31}$$

FIXED : Nodes in this group are fixed. They have null displacement vector and hence do not move during the simulation. To enforce this constraint during simulation, a penalty method is applied to the respective nodes. In this method, for a desired null displacement of node i, a penalizing term $\alpha \gg 0$ is added to the diagonal term K_{ii} in the state matrix and $R_i = 0$.

$$K_{ii} = \alpha\ ,\ R_i = 0 \tag{3.32}$$

Model Illustration Example. Let us take our deformable tetrahedra as an example again. If node 3 is fixed, then the following is obtained in the state matrix

and load vector:

$$\boldsymbol{K} = \begin{pmatrix} \frac{B_1\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} + \frac{B_{1j}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_1} & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_2} & \frac{B_2\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} + \frac{B_{2j}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_3} & \infty & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_3} & -1 \\ \frac{-B_{41}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_4} & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1 & \boldsymbol{\mathcal{A}}_2 & \boldsymbol{\mathcal{A}}_3 & \boldsymbol{\mathcal{A}}_4 & 0 \end{pmatrix}$$

$$\boldsymbol{R} = \begin{pmatrix} \rho_1 g \delta_1 \\ \rho_2 g \delta_2 \\ 0 \\ \rho_4 g \delta_4 \\ 0 \end{pmatrix} \tag{3.33}$$

CONSTRAINED : The nodes belonging to this last group are subjected or constrained to a known displacement vector $\mathbb{C}$. This happens when the nodes of our deformable object is fixed to a moving part. Again, to enforce this constraint, a penalizing term $\alpha \gg 0$ is added to the diagonal term K_{ii} and $R_i = \alpha\mathbb{C}$.

$$K_{ii} = \alpha \ , \ R_i = \alpha\mathbb{C} \tag{3.34}$$

Model Illustration Example. Again, if node 1 in our deformable tetrahedra is temporarily attached to a tool and thus constrained by a displacement $\mathbb{C}$ of the tool, then the state matrix and load vector will be as follows:

$$\boldsymbol{K} = \begin{pmatrix} \infty & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_2} & \frac{B_2\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} + \frac{B_{2j}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_2} & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_3} & \frac{B_3\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} + \frac{B_{3j}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_3} & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_3} & -1 \\ \frac{-B_{41}\boldsymbol{\mathcal{A}}_1}{\mathcal{V}_4} & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1 & \boldsymbol{\mathcal{A}}_2 & \boldsymbol{\mathcal{A}}_3 & \boldsymbol{\mathcal{A}}_4 & 0 \end{pmatrix}$$

$$\boldsymbol{R} = \begin{pmatrix} \alpha\mathbb{C} \\ \rho_2 g \delta_2 \\ \rho_3 g \delta_3 \\ \rho_4 g \delta_4 \\ 0 \end{pmatrix} \tag{3.35}$$

3.3 System Resolution

In the previous section, the mathematical foundations of the VDM model was presented. The system of equations describing the equilibrium of the system was derived. In this section, we are interested in the methods of solving this set of equations for linear and nonlinear static analysis. A static analysis is sufficient because soft tissue is known to be well-damped and thus the viscoelastic effects can be neglected. Furthermore, a static resolution does not suffer from numerical instabilities related to the convergence of dynamic systems.

3.3.1 Linear Analysis

The linear analysis amounts to calculating small deformations and small strains due to external loads. Let us recall the equation of equilibrium of the VDM model once again:

$$\frac{B_i}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i\Big) + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i - \boldsymbol{\mathcal{A}}_j\boldsymbol{\Delta L}_j\Big) - \Delta P = \rho_i g \delta_i \tag{3.36}$$

where $i = 1 \dots \aleph$ and $\Delta P = P_{fluid} - P_{ep}$. For small deformations, we assume that the area $\boldsymbol{\mathcal{A}}^{facet}$ of our facets do not change. Hence, the boundary conditions can be reduced such that:

$$\sum_i^{\aleph} \boldsymbol{\mathcal{A}}_i \, \boldsymbol{\Delta L}_i = 0 \tag{3.37}$$

Property : If $|\boldsymbol{\Delta L}_i| < \epsilon \ \ \forall \ i$, then $\boldsymbol{\mathcal{A}}_i$ can be considered constant and the state matrix $\boldsymbol{K}$ is constructed only once in the beginning. This matrix is reused at each time step. The problem $\boldsymbol{K\Delta L} = \boldsymbol{R}$ is then solved optimally using standard numerical methods to obtain a solution.

Numerical Resolution : A technique that is frequently used to precondition the state matrix $\boldsymbol{K}$ is the lower-upper (LU) decomposition. In this method, we suppose that $\boldsymbol{K}$ can be written as:

$$\boldsymbol{LU} = \boldsymbol{K} \tag{3.38}$$

where $\boldsymbol{L}$ and $\boldsymbol{U}$ are the lower and upper triangular elements of $\boldsymbol{K}$. In the case of our deformable tetrahedra, we have a 5×5 state matrix and the LU decomposition

will be as follows:

$$\begin{pmatrix} L_{11} & 0 & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} & 0 \\ L_{51} & L_{52} & L_{53} & L_{54} & L_{55} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} & U_{14} & U_{15} \\ 0 & U_{22} & U_{23} & U_{24} & U_{25} \\ 0 & 0 & U_{33} & U_{34} & U_{35} \\ 0 & 0 & 0 & U_{44} & U_{45} \\ 0 & 0 & 0 & 0 & U_{55} \end{pmatrix} = \begin{pmatrix} K_{11} & K_{12} & K_{13} & K_{14} & K_{15} \\ K_{21} & K_{22} & K_{23} & K_{24} & K_{25} \\ K_{31} & K_{32} & K_{33} & K_{34} & K_{35} \\ K_{41} & K_{42} & K_{43} & K_{44} & K_{45} \\ K_{51} & K_{52} & K_{53} & K_{54} & K_{55} \end{pmatrix} \tag{3.39}$$

By using the decomposition $\boldsymbol{LU} = \boldsymbol{K}$ and after substitution into $\boldsymbol{K\Delta L} = \boldsymbol{R}$, we get:

$$\boldsymbol{K\Delta L} = (\boldsymbol{LU})\boldsymbol{\Delta L} = \boldsymbol{L}(\boldsymbol{U\Delta L}) = \boldsymbol{R} \tag{3.40}$$

We first solve for the vector $\boldsymbol{Y}$ such that:

$$\boldsymbol{LY} = \boldsymbol{R} \tag{3.41}$$

and then solve:

$$\boldsymbol{U\Delta L} = \boldsymbol{Y} \tag{3.42}$$

Since $\boldsymbol{K}$ is constant, hence the matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ are also constant. Thus, these matrices are also precomputed. The pseudo-code of this resolution procedure is described in algorithm 1.

Data : $\boldsymbol{K}$, the state matrix and $\boldsymbol{R}$, the external load vector
Result: $\boldsymbol{\Delta L}$, the nodal displacement vector
begin
  $\boldsymbol{A} \longleftarrow \boldsymbol{K}$
  $\boldsymbol{B} \longleftarrow \boldsymbol{R}$
  if ($\boldsymbol{LU} = \boldsymbol{A}$) **then**
    while ($|\boldsymbol{B}| \neq 0$) **do**
      $\boldsymbol{LY} = \boldsymbol{B}$
      $\boldsymbol{U\Delta L} = \boldsymbol{Y}$
end

Algorithm 1: *Resolution of* $\boldsymbol{K\Delta L} = \boldsymbol{R}$ *using a linear approach.*

3.3.2 Quasi-Linear Analysis

The analysis presented earlier is only valid for small deformations and small strains ($< 10\%$) where we assume that $\boldsymbol{\mathcal{A}}_i$ is constant. For well-damped soft tissue, this is perhaps a valid assumption for small deformations. However, for other kinds

of large deformations, this assumption becomes invalid and volume conservation needs to be restated as follows:

$$\sum_{i}^{\aleph} \Big(\boldsymbol{\mathcal{A}}_i \boldsymbol{\Delta L}_i + \boldsymbol{L}_i \boldsymbol{\Delta \mathcal{A}}_i \Big) = 0 \tag{3.43}$$

Properety : In this case, $\boldsymbol{\mathcal{A}}_i$ is the nonlinear term and it has to be updated regularly. In the quasi-linear analysis, a *selective* method is used to update the nonlinear terms. Since the nonlinear $\boldsymbol{\mathcal{A}}_i$ terms appear in almost all the nonzero terms of $\boldsymbol{K}$, a choice has to be made on which terms that needs to be updated. To ensure that volume is conserved during our simulation, it is imperative that equation 3.43 is satisfied at all times. This is done by updating the $\boldsymbol{\mathcal{A}}_i$ terms at each time-step. These terms only appear in the bottom row of matrix $\boldsymbol{K}$, allowing us to use an optimal method to solve the system.

Numerical Resolution : The Sherman-Morrison (SM) method can be used for this purpose. In this method, we suppose that the inverse matrix of the initial state matrix $\boldsymbol{K}^{-1}$ has been found. This can be done using any numerical method. Then the solution to the system is given as follows:

$$\boldsymbol{\Delta L} = \boldsymbol{K}^{-1} \boldsymbol{R} \tag{3.44}$$

Since $\boldsymbol{K}$ is updated at each time-step with the new $\boldsymbol{\mathcal{A}}_i$ terms, $\boldsymbol{K}^{-1}$ has to be recomputed to obtain a new solution. Now, given a change to our original matrix $\boldsymbol{K}$ of the form:

$$\boldsymbol{K} \longmapsto (\boldsymbol{K} + \boldsymbol{u} \otimes \boldsymbol{v}) \tag{3.45}$$

where in our case, vector $\boldsymbol{u}$ is a unit vector and $\boldsymbol{v}$ is the components of the error vector that must be added, then the desired change in the inverse is given by:

$$\boldsymbol{K}^{-1} \longmapsto \boldsymbol{K}^{-1} + \frac{\{\boldsymbol{K}^{-1} \cdot \boldsymbol{u}\} \otimes \{(\boldsymbol{K}^{-1})^T \cdot \boldsymbol{v}\}}{1 + (\boldsymbol{K}^{-1} \cdot \boldsymbol{u})} \tag{3.46}$$

Model Illustration Example. Let us take our deformable tetrahedra as an example. It has a 5×5 state matrix and an initial distributed area denoted as $\boldsymbol{\mathcal{A}}_i^0$. At each time-step, $\boldsymbol{\mathcal{A}}_i$ of each node can be recalculated as follows:

$$\boldsymbol{\mathcal{A}}_i \longmapsto (\boldsymbol{\mathcal{A}}_i^0 + \boldsymbol{\Delta \mathcal{A}}_i) \tag{3.47}$$

Then, for this tetrahedra, the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are given as follows:

$$\boldsymbol{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \boldsymbol{v} = \begin{bmatrix} \boldsymbol{\Delta A}_1 & \boldsymbol{\Delta A}_2 & \boldsymbol{\Delta A}_3 & \boldsymbol{\Delta A}_4 & 0 \end{bmatrix} \tag{3.48}$$

Now by using the state matrix of our deformable tetrahedra as given in equation 3.33 and by updating with these $\boldsymbol{u}$ and $\boldsymbol{v}$ vectors, we obtain the following:

$$\begin{pmatrix} \frac{B_1\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_1} + \frac{B_{1j}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_1} & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_2} & \frac{B_2\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_2} + \frac{B_{2j}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_2} & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_3} & \frac{B_3\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_3} + \frac{B_{3j}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_3} & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_3} & -1 \\ \frac{-B_{41}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_4} & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1^0 & \boldsymbol{\mathcal{A}}_2^0 & \boldsymbol{\mathcal{A}}_3^0 & \boldsymbol{\mathcal{A}}_4^0 & 0 \end{pmatrix}$$

$$+ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} \boldsymbol{\Delta A}_1 & \boldsymbol{\Delta A}_2 & \boldsymbol{\Delta A}_3 & \boldsymbol{\Delta A}_4 & 0 \end{bmatrix} =$$

$$\begin{pmatrix} \frac{B_1\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_1} + \frac{B_{1j}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_1} & \frac{-B_{12}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_1} & \frac{-B_{13}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_1} & \frac{-B_{14}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_1} & -1 \\ \frac{-B_{21}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_2} & \frac{B_2\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_2} + \frac{B_{2j}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_2} & \frac{-B_{23}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_2} & \frac{-B_{24}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_2} & -1 \\ \frac{-B_{31}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_3} & \frac{-B_{32}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_3} & \frac{B_3\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_3} + \frac{B_{3j}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_3} & \frac{-B_{34}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_3} & -1 \\ \frac{-B_{41}\boldsymbol{\mathcal{A}}_1^0}{\mathcal{V}_4} & \frac{-B_{42}\boldsymbol{\mathcal{A}}_2^0}{\mathcal{V}_4} & \frac{-B_{43}\boldsymbol{\mathcal{A}}_3^0}{\mathcal{V}_4} & \frac{B_4\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_4} + \frac{B_{4j}\boldsymbol{\mathcal{A}}_4^0}{\mathcal{V}_4} & -1 \\ \boldsymbol{\mathcal{A}}_1^0 + \boldsymbol{\Delta A}_1 & \boldsymbol{\mathcal{A}}_2^0 + \boldsymbol{\Delta A}_2 & \boldsymbol{\mathcal{A}}_3^0 + \boldsymbol{\Delta A}_3 & \boldsymbol{\mathcal{A}}_4^0 + \boldsymbol{\Delta A}_4 & 0 \end{pmatrix} \tag{3.49}$$

Hence, adding the matrix $\boldsymbol{u} \otimes \boldsymbol{v}$ to the original state matrix is equivalent to updating the bottom row of $\boldsymbol{K}$ with the new distributed areas. The pseudo-code of this resolution procedure is described in algorithm 2.

Data : $\boldsymbol{K}$, the state matrix and $\boldsymbol{R}$, the external load vector
Result: $\boldsymbol{\Delta L}$, the nodal displacement vector
begin
  Given $\boldsymbol{K} \longmapsto (\boldsymbol{K} + \boldsymbol{u} \otimes \boldsymbol{v})$
  $\boldsymbol{A} \longleftarrow \boldsymbol{K}$
  $\boldsymbol{B} \longleftarrow \boldsymbol{R}$
  while ($|\boldsymbol{B}| \neq 0$) **do**
    Compute Error Vector $\boldsymbol{v}$
    Compute $\boldsymbol{K}^{-1}$
    Solve $\boldsymbol{\Delta L} = \boldsymbol{K}^{-1}\boldsymbol{R}$
end

Algorithm 2: *Resolution of $\boldsymbol{K\Delta L} = \boldsymbol{R}$ using a quasi-linear approach.*

3.3.3 Nonlinear Analysis

In the nonlinear analysis, all nonlinear terms are updated at each time-step. This amounts to simulating large deformations and large strains. In this case, for large systems, a simple inversion or preconditioning of the state matrix at each time-step may be computationally expensive for interactive-time applications. However, the rapid increase in computational power has popularized iterative methods as a resolution scheme.

Property : The basic idea behind this iterative method is the minimization of the residual r^i at each iteration i, defined as:

$$\boldsymbol{r}^i = \boldsymbol{R} - \boldsymbol{K\Delta L}^i \tag{3.50}$$

It can be shown that this guarantees exact convergence for linear systems of equation in at most n iterations, where n is the size of the linear system. However, due to floating point errors, exact convergence is improbable in practice and the solution is obtained when the error drops below a defined tolerance.

Numerical Resolution : We chose the Bi-Conjugate Gradient (BCG) iterative method as the optimal resolution scheme. This method is attractive for large sparse systems because only the nonzero terms of the state matrix is stored; hence minimal memory. For real-time solution of very large systems, even a solution in n iterations may be too expensive. However, in interactive-time applications the solution $\boldsymbol{\Delta L}$ only changes minimally from one time-step to another. Then, by using the previous result of the displacement vector as the starting guess for $\boldsymbol{\Delta L}^0$, we can achieve

dramatic gains in speed after finding the first solution. The number of iterations needed to minimize the error below a certain tolerance is very much smaller than the value of n. The pseudo-code of this iterative resolution procedure is described in algorithm 3.

Data : $\boldsymbol{K}$, the state matrix and $\boldsymbol{R}$, the external load vector
Result: $\boldsymbol{\Delta L}$, the nodal displacement vector
begin
  $\boldsymbol{A} \longleftarrow \boldsymbol{K}$
  $\boldsymbol{B} \longleftarrow \boldsymbol{R}$
  while ($|\boldsymbol{B}| \neq 0$) **do**
    while ($|\boldsymbol{\Delta L}^{i+1}| - |\boldsymbol{\Delta L}^{i}| > \epsilon$) **do**
      Improve $\boldsymbol{r}^i = \boldsymbol{B} - \boldsymbol{A}\boldsymbol{\Delta L}^i$
      Improve Estimated Solution $\boldsymbol{\Delta L}^{i+1}$
end

Algorithm 3: *Resolution of* $\boldsymbol{K}\boldsymbol{\Delta L} = \boldsymbol{R}$ *using an iterative approach.*

3.4 Simulation Results

In this section we presents the simulation results of the VDM model. We present the following aspects:

- Anisotropic Behavior
- Stress Distribution
- Comparison with FEM

Anisotropic Behavior. To test this behavior, we compared deformation curves of three points on a cube at rest that were placed on the xy, xz and yz planes respectively (see figure 3.7). In this test, a force was applied to compress the cube at rest. We first allowed deformation in all directions. We then changed the bulk modulus to observe the behavior of the cube. The results from the deformation curves show that the VDM model respects the imposed anisotropic constraints.

From the results shown in figure 3.8, we can see that the cube behaves differently when the bulk modulus is changed. In the first test, the displacement of points B and C coincide. This is because of the uniform bulk modulus. Hence, these points should move equally to maintain conservation of volume. In the second test, point

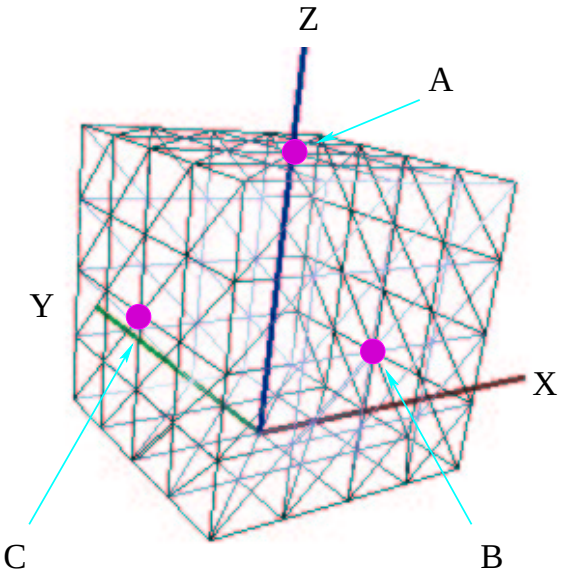
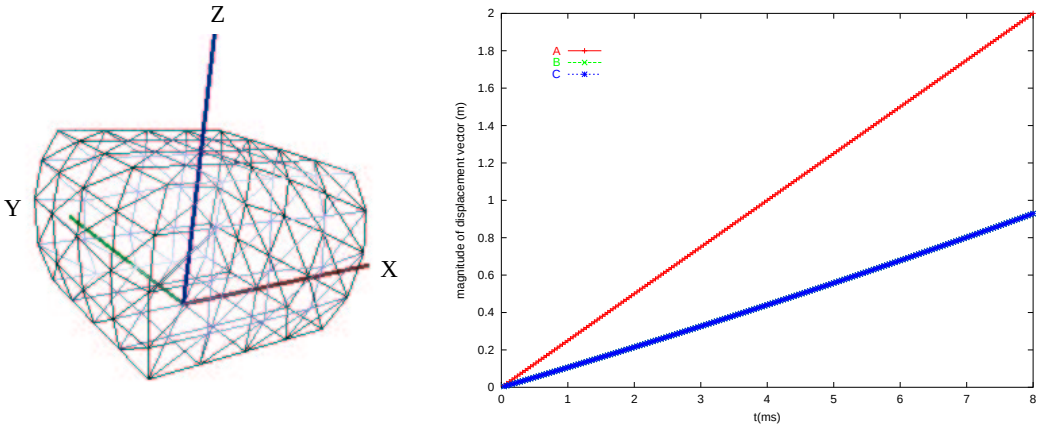


Figure 3.7: *A test cube at rest was used as an example for the anisotropic behavior test. 3 test points were chosen; A on the xy-plane, B on the xz-plane and C on the yz-plane. The bottom part of the cube is at rest and fixed. A force to compress to cube was applied along the z-axis. The displacement vectors of the 3 test points normal to their respective planes were observed.*

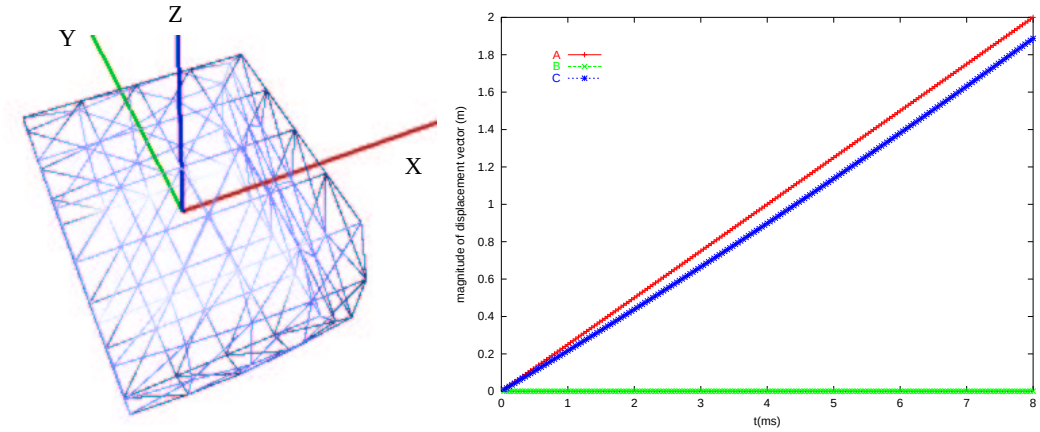
B is on the xz-plane which has bulk modulus set to infinity. From the displacement curves, point B is seen to have nearly zero displacement. On the other hand, point C has a larger displacement vector in time. This is due to the constraint of conservation of volume. In the last test, deformation was preferred in the y-direction. This is observed in the displacement curve of point B by comparing with the first test. Point C however, has a smaller displacement vector to maintain volume conservation. We note that point A has a constant displacement curve in all the three tests. This is due to the constraint of type *contact* applied to this node. In conclusion, by changing the bulk modulus that is associated to the nodes, deformation can be preferred in a particular direction.

Stress Distribution. To plot the stress distribution in our VDM model, we used the cube at rest as an example again (see figure 3.9). This time, a force to compress the cube was applied and deformation was allowed in all directions without preference. Three test points were used to observe the magnitude of stress of the cube. The results are presented.

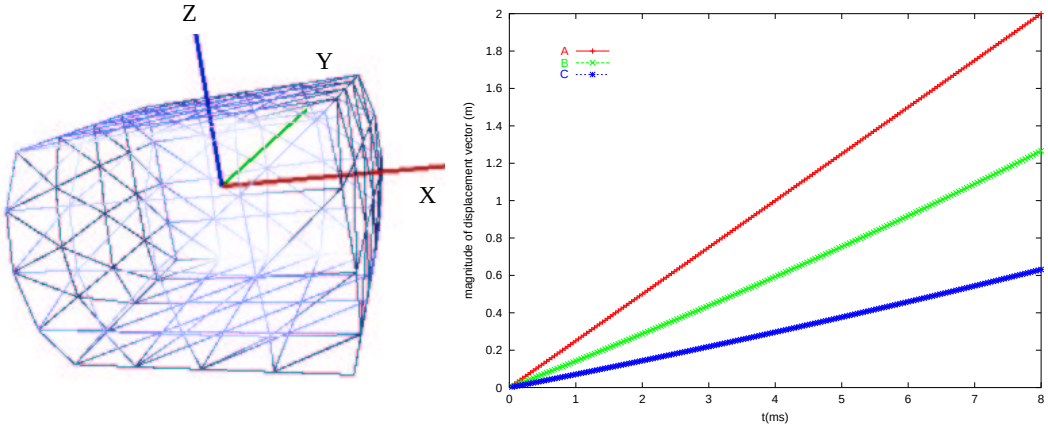
The results in figure 3.10 show that the stress experienced by the cube due to volumic tension is indeed dependent on the displacement and the distributed



(a)

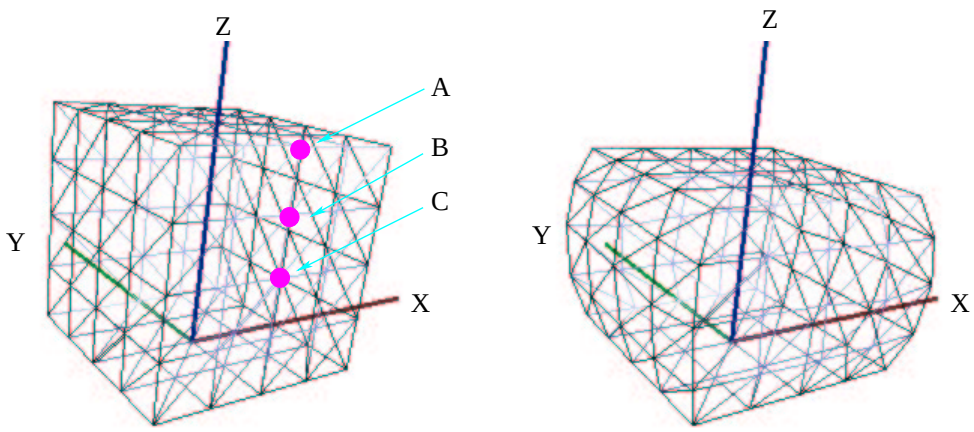


(b)



(c)

Figure 3.8: *(a) Compression with uniform bulk modulus. (b) The bulk modulus along the x-direction was set to infinity. (c) The bulk modulus of the cube was set such that deformation is preferred in the y-direction.*

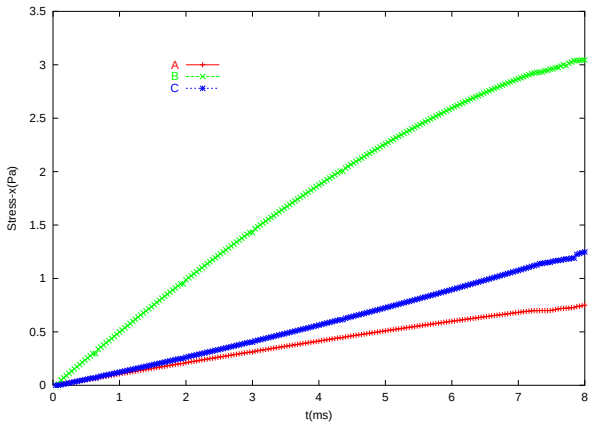


(a) (b)

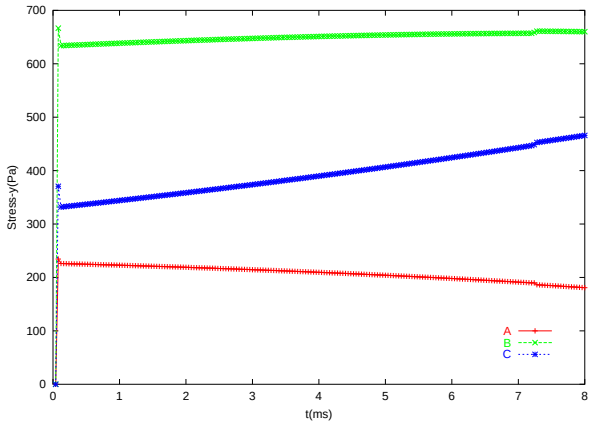
Figure 3.9: *(a) A test cube at rest was used as an example for the stress distribution test. 3 test points were chosen; A, B and C on the xz-plane. The bottom part of the cube is at rest and fixed. A force to compress to cube was applied along the z-axis. The magnitude of stress in the x, y and z directions were observed. (b) This cube at rest with its bottom fixed was compressed using a force along the z-direction.*

surface area. In other words, they are a function of volumic change. The stress in the x-direction is very minimal for all the test points because these points have displacement vectors with small x-components. On the other hand, since all the test points have significant displacement in the other directions, stress is observed to increase in the y and z directions. Point C has almost zero stress in the z-direction because during compression, this point has minimum distributed area in this direction. For the stress in the y-direction, point A has minimum stress. This is due to the fact that this point is constrained not to move in this direction. On the other hand, points B and C are displaced in the y-direction but point B has a higher stress which is due to the higher net volumic change experienced as compared to point C.

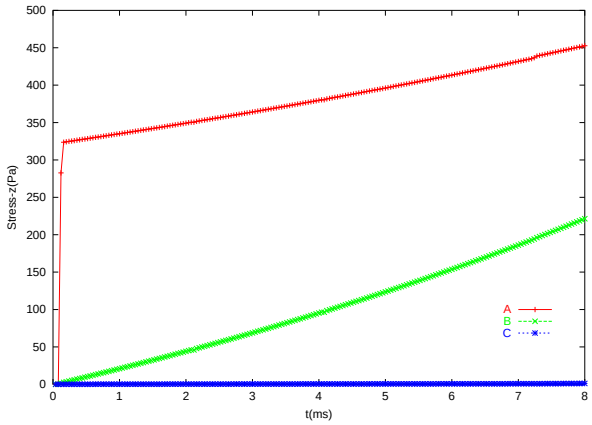
Comparison with FEM. To investigate the accuracy of the VDM model, we decided to compare it with a well known classical model like the FEM model. In these experiments, two beam mesh as shown in figure 3.11 with similar rigidity characteristics was used. One end of the beam was fixed and the other end was subjected to a displacement vector describing various types of large deformations. The final configuration of the beam was observed and the displacement vectors of several points along the beam was compared.



(a)



(b)



(c)

Figure 3.10: *(a) Stress distribution along the x-direction. (b) Stress distribution along the y-direction. (c) Stress distribution along the z-direction.*

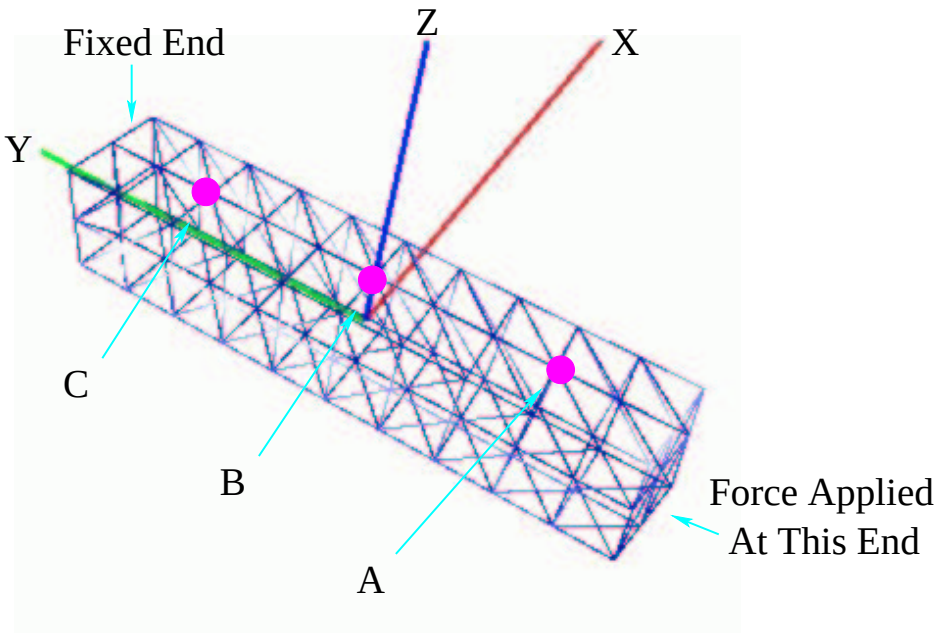
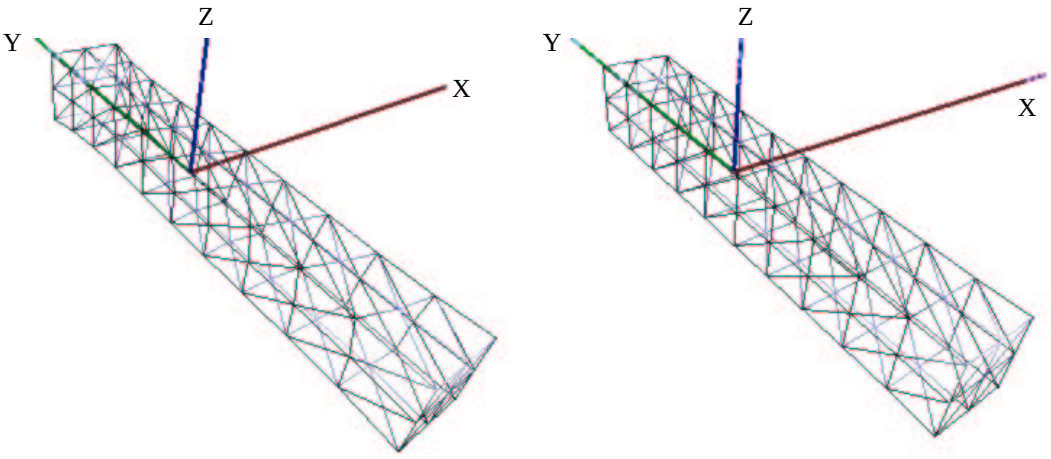


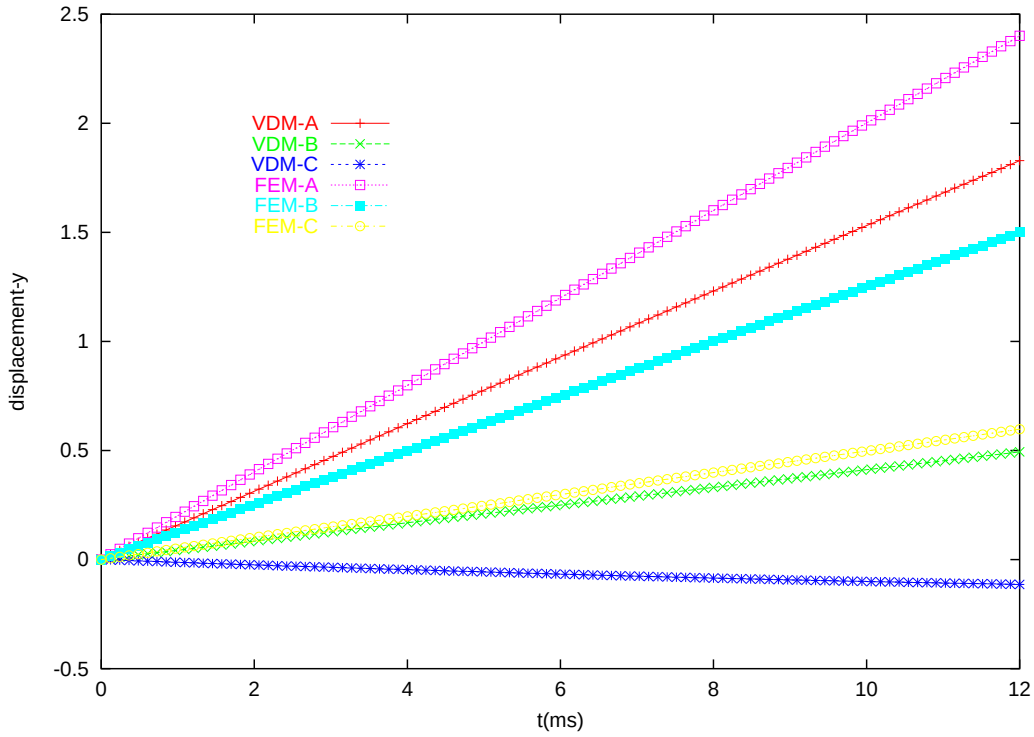
Figure 3.11: *A test beam with one end fixed was used as an example for the comparison test. 3 test points were chosen; A, B and C along the y-axis. A force to deform the beam was applied. 3 types of deformation were tested; stretching, bending and twisting. The displacement vectors of the 3 test points were compared.*

From the results, we can see that there is a difference in the behavior of the nodes but the general shape of the beam seem to be identical. We observed the difference from one tine step to another. In the first test for stretching, a systematic increase of about 5% in the average error of the curves are observed for all the test points. When a force to bend the beam was applied, a constant systematic increase of about 2% in the average error is observed between all the test points. Twisting was applied in the last test where again a systematic increase of about 5% in the average error of the curves are observed for all the test points.

The difference between the models is nevertheless expected. FEM is a volumic model as compared to VDM which is surface based. Also, the physical parameters of FEM and VDM are not easily matched. The error in the *rigidity* constant is another source of error in the results. It is unclear how a deformable beam would behave under externally applied forces. But, we would like to note that realism was rather observed in the VDM model. At each time-step, we calculated the volume of the beam of the two models and found that volume conservation was rather observed in the VDM model. In conclusion, there is a difference between the two models, but if we would like to observe volume conservation, VDM seems to be a better choice.

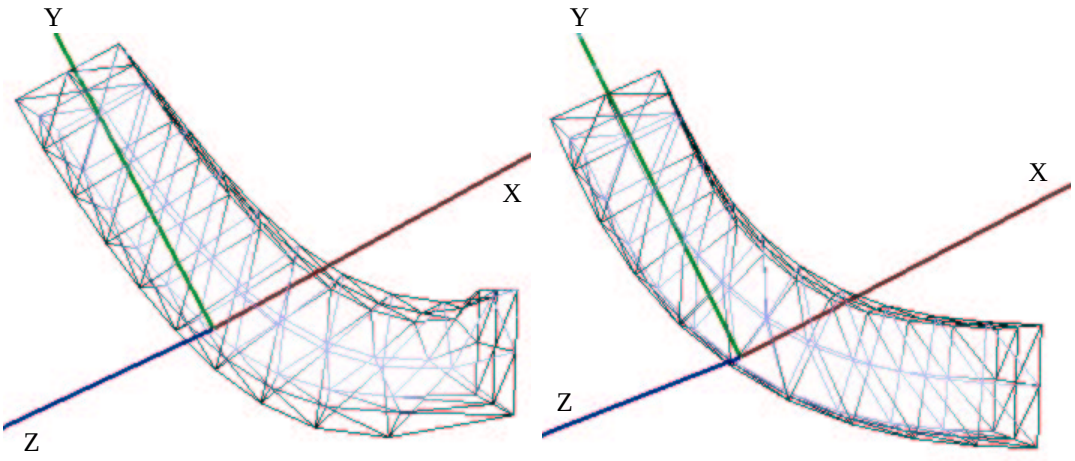


(a) (b)

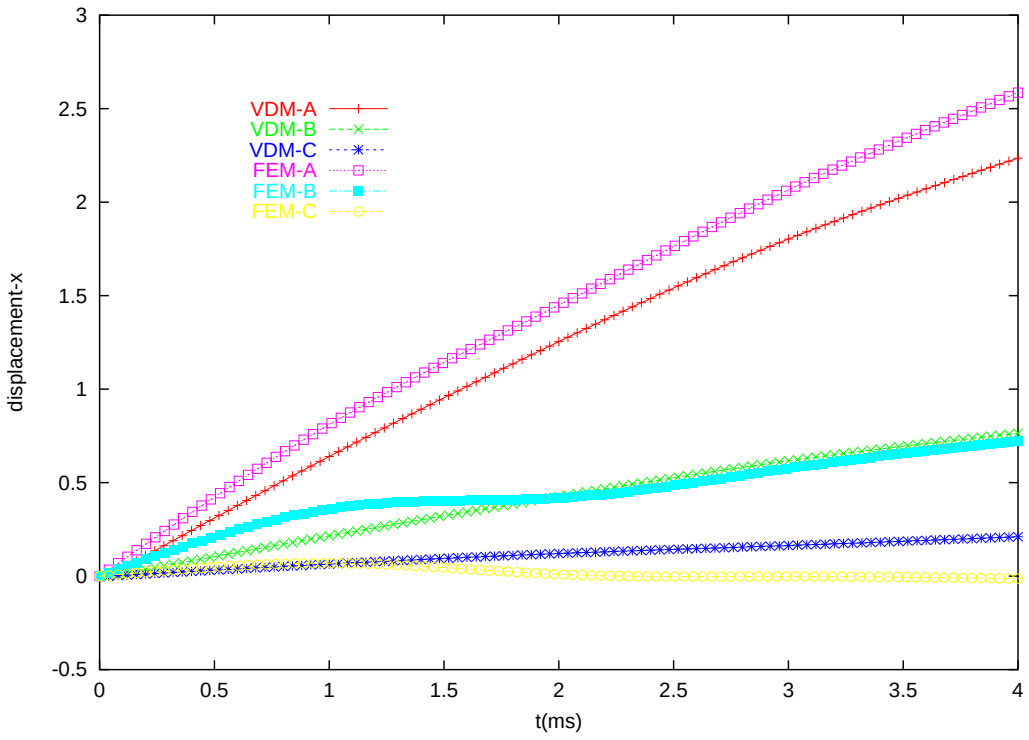


(c)

Figure 3.12: *Stretching of a beam with one end fixed and a force applied along the y-axis at the other end. Final configuration of the beam with the, (a) VDM model and (b) FEM model. (c) The variation of the magnitude of the displacement vector of the 3 test points.*

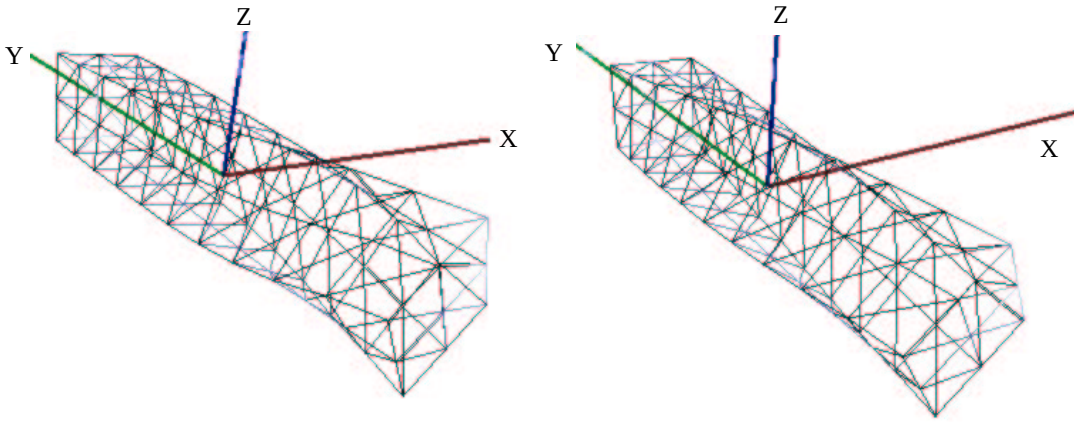


(a) (b)

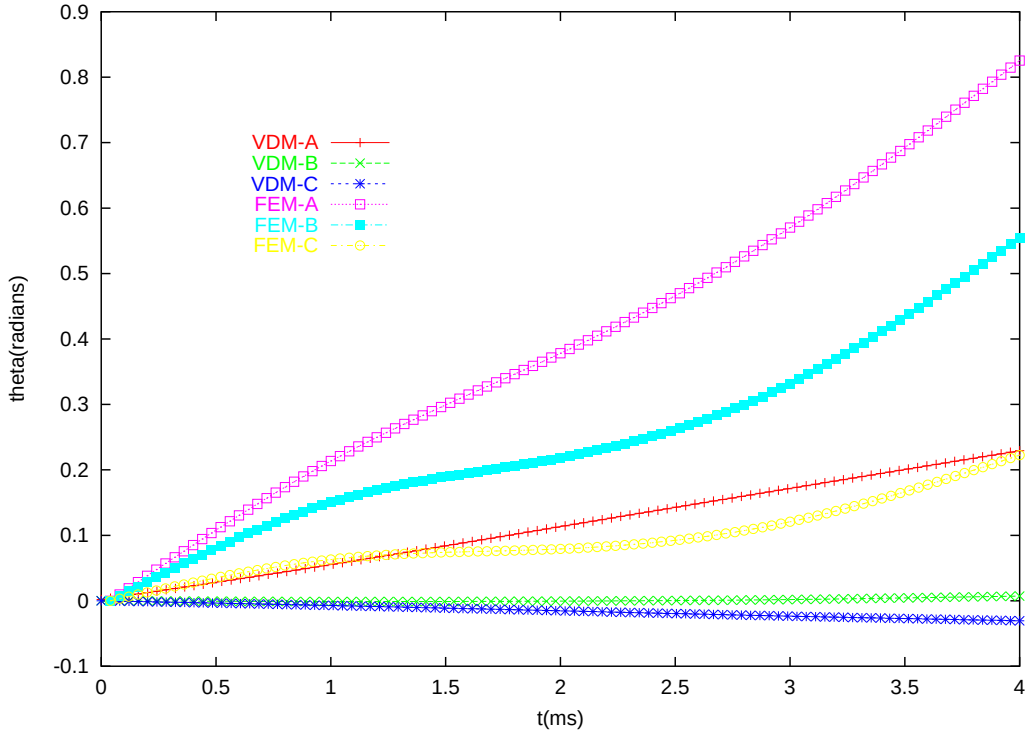


(c)

Figure 3.13: *Bending of a beam with one end fixed and a force applied along the x-axis at the other end. Final configuration of the beam with the, (a) VDM model and (b) FEM model. (c) The variation of the displacement vector of the 3 test points.*



(a) (b)



(c)

Figure 3.14: *Twisting of a beam with one end fixed and a force applied to rotate along the xz-plane at the other end. Final configuration of the beam with the, (a) VDM model and (b) FEM model. (c) The variation of the displacement vector of the 3 test points.*

3.5 Summary

In this chapter we have presented VDM, a new physical model suitable for soft tissue simulation. This model is surface based, hence it's complexity is in general one order of magnitude lower than a classic volumic model like FEM. VDM uses bulk variables like pressure, volume and bulk modulus as model parameters. Pascal's principle and volume conservation are used as boundary conditions.

A key feature of VDM is the absence of discretization of the interior. Any surface based polygonal mesh can be used as input data. An advantage in the medical field is that these types of triangulaire meshes are common. They are obtained from segmented MRI images. The required parameters for the VDM model like volume and area of the facets are easily extracted from these polygonal mesh.

The intrinsic parameters in VDM for soft tissue depends on the organ being simulated. If for example, a liver is being modeled, it being 95% irrigated by blood, requires only the density ρ and bulk modulus B of blood. Such parameters can be obtained from the literature. Furthermore, experiments to determine these values can be conducted for verification purposes. These advantages make VDM an interesting alternative for soft tissue simulation.

We have also presented three possible resolution methods for static analysis of soft tissue simulation using the VDM model. A static analysis is sufficient in many cases as soft tissue is known to be well damped and the viscoelastic effects can be neglected. The linear analysis is suitable for soft tissue undergoing small deformation. There are many examples of medical simulation that restrict deformation within a linear analysis. The quasi-linear analysis allows a quick resolution technique for any deformation with volume conservation as the main aspect. In this method there is a trade-off between the computational time and realism. On the other hand, the nonlinear method allows us to solve for large deformations and large strains without much loss of accuracy. Since it is an iterative method, the accuracy of the results depend on the size of the system solved and the constraint of interactive-time in the application concerned.

Experimental results to show the behavior of the VDM model has been presented. We showed results from anisotropic behavior, stress distribution and large deformations like stretching, bending and twisting. Finally, we compared the VDM model with the classical FEM model for various types of deformations using a simple beam mesh.

Part II

Collision Detection

Chapter 4

State of the Art

4.1 Introduction

Interaction between virtual objects within a dynamic environment is inevitable. Even more, these interactions must be detected and treated at interactive rates. Hence, 3D collision detection techniques that are fast, robust and efficient are required. Since the user going to interact with the virtual environment, in ways which we do not know *a priori*, these techniques must be able to support various types of scenarios like contact, fictive interpenetration, large deformation, self-collision, topology modification, *etc.*

In the last few years, many techniques that have their roots in the field of computational geometry, computer graphics and robotics have been proposed as solutions. These techniques can be grouped into four different interference approaches [Jimenez et al., 2001]; space-time volume, swept volume, multiple checks and trajectory parameterization. The task of choosing which sort of technique depends normally on three important factors; **efficiency**, **robustness** and **flexibility**. It is interesting to note that the first three techniques mentioned require a **static interference test** (SIT) algorithm as a basic test.

For interactive animation and simulation, the best SIT algorithm is desired. This is currently a problem, because most of the existing algorithms treat specific cases under specific conditions. For example, some algorithms can only be applied to a set of points that define the convex hull of the object; which is a problem because most deformable bodies are concave. Hence, the best algorithm, fit for any case is almost impossible. A naive but possible solution, would be to have different implementations for various scenarios with a corresponding algorithm, but this is clearly not an option, computationally speaking. Hence, the next best solution, would be

to settle for a compromise between efficiency, robustness and flexibility. However, interactivity and accuracy must not be lost at the expense of this compromise.

In general, SIT algorithms are developed with the following characteristics:

- **Efficiency.** The efficiency of an algorithm is not necessarily a function of its complexity. It also depends on the frequency of its application. Thus, an intelligent strategy is required to decide when and where the interference test should be executed. Several strategies have been developed to this end and we shall see this in the proceeding sections. Some of the important techniques of this *selection* phase is by using lower time bounds, distance computation algorithms, hierarchical object representations, orientation based pruning criteria, and space partitioning schemes.
- **Robustness.** Robustness is another important criterion of algorithms in dynamic simulators. Anatomical meshes that are obtained from segmentation of 3D scanned images are often complex and degenerate. These meshes are often put through a *cleaning-up* phase to remove bad data like overlapping facets and unwanted points. The final mesh, although topologically correct, may still contain some undesirable features like approximately coplanar vertices, degenerate facets, *etc.* As a consequence, collision detection algorithms must be shaped not only by the application itself, but also by the challenging inputs arising in practice.
- **Flexibility.** In an application, flexibility largely delimits the kind of algorithms that can be applied. For example, efficient algorithms used for collision detection of convex rigid bodies becomes useless in medical simulation where the objects are deformable. Algorithms that are intended for deformable bodies based on a fixed bounding volume representation becomes useless when the deformable object undergoes large deformation. Hence, specific algorithms under specific conditions will not suffice. An algorithm which unifies various interactions under one data structure is required. This subsequently has a direct effect on memory usage and efficiency.

In this chapter, we will try to give a comprehensive overview of the general approaches pertaining to the SIT technique. Here, we note that since our work has been concentrated on treating polyhedral polygons soups (vertices, edges and facets) as the input data, we will only present previous works that use similar input data. In addition, we exclude all work that use dynamic and kinematic variables (we only consider geometrical data). We also assume without loss of generality that neighborhood topology can be calculated with respect to the input data. When possible, we will highlight some of the well-known algorithms that originate from

these approaches. In the end of this chapter, we will summarize and reiterate the need for a unifying algorithm, that exploits the trade-off between efficiency and computation time.

4.2 Software Models

In a general virtual environment where rigid or deformable objects are present, the general approach for collision detection in the animation or simulation domain is divided into three phases:

- **Broad Phase.** In this first optimization phase, pairs of objects which are *probably* in collision are selected. These pairs will then be tested for inteference. The algorithms used in this phase are mostly based on *spatial decomposition*.
- **Narrow Phase.** This phase does not compute the exact colliding entities for a pair of objects, but it gives a rough idea about the region within each object where a collision might exist. This region is called the zone of collision and the algorithms in the phase are mostly based on *object decomposition*. Again, this is another optimization phase.
- **Exact Phase.** This last phase obtains the exact colliding entities, if it exists. The algorithms in this phase check for intersection between two polygons.

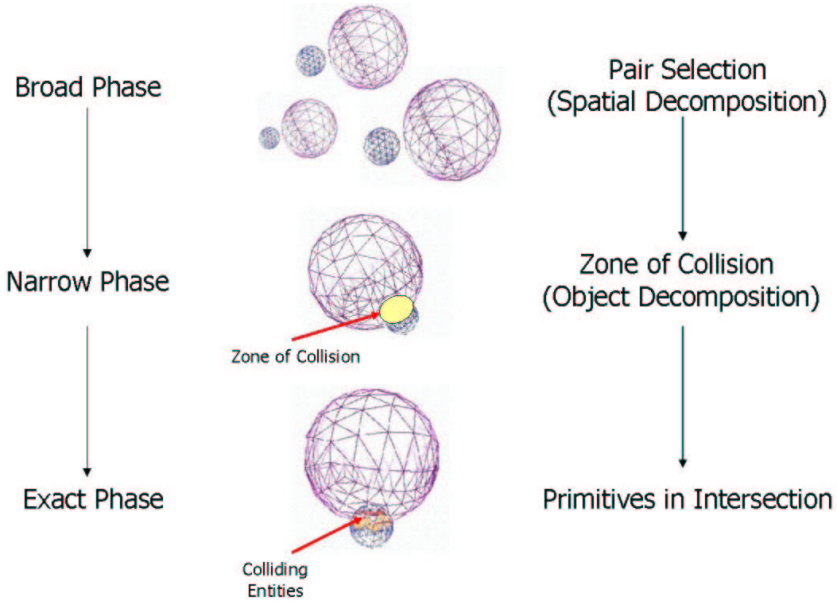


Figure 4.1: *The general approach for collision detection in the animation and simulation domains.*

4.2.1 Pair Selection

The aim of the broad phase in the collision detection algorithm is to quickly eliminate parts of the virtual environment where interaction could not be possible. In other words, the most probable pairs of objects that could be in collision are selected. The SIT algorithms in this phase normally use a spatial decomposition approach.

Voxel Approach. The algorithms based on this approach decomposes the *space* into uniform or adaptive grids or cells which are also known as *voxels* [Garcia-Alonso et al., 1994]. A common voxel approach that have been widely used are octrees [Hamada and Hori, 1996], [Derek and Gupta, 1996]. These cells generally never overlap, never extend beyond the space of their parent and their union always cover their parent space. An exception here is *Loose-Trees* [Ulrich, 2000] in which the octree cells are made to overlap.

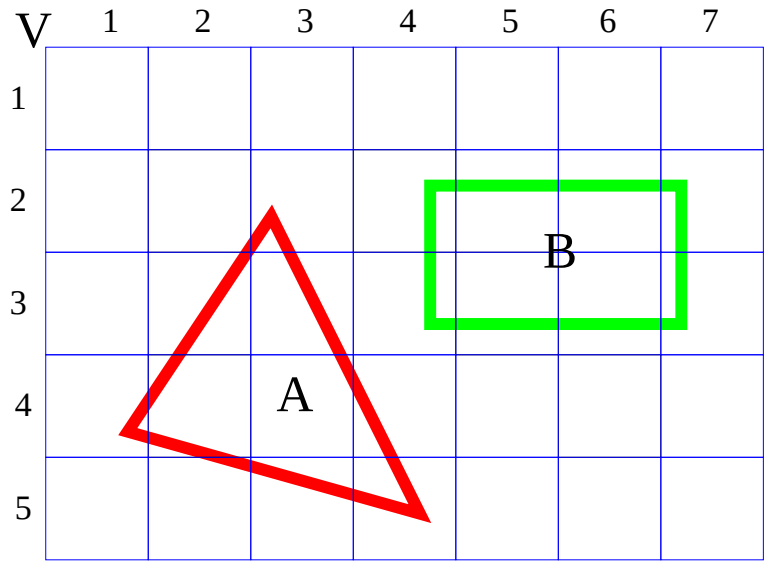


Figure 4.2: *Voxels in the grid are associated to objects A and B. If a common voxel is not found (like in this case), there can be no interaction, else a collision might exist and a further resolution check can be performed.*

For example, we can set up a grid of equal-sized boxes (see figure 4.2) over the workspace. For each box in this grid, we can store a list of the objects that intersect the voxel. Since a single object may be stored in the list of many voxels, the size of the resulting data structure could be large in comparison with the size of the input data. Processing a collision query is done by identifying common voxels between a

pair of objects. If a common voxel is found, then the pair of objects, associated with each of these voxels is further processed to determine interaction. An absence of a common voxel indicates no probable pair of objects.

Tree Approach. Another simple approach to representing spatial data is the binary space partition (BSP) tree [Naylor et al., 1990], [Thibault and Naylor, 1987]. It consists of selecting an arbitrary object from the virtual environment, and making it the root of the tree. Next, a plane spanned by this object is used to divide the rest of the virtual environment into two sets: front and back. Objects crossing this plane are split. This process is repeated recursively with the front and back sets, creating the front and back subtrees respectively. An incremental approach to construct this tree is also possible. The major disadvantage of this method is that BSP trees are hard to maintain for dynamic scenes.

Sweep & Prune Approach. This technique [Cohen et al., 1995] seems to be the most efficient for dynamic and multi body environments. In this algorithm, fixed bounding cubes (rigid objects) or dynamic bounding cubes (deformable objects) are used to globally bound the objects in the virtual environment. These bounding cubes are axis-aligned. Then a *sweep and prune* technique is applied to check overlap in the x,y and z axis. The presence of an overlap in all axes indicate collision. These overlaps are stored in a list and they are sorted at each time-step to exploit temporal coherence.

4.2.2 Zone of Collision

After a pair of objects has been selected by the broad phase for collision testing, the narrow phase is in charge of finding a region within these objects, where collision could be present. Distance computation can be used to any convex hull data set; a single convex rigid object or the convex decomposition of a concave rigid object. For deformable objects though, an underlying data structure is often used if interaction is to be detected accurately and efficiently. In the literature, the bounding volume approach is widely used as the underlying data structure to find the zone of collision. At the end of this stage, the list of pairs of *possibly* interacting primitives are passed on to the exact phase.

The aim of this section is to present a review of the following methods:

- Collision detection between rigid objects using distance computation.

- Collision detection between deformable objects using bounding volume hierarchies.

4.2.2.1 Collision Detection between Rigid Objects

Distance Calculation. The main idea behind this technique is to determine the closest features of two polyhedra, and then compute the euclidean distance between them. If this distance is positive, then there can be no interaction, otherwise, interaction can be confirmed. So far, three efficient methods have been proposed to compute the closest features. Two of them proceed by expanding an incremental representation in the direction of the minimum distance, while the third navigates along the boundaries of the polyhedra to find the closest features. These methods are described in the following paragraphs.

Dobkin and Kirkpatrick (DK).

Principle : In the DK algorithm [Dobkin and Kirkpatrick, 1990], the polyhedra P with vertices $V(P)$ is preprocessed to build a hierarchical representation $H(P) = P_i, \ldots, P_k, \quad i = 1, \ldots, k$. At each time-step, a closest point search procedure is executed. Every step in the closest points search procedure corresponds to a level in the hierarchical representation.

In this algorithm, preprocessing is done as follows:

- The lowest level P_1, is chosen to consist the original polyhedra and similarly the highest level P_k, is a d-simplex.
- At each level, vertices are removed such that $P_{i+1} \in P_i$ and $V(P_{i+1}) \in V(P_i)$ for $1 \leq i < k$
- The vertices $V(P_i) - V(P_{i+1})$ form an independent set (*i.e.* , are not adjacent) at each hierarchical level.
- The corresponding edge and face adjacency relationships are updated at each level.

In the first step, the closest points of two tetrahedras (the highest level in the hierarchy) are trivially determined. Now, if we consider the direction of the segment that joins the closest points found at a given step, two planes perpendicular to this direction that touch each polyhedron can be determined. These planes bound the zone where the next closest pair has to be searched for. The intersection of this zone with the polyhedra expanded at the next level may consist of either two simplices,

one simplex or the empty set. If the closest points are not the same as in the previous step, then at least one of them belongs to one of these simplices. Therefore, every search step is restricted to at most two simplices. Interference is detected implicitly when the separation between these simplices turns out to be null.

For two polyhedra with n and m vertices, the complexity of the preprocessing stage is $\mathcal{O}(n+m)$. Using this hierarchical polyhedral representation, since the number of steps is bounded by $log\ n * log\ m$, minimum distance computation can be performed in optimal $\mathcal{O}(log\ n * log\ m)$ time. Figure 4.3 shows an example of this algorithm applied to a simple polyhedra represented in this hierarchical representation.

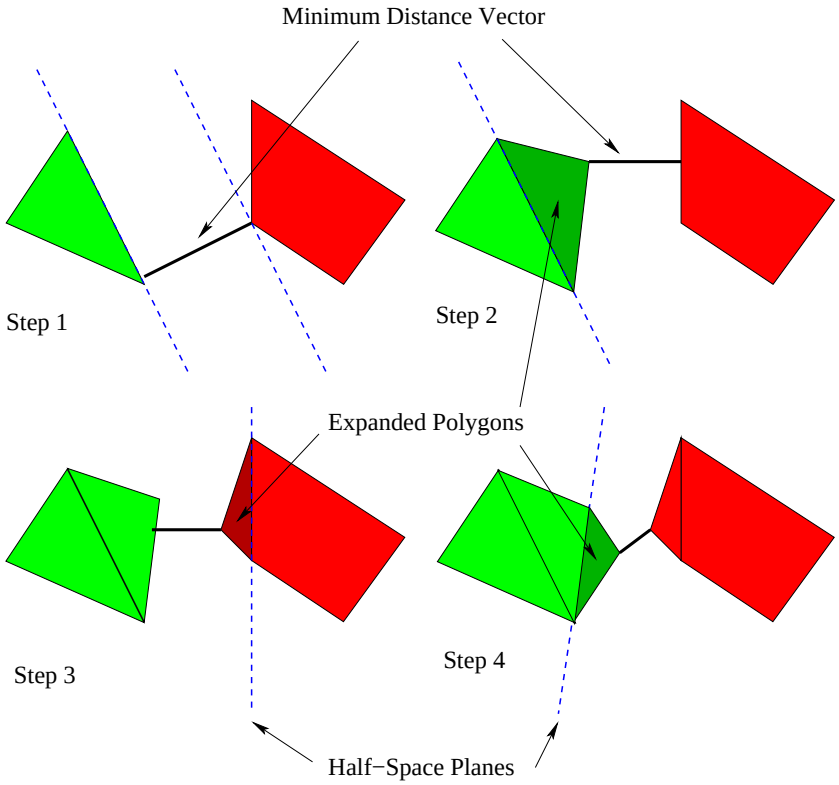


Figure 4.3: *In the DK algorithm, the closest features are found by searching only those parts of the hierarchy (expanded polygons) that intersects with the half-space defined by the planes obtained from the minimum distance vector at each level.*

Gilbert, Johnson and Keerthi (GJK).

Principle : This algorithm [Gilbert et al., 1988] calculates the distance between two convex envelope defined by two sets of points X and Y. It returns the value of this distance in the case where there exists a separation between the two envelopes, otherwise it gives an approximation of the interpenetration distance.

To understand this algorithm, we begin with some mathematical notations. The convex envelope of a set of points X, (C_X) is given by:

$$C_X = \sum_{i=1}^{n} \lambda^i \boldsymbol{x}_i : \boldsymbol{x}_i \in X,\ \lambda^i \geq 0,\ \lambda^1 + \lambda^2 + \ldots + \lambda^n = 1 \tag{4.1}$$

The euclidean distance between two convex envelopes is defined as follows:

$$d(C_X, C_Y) = \min\{|\boldsymbol{x} - \boldsymbol{y}|,\ \boldsymbol{x} \in C_X,\ \boldsymbol{y} \in C_Y\} \tag{4.2}$$

This distance is equal to the distance between the Minkowski difference of X and Y and the origin of the reference frame $\mathbb{O}$. The Minkowski difference $C_X \ominus C_Y$, is defined as follows:

$$C_X \ominus C_Y = \{\boldsymbol{z} : \boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y},\ \boldsymbol{x} \in C_X,\ \boldsymbol{y} \in C_Y\} \tag{4.3}$$

Hence, we have:

$$d(C_X, C_Y) = \min\{|\boldsymbol{z}| : \boldsymbol{z} \in C_X \ominus C_Y\} \tag{4.4}$$

The Minkowski difference of two convex polyhedra is a convex polyhedra. To compute the distance between $C_X \ominus C_Y$ and $\mathbb{O}$, we introduce the support function $\mathcal{H}$ defined for any set Z by:

$$\mathcal{H}_Z(\boldsymbol{\eta}) = \max\{\boldsymbol{z} \cdot \boldsymbol{\eta} : \boldsymbol{z} \in Z\} \tag{4.5}$$

where $\boldsymbol{z} \cdot \boldsymbol{\eta}$ is the projection of point $\boldsymbol{z}$ on axis $\boldsymbol{\eta}$. This means that the function $\mathcal{H}$ gives the furthest projection of a point in the direction of $\boldsymbol{\eta}$. The point $\boldsymbol{M}_Z$ ($\boldsymbol{z}$ of Z) which is closest to $\mathbb{O}$ is closest in one direction and furthest in the opposite direction. Hence this point satisfies the following:

$$|\boldsymbol{z}|^2 + \mathcal{H}_Z(-\boldsymbol{z}) = 0 \tag{4.6}$$

In other words, in the GJK algorithm, to calculate the distance between the convex envelopes of two sets of points X and Y, we compute the distance between $C_X \ominus C_Y$ and $\mathbb{O}$. The step that takes most of the processing time is the computation of $C_X \ominus C_Y$. This normally takes $n * m$ operations where n and m is the number of points in X and Y respectively. This complexity can be reduced by the following interesting property of h:

$$h_{C_X \ominus C_Y}(\eta) = h_{C_X}(\eta) - h_{C_Y}(\eta) \tag{4.7}$$

Now, using this property, the computation of $h_{C_X \ominus C_Y}$ only requires $n+m$ operations. Nevertheless, the number of iterations actually executed depends on the initial choice of Z_0. In the worst case, $n+m$ iterations are executed; which gives GJK a complexity of $\mathcal{O}(n+m)$.

Taking the polyhedra in figure 4.4 as an example for a compact set Z, to find the point closest to $\mathbb{O}$, we proceed as follows:

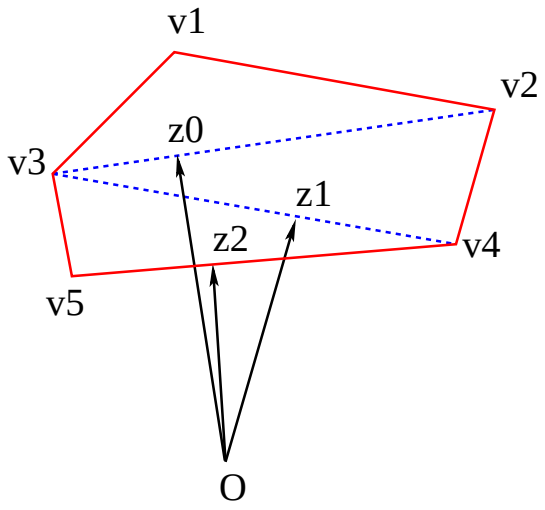


Figure 4.4: *In this example, the GJK algorithm converges in three iterations*

- We start with a initial set of points $Z_k = z_1, z_2, \ldots, z_m$. For $k = 0$, Z_0 is constructed with any three non-aligned points in $2D$ or any four non-coplanar points in $3D$.
- We then determine the point $\boldsymbol{\nu}_k$ of Z_k which is the closest to $\mathbb{O}$. This can be done is a very efficient way by using Johnson's algorithm [Johnson, 1987].
- If $|\boldsymbol{\nu}_k|^2 + \mathcal{H}_Z(-\boldsymbol{\nu}_k) = 0$, then $\boldsymbol{\nu}_k$ is the point of C_Z closest to $\mathbb{O}$.
- Else, $Z_{k+1} = \hat{Z}_k \cup S_Z$ where S_Z is a solution of $h_Z(\boldsymbol{\nu})$ and $\hat{Z}_k \subset Z_k$ is reduced by m elements. S_Z also satisfies $S_Z \in C_{\hat{Z}_k}$. We reiterate from the second step.

Several extensions have been done to the GJK algorithm for specific applications. Some of the main ones are described in the following:

Enhanced GJK (EGJK) : Cameron in [Cameron, 1997] showed that in the case where the two objects move continuously, for small changes, the new closest points will be in the neighborhood of the previously obtained closest point. In such cases, with some preprocessing done to the objects to gather information about neighborhood topology, it is possible that by using Z_k obtained from the previous iteration, the complexity can be reduced to $\mathcal{O}(1)$. The technique is referred to as *hill-climbing*.

Joukhadar and Laugier (JL) : This algorithm [Joukhadar et al., 1996] gives the negative distance[1] between two convex envelopes sets X and Y. This negative distance can be used as a measure of fictive interpenetration when X and

[1]The distance between two polyhedra is called positive when the two polyhedra do not intersect. When they do, their *negative distance* is the length of the smallest translation vector required to separate them.

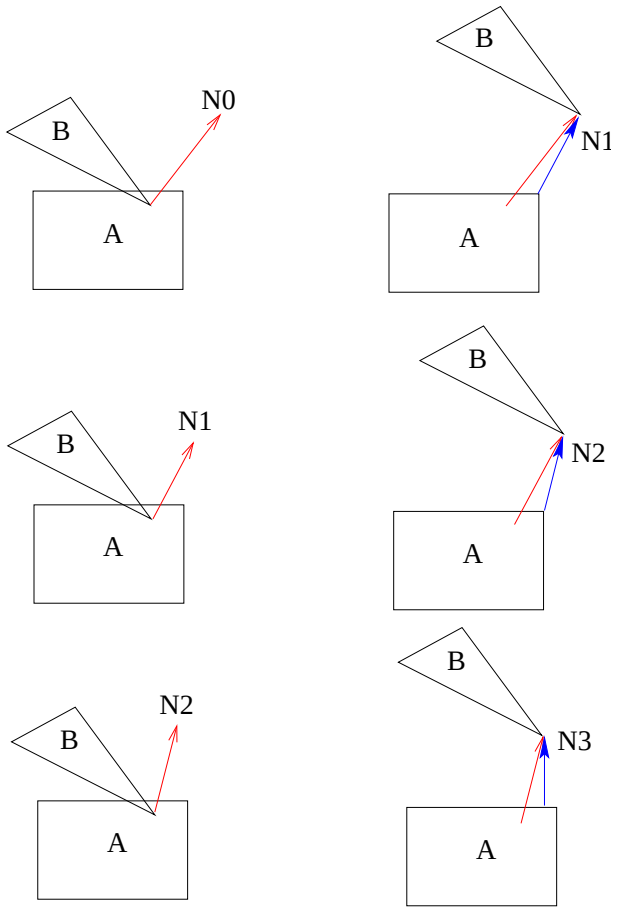


Figure 4.5: *To find the negative distance using the JL algorithm, we separate A and B using the direction $\boldsymbol{N}_{i-1}$ such that the convex envelopes are not in collision. Then we apply the GJK algorithm to obtain $\boldsymbol{N}_i$. In this example $\boldsymbol{N}_4 = \boldsymbol{N}_3$, hence the negative distance is given by $\boldsymbol{N}_4 - \boldsymbol{N}_3$.*

Y are in collision. According to this algorithm, to find the negative distance, we proceed as follows:

- We start by first separating X and Y using the last contact direction $\boldsymbol{N}$.
- We apply the GJK algorithm to obtain a new contact direction $\boldsymbol{N'}$.
- If $\boldsymbol{N} == \boldsymbol{N'}$, then the negative distance is $|\boldsymbol{N}| - |\boldsymbol{N'}|$. Else $\boldsymbol{N} = \boldsymbol{N'}$ and we reiterate from step 1.

At each iteration i, the value of $|\boldsymbol{N}_i|$ is closer to $|\boldsymbol{N}_{i-1}|$. Thus, this algorithm converges after a fixed number of iterations to obtain $\boldsymbol{N}$. The complexity of this algorithm is $\mathcal{O}\{k(m+n)\}$ where k is the number of iterations required for convergence. Figure 4.5 shows an example.

ISA-GJK : This algorithm [Van der Bergen, 1999] incorporates the GJK

algorithm with some modifications. The key argument for these modifications is that GJK has a tendency to generate simplices that are progressively more oblong *i.e.* , closer to being affinely dependent, as the number of iterations k increase. Hence the error in the minimum distance vector $\boldsymbol{\nu}_k$ cannot be easily estimated.

This algorithm estimates the error in $\boldsymbol{\nu}_k$ by maintaining a lower bound for Z. This lower bound is the signed distance from the origin to the supporting plane $\mathcal{H}_Z\{-\boldsymbol{\nu}_k, (\boldsymbol{\nu}_k \cdot \boldsymbol{\eta})\}$, which is given by the following:

$$\delta_k = \frac{\boldsymbol{\nu}_k \cdot \boldsymbol{\eta}}{|\boldsymbol{\nu}_k|} \tag{4.8}$$

This is a proper lower bound since for positive δ_k, the origin lies in the positive half-space, whereas $C_X \ominus C_Y$ is located in the negative half-space. But this lower bound may not be monotonic in k, *i.e.* it is possible that $\delta_j < \delta_i$ for some $j > i$. Hence the following is used as the lower bound which is often tighter than δ_k:

$$\mu_k = \max\{0, \delta_0, \delta_1, \ldots, \delta_k\} \tag{4.9}$$

Given ϵ, a tolerance for the absolute error in $|\boldsymbol{\nu}_k|$, the algorithm terminates as soon as $|\boldsymbol{\nu}_k| - \mu_k \leq \epsilon$.

Another important modification in this algorithm is the use of a test (called the Separating Axis Test) to speed up the terminating conditions of GJK. The basis of this improvement is that if two objects do not collide, we do not have to explicitly calculate the distance between them to ascertain this. We merely need to know whether the distance is equal to zero or not. Thus, the lower bound δ_k can be used as a terminating parameter to return non-intersection.

The lower bound is positive iff:

$$\boldsymbol{\nu}_k \cdot \boldsymbol{\eta} > 0 \tag{4.10}$$

i.e. $\boldsymbol{\nu}_k$ is a separating axis of X and Y. In general GJK needs fewer iterations to find a separating axis for a pair of non-intersecting objects than to compute an accurate approximation of $\boldsymbol{z}$:

$$\lim_{k \to \infty} \boldsymbol{\nu}_k = \boldsymbol{z} \tag{4.11}$$

In conclusion, besides requiring fewer iterations in the case of non-intersecting objects, this algorithm performs better because the value of $|\boldsymbol{\nu}_k|$ is not required to be calculated or initialized. The computation of $|\boldsymbol{\nu}_k|$ involves evaluating a square-root, which is an expensive operation. Hence, a single iteration of this algorithm is significantly cheaper than the original GJK. This package is also more robust because the error in the minimum distance vector is better quantified.

EPA-GJK : Van der Bergen in [Van der Bergen, 2001] has proposed another method to compute the negative distance between convex polytopes. He used an algorithm called the *expansion polytope algorithm* (EPA) to calculate the minimum translation vector required to separate two convex polytopes.

Lin and Canny (LC).

Principle : This algorithm [Lin and Canny, 1991] also calculates the two closest features of two convex polygon soup, X and Y with p and q features (vertex, edge and facet) respectively. The basic idea of this algorithm is the concept of a *voronoi region*. Every feature of a polyhedron is associated one such region, consisting of all the points that are closer to it than to any other feature. In each iteration, we apply a local criteria to verify if the characteristic features (the closest feature at the end of each iteration are called characteristic features) contain the closest points. This is done as follows:

- The closest point of the characteristic features in sets X and Y are first calculated.
- If the closest point in the characteristic feature of X is inside the voronoi region of the characteristic feature of Y, and the closest point in the characteristic feature of Y is inside the voronoi region of the characteristic feature of X, then the closest point of the characteristic features are indeed the closest points of X and Y.
- Else using neighborhood topology, a new characteristic feature in X and Y is found. The new updated features must be closer than the previous features and we reiterate from the first step.

At any one time, since the closest point of a characteristic feature is tested against the voronoi region of another characteristic feature, only three possibilities can exist; point/point, point/edge and point/facet. The voronoi region of a point, edge and facet of a polyhedra is shown in figure 4.6.

The complexity of this algorithm is linear with respect to the total number of features; in other words $\mathcal{O}(p+q)$. Again, as with GJK, this complexity depends on the initial choice of features. In the case where the closest features of the previous time-step are re-used and the objects have not moved much, this algorithm runs in $\mathcal{O}(1)$ time.

In conclusion, this algorithm has a similar complexity as with GJK. However, in contrast to GJK, this algorithm requires some preprocessing to gather information

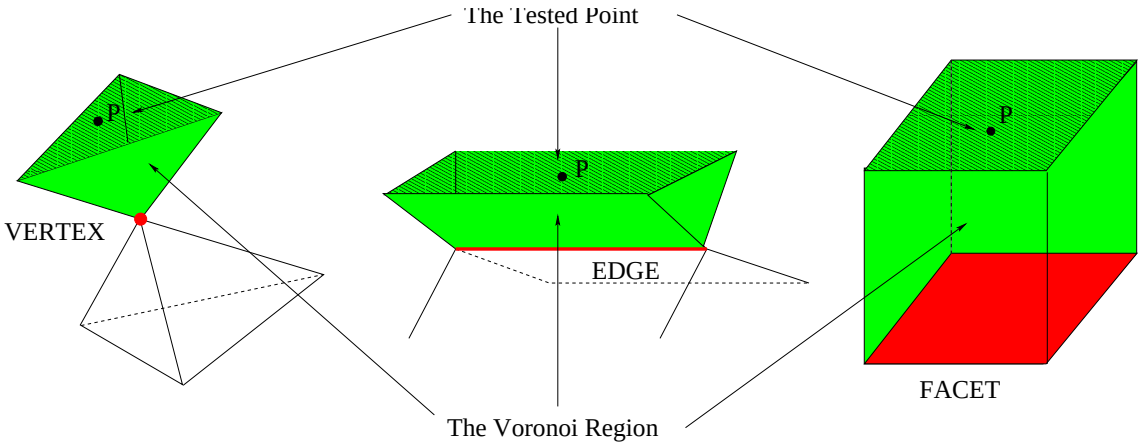


Figure 4.6: *Voronoi regions for a point, edge and facet of a convex polyhedra. The point P which is the closest point of the characteristic feature of another object is tested against these regions.*

about neighborhood topology. Furthermore, this algorithm only gives the two closest features in the case where there is a separation between the two convex envelopes. If the objects are in collision, this algorithm enters into an infinite loop. To remedy this problem, the notion of *pseudo voronoi regions* [Ponamgi et al., 1995] to define voronoi regions that are interior to the object has been used to determine if the objects intersect or not. Several extensions have also been done to the LC algorithm in the following implementation:

V-Clip : This algorithm [Mirtich, 1998] is a robust implementation of the LC algorithm with some minor changes. Instead of calculating the closest points of the two characteristic features at each iteration, some simple clipping operations together with scalar derivative tests are used as replacements. The code is simple and its implementation does not require the specification of any numerical tolerance. V-Clip also handles the case where the polyhedra interpenetrates. The complexity of this algorithm is similar to LC; $\mathcal{O}(1)$ time.

The general idea behind this algorithm is called *Voronoi Clipping* which is presented in an example shown in figure 4.7. In this example, once edge E is clipped (points P and Q are identified), the next step would be to determine if the closest point on E to F lies within $K \supseteq \mathcal{VR}(X)$, where K is the convex region spanned by the planes that define $\mathcal{VR}(X)$, and if not, how to update F (in this case, edges M and N are the potential candidates).

Updating the features is done by checking the signs of the derivative of the distance function between E and F noted as $D_{E,F}$ at P and Q. If $N \neq \emptyset$ and $D_{E,F}(P) > 0$ then F is updated to N, else if $M \neq \emptyset$ and $D_{E,F}(Q) < 0$ then F is

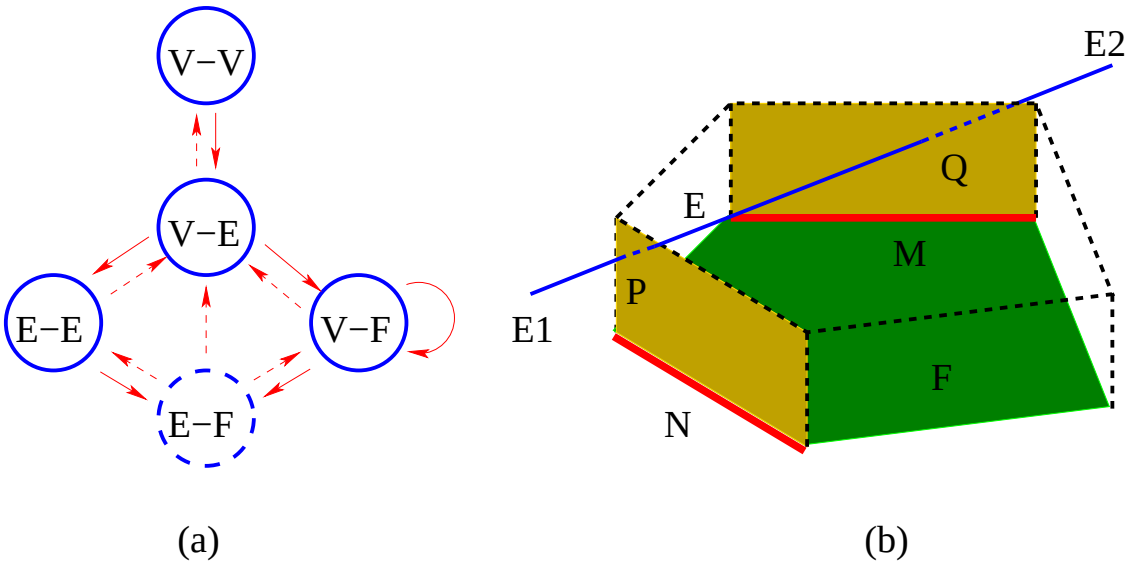


Figure 4.7: *(a) Possible state transitions of the V-Clip algorithm between vertex(V), edge(E) and facet(F). (b) The clipping of the edge(E) spanned from E1 to E2 against the facet F. Edge E intersects $\mathcal{VR}(F)$ at two planes at points P and Q respectively. These points also give the neighboring features for state transition, edges M and N. Taken from [*Mirtich*, 1998].*

updated to M. Similar distance functions exist for other possible combinations of features. For degenerate cases where the distance function is not differentiable, the algorithm simply reports penetration indicating intersection between edge E and the other feature.

We note that not all possible combinations appear in this algorithm. Geometrically speaking, for convex polyhedra, the minimum distance can always be expressed between two features of type $V-V$, $V-E$, $V-F$ and $E-E$. $E-F$ can always be reduced to features of a lower dimension. Since the solution for a vertex against voronoi planes of V,E or F is trivial, the only difficulty is when an edge is checked against a V,E or F. An F almost never appears (see figure 4.7).

DEEP : Kim *et al.* in [Kim et al., 2002b] has also improved on the LC algorithm to calculate the negative distance. This algorithm incrementally seeks an optimal solution by walking on the surface of the Minkowski sum (of the objects) which is constructed by using Gauss maps.

Lower & Upper Bounds. This technique is applicable to any two general polyhedras that is represented by a *bounding volume hierarchy*. In [Quilan, 1994] a *sphere-tree* is used while in [Johnson and Cohen, 1998], [Johnson and Cohen, 1999]

an *LUB-Tree* is used. The main advantage here is that a convex decomposition of the objects are not required. The key principle in these algorithms is to descend the bounding volume hierarchy using lower and upper bounds on the distance as the criteria without exploring all the nodes in each hierarchy.

4.2.2.2 Collision Detection between Deformable Objects

Bounding Volume (BV) Hierarchy Approach. A bounding volume encloses the virtual object of interest. For example, it can be an sphere or a box enclosing a primitive polygon, a set of polygons or a complete object. The collision problem is simplified to determining whether the two bounding volumes (rather than the two objects) overlap. When the complexity of the object increases a bounding volume hierarchy is used. A volume hierarchy is a tree of bounding volumes, such as spheres or boxes. Each volume encloses a or several geometric primitive. A set of these volumes, called a *parent*, spatially encloses all the geometric primitives covered by its leaf nodes. Generally, each bounding volume in the hierarchy is optimized in terms of volume, surface area, diameter, *etc.* with the aim of having a tight fitting around the enclosed primitive. Depending on the choice of design, the leaves of a BV tree may contain a unique geometric primitive, or a collection of primitives. A survey regarding these aspects have been made in [Ehmann and Lin, 2001].

The tightness or fit of a bounding volume has a dramatic effect on the overall effectiveness of the bounding-volume hierarchy. A tighter fit usually results in smaller number of overlap tests while requiring a slightly more expensive overlap test. The overlap test used will depend on the type of the hierarchy; axis-aligned bounding boxes (AABBs) [Van der Bergen, 1997], arbitrary oriented bounding boxes (OBBs) [Gottschalk et al., 1996], or sphere hierarchies [Hubbard, 1996]. Since the overlap test for spheres is trivial, we concentrate our discussion on boxes as BVs.

There are two major decisions to be made when using a box as a bounding-volume; choosing the orientation of the box to get the best fit and choosing how to split the children. OBBs are normally oriented using the statistical principal components as the axes (eigenvalues of the covariance matrix of the vertex distribution). On the other hand AABBs are oriented parallel to the local axis frame (see figure 4.8). The partitioning of the children depends on the choice of construction; top-bottom or bottom-top. Normally, bottom-top hierarchy building strategy is considered flexible and top-bottom is considered efficient, but various sibling merging rules and child partitioning exist that makes this comparison very qualitative. As a general rule of thumb, the efficiency of a BV hierarchy is evaluated using the

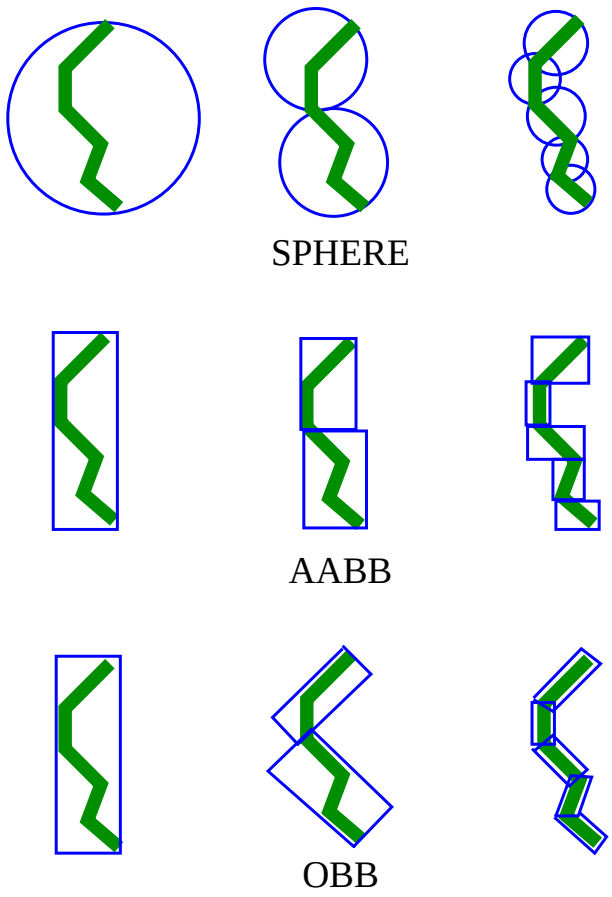


Figure 4.8: *Bounding volume hierarchies. Bounding spheres have the worst fit but the fastest overlap test, AABBs have better fit but more overlap tests are required, OBBs have the best fit and require the least number of overlap tests. We note that both the AABB and OBB share the same cost of a overlap test.*

following criterion [Gottschalk et al., 1996]:

$$\mathcal{T} = (\mathcal{N} \times \mathcal{B}) + (\mathcal{P} \times \mathcal{C}) \tag{4.12}$$

where $\mathcal{T}$ is the total cost function for interference detection, $\mathcal{N}$ is the number of bounding volume pair overlap tests, $\mathcal{B}$ is the cost of testing a pair of bounding volume for overlap, $\mathcal{P}$ is the number of primitive pairs tested for interference and $\mathcal{C}$ is the cost of testing a pair of primitives for interference. To lower $\mathcal{N}$ and $\mathcal{P}$, the bounding volume should fit the original model as tightly as possible. To lower $\mathcal{B}$, the algorithm used for the box/box overlap test should be optimal. $\mathcal{C}$ is minimum if the best algorithm for facet/facet interference is used. An interesting result concerning the choice of algorithm for the box/box overlap test is given in table 4.1.

But a bottle neck is the continuous deformation of deformable bodies. This implies that the precomputed data has to be recalculated as and when necessary

Separating Axis Test (SAT)	Closest Feature (LC/GJK)	Linear Programming
$5-7\ \mu s$	$45-105\ \mu s$	$180-230\ \mu s$

Table 4.1: *Performance of Box Overlap Algorithms. Taken from [Gottschalk et al., 1996].*

online. A direct consequence is the increased computation burden that slows down the simulation to a point that interactive-time requirements cannot be satisfied.

Not much work has been significantly reported as remedies to this problem. The most recent propositions to solve this problem comes from [Van der Bergen, 1997] and [Larsson and Moller, 2001]. In both these articles, a bounding volume hierarchy of type AABB is used as the underlying structure of the deformable object. It is reported that the cost of updating an OBB tree seems much more costly than an AABB tree (about 3 times more). Hence AABB is a suitable choice when issues related to updating the hierarchy is concerned. The two main techniques involved in modifying a hierarchy is as follows:

- **Rebuilding.** This process is the entire reconstruction of the tree from parent to child (top-bottom) or child to parent (bottom-top).
- **Refitting.** This process only changes the affected bounding volumes. The changes are then propagated from node to node until the root or leaf is reached.

Algorithm of Van der Bergen.

Principle : Van der Bergen in [Van der Bergen, 1997] proposes a fast method to update the AABBs using the refitting technique. It is reported that a top-bottom approach is used to build the hierarchy. This refitting algorithm can be summarized by the following steps:

- The bounding boxes of the affected leaves are first recomputed.
- Each parent of these bounding boxes is then recomputed in a bottom-top order.

Refitting in an AABB tree can be sometimes disadvantages. Due to the relative position changes of primitives after deformation, the boxes in a refitted tree may have a higher degree of overlap than the boxes in a rebuilt tree. The overhead of recursive function calls are minimized by using an array of nodes where the internal child node's index number is greater than its parent's index number. In this way, the internal nodes are refitted properly by iterating over the array in reversed order.

A comparison of performance shows that AABB trees are refitted and rebuilt in less that 5% and 33% respectively of the time taken to rebuild OBB trees.

Algorithm of Larsson & Moller.

Principle : In [Larsson and Moller, 2001] the updating process for an AABB tree is slightly improved. The basic idea is similar to that of [Van der Bergen, 1997] but instead of doing a bottom-top update sequence, an intermediate level in the hierarchy is chosen as the starting point for the update process. Hence, to update the necessary bounding volumes in the hierarchy after deformation, a combination of an *incremental* bottom-top and a *selective* top-bottom is used. This is referred to as an **hybrid-update**.

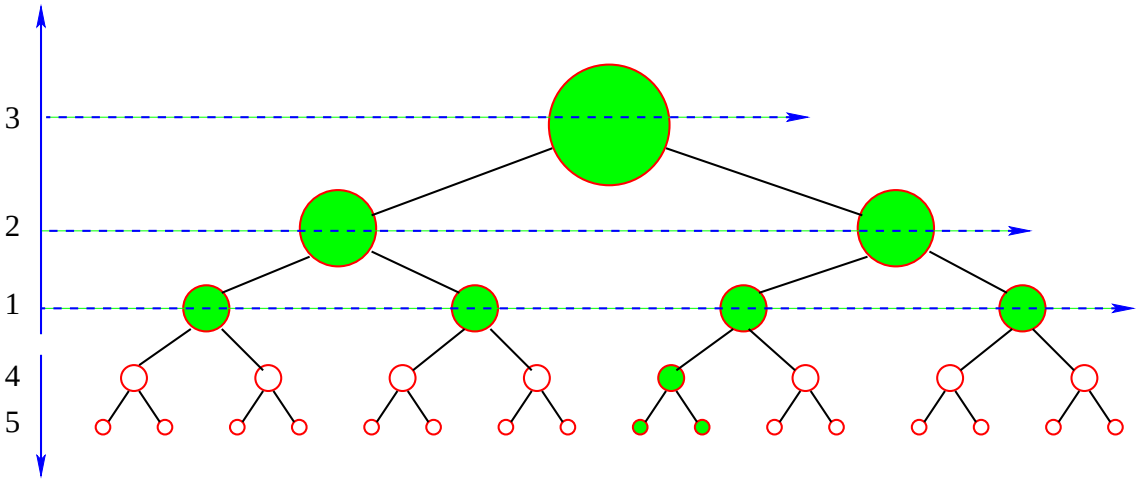


Figure 4.9: *Example of a hybrid tree update method, combining the bottom-top and top-bottom strategy. Taken from [Larsson and Moller, 2001].*

This algorithm can be summarized by the following steps:

- For a tree with depth n, the first $n/2$ levels are updated using the bottom-top strategy.
- The remaining levels are examined during collision traversal where, when non-updated nodes are reached, they can be either updated top-bottom as needed or a specific number of levels in the child trees can be updated bottom-top.

An example is shown in figure 4.9, where the three top-most levels (1 to 3) are updated bottom-top and the remaining levels (4 and 5) are updated on the fly as and when necessary. In this example, there are a total of 31 nodes, but only 11 of them are updated (those that are shaded).

Other Works.

Some of the other related works that have some relation to deformable objects are [Hubbard, 1996], [Quilan, 1994], [Meseure and Chaillou, 1997] and [Davanne et al., 2002] which are based on spheres as the bounding volume. In these works, not much attention has been given to cases where the hierarchies are updated except in [Davanne et al., 2002], where a combination of voxel grids and spheres are used as the underlying data structure.

4.2.3 Colliding Entities

In the exact phase, the SIT algorithm that is used to obtain the colliding entities is the intersecting test. Interaction can be detected if at least two primitives intersect. We shall consider three pairs of primitives; sphere/sphere, box/box and triangle/triangle. These pairs are considered because they appear in the final *checking-phase* of most of the collision detection algorithms which is explained in the next section.

Sphere/Sphere.

Principle : The sphere-sphere intersection is the most simplest and fastest. Given two spheres A and B with radius r_A and r_B respectively, intersection is confirmed when:

$$|\boldsymbol{p}_A - \boldsymbol{p}_B| < r_A + r_B \tag{4.13}$$

where $\boldsymbol{p}$ denotes position.

Triangle/Triangle.

This test is important because most of the geometrical data that is obtained from MRI segmented images are surface based triangulaire meshes. Furthermore, rendering hardwares are generally tailored for triangles, which is another reason for the common use of triangulaire meshes. Hence, an efficient intersection test between triangulaire primitives which are commonly referred to as *facets* is essential. Currently, the most efficient intersection test that is used between two facets to determine interaction is the algorithm by [Moller, 1997].

Principle : The basic idea in this algorithm is to determine the existence of a line that intersects the two triangles. If such a line is found, then the intervals of intersections are compared to determine if the triangles intersect. Figure 4.10 will help clarify the concept of intersection of intervals on L_{AB}.

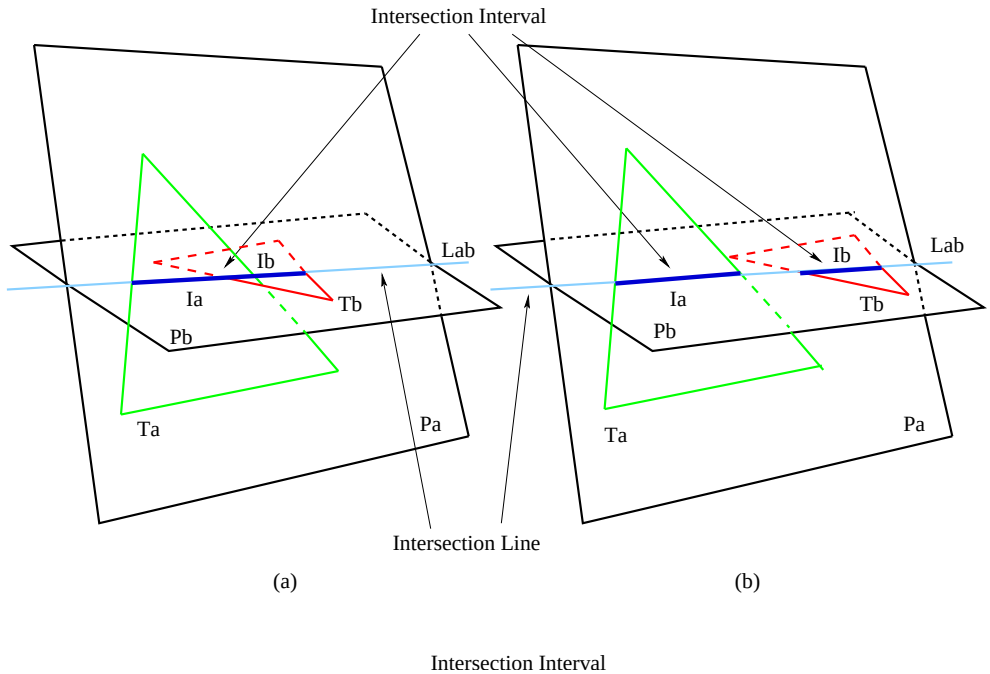


Figure 4.10: *Collision detection between triangle T_a and T_b on planes P_a and P_b respectively. (a) The intervals I_a and I_b overlap, hence intersection. (b) No overlap detected on the intersection line L_{ab}, hence no intersection.*

In this algorithm, the procedure to determine intersection between facet A and B is as follows:

- Compute plane equation of facet B, P_B.
- If all points of facet A are on the same side of P_B, exit.
- Else, compute plane equation of facet A, P_A.
- If all points of facet B are on the same side of P_A, exit.
- Else, $\exists$ an intersection line L_{AB} that passes through A and B.
- Project L_{AB} onto largest axis and compute interval I_A and I_B that defines $L_{AB} \cap (A \cup B)$.
- If $I_A \cap I_B \neq \emptyset$, A and B intersect, else A and B do not intersect; exit.

Box/Box.

The box/box intersection test is another important test. Hierarchies built using bounding boxes require this test to descend the bounding volume tree. The fastest algorithm for this procedure is the Separating Axis Test (SAT) [Gottschalk et al., 1996].

Principle : The basic idea in this algorithm is the detection of a separating axis between the two boxes. It can be shown that 15 axial projections are sufficient to determine the contact status of two arbitrarily positioned and oriented boxes. For these 15 axes, in the worst case about 200 operations are executed. Note that during execution, not all 15 axes are checked. The algorithm exits upon the detection of the first separating axis.

4.3 Implementation Issues

In the previous sections, we saw how the basic SIT algorithms have been implemented in various routines and applied to solve different problems. Another important aspect that is interesting to consider is the cost involved in implementing these procedures. We discuss and make comparisons with respect to three criteria:

- Complexity
- Memory Storage
- Frame Coherence

4.3.1 Complexity

The complexity involved in most the mentioned applications is in the order of $\mathcal{O}(N)$ where N is the number of elementary primitives. An exception is the application to calculate distance between rigid convex objects or decompositions of rigid convex objects where the complexity is reduced to constant time [Gilbert et al., 1988], [Lin and Canny, 1991]. Distance calculation between concave objects that are represented by bounding volume hierarchies such as the work in [Quilan, 1994] only give a worst case performance (screening all the nodes). This is because the performance of this algorithm depends on where *actually* the points that realize the minimum distance is located and how long it will take to descend each hierarchy tree to arrive at these respective points. Supposing that the best SIT algorithm is used to determine descent, then the complexity of the algorithm becomes a function of the way these hierarchies are constructed and the resolution of each hierarchy, which is entirely user dependent.

4.3.2 Memory Storage

Distance calculation between convex objects using [Gilbert et al., 1988] requires the least memory. The only requirement is the points that make up the convex hull of the object. On the other hand, the algorithm by [Lin and Canny, 1991] and the modifications proposed by [Mirtich, 1998] require additional topology information to be stored and available as the minimum input data. With the current progress in memory manufacturing and mass production, the cost and size of memory required may be negligible, but nevertheless for comparison sake, it is a point worth noting. For all the other type of applications, BV's are required to be stored. Assuming a fixed number to nodes $\aleph$ in the trees to facilitate comparison, sphere trees require only the center and radius to be stored for each node. The same applies for the AABB tree where the center and diagonal of each box is stored. Hence, for both cases, storage is of order $\mathcal{O}(\aleph)$. On the other hand, OBB trees requires an addition local 3x3 orientation matrix R to be stored for each node which leads to a storage of order $\mathcal{O}(\aleph + \aleph * R)$. For large-scale models or environments, this amount of storage can be significantly large. For each parent, pointers to their respective children are required for descent. Memory storage for this stage is of the order $\mathcal{O}(\aleph * Q)$ where Q is the number of children of each parent. In some cases, an additional pointer to the parent is also required. The total cost of memory in this stage is of order $\mathcal{O}(\aleph)$ for this case.

4.3.3 Frame Coherence

Frame coherence refers to the fact that if the simulated environment does not change much between one time-step to another, then there exists a coherence between them. This coherence can be exploited in the simulated frames for detecting interaction. Algorithms to calculate distance [Gilbert et al., 1988], [Lin and Canny, 1991] have made provisions to exploit frame coherence in the case when the objects move very little. The assumption here is that between two time-steps, the closest features either remain the same or lie in the neighborhood of the previous closest features. Hence, at each iteration, the previous closest features are used as the starting point of these algorithms which enables them to converge much faster. In [Van der Bergen, 1999], the fact that the computation of the distance is not necessarily required to determine interaction is exploited for frame coherence. In this case, instead of using closest features, the separating axis is *cached* at each iteration.

4.4 Hardware Models

Alternatively, an approach for detecting interaction between objects is by using the graphics hardware.

4.4.1 Z-Buffer Comparisons

The first proposed idea is if two objects share some common *drawn* volume, then these objects are in collision. This drawn volume is measured by using the *Z-buffer* of the graphics card. The Z-buffer contains a measure of depth with respect to the viewing plane *i.e.* the notion of object volume. Lombardo in [Lombardo et al., 1999] proposed such a scheme for deformable and rigid objects of rectangular shape. It makes use of the OpenGL hardware functions to detect the polygons within a bounding box. This approach may be very fast due to hardware acceleration but it imposes very strict conditions on the type of possible interactions: the shape of one of the objects is limited by the type of bounding box that OpenGL can construct. Besides the method is highly dependent of the configuration of the system, and performance can be lost while changing to another PC-class/MS Windows graphics environment [Vepstas, 1996].

Another similar idea but applied to two oriented bounding boxes is presented in [Baciu et al., 1998]. In this work, the concept of *minimal overlapping region* (MOR) is presented. A soft assumption that the objects are convex is used. Hence, concave objects are assumed to be decomposed in convex subobjects. In this algorithm, the frame buffer and Z-buffer are examined to determine interaction. An important part of this algorithm is the possibility of finding the contact points by finding the overlapping regions of bounding boxes.

4.4.2 Distance Fields

A second idea that has been recently proposed is the use of distance fields generated by voronoi diagrams [Hoff et al., 2001], [Hoff et al., 2002]. A distance field specifies the minimum distance to a shape for all points in the field. In this algorithm, an image-space proximity query is used to detect interactions between concave objects. These queries operate on a uniform grid of sample points in regions of space containing potential interactions. The graphics hardware pixel frame-buffer is used as the grid and the queries become pixel operations. As such, this algorithm gives dramatically different results for different pixel resolution which is clearly a handicap in accuracy.

On the other hand, recent contributions handle multi-object and self-collisions by making various manipulations to the stencil, frame and Z-buffers along with the *accelerated* computation of distance fields. It is also reported that this algorithm can return the closest points, magnitude and direction of penetration and separation. Nevertheless, to avoid excessive load, a geometric localization step (like in the coarse phase previously mentioned) is used to localize regions of potential collision or as a trivial rejection stage. Hence, the system is rather a hybrid approach; combining software and hardware.

4.5 Summary

In this chapter, we have presented some important aspects of collision detection. By considering no *a priori* information on the **motion** of the virtual objects, we believe SIT is the optimal method to detect interference. To efficiently apply the SIT algorithms for collision detection, three phases are executed; *broad*, *narrow* and *exact*.

In the broad phase, pairs of objects to be tested for collision are selected. The main idea here has been to use a spatial decomposition approach. An important algorithm in this phase for dynamic and multi body environment is the Sweep-Prune technique.

We then presented two important techniques in the narrow phase category; distance calculation and bounding volume interference. The main aim here is to zoom directly to the most probable areas where collision could be present. Distance computation can be effectively used between any two convex hull. Several important algorithms to this end and their respective improved variations were described in detail, namely DK, GJK, EGJK, ISA-GJK, JL, EPA-GJK, LN, V-Clip and DEEP. These algorithms exploited the convexity of the objects to either decide interaction by using minimum distance or by detecting the absence of a separating plane. In these algorithms, the main concern was robustness and speed. However, for deformable objects, an underlying data structure is often used to speed up collision detection. The common form of data structure used are bounding volumes. We have concentrated on the details about using sphere, axis-aligned boxes and oriented boxes as the bounding volume. Two algorithms that have been used to update this hierarchy have been presented.

In the exact phase, we presented algorithms for intersection test between spheres, boxes and triangles. These, we believe are the common elementary primitives that appear in the geometrical model of an object.

We also presented some hardware models for collision detection. These techniques are very graphics hardware and operating system based. Currently, hardware based models have not been used extensively in the simulation and animation field.

But medical simulators require these routines be adaptable for deformable bodies such that they are extremely efficient, robust and optimized. We have seen that currently the main idea here is to update the underlying data structure. To this effect, we concluded by stressing that for deformable objects, the AABB hierarchy is best suited (less compact than OBB but much easily updated) and presented some algorithms that update this type of hierarchy. Furthermore, the currently available packages (see appendix A) which have implemented some of these routines do not consider concave objects or deformable objects (see figure 4.11). No **single** library has been specifically designed for medical simulators taking into account the constraints of deformable objects. Another point worth noting is that no library has considered collision treatment as a simulation stage following collision detection for deformable objects. The is partly because collision treatment for deformable objects is still a complex field.

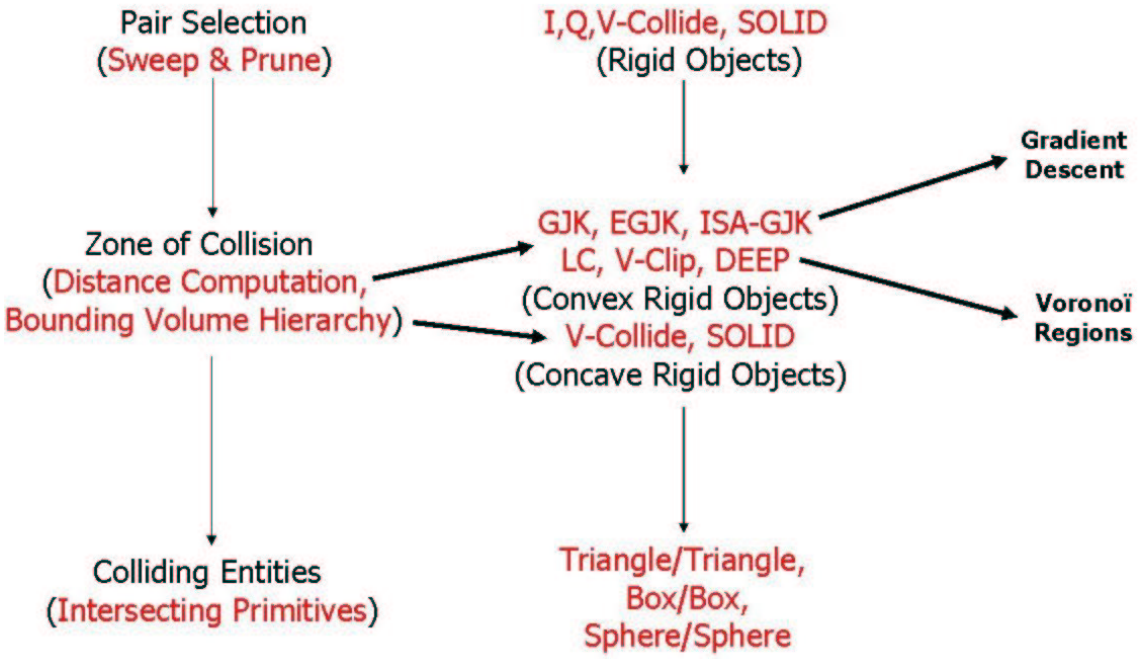


Figure 4.11: *The exisiting collision detection libraries and their characteristics (note that this list is not exhaustive).*

In the next chapter, we will present the requirements of a collision detection library for medical simulators. Our considered objects are all concave and deformable. Our aim is to propose a single underlying data structure that can handle efficiently

interference of deformable objects. Another aim is to have routines that are robust and efficient.

Chapter 5

Collision Detection for Medical Simulators

5.1 Introduction

In the previous chapter, we presented the state of the art in collision detection limited to the case of static interference tests (SIT). We concluded that no single collision detection library is currently available for medical simulators where the main concern is interaction between deformable objects. In this chapter, we focus on presenting the algorithms for collision detection in medical simulators.

Medical simulators imposes somewhat a different virtual environment. Firstly, the objects are deformable and are continuously deforming. Hence, algorithms that are efficient are required. Secondly, the resolution of each object in terms of the number of facets is very high; very often in the thousands. In this case, optimized algorithms are required. Thirdly, since the nature of these applications are *life* related, robust algorithms are desired.

In such an application, the following combination of pairs of objects that needs testing can be found:

- Rigid objects - for example, between surgical tools.
- Rigid and deformable objects - for example, between a surgical tool and a virtual organ.
- Deformable objects - for example, between virtual organs.

Besides this geometrical restriction, no additional hypothesis has been made on the trajectory or velocity of the object. The movement of the objects in a medical simulator cannot be predicted with enough accuracy as the motion of the surgeon

and the behavior of the organs are very complex in nature. Given two objects to be tested, the SIT routines in a medical simulator must be able to handle the following situations:

- Collision detection between two rigid (convex or concave) objects.
- Collision detection between a rigid (convex or concave) object and a deformable object.
- Collision detection between two deformable objects.

We have mentioned before that for rigid convex objects, collision detection can be done by calculating the distance between the polygonal meshes. For rigid concave objects, we assume that the object can be represented by its convex decomposition. Thus distance computation can be applied to this convex decomposition. The best method so far for collision detection between deformable objects is to detect interference between primitives. This is done efficiently by applying the *coarse* and *exact* phases as mentioned in the previous chapter.

After collision detection, collision treatment follows. But this stage requires detecting all the primitives that are in contact. Primitives in contact are the primitives that make up the volume of fictive interpenetration. We call these primitives *contact elements*. Most of the collision detection libraries only report a yes or no to the interference state of two objects, the do not provide the contact elements that are required in the collision treatment phase.

In this chapter, we detail out the algorithms for collision detection in medical simulators. In such an application, the frequency of collision detection is very high, for example the organs in the human body are almost always in contact. Thus, these algorithms have to be robust, efficient and optimized. The following algorithms will be presented:

- Distance computation of rigid convex objects.
- Distance computation of rigid concave objects.
- Collision detection of deformable objects.
- Contact localization for collision treatment.

We have implemented all these algorithms under a single library for collision detection in medical simulators which is called **ColDetect** (see appendix B). This library uses the AABB hierarchy as the underlying data structure. This hierarchy has been constructed bottom-top with volume minimization as the criteria to combine the boxes at each level.

5.2 Distance Computation of Convex Objects

The algorithm [Sundaraj et al., 2000] presented in this section computes the minimum distance between two convex envelopes of objects X and Y.

Problem. Although algorithms that compute the minimum distance exist, there are still ares where we can improve aspects like efficiency and robustness.

Proposed Solution. The main idea behind this algorithm is to descent the distance gradient. This is done by alternatively forming simplexes (point, edge or facet) of each object such that at each iteration, the formed simplexes are more closer (by computing the distance) to each other. The simplexes are updated at each iteration by using the *support function* and *optimized local methods* are used to compute the distance between simplexes. Since the distance gradient has a single local minimum, convergence is expected after several iterations.

5.2.1 Description of the Algorithm

General Procedure. In this section, we will explain how our algorithm is executed. We assume that the convex envelopes of objects X and Y, both described in a global frame, are initially separated. We proceed as follows:

- We begin with two random points, one from each object. They will be the initial simplex denoted as S_X and S_Y and the initial witness point denoted as W_X and W_Y.
- The support vector, $SV_{XY} = W_X - W_Y$ is constructed. Using the support function, by sweeping in the direction of SV_{XY}, a support point that forms a new simplex in Y is obtained, S_Y.
- The optimized local methods are then used to solve the minimum distance between W_X and the new S_Y. This minimum distance is denoted as D_{ub} and has a upper-bound norm d_{ub}.
- The features that realize the minimum distance are updated and the points that give the minimum distance are updated as the new witness points W_X and W_Y.
- Again, using the support function, by sweeping in the direction of $-D_{ub}$, a support point in X is obtained. The vector between this support point and a point in S_Y is constructed. The dot product between this vector and D_{ub} is denoted as D_{lb} and has a lower-bound norm d_{lb}.

- If $(d_{ub} - d_{lb}) < (\epsilon * d_{ub})$, then D_{ub} is indeed the minimum distance vector, and we exit the algorithm.
- Else, $SV_{YX} = W_Y - W_X$ is constructed with the new witness point W_Y and the process is repeated in object X from the second step.

We note here that with some neighboring adjacency information of the objects, the cost of computing a support point of a convex polyhedra can be reduced to almost constant time. This technique is well known as *hill-climbing* and has been implemented in our algorithm. Our experiments were carried out with this option available. With this option, our algorithm has a complexity of order $\mathcal{O}(1)$, *i.e.* constant time when the witness points are reused at each call.

Support Function. The support function $\mathcal{H}$ defined for any set Z is given by:

$$\mathcal{H}_Z(\boldsymbol{\eta}) = \max\{\boldsymbol{z} \cdot \boldsymbol{\eta} : \boldsymbol{z} \in Z\} \tag{5.1}$$

where $\boldsymbol{z} \cdot \boldsymbol{\eta}$ is the projection of point $\boldsymbol{z}$ in the direction of $\boldsymbol{\eta}$. $\boldsymbol{\eta}$ is called the *support vector*. This means that the function $\mathcal{H}$ gives the furthest projection of a point in the direction of $\boldsymbol{\eta}$. The result of the furthest projection gives us the *support point*.

Optimized Local Methods. Since the only possible simplex pairs that appear in the MS algorithm is point-point, point-edge and point-facet, we only need to formulate efficient and robust methods to solve them. The advantage of using optimized methods is that we arrive at the solution for each possible feature pair with at most performing one division. Except for the point-point case, the others need detailed analysis. We begin with the analysis of point-point.

- *Point-Point :* The divisionless distance function Q, between point P_0 and P_1 is given by:

$$Q = |P_0 - P_1| \tag{5.2}$$

- *Point-Edge :* Let the point be P. The edge, E can be expressed as a line segment by:

$$E(t) = B + tM \tag{5.3}$$

where B is a point on the line, M is the line direction with $t \in [0, 1]$. If $|M|^2 \leq \epsilon$, where ϵ is a user defined value to denote minimum edge length,

then $t = 0$ and we consider the edge as a point and use point-point analysis. Otherwise, let:

$$\begin{aligned} t' &= M \cdot (P - B) \\ t'' &= M \cdot M \end{aligned} \tag{5.4}$$

The distance function $Q(t)$, from point P to the edge E is given as:

$$Q(t) = \begin{cases} |P - (B + M)| & \text{if } t' > 1,\ t'' < t' \\ |P - B| & \text{if } t' \leq 0 \\ |P - (B + \frac{t'}{t''}M)| & \text{otherwise} \end{cases} \tag{5.5}$$

The operation $\frac{t'}{t''}$ is left to the end and is only performed if necessary.

- *Point-Facet :* The problem of finding the minimum distance between a point P and a facet F, defined as:

$$F(s,t) = B + sE_0 + tE_1 \tag{5.6}$$

where E_0 and E_1 are two edges of the facet, $(s,t) \in R = \{(s,t) : s \in [0,1],\ t \in [0,1],\ s + t \leq 1\}$ is obtained by computing $(s',t') \in R$ corresponding to a point on the facet closest to P. Let $a = E_0 \cdot E_0,\ b = E_0 \cdot E_1,\ c = E_1 \cdot E_1,\ d = E_0 \cdot (B - P),\ e = -E_1 \cdot (B - P)$ and $f = (B - P) \cdot (B - P)$. If $ac - b^2 < \mu$, where μ is a user defined value, then the two edges of the facet are not linearly independent and we will then treat the facet as an edge and use point-edge to solve the problem. Else, the minimum squared-distance function is given as:

$$\begin{aligned} Q(s,t) &= |E_0(s) - E_1(t)|^2 \\ &= as^2 + 2bst + ct^2 + 2ds + 2et + f \end{aligned} \tag{5.7}$$

The aim is to minimize $Q(s,t)$ in R. The method to obtain the minimum distance for each region can be obtained by examining the values of $a \ldots f$. But, we would like to note that at most one division is computed with the denominator as $ac - b^2$.

Collision Detection. Collision is detected when a separating plane cannot be identified at any moment during the execution of the algorithm. This is verified when the distance between a witness point W and the simplex S is computed. Let the support vector be denoted as SV and the support point obtained by sweeping in the direction of the support vector be denoted as SP. A separating plane exists between W and S iff:

$$SV \cdot SP > 0 \tag{5.8}$$

for all instances of S.

Robustness. Depending on the kind of features that we cache at each iteration, the local distance method will calculate the minimum distance between these features and if necessary, update the cached features. It is at this stage that we ensure robustness. The updated features are verified if they are *stable* or not. A feature that is stable must meet a certain criteria. For example, an edge must have a certain minimum length and a facet must have a minimum area and non-parallel edges. If the formed feature is not stable, there could only be two possibilities; the object size is too small (can be considered as a point) or the feature of that object is too small (descent is not possible). In either case we terminate immediately without moving on to the verification stage. If the feature is stable, then we apply our local distance methods to get the distance. If this distance is not the minimum, we proceed to the other object and repeat the process. Such a method ensures that all the features formed are entirely stable. Then our local distance methods have no problems getting the distance values. Hence, the entire approach becomes stable.

Terminating Conditions. The algorithm exits for four main reasons:

- Minimum distance is too large to be of interest: $d_{lb} > \mu$
- Minimum distance is too small that collision is assumed: $d_{ub} - d_{lb} < \epsilon * d_{ub}$
- Objects are identified to be in collision.
- The cached features are not stable indicating that further descent is not possible.

The lower bound on the minimum distance gives us the first expression to exit, where μ is a value that indicates separation. The second expression to exit is given by the difference between the lower and upper bounds, where ϵ is a value that denotes relative precision and is machine dependent. The third exiting condition is the absence of a separating axis. The fourth and final condition is related to the robust test applied to the cached features.

5.2.2 Experimental Results

We present some experimental results here to compare our algorithm. We compared the execution times of the algorithm with Cameron's enhanced EGJK hill-climbing algorithm[1]. The objects tested are randomly generated convex polyhedra. For a pair of objects, one is placed at the origin while we apply a continuous translation or rotation to the other object through little incremental displacements vectors.

[1]The code used was downloaded from http://www.comlab.ox.ac.uk/~cameron/distances.html

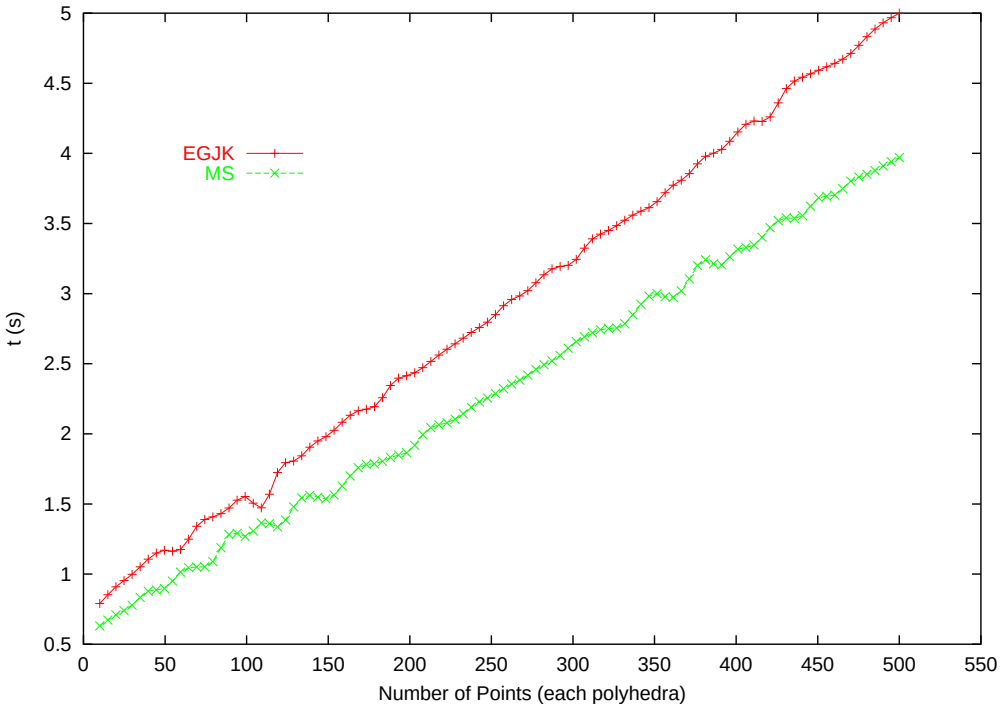


Figure 5.1: *EGJK vs MS: Execution time for continuous translation without tracking (memory) information.*

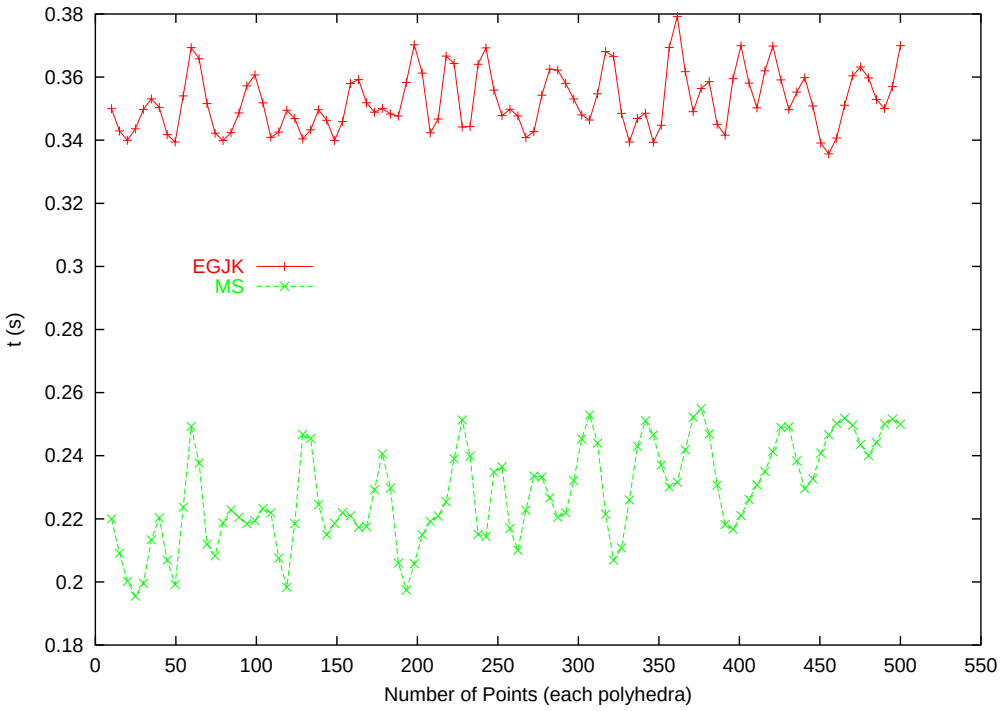


Figure 5.2: *EGJK vs MS: Execution time for continuous translation with tracking (memory) information.*

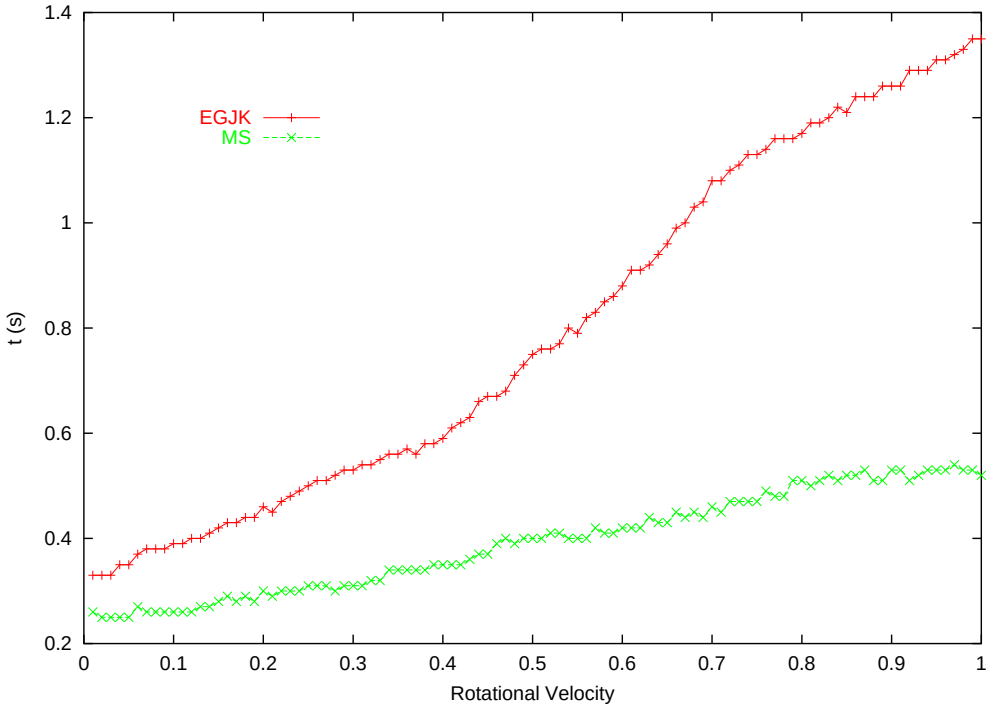


Figure 5.3: *EGJK vs MS: Evolution of the time for a continuous rotation with different rotational velocities (x-axis).*

Figure 5.1 shows how the timings evolve with increasing object complexity. When no tracking information is used we obtain a linear relationship between complexity and execution time in both the algorithms. However, when making use of this tracking information, the time is almost constant ($0.36s$ on average) for the EGJK algorithm. On the other hand, with our approach the time increase is slower with rising complexity of the objects, and the near-constant time drops to $0.24s$ on average. This can be seen in figure 5.2. For the experimental results shown in figure 5.3 we placed two objects with 50 points each a given distance apart and then applied a continuous rotation to one of them. The plot shows how the timings evolve for different rotational velocities. Tracking information is used, of course. We observe that our approach gives lower timings even at high rotational speeds as compared to the EGJK method. These results were obtained using a SGI-Octane 195MHz, and the code was compiled using the *CC* compiler with *-Ofast* optimization. However, some caution should be taken when interpreting these results since they represent a purely practical analysis.

5.3 Distance Computation of Concave Objects

The algorithm that we present has been inspired from [Quilan, 1994]. In this section, an algorithm that computes the minimum distance between two concave objects X and Y represented by an AABB hierarchy is detailed.

Problem. Minimum distance computation can be used for collision detection between rigid concave objects. Normally, the underlying data structure used is a convex decomposition or a sphere tree [Quilan, 1994]. We on the other hand have opted to represent our concave object using an AABB hierarchy. Hence, a method to compute the minimum distance between two AABB trees is required.

Proposed Solution. The main idea in this algorithm is to descend the hierarchy using a minimum distance criteria between the bounding boxes without exploring all the facets (bounding box leaves) of each object. If the minimum distance between two nodes is greater than a reference value, then the children will not be explored any further. A point to note is that the choice to descend the tree is of paramount importance to ensure efficiency. It is in this sense that we differ from [Quilan, 1994]. We have decided to descend to the child that is closer to the root of the other object. The assumption here is that the child closer to the root of the other object has more probability of being nearer, as the root of the hierarchy corresponds to the largest enveloping bounding box. We have found this to produce more efficient results.

5.3.1 Description of the Algorithm

In this section, we will explain how our algorithm is executed. We assume that the concave objects X and Y, both described in a global frame, are represented by an AABB underlying data structure hierarchy. We proceed as follows:

- We initialize the reference distance to ∞.
- We descend the two AABB trees to search for a leaf bounding box pair that has minimum distance inferior to this reference value. The minimum distance between the corresponding facets is determined using the algorithm presented in section 5.2.
- If we find a leaf pair facets that have null minimum distance, collision is detected and we exit the algorithm. Else, the reference value is updated to the current minimum distance value and we continue searching.

- If the found minimum distance between nodes is greater of equal to the reference value, we stop further descent between these nodes.
- Else, the reference value is updated and we continue searching until a leaf bounding box is reached.
- Searching stops when descent is not possible.

5.3.2 Experimental Results

We initially tested our algorithm with several complex meshes, the execution times are tabulated in table 5.1. We then used a single complex mesh with various different resolutions with a fixed orientation. The time taken to localize the pair of bounding boxes that realize the minimum distance is shown in figure 5.5. From the results, we can conclude that the execution time is approximately linear to the time taken to search the AABB hierarchy, which is expected. It there are n leaf nodes, we can expect n interior nodes of the tree with a maximum depth of n levels and a minimum depth of $log\ n$ levels, depending on how well the tree is balanced. Hence, in the ideal case we have a complexity of $\mathcal{O}(n\ log\ n)$ and in the worst case, we have a complexity of $\mathcal{O}(n^2)$.

Mesh	No. of Vertices	No. of Facets	Execution Time (ms)
Tea Pot	530	1024	6.43
Liver (simplified)	402	800	6.012
Liver (original)	1691	3366	11.625
Femur Bone	1998	3992	20.426

Table 5.1: *Execution time for distance calculation between two identical meshes.*

5.4 Collision Detection of Deformable Objects

The proposed algorithm in this section, is responsible of detecting collision and updating the AABB hierarchy in an efficient and fast method. This algorithm works on two deformable objects X and Y that have an AABB underlying data structure.

Problem. Updating the underlying data structure is of paramount importance to the accuracy of collision detection in deformable objects. The time spent to do this has to be minimized. Hence, an efficient manner is required for this purpose. We recall that our AABB hierarchy underlying data structure has been constructed

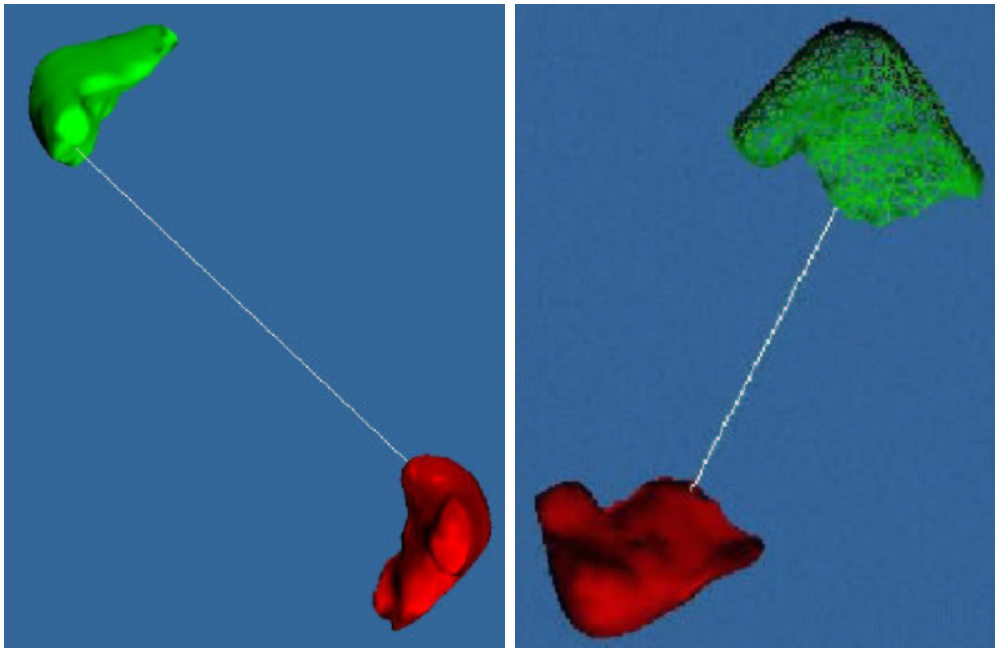


Figure 5.4: *Distance calculation between two deformable meshes. The used meshes have each one 800 facets and 402 vertices. The execution time is on average about* $30ms$*. The time was obtained using codes written for a Java3D platform.*

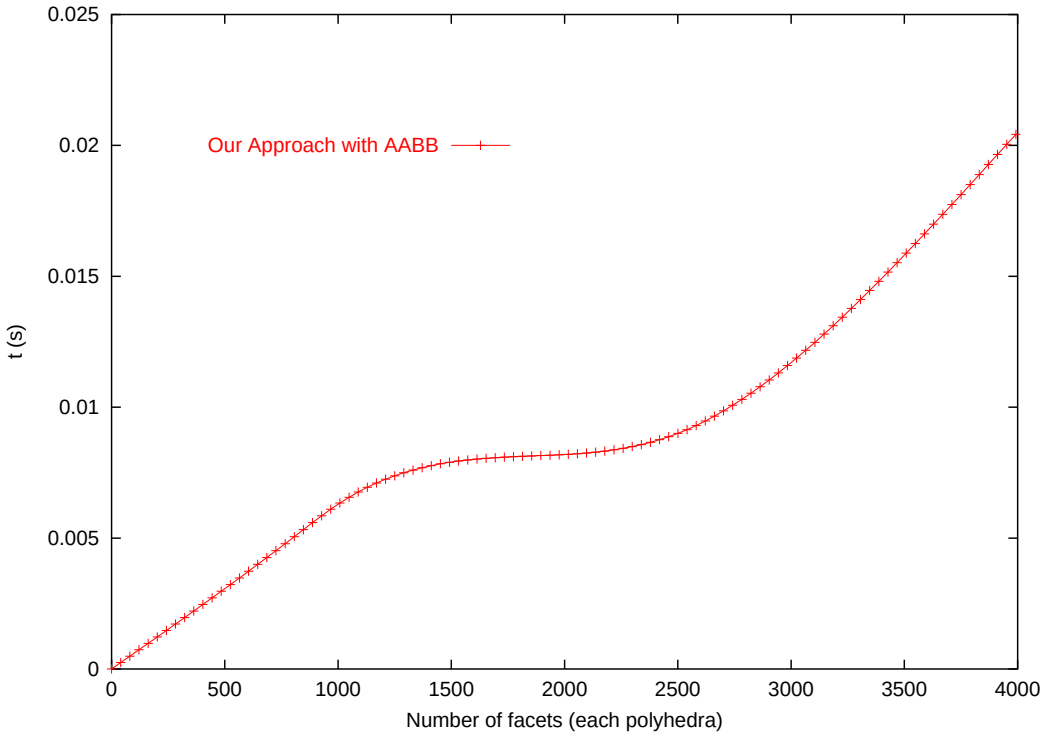


Figure 5.5: *Execution time for distance calculation between a single mesh of various resolution with a fixed orientation.*

bottom-top and volume minimization has been used as the criteria to combine the boxes at each level. This gives us a compact but not necessarily equilibrium hierarchy around the surface of the object. Given such a tree, the aim is to update this tree without rebuilding or refitting the entire hierarchy.

Proposed Solution. The main idea of this algorithm is the updating of the bounding boxes without exploring all the facets of each object and without exploring the entire tree. This is done by firstly reconstructing the boxes of all the leaves which have at least one of its vertices subjected to deformation. This *change* is then propagated to the parent box at each level. A parent box is only updated if at least one of it's child has been updated and this is done only *after* all it's child has been already treated. It is in this sense that we differ from [Van der Bergen, 1997] and [Larsson and Moller, 2001]. In the earlier, all the nodes are revisited, in contrary to ours. This is due to the fact that their hierarchy is constructed top-bottom using a subdivision criteria. In the latter, an approximated update of the AABB hierarchy is constructed, unlike ours where the exact hierarchy is maintained at all times.

5.4.1 Description of the Algorithm

Let the deformation vector of the two deformable bodies be known at each time-step. To detect collision, we proceed as follows:

- We reconstruct the boxes of all the leaves for both the bodies which have at least one of its vertices subjected to deformation.
- We determine the nodes of type parent that must be updated. Recall that a bounding box is defined by a center and a diagonal. Hence, a bounding box is defined by it's maximum and minimum limits in each axis. In this case, a parent need only be updated if and only if it's limits are affected by the limits of it's children.
- The affected parents are updated in a bottom-top approach only after all of the children are treated. This avoids revisiting the parents.
- Once updating is completed for both objects, a search for interference is started between the two updated AABB trees. The bounding box leaves that are in intersection are stored in a list to be further treated.

Once we have localized all possible pairs of bounding box pairs that intersect between the two objects, we examine the corresponding facets using the algorithm presented

for the intersection test between triangles in chapter 4 to determine the presence of interference. These facets are stored in another list to be used in the collision treatment phase.

5.4.2 Experimental Results

The results of our experiments are presented in this section. In table 5.2, we applied our algorithm on various complex meshes with a randomly applied deformation vector. In figure 5.7, we took a single mesh of fixed resolution and applied a random deformation vector to various number of nodes. The graph shows the time taken to update the AABB hierarchy. From the results in the shown graph, we can conclude that the execution time, as expected, is linear to the number of affected nodes in the AABB hierarchy.

Mesh	No. of Vertices	No. of Facets	Execution Time (ms)
Tea Pot	530	1024	7.867
Liver (simplified)	402	800	9.917
Liver (original)	1691	3366	27.858
Femur Bone	1998	3992	28.762

Table 5.2: *Execution time for the update process of the underlying AABB hierarchy of different meshes for a randomly applied deformation vector.*

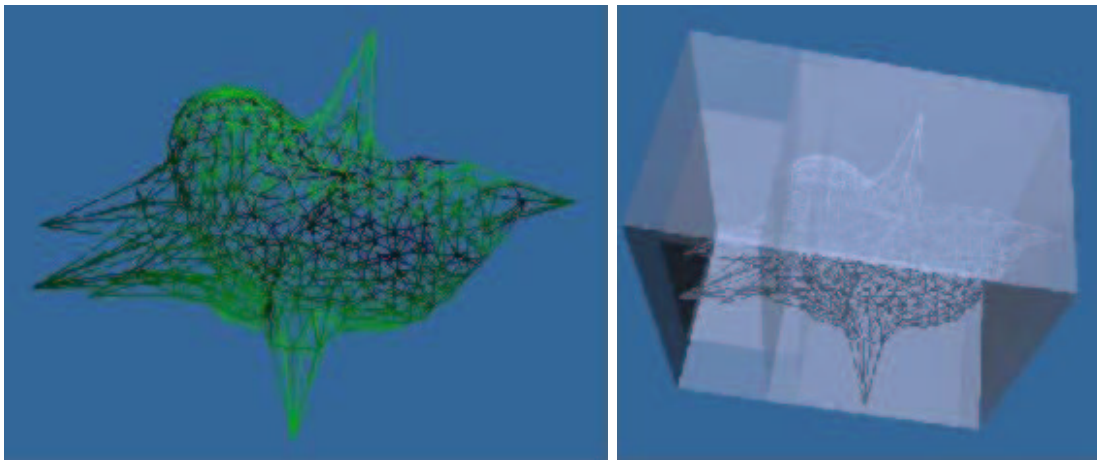


Figure 5.6: *Example of the updating process of the AABB hierarchy. The used mesh has 800 facets and 402 vertices.*

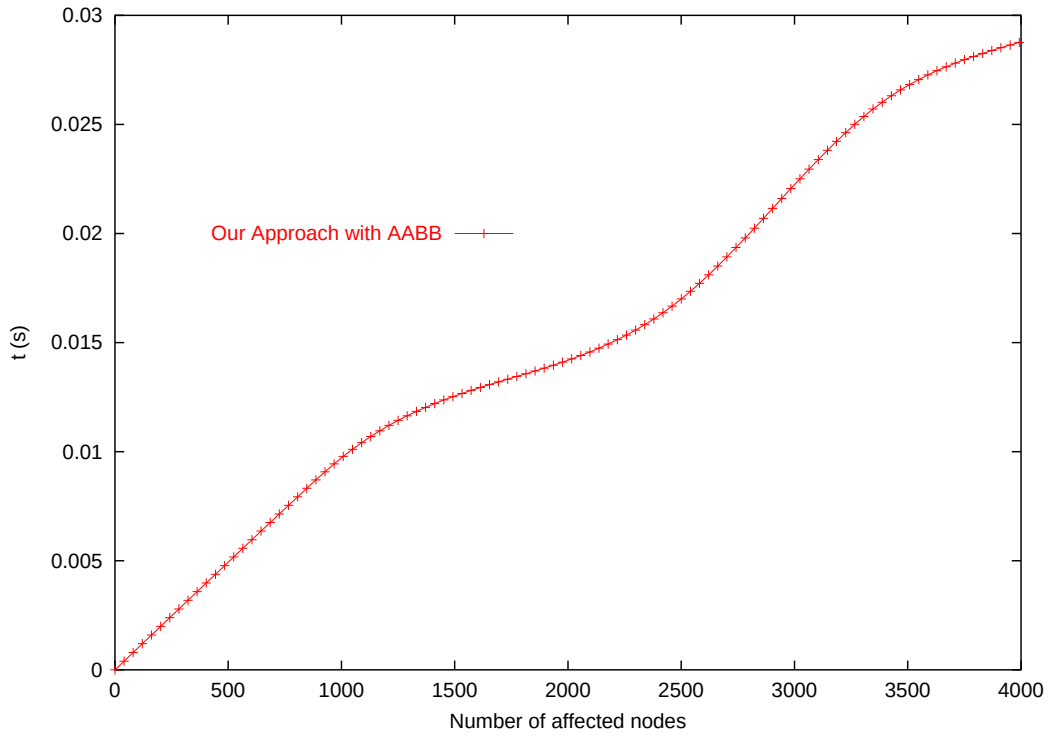


Figure 5.7: *Evolution of the time for the updating process of a fixed resolution mesh with a random deformation vector applied to various number of nodes.*

5.5 Contact Localization for Collision Treatment

We will present in this section an algorithm [Sundaraj and Laugier, 2000], to localize the fictive zone of interpenetration. The primitives that form this zone are called *contact elements* and they are used during collision treatment.

Problem. Most of the collision detection algorithms that we have mentioned in the previous chapter, generally return only intersecting elements and not all the contact elements. By contact elements, we mean elements that are not intersecting but nevertheless are interior to an object (see figure 5.8). Collision treatment can then be applied to these elements if we know their extent of deformation. This extent of deformation, for example can be the measure of fictive interpenetration of each contact element.

Proposed Solution. The main idea of this algorithm is by first locating the contour of the zone of interpenetration which is the set of all intersecting facet pairs. Following this, a recursive search for all the contact elements can be done using the

contour. To avoid searching in the wrong areas, thereby optimizing this process, we construct an inner and a outer boundary which is interior and exterior respectively to the contour. These boundaries will serve as the search limit. Finally, we search for all the contact elements. In doing so, we actually have formed the two collision surfaces which envelopes the volume of interpenetration. These surfaces will be highlighted in our results. Flowchart 5.9 shows the different stages of this algorithm.

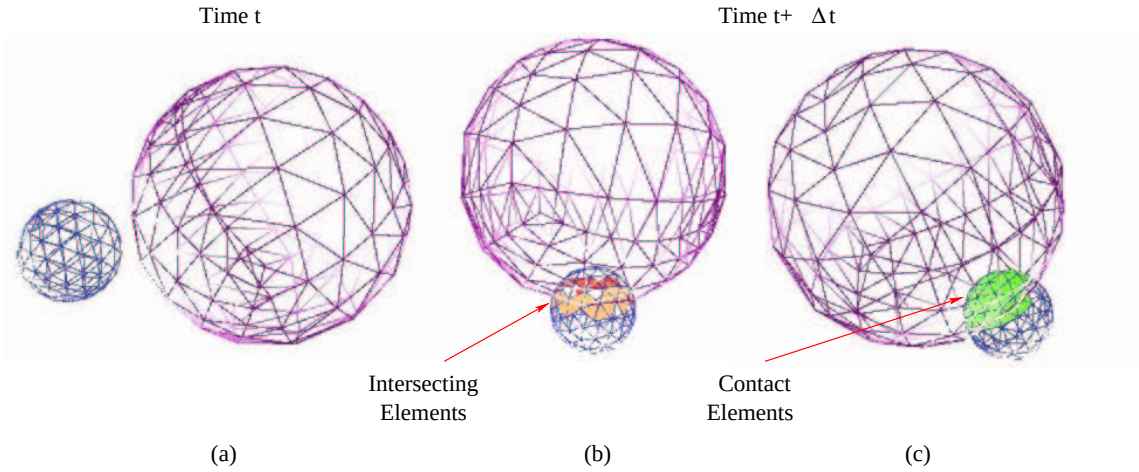


Figure 5.8: *(a) At time t, the spheres are not intersecting. (b) At time $t + \Delta t$, the spheres are intersecting. The available collision detection algorithms return the intersecting elements. (c) Our algorithm returns the contact elements; elements that are not in intersection with other primitives but nevertheless interior to each other.*

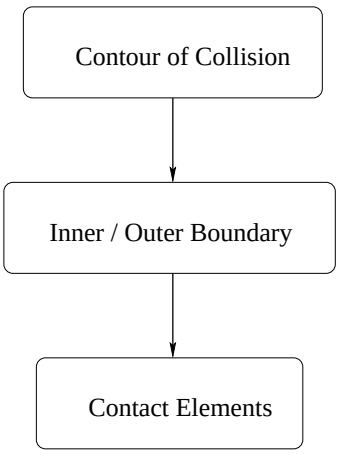


Figure 5.9: *Flowchart showing how to arrive at detecting all contact elements.*

5.5.1 Description of the Algorithm

In this section, we will explain how our algorithm is executed. We assume that the deformable objects have an AABB underlying data structure. We proceed as follows:

- **Contour of Collision.** To get the collision contour, we use algorithm presented in section 5.4 to localize all the pairs of facets that are intersecting. Figure 5.10 shows an example of our algorithm on two convex objects. This step is linear in time with respect to the number of collision pairs.

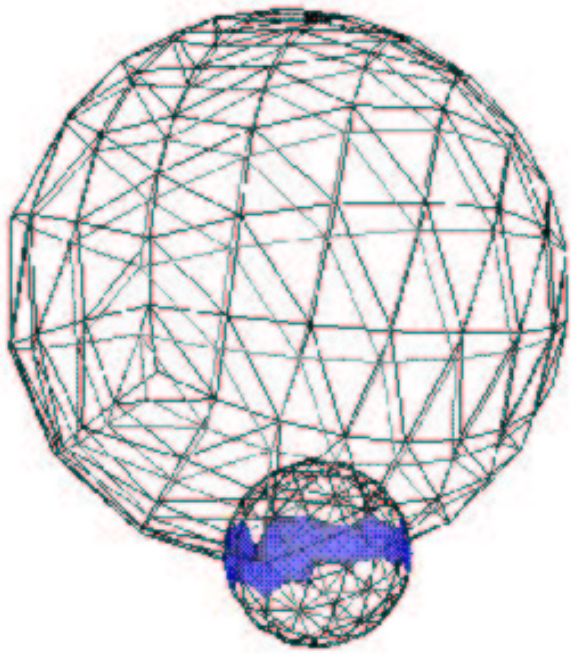


Figure 5.10: *The collision contour of the smaller sphere is shaded.*

- **Inner and Outer Boundary.** Our aim here is to construct an inner and outer boundary which is interior and exterior respectively to the collision contour. The outer boundary server as a limit to the collision surface. In other words, facets beyond this limit need not be considered in the search for the contact elements. The inner boundary then serves as the starting point to start searching. Consider a collision pair, (F_{A^i}, F_{B^j}) whose normals are oriented towards the outside.

  DEFINITION 1. *We denote $\mathcal{P}_{A^i}$ and $\mathcal{P}_{B^j}$ as the planes formed by F_{A^i} and F_{B^j} respectively.*

DEFINITION 2. *We denote $\mathcal{E}^+_{A^i}$ and $\mathcal{E}^-_{A^i}$ the half-space in $\mathbb{R}^3$ defined by the plane $\mathcal{P}_{A^i}$; similarly we denote, $\mathcal{E}^+_{B^j}$ and $\mathcal{E}^-_{B^j}$ the half-space in $\mathbb{R}^3$ defined by the plane $\mathcal{P}_{B^j}$. The positive half-space denotes exterior and the negative half-space denotes interior.*

DEFINITION 3. *A neighboring facet $F_{A^{i'}}$ or $F_{B^{j'}}$ of a collision pair (F_{A^i}, F_{B^j}) is labeled as exterior to the collision contour if:*

$$\begin{aligned} F_{A^{i'}} &\subset \mathcal{E}^+_{B^j} \\ F_{B^{j'}} &\subset \mathcal{E}^+_{A^i} \end{aligned} \tag{5.9}$$

DEFINITION 4. *Similarly, a neighboring facet $F_{A^{i'}}$ or $F_{B^{j'}}$ of a collision pair (F_{A^i}, F_{B^j}) is labeled as interior to the collision contour if:*

$$\begin{aligned} F_{A^{i'}} &\in \mathcal{E}^-_{B^j} \\ F_{B^{j'}} &\in \mathcal{E}^-_{A^i} \end{aligned} \tag{5.10}$$

Note that the inner boundary list has a more tighter constraint when compared to the outer boundary list. In other words, a facet cannot appear on both the list, elimination has to take place in the inner boundary list due to the tighter constraint. This step is linear time with respect to the number of collision pairs in the contour of collision. Figure 5.11 shows the detection of the inner and outer boundary facets for a concave and convex object in collision. Note that the facets that have been localized only belong to a single list.

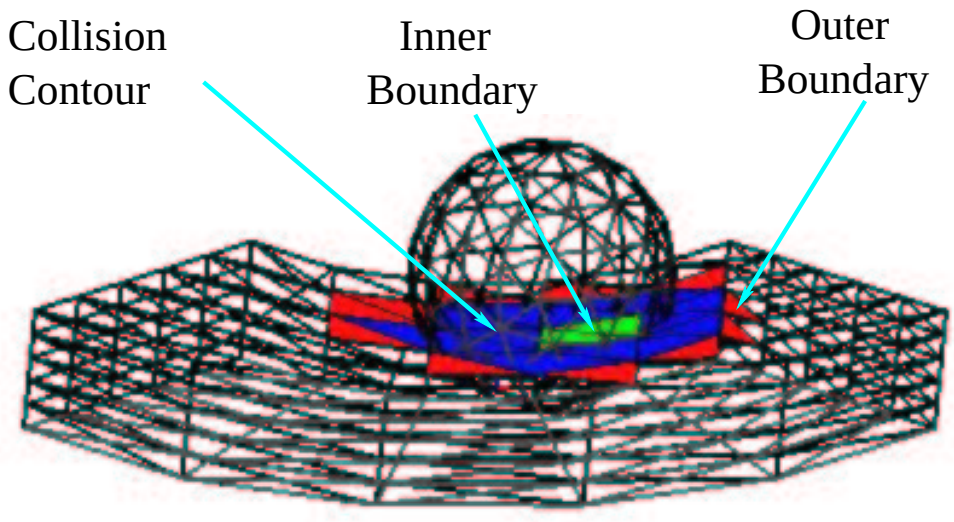


Figure 5.11: *Inner and outer boundary detection with respect to the collision contour for the concave object.*

- **Contact Elements.** We use the inner boundary facet list as the starting point to search for all contact elements. We recursively search all neighboring facets of each inner boundary facet. If these facets do not appear in the contour list or the outer boundary list, then we consider them as a contact element. This step is linear with respect to the number of inner boundary facets. Figure 5.12 shows an example of this recursive search output.

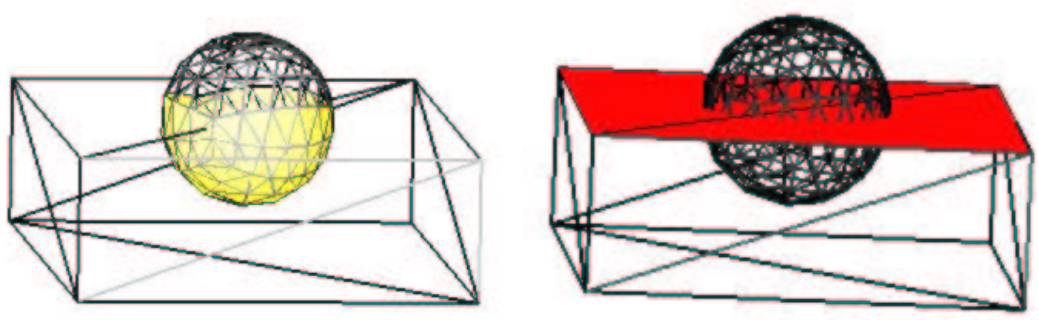


Figure 5.12: *Contact elements (shaded) detection of a cube and a sphere.*

Generally, the entire algorithm is linear with respect to the number of contact elements, *i.e.* $\mathcal{O}(p)$ where p is the number of contact elements, which is proportional to the volume of interpenetration, depending on the extent of discretization. Once we have localized all the contact elements, we actually have the two collision surfaces that encompass the volume of fictive interpenetration. The volume can be used as a measure of fictive interpenetration to calculate the collision forces $\boldsymbol{F}_c$ by the *penalty method* [Deguet et al., 1998b], [Deguet et al., 1998a]:

$$\boldsymbol{F}_c = \begin{cases} (-\lambda v - \mu \dot{v})\boldsymbol{k} & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

where λ is the rigidity factor of the collision, μ is a damping factor (which represent the rate of dissipation of energy), v the volume of interpenetration between these two objects, and $\boldsymbol{k}$ is the contact direction. This force is supposed to act on the inertial center of the fictive interpenetration volume.

5.5.2 Experimental Results

We tested our algorithm on several complex meshes (see table 5.3). The execution time as a function of the number of contact elements was investigated. The results are shown in figure 5.14. The experiments were run through to the final stage of the algorithm, that is from collision detection until all contact elements were localized.

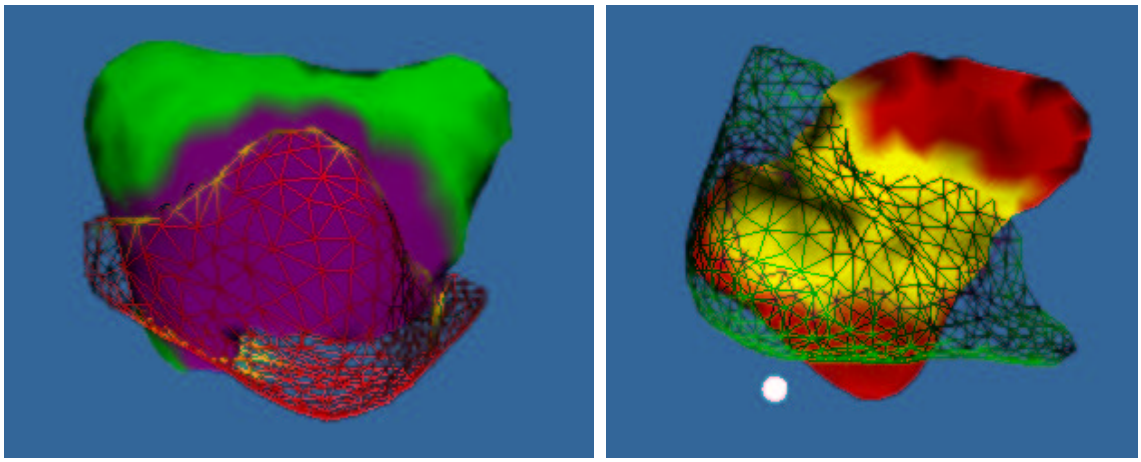


Figure 5.13: *Example of the detection of the volume of interpenetration using algorithm. The used mesh has 800 facets and 402 vertices.*

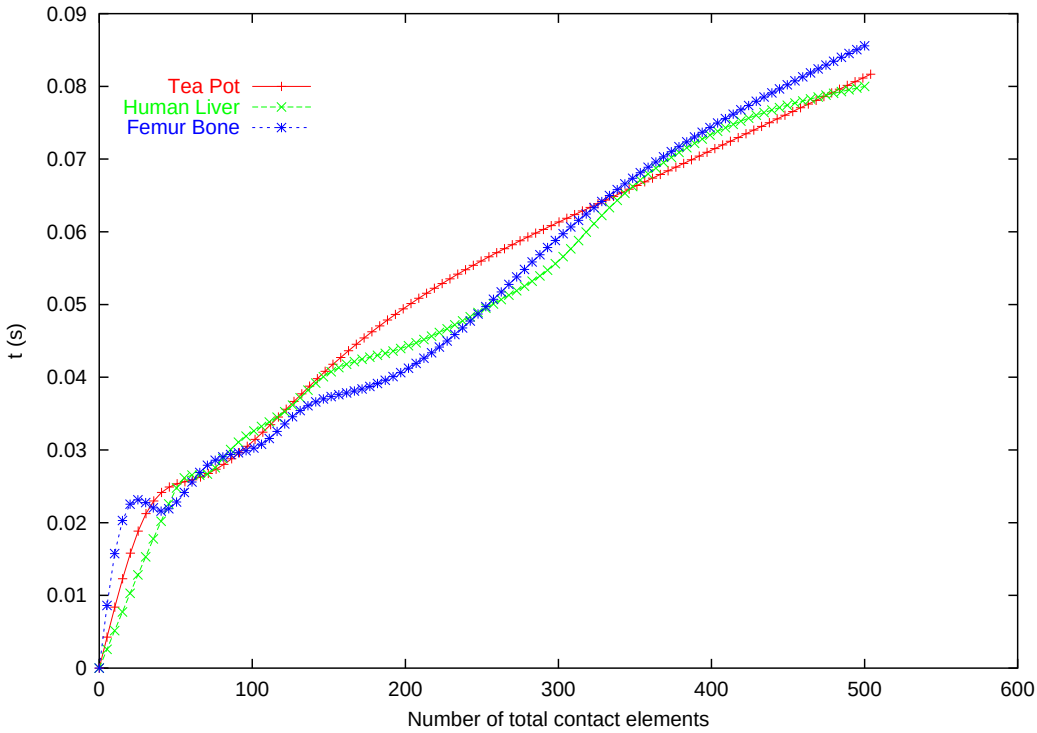


Figure 5.14: *Evolution of the time for a complete detection of the zone of fictive interpenetration as a function of the number of contact elements for various meshes.*

Mesh	Number of Vertices	Number of Facets
Tea Pot	530	1024
Human Liver	1691	3366
Femur Bone	1998	3992

Table 5.3: *Details of the meshes used for testing the algorithm.*

From the results, we can conclude that the complexity of this algorithm is indeed linear with respect to the number of contact elements. Even for high resolution meshes, interactive-time compliance can be met. The timings for this experiment were obtained on a Pentium-4 1.2GHz machine. These experiments were run on a Java3D platform.

5.6 Summary

In this chapter, we have presented various algorithms for collision detection of rigid and deformable bodies. These algorithms have the main aim of being efficient, optimized and robust such that it can be used in interactive-time medical interventions.

We firstly presented a new algorithm for distance computation between rigid convex objects. This algorithm has a complexity of constant time *i.e.* $\mathcal{O}(1)$ in the best case. We have compared this algorithm with the EGJK algorithm and found it to be more efficient. We have also improved the robustness of this algorithm by using local methods to compute distance between simplexes.

We then presented an algorithm for distance computation between rigid concave objects. We have opted to use an AABB hierarchy as the underlying data structure to solve this problem. Within this context, we have improved on the descending criteria in this algorithm. This has improved the execution time of the algorithm. We have also optimized memory usage by using a compact data structure to represent the AABB tree.

Following this, we presented an algorithm for collision detection of deformable objects. This algorithm uses an AABB hierarchy as the underlying data structure for collision detection. We have already mentioned that for deformable objects the AABB tree is the most efficient. This hierarchy is constructed using a bottom-top approach. The main contribution in this algorithm is the updating of the AABB tree in an efficient manner. This is done by verifying and only visiting the nodes of the tree that must be updated. Hence, the complexity of this algorithm is a function of

the number of affected nodes (nodes that must be updated) in the AABB hierarchy.

In the case where fictive interpenetration is allowed and the user would like to have all the contact elements as input data to the collision treatment phase, we have presented an algorithm to localize all contact elements. This algorithm has the complexity which is linear to the number of contact elements. Using all the contact elements, the fictive volume of interpenetration can be used as a measure of deformation to compute the collision forces.

These algorithms have been implemented in a collision detection library called **ColDetect**. ColDetect is described in appendix B and can be downloaded from the following website http://www.inrialpes.fr/sharp/coldetection. This library has also been integrated successfully into our 3D dynamic simulator, AlaDyn3D and is independent of the 3D rendering engine and the operating system platform.

Part III

Virtual Reality Applications

Chapter 6

Medical Simulators

6.1 Introduction

In the previous parts, we presented two important aspects of medical simulators; physical models for soft tissue simulations and algorithms for collision detection. Indeed, medicine is one of the major application areas for virtual reality (VR). The medical application of VR was stimulated by the need of medical staff to visualize complex medical data, particularly during surgery, for surgery planning and for medical training. Needless to say, medical planning and training has been one area in which VR has made a significant contribution. The development of commercial and prototype medical simulators is a testimony to this.

In this chapter, we present the application of our findings in two prototype medical simulators. We shall conduct two case studies; an echographic thigh exam (ETE) simulator and an arthroscopy knee reconstruction (AKR) simulator. In the earlier, the trainee pushes an echographic probe gently on the thigh and stops when sufficient resistance is felt. Hence, our main concern here is global small deformation and force feedback. In the latter, the surgeon wishes to know the deformation of the knee ligament and the amount of contact with it's surroundings. In this case, our main concern is global large deformations and collision detection.

We begin this chapter by presenting the ETE training simulator with emphasis on the new physical model, collision detection and haptic feedback integration. Then the details regarding the development of a medical simulator for AKR will be presented. In this part, we will explain the procedure of automatically and intra-operatively generating the physical model of the ligament according to the patient acquired anatomy and knee kinematics. We end this chapter with a discussion of the results.

6.2 Echographic Thigh Exam

6.2.1 Motivations

The exam of the thigh using ultrasound is called an echographic thigh exam. A common exam is the echography of the thigh to detect a thrombosis in the vein. A thrombosis is the formation of blood clot formed from platelets and other elements which may obstruct the flow of blood in the vessel. It may also travel to other areas of the body. A healthy vein will compress under the influence of an external force, while a vein affected by thrombosis will only partially or hardly compress, depending on the stage in the evolution of the illness. The medical professional executes the diagnosis by applying a pressure on the thigh with an echographic probe. From the instantaneous echographic images of the vein, the presence of a thrombosis can be ascertained.

The learning process of this procedure is somehow long and only after approximately 1000 echographic exams an acceptable competence in acquired. The first 500 exams will have to be carried out under the supervision of an experienced practitioner. However, learning this procedure can be much quicker and improved by using a virtual environment as an alternative form of training tool. With a dedicated medical simulator for the examination of the thigh, an interactive and 3D virtual thigh model can be created, where medical professionals can manipulate the virtual surgical tools with an haptic interface and train adequately before coming in contact with patients.

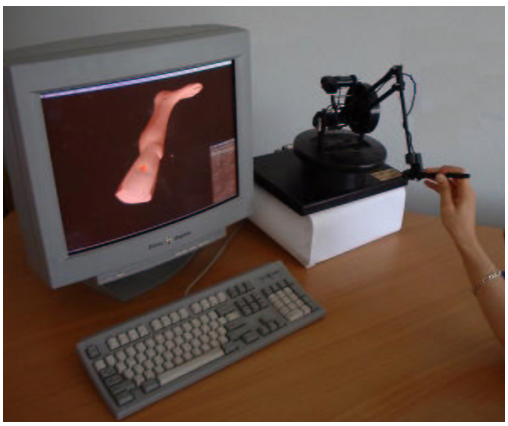


Figure 6.1: *The developed ETE prototype. The user interacts with the virtual thigh using an haptic interface.*

Indeed, these motivations were the reason why our research group developed a prototype echographic training simulator [Aulignac, 2001] using mass-spring net-

work (MSN) as the physical model of the thigh (see figure 6.1). But this model only produced local deformations and force feedback was produced using a simple method. Now, we would like to explore the feasibility of using the VDM physical model in this simulator. This is due to the fact that the thigh can be considered to consist mainly of muscles and blood. Within this context, we investigate interactive-time global small deformations and realistic force feedback reproduction.

6.2.2 Thigh Deformable Model

Graphical Model. Since the main aim of the simulator is for the purpose of training, the use of a generic mesh of a human leg as the graphical model is more than sufficient. This generic model is divided into two parts; the thigh and the rest of the leg. To accelerate the graphical rendering procedure, the lower part of the leg is considered fixed. Thus, this data is transfered only once to the graphics hardware, and consequently, hardware acceleration can be used to render them. The thigh however is deformable and has to be simulated in interactive-time. Moreover, a complete generic leg mesh is used so that the trainee can have an increased sense of realism and also have landmarks for orientation in the virtual environment. Figure 6.2 shows the used generic model of the thigh.

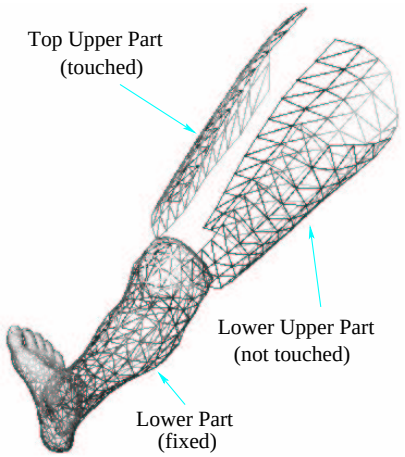


Figure 6.2: *Decomposition of the lower and upper parts of the leg. The top part of the thigh is divided into two parts; the area touched by the practitioner and vice-versa.*

Physical Model. We were inspired to patch a VDM model on the geometric mesh of the thigh. As mentioned before, VDM is used to model objects filled by some incompressible fluid, which seems to be a good approximation for soft biological

tissue (see figure 6.3). In contrast to the model proposed by [Aulignac, 2001], we have not limited the movements of the lower upper part of the thigh.

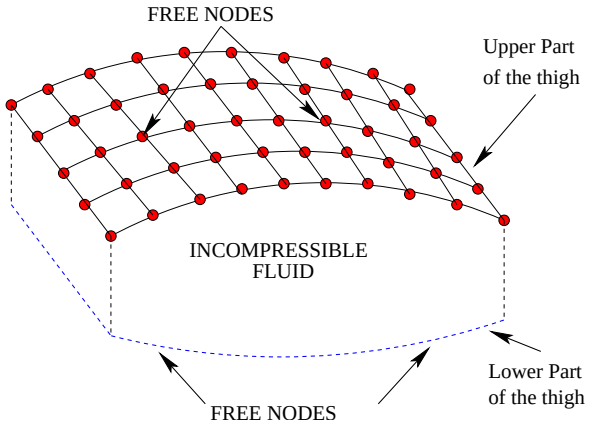


Figure 6.3: *The VDM model of the thigh.*

For the upper part of the geometrical mesh, the VDM model is used as the physical model. Given the geometrical model, the distributed volume $\mathcal{V}_i$ and distributed area $\boldsymbol{\mathcal{A}}_i$ can be extracted. Then, each node i which is connected to neighboring nodes j is governed by the following equation:

$$\frac{B_i}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i\Big) + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i - \boldsymbol{\mathcal{A}}_j\boldsymbol{\Delta L}_j\Big) - \Delta P = \rho_i g \delta_i \tag{6.1}$$

and the following boundary condition:

$$\sum_i^{\aleph} \Delta\mathcal{V}_i = \sum_i^{\aleph} \Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i + \boldsymbol{L}_i\boldsymbol{\Delta\mathcal{A}}_i\Big) = 0 \tag{6.2}$$

where for each node i, B_i is the bulk modulus, B_{ij} is the connectivity bulk modulus, $\boldsymbol{\Delta L}_i$ is the displacement vector, ρ_i is the associated density, g is the gravitational constant and δ_i is the hydrostatic distance. This physical model is mapped directly onto our surface based geometrical model of the upper part of the thigh.

Parameter Estimation. The parameters of the VDM model namely the bulk modulus B_i, and the connectivity bulk modulus B_{ij} need to be estimated so that we can have reasonable accuracy in our results. We decided to use the data from the literature as our source. We assumed that the human thigh is full of blood (incompressible) and contains mainly muscles. The human skin was considered as the elastic container of our VDM model. The parameters used during the simulation are shown in table 6.1.

Table 6.1: *Physical data of the virtual human thigh*

Surface Area ($\mathcal{S}$)	225000 mm^2
Volume ($\mathcal{V}$)	8835730 mm^3
Resolution (Q)	260 elements
Density (ρ) - Blood	1055 kgm^{-3}
Bulk Modulus (B_i) - Muscle	2.7 x 10^9 Nm^{-2}
Connectivity Bulk Modulus (B_{ij}) - Skin	1.4 x 10^6 Nm^{-2}

System Resolution. In this application, normally the surgeon places the patient on a table before examining the thigh. In this case, the nodes of the mesh in the lower upper part of the thigh are fixed and do not move due to contact with the table. The nodes on the top upper part of the knee that are in contact with the echographic probe are constrained to move according to the probe. These respective nodes $i^*_{upperthigh}$ and $i^*_{lowerthigh}$ are constrained by the following equations:

$$K^*_{ii_{lowerthigh}} = \alpha, \;\; R^*_{i_{lowerthigh}} = 0 \tag{6.3}$$

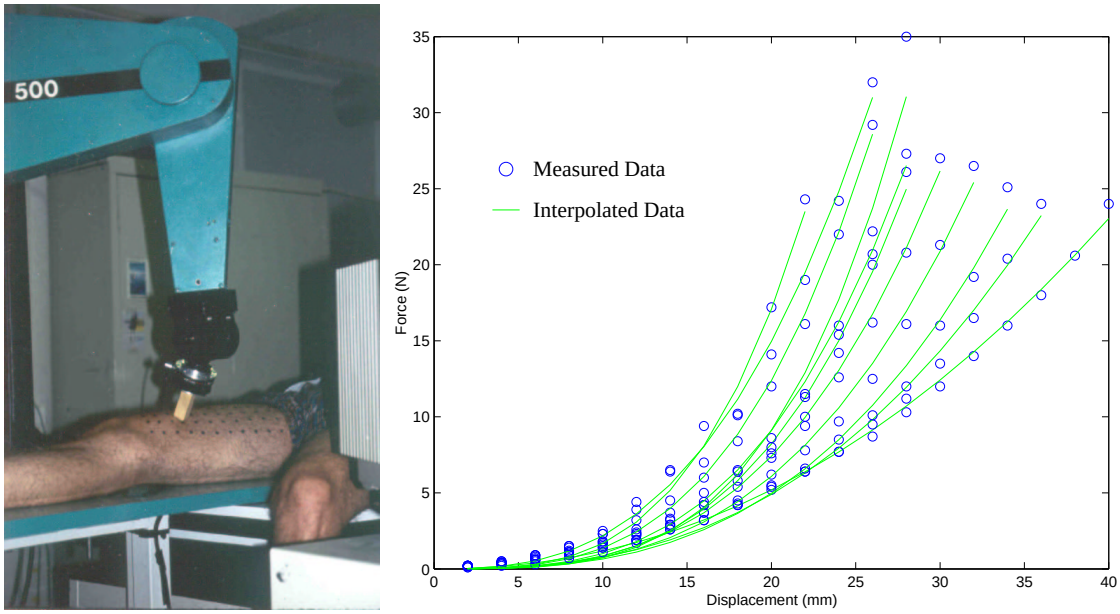
$$K^*_{ii_{upperthigh}} = \alpha, \;\; R^*_{i_{upperthigh}} = \alpha \mathbb{C}_{probe} \tag{6.4}$$

where $\alpha \gg 0$.

A quasi-linear analysis using the Sherman-Morrison method which has been presented earlier is used to solve the system. Since we only expect small deformation in an echographic thigh exam, a quasi-linear analysis is sufficient. This can be explained from the force-displacement measurements taken by [Aulignac, 2001] (see figure 6.4). For small displacements, a linear relationship is observed. Hence, the state matrix $\boldsymbol{K}$ is formed in the beginning and inversed. As we have shown before, the VDM model gives us $\aleph + 1$ equations and $\aleph + 1$ unknowns: $\boldsymbol{\Delta L}_i$ from $i = 1 \ldots \aleph$ and ΔP. This system can be written in the form $\boldsymbol{K \Delta L} = \boldsymbol{R}$ where $\boldsymbol{K}$ is the sparse state matrix of the virtual thigh, $\boldsymbol{\Delta L}$ is the displacement vector of the nodes and $\boldsymbol{R}$ is the external load vector. Given the displacement of the echographic probe during examination, the deformation of the thigh is computed in real-time and the bottom row of the state matrix is updated. The pseudo-code for this resolution procedure is given in algorithm 2 of chapter 3.

6.2.3 Interaction

In the previous section, we mapped a physical model onto a deformable geometrical model. During an echographic exam of the thigh, besides interactive-time



(a) (b)

Figure 6.4: *(a) Acquisition of thigh physical parameters using a robot with force sensors. (b) The circles show the experimental data obtained from a real thigh. The curves are the obtained from the model using a linear square error estimation method.*

deformation, realistic force feedback is required. Force feedback is felt by the user due to interaction of the virtual thigh with a virtual echographic probe. In this section, we explain how force feedback is produced.

In the architecture of the prototype developed, deformation of the thigh is handled by the simulation loop and the force feedback to the user is handled by the haptic loop. To ensure visual and tactile realism, the simulation loop and haptic loop must be updated at 30Hz and 1KHz respectively. Due to the difference in update rates between the two loops, a **deformable buffer model** [Mendoza et al., 2003] that shares a common memory area is used to ensure stability. This buffer model is in the haptic loop. It is a local representation of the virtual object at the area of contact. Figure 6.5 shows the concept of this dual thread architecture.

Collision Detection. The collision detection routine is embedded in the simulation loop and is thus updated at a rate of 30Hz. To avoid computational overhead in this loop, the virtual echographic probe was modeled as a rigid convex cylinder. The algorithm in section 5.4 is used to detect all colliding facets and then we localize the zone of interpenetration by using the algorithm in section 5.5. The facets in this

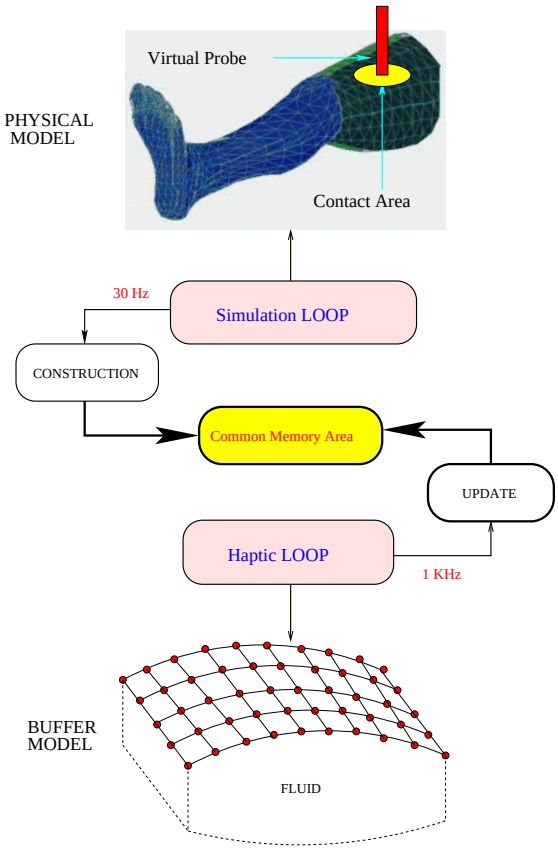


Figure 6.5: *The physical model is reduced to a buffer model at the area of contact [*Mendoza et al., 2003*]. The two threads in the dual architecture are independent; each controlled by a different loop. Both loops share a common memory area.*

zone of interpenetration are transfered to the common memory area at a rate of 30Hz. Figure 6.6 shows an example of detecting and localizing contact between a virtual echographic probe and a virtual thigh.

Force Feedback. Our simulation runs at approximately 30Hz on a PC-Linux 450MHz. At each time-step, we check for collision, calculate deformation and update the position of the vertices of the deformable thigh. The common memory area is also updated at this frequency and the stored primitives are used to form the deformable buffer model. In this section, we present our approach of using this deformable buffer model to render haptic forces to the user. From our experience, this problem is divided into three phases; *CONSTRUCT*, *UPDATE* and *LINK*.

- **CONSTRUCT :** Upon contact, the primitives (W) in the zone of fictive interpenetration are transfered to the haptic loop through the shared memory area. The state matrix $\boldsymbol{K}_{buffer}$ for these primitives is then formed and inversed.

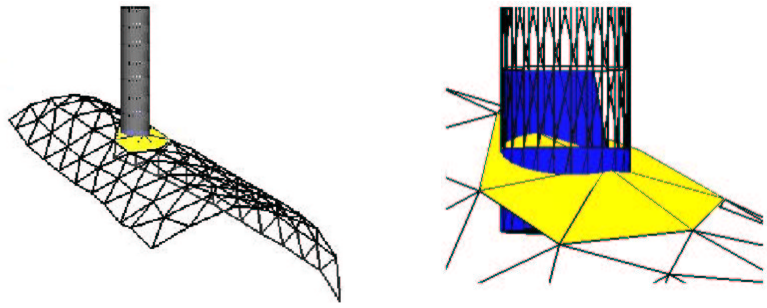


Figure 6.6: *The virtual echographic probe is modeled as a cylinder with a known surface area at the tip. Upon contact, all the interaction primitives are localized using the algorithm in section 5.5 and transfered to the common memory area.*

Since the number of facets in this loop is limited, this process is fast. To further improve computational speed, we can first test the maximum size of $\boldsymbol{K}_{buffer}$ that can be inversed in the haptic loop. This will then give us the maximum number of primitives W allowed for transfer. If memory is not a constraint, the inverse matrix for the entire set of possible primitives can be precalculated and stored. This look-up table is then used to get the inversed elements when contact is detected. We assume that the set W does not change much during the simulation due to the slow gesture of the surgeon.

- **UPDATE :** The simulation loop updates the deformable buffer model in the haptic loop at 30Hz. Locally, within the haptic loop, there are several processes being executed at 1KHz:

  1. Collision detection between the virtual echographic probe and the deformable buffer model defined by the set W.
  2. Modifying the $\boldsymbol{R}_{buffer}$ matrix.
  3. Resolution of the problem: $\boldsymbol{K}_{buffer}\boldsymbol{\Delta L}_{buffer} = \boldsymbol{R}_{buffer}$.

  These processes must not be interrupted when the buffer model is updated. This is done by keeping a copy of shared data. An important observation in this phase is that the values are written at 30Hz but read at 1KHz by the different loop threads. To avoid read-write problems, a flag is used to indicate if the values are safe to be read. When the buffer model is being updated, the copied data is temporarily used in the haptic loop. Upon completion, the copied data is switched with the updated data.

- **LINK :** The problem $\boldsymbol{K}_{buffer}\boldsymbol{\Delta L}_{buffer} = \boldsymbol{R}_{buffer}$ is solved in the haptic loop to obtain the deformation vector $\boldsymbol{\Delta L}_{buffer}$. But as we have mentioned before, in the VDM model, the last component in this deformation vector is the change in pressure ΔP_{buffer}. Since the number of elements in the simulation loop and haptic loop differ, ΔP_{buffer} will not be the real change in pressure experienced by the deformable thigh. Since the fluid is incompressible, we can approximate the rendered haptic force to the user ΔP_{haptic} by the following definition:

$$\Delta P_{buffer}\ \mathcal{V}_{buffer} = \Delta P_{thigh}\ \mathcal{V}_{thigh} \tag{6.5}$$

where $\Delta P_{thigh} = \Delta P_{haptic}$ is the real change in pressure experienced by the full physical model. $\mathcal{V}_{buffer}$ and $\mathcal{V}_{thigh}$ are the distributed volumes (total sum) of the nodes in the respective haptic and simulation loop. By using $P = |\boldsymbol{F}|/|\boldsymbol{\mathcal{A}}|$, the rendered force to the user at 1KHz is then obtained using:

$$\begin{aligned} \Delta P_{haptic} = \Delta P_{thigh} &= \frac{\Delta P_{buffer}\ \mathcal{V}_{buffer}}{\mathcal{V}_{thigh}} \\ \boldsymbol{F}_{haptic} &= \frac{\Delta P_{buffer}\ \mathcal{V}_{buffer}\ \boldsymbol{\mathcal{A}}_{haptic}}{\mathcal{V}_{thigh}} \end{aligned} \tag{6.6}$$

where $\boldsymbol{\mathcal{A}}_{haptic}$ is the distributed area (total sum) of all the elements in the haptic loop. Since this force is derived from the buffer model, it is computed at 1KHz.

6.2.4 Experimental Results

We used the echographic thigh simulator originally developed by Aulignac in [Aulignac, 2001] to test the VDM thigh model and the VDM deformable buffer model. The visual-haptic platform uses a PC-Linux 450MHz Intel processor with 256Kb of RAM connected to a haptic interface type PHANToM. The virtual probe was positioned using the haptic interface, at various different points. At each of these points, the probe was used to push perpendicularly to the surface. We did this because in an echographic thigh exam, this is the standard procedure.

Deformation. In these experiments, the deformation obtained using the VDM model is satisfactory because volume conservation of the thigh was observed throughout the entire history of deformation. Although this is not the main physical characteristic of the thigh, our aim was to observe volume conservation, global small deformations and stability in the numerical resolution methods. A corresponding deformed echographic image has also been successfully produced at all points and orientations (see figure 6.7).

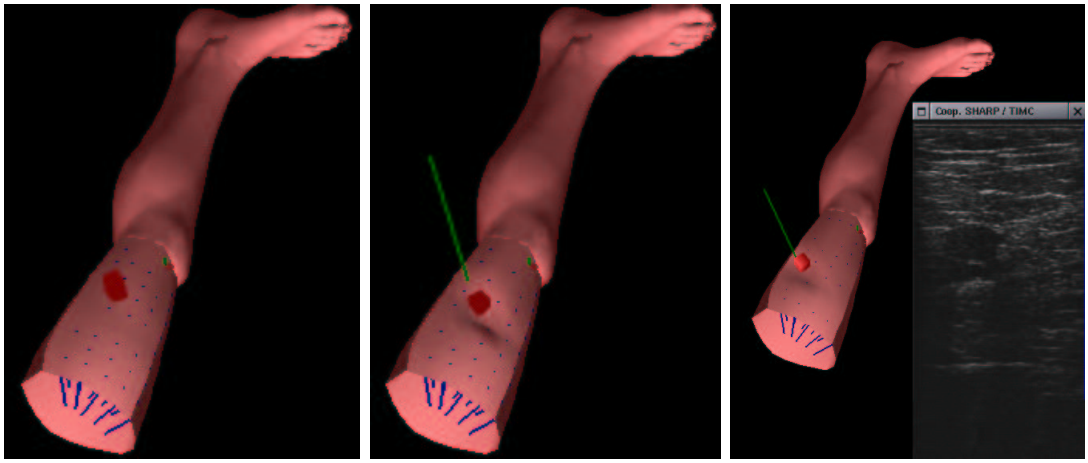


Figure 6.7: *Quasi-linear simulation of the human thigh with echographic image integration.*

Force Feedback. Figure 6.8 shows the curve of the force obtained by the physical model and the haptic force during the contact process. In this experiment, the user gently pushed the virtual probe at one of the points on the surface of the thigh mesh with an arbitrarily displacement vector in a quasi-sinusoidal form of motion. It is possible to see that the force obtained from the physical loop is not continuous while the force obtained from the buffer model is continuous and approximates the force produced by the physical model. The close approximation of the two curves, show that we have correctly approximated the force produced in the full physical model and the reduced buffer model.

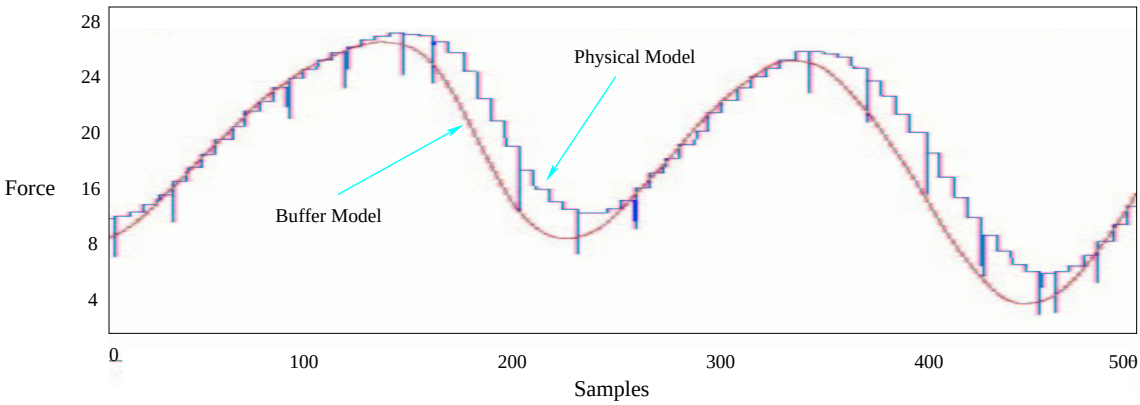


Figure 6.8: *The curve of the force produced by the buffer model in the haptic loop shows a smooth behavior (1KHz) while the force from the physical model is discontinous (30Hz). The value of both the curves are approximately the same.*

6.2.5 Conclusion

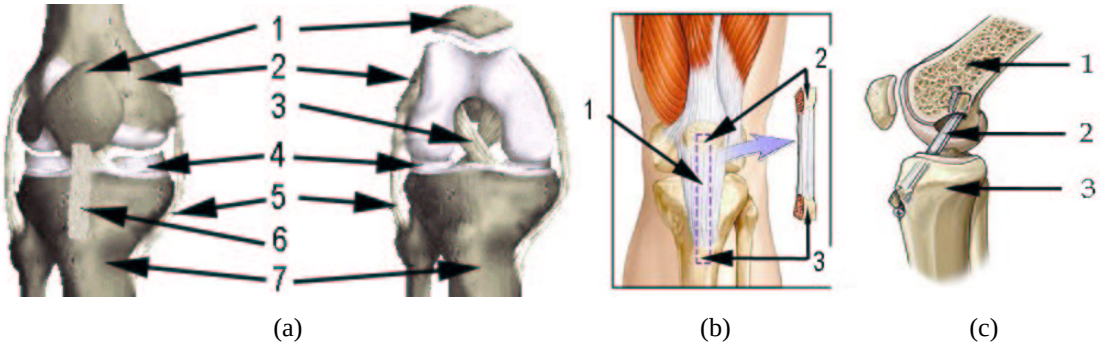
We have presented the application of the VDM physical model in an echographic thigh exam simulator. We have investigated global small deformations and faithfull force feedback. We have found that the volume of the thigh has been conserved throughout the entire history of load application. Feedback to the user has been continuous and realistic. There are nevertheless some shortcomings in this prototype. The applied VDM model has been tested in a limited context. This is because we assumed that the medical professional moves the probe perpendicular to the surface of the thigh. This is nevertheless the normal practice in an echographic thigh exam. Furthermore, for validation purposes, we only had measured data for perpendicular movements. It would be interesting to extend the VDM thigh model to an arbitrary displacement case. For this new measurements must be obtained to verify the results.

6.3 Arthroscopy Knee Reconstruction

6.3.1 Motivations

The replacement of an injured anterior cruciate ligament (ACL) using minimum invasive techniques is called an arthroscopy knee reconstruction (AKR). The ACL (see figure 6.9) is the primary stabilizer of the knee joint which is frequently injured by a twisting or pivoting movement. Left untreated, an ACL injury can allow a process of deterioration and dysfunction of the knee to occur. The replacement of the damaged ACL with a strong biologic substitute is necessary to restore this primary stabilizing structure of the knee. To perform this, a natural graft is first harvested, then two bone tunnels are drilled in the tibia and the femur. Finally, the graft is inserted inside these tunnels and fixed using interference screws. A wrong positioning of one or both tunnels can easily lead to a failure of the graft. As the ACL graft geometry does not fit the original ACL shape, the goal, when placing the graft, is to obtain an isometric behavior for it during knee flexion.

However, because only a small area of the graft section is isometric, even in the best case, the graft is subjected to stress during flexion, and may fail if this stress is above its failure threshold. Therefore, the application developed [Sundaraj et al., 2003] has the goal of automatically generate, intra-operatively, a dynamic physical model of the graft in order to be able to predict failure in case of too high stress during flexion *before* the tunnels are drilled. Since the ACL graft contains mainly of blood and elastic fibers, the VDM physical model can be considered a suitable choice. This model must be built and used during surgery. We aim to



(a) (b) (c)

Figure 6.9: *(a) The knee joint anatomy with extension on the left and flexion on the right, (1)Patella (2)Femur (3)ACL (4)Meniscus (5)Collateral Ligament (6)Patella Tendon (7)Tibia. (b) Bone-Tendon-Bone ACL graft harvesting, (1)Patella Tendon (2)Patella (3)Tibial Tubercule. (c) ACL graft insertion and fixation using screws, (1)Femur (2)ACL Graft (3)Tibia.*

avoid tibial or femoral tunnel positions which will lead to failure of the graft, due to high stress during knee flexion. Within this context, we investigate interative-time global deformations of the graft and collision detection with the surroundings which are the main factors that contribute to stress in the graft.

6.3.2 AKR using OrthoPilot

Aesculap's OrthoPilot CT-Free navigation system is compounded by a computer, a localizer (a Polaris[1] Infrared Camera) and two foot-switches as shown in figure 6.10. This system helps surgeons to find the tibia and femur tunnels positions and orientation in order to avoid impingement while providing isometric graft positioning [Saragaglia et al., 2003], [Eichhorn, 2002].

To use this system, firstly, markers are placed on the tibia and the femur in order to localize the bones. Following this, specific additional landmarks are acquired by the surgeon. The leg's kinematics is also acquired, by performing a flexion-extension motion. With these measurements, the surgeon queries the OrthoPilot system for the best possible configuration to navigate the tibia and femur drill guides such that the graft is isometric and there is no impingement with the surroundings. Isometric positioning refers to the insertion areas on the tibial and femoral bones that maintain a constant length of the ACL graft during flexion. Dessenne in [Dessenne, 1996] has shown that these isometric areas presents an ellipsoid shape. In the current system, the tunnel sites are navigated in order to have the most isometricity between the

[1]Polaris camera are built by Northern Digital Inc.

Figure 6.10: *The Aesculap OrthoPilot workstation.*

posterior side of the tibial tunnel and the *anterior* side of the femoral tunnel. This choice seems to be most appropriate from the surgical point of view.

This system has been used in the operating room for more than 90 times, and has proven it's usefulness for precise ACL graft replacement. But the correct isometric insertion areas are very much smaller than the graft cross-sectional area (isometric insertion areas are often less than $5mm^2$ and a graft cross-sectional area is about $50mm^2$). That is to say that, even if it is implanted over the isometric insertion areas, the **graft is subjected to stress** during leg flexion. Hence, there is a need to model the physical behavior of the ACL graft in order to know if it will be subjected to high stresses during knee flexion. This model must be generated intra-operatively, produce physically realistic deformation and must be simulated in interactive-time.

6.3.3 ACL Deformable Model

Geometrical Model. The geometrical model of the virtual ACL graft comes from the patient. During the ACL reconstruction procedure, the surgeon begins by an arthroscopic inspection of the knee joint. This examination allows the surgeon to practice, if necessary, meniscus resection and to remove the damaged remaining pieces of the torn ACL (on tibia and femur sides). Then, the graft is harvested. The graft may be compounded by a piece of bone, taken from the patella, a piece of the patella tendon and a piece of bone taken on the tibia (see figure 6.9). Other types of grafts can be used, according to the choice of the surgeon. Once the graft has been harvested, it's measurements can be determined. We note that this step is

per operation. The harvested graft is generally a beam shape deformable object. This object is then discretized according to the surgeon's choice. Given the planned orientation of the graft with respect to the patient's knee bones, the two ends of the graft are modified to reflect the positioning with respect to the generic femur and tibia bones (see figure 6.12). This is done by tapering off both the ends of the graft according to the surgeon's choice. A tapered virtual ACL graft is shown in figure 6.11.

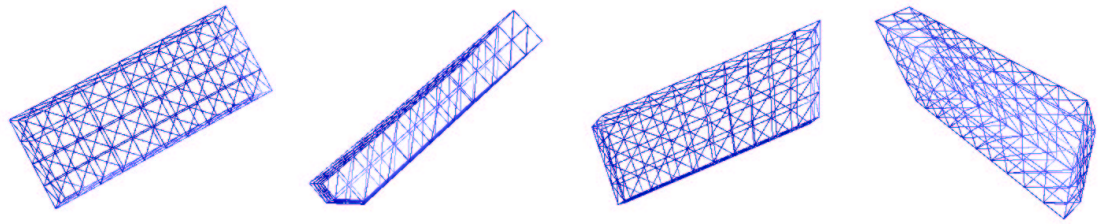


Figure 6.11: *The virtual ACL graft seen from various views. The extreme left view is the original ACL graft without modifications.*

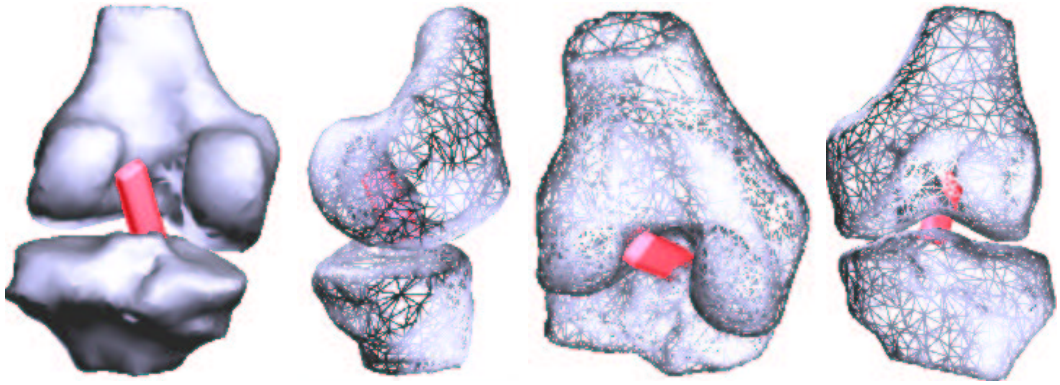


Figure 6.12: *Positioning of the virtual ACL graft with respect to the generic tibia and femur seen from various views.*

Physical Model. The VDM model is used as the physical model for the ACL graft. Given the geometrical model, the distributed volume $\mathcal{V}_i$ and distributed area $\boldsymbol{\mathcal{A}}_i$ can be extracted. Then, each node i which is connected to neighboring nodes j is governed by the following equation:

$$\frac{B_i}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i\Big) + \sum_j \frac{B_{ij}}{\mathcal{V}_i}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i - \boldsymbol{\mathcal{A}}_j\boldsymbol{\Delta L}_j\Big) - \Delta P = \rho_i g \delta_i \tag{6.7}$$

and the following boundary condition:

$$\sum_i^{\aleph} \Delta\mathcal{V}_i = \sum_i^{\aleph}\Big(\boldsymbol{\mathcal{A}}_i\boldsymbol{\Delta L}_i + \boldsymbol{L}_i\boldsymbol{\Delta\mathcal{A}}_i\Big) = 0 \tag{6.8}$$

where for each node i, B_i is the bulk modulus, B_{ij} is the connectivity bulk modulus, $\boldsymbol{\Delta L}_i$ is the displacement vector, ρ_i is the associated density, g is the gravitational constant and δ_i is the hydrostatic distance. This physical model is mapped directly onto our surface based geometrical model of the virtual ACL graft.

Parameter Estimation. This is the most difficult part of our work. The parameters of the VDM model namely the bulk modulus B_i, and the connectivity bulk modulus B_{ij} need to be estimated so that we can have reasonable accuracy in our results. Since we do not have any means and expertise to really conduct experiments to determine these values on the operated patients, we had to resort to data available in the literature. We considered that the ACL graft is full of blood (incompressible) and contains mainly elastic fibers. A problem here is that we had to match physical parameters that were only available for finite element models of high complexity to our VDM model. We used [Pioletti, 1998] and [Tumer and Engin, 1993] as sources. The parameters are shown in table 6.2.

Table 6.2: *Physical data of the virtual ACL graft*

Height (H)	5 mm
Width (W)	10 mm
Cross-Section Area ($\mathcal{A}$)	50 mm^2
Length (L)	25 mm
Surface Area ($\mathcal{S}$)	850 mm^2
Volume ($\mathcal{V}$)	1250 mm^3
Resolution (Q)	138 elements
Density (ρ) - Blood	1055 kgm^{-3}
Bulk Modulus (B_i) - Ligament	0.3 x 10^6 Nm^{-2}
Connectivity Bulk Modulus (B_{ij}) - Fiber	4.0 x 10^3 Nm^{-2}
Angle of Flexion (θ)	$0° - 90°$

System Resolution. Due to the nature of the simulated environment, the two ends of the virtual ACL graft is subjected to constraints. The end which is in contact with the femur is fixed while the other end is subjected to the displacement of the tibia bone, $\mathbb{C}_{tibia}$. These respective nodes i^*_{femur} and i^*_{tibia} are constrained by the following equations:

$$K^*_{ii_{femur}} = \alpha, \;\; R^*_{i_{femur}} = 0 \tag{6.9}$$

$$K^*_{ii_{tibia}} = \alpha, \;\; R^*_{i_{tibia}} = \alpha \mathbb{C}_{tibia} \tag{6.10}$$

where $\alpha \gg 0$.

A nonlinear analysis using the Bi-Conjugate Gradient (BCG) method which has been presented earlier is used to solve the system. The distributed area $\boldsymbol{\mathcal{A}}_i$ is updated at each time-step. As we have shown before, the VDM model of the ACL graft gives us $\aleph + 1$ equations and $\aleph + 1$ unknowns: $\boldsymbol{\Delta L}_i$ from $i = 1 \ldots \aleph$ and ΔP. This system can be written in the form $\boldsymbol{K}\boldsymbol{\Delta L} = \boldsymbol{R}$ where $\boldsymbol{K}$ is the sparse state matrix of the virtual ACL graft, $\boldsymbol{\Delta L}$ is the displacement vector of the nodes and $\boldsymbol{R}$ is the external load vector. Given the displacement of the tibia during flexion, the deformation of the ACL is computed at each time-step. The pseudo-code for this resolution procedure is given in algorithm 3 of chapter 3.

6.3.4 Intra-Operative Surgery

Data Acquisition. The first step in using the simulator is getting the dimensions of the virtual ACL graft. This is obtained from the harvested graft taken off the patella tendon of the patient. These dimensions are fed into the simulator to generate a virtual geometrical mesh of the ACL graft. Then the VDM physical model is mapped onto the geometrical model.

Then specific landmarks and the patient's leg kinematics are acquired by the surgeon (see figure 6.13) [Saragaglia et al., 2003], [Eichhorn, 2002]. The kinematic acquisition is actually a set of spatial transformations obtained at 1.5° intervals, from the tibial marker to the femoral one. Each marker is a set of diodes, fixed to the bone using a screw. As the markers positions are surgeon dependent, it not possible to know them *a priori*. Each marker's position and orientation can be read using the infra-red camera. This information is fed into the OrthoPilot system to obtain the **planned** positions and orientations of the tunnels. This information is then used to adjust the previously generated geometric model of the graft such that the virtual ACL graft is oriented correctly with respect to the patient's femur and tibia.

Interactive-Time Diagnostic. Diagnostic is done in real-time by checking the state of the virtual ACL graft for the entire knee flexion using the graphical user interface of the AKR medical simulator as shown in figure 6.14. This interface is a dynamic simulator developed within our research team called **AlaDyn3D**. It will be linked to the OrthoPilot system through a serial interface for data transfer. The simulator consists of two views; one for the graphical view of deformation and another for the surgeon to manipulate the simulation.

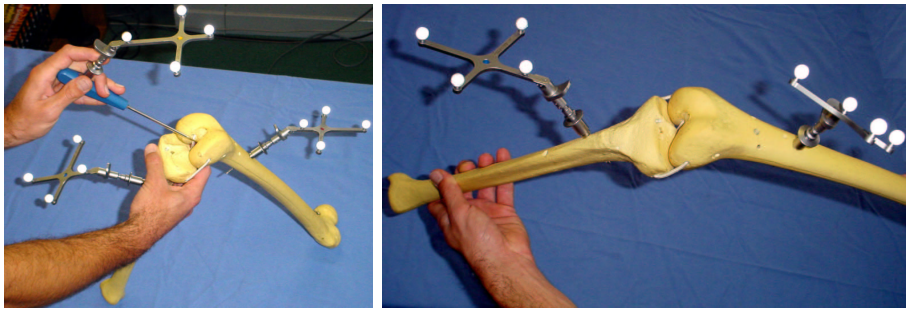


Figure 6.13: *Acquisition of specific landmarks on the right and the patient's knee kinematics on the left. The markers can be seen attached to the bones.*

During surgery, the surgeon basically has two options; he/she can either start the simulation by clicking on the *Animation* button of the AKR control view or he/she can flex the tibia using the *Flexion* slide. Both options flex the tibia at an 1.5° interval, the difference being automatic or manual. Similar procedures are applied if the surgeon wishes to rotate the tibia with respect to the femur using the *Rotation* slide. If the surgeon is not happy with the results of the stress state of the virtual ACL graft for a particular angle of flexion, he/she can inquire OrthoPilot for another suitable position and orientation of the ACL graft. The new data is fed into the AKR simulator and the process is repeated until the surgeon finds an optimal configuration.

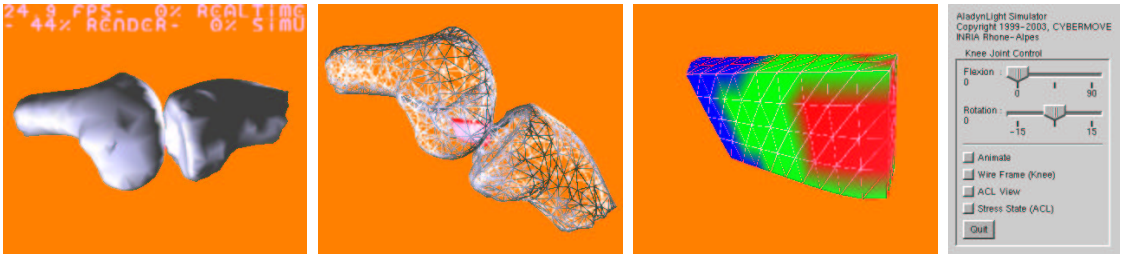


Figure 6.14: *The AKR medical simulator. A generic mesh of the femur and tibia is included so that the surgeon can better visualize the ACL graft. Stress distribution of the ACL ligament is represented by a color code.*

6.3.5 Experimental Results

To test our model, the virtual ACL graft with its ends fixed at the femoral and tibial tunnels outlets, was subjected to a sample set of transformations obtained from OrthoPilot's database. The acquired leg kinematics is used as position boundary

conditions for the physical model. We will assume that only the tibia moves with respect to the femur during flexion. We note here that OrthoPilot has already been successfully used in more that 90 operations. A sample configuration of the **planned** position and orientation of the ACL graft was also obtained from this system.

Deformation and Stress. In figure 6.15 and figure 6.16, we show the results given by our dynamic simulator. We would like to note that the bones of the knee joint are *not* acquired by the system because this is a CT-Free procedure. The system only generates the ACL graft mesh. We included these bones to show the site of the virtual ACL graft with respect to a patient's knee. To better visualize the simulation, we have found that it is better to include a generic mesh of the femur and tibia. At each transformation of angle 1.5°, the deformation and the stress state of the ACL was calculated and analyzed to know if the failure threshold has been reached. Thus it is possible to know where the graft will fail, and for which angle of flexion.

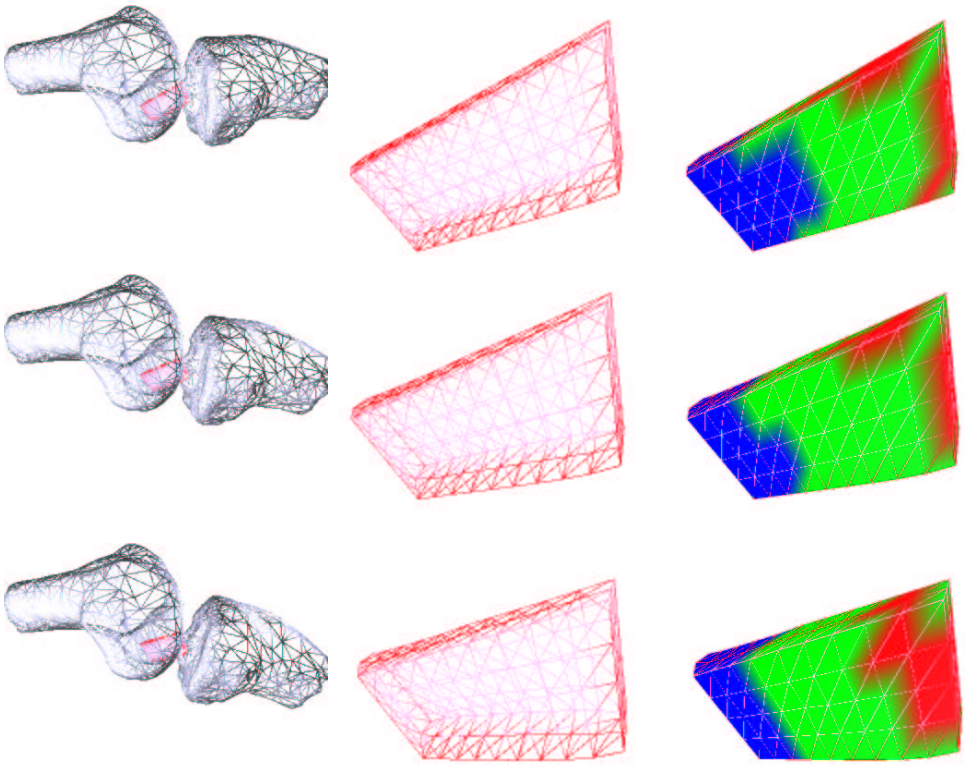


Figure 6.15: *Deformation at* 10°, 20° *and* 30°. *3 views are shown; knee in flexion, ACL graft deformation state and the stress state.*

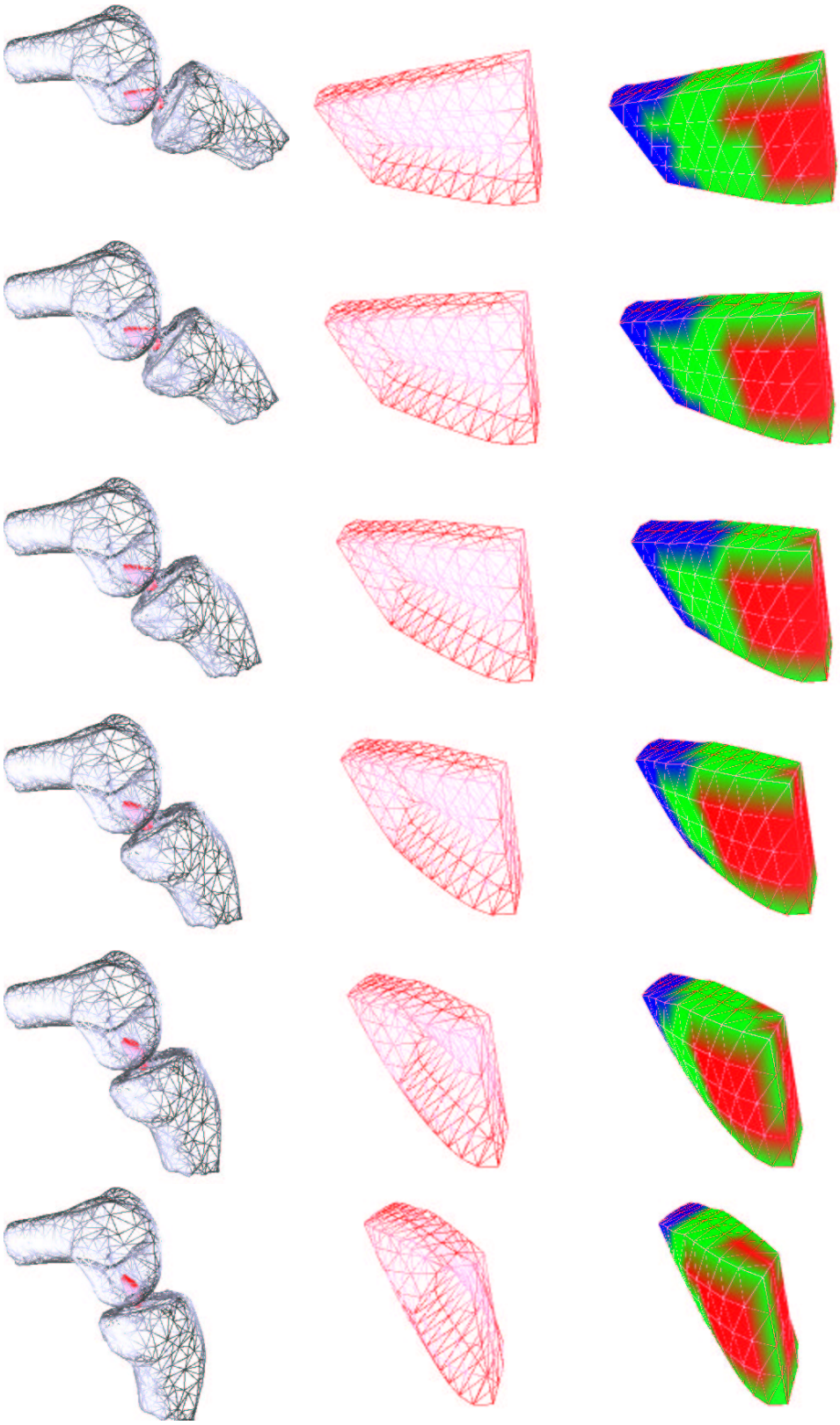


Figure 6.16: *Deformation at* 40°, 50°, 60°, 70°, 80° *and* 90°. *3 views are shown; knee in flexion, ACL graft deformation state and the stress state.*

Collision Detection. A major reason for the failure of the graft is stress due to contact with the bones and surroundings. We have currently omitted the surroundings in our collision detection test. We have tested the interaction between the deformable ACL graft and the rigid bones using the algorithms presented in chapter 5. Our aim is to observe the change in the area of contact given the **planned** position and orientation of the graft. The results are shown in figure 6.17:

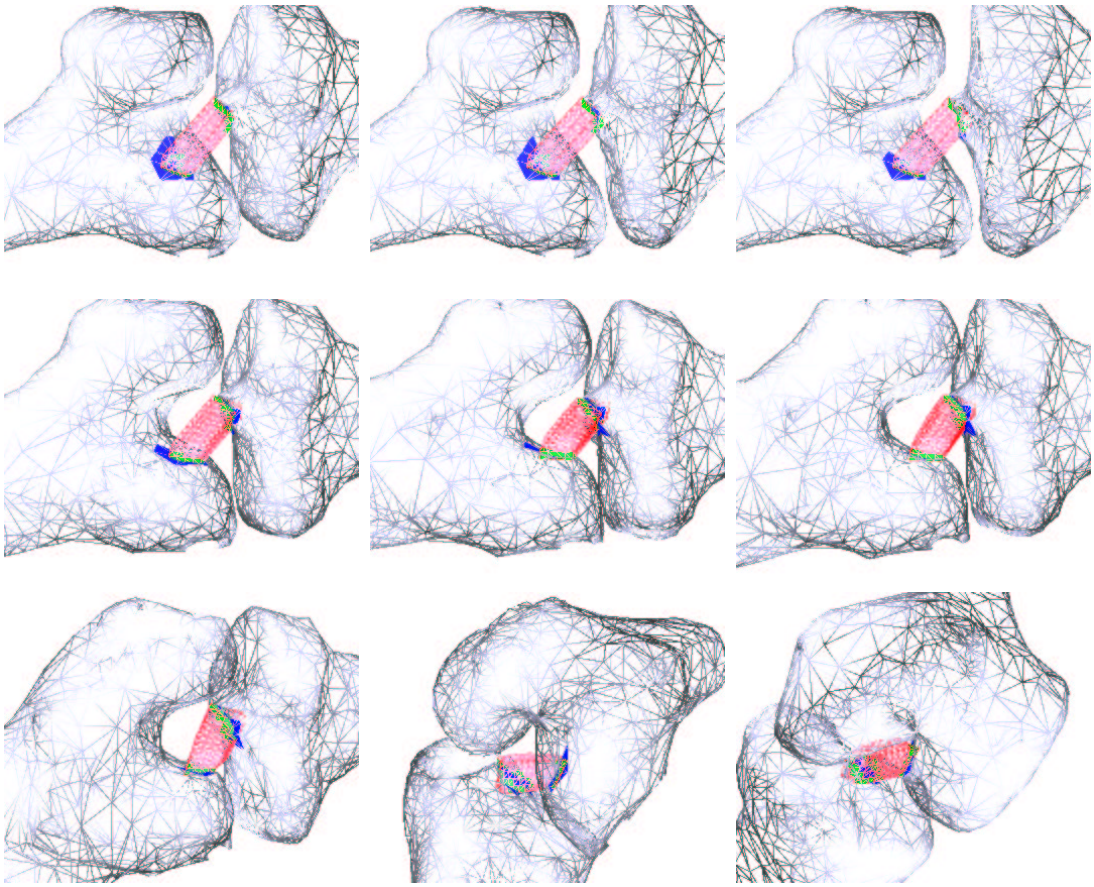


Figure 6.17: *Collision detection at* 10°, 20°, 30°, 40°, 50°, 60°, 70°, 80° *and* 90° *(from left to right). The change in contact area is observed to be very minimal.*

6.3.6 Conclusion

From the mesh configuration and stress state in figure 6.15 and figure 6.16, we can deduce that the graft is stressed at the tibia and femur end the most. This was found to be correct by comparing results from Pioletti in [Pioletti, 1998] and with discussions with surgeons. But Pioletti used a simplified geometrical model of the real ACL ligament constructed from femurial and tibial insertion sites. We on the other hand are interested in the ACL graft, whose dimensions are known for each

patient. A point to note is that since the graft is bigger than the real ligament, more areas are subjected to stress. This is because of the difference between the real isometric area and the area occupied by the graft. Furthermore, the difference in size causes the graft to be more in contact with the knee bones and the surroundings during flexion. This contact also causes stress. The deformation of the ACL graft seems consistent with the predicted results whereby there is very little change in the length of the graft. There is also very little change in volume observed during flexion.

6.4 Summary

In this chapter, we have presented the results of our findings in two medical simulators; an echographic thigh exam (ETE) simulator and an arthroscopy knee reconstruction (AKR) simulator.

Echographic Thigh Exam (ETE). We have implemented a VDM model of the human thigh in an ETE simulator. This model has been calibrated using physical parameters available in the literature. The VDM model is suitable for the thigh because the thigh consists mainly of muscles (blood); hence volume conservation can be assumed. This echographic thigh simulator has been constructed using a dual thread architecture. This architecture has been chosen to solve the problem of different update rates between the physical loop (30Hz) and haptic loop (1KHz). These two loops share a common memory area.

The solution to the interactive-time deformation problem is solved by the physical loop at 30Hz. We believe that this deformation is satisfactory and find that volume of the thigh is conserved throughout the entire history of load application. The deformation vector has been obtained using a quasi-linear resolution approach tailored for nonlinear applications. The algorithms for this approach has been presented. Furthermore, we have shown that the physical loop is in charge of collision detection and the transfer of colliding primitives to the shared memory area.

A deformable buffer model constructed from the colliding primitives in the shared memory area has been integrated in the haptic loop. We divided the simulation in this loop into three parts; CONSTRUCT, UPDATE and LINK. The tasks assigned for each of these stages had been explained. With proper care, we have shown how to avoid read/write problems that may cause trembling at the user end. We have shown how to link the feedback force (to the user) obtained from the buffer model to the feedback force in the physical model. The matching process has been satisfactory.

Arthroscopy Knee Reconstruction (AKR). We have presented the development of a prototype AKR simulator tailored for a CT-Free procedure. The virtual ACL graft in this simulator has been implemented using the VDM model. This model has been calibrated using physical parameters available in the literature. The VDM model is suitable for the ACL graft because it is known that the ACL ligament undergoes very little change in volume during flexion. Furthermore, since our application requires interactive-time compliance, a computationally fast model like VDM is very appropriate.

A graphical user interface has also been designed for the surgeon. This interface is user friendly and allows the surgeon to have the maximum possible information during surgery. More particularly, the surgeon will have information about the state of stress of the ACL graft and the shape of deformation. This will enable the surgeon to **predict** if the planned position is acceptable or not. If the planned position is unsatisfactory for a specific angle of flexion or the deformation of the graft is such that too much friction with the surroundings is expected, the surgeon can query the OrthoPilot system for a new planned position. In addition, we have integrated a generic femur and tibia bone mesh into our prototype to help the surgeon better visualize the positioning of the ACL graft and the state of deformation. We find that the surgeon can better decide and is more comfortable when the generic bones are present. The graft is better visualized in 3D when it is rotated in space together with the bones.

The solution to the interactive-time deformation problem is solved by the physical loop at 30Hz. We believe that this deformation is satisfactory and find that volume of the ACL is conserved throughout the entire history of tibia flexion. The deformation vector has been obtained using a static nonlinear approach tailored for applications involving large deformations. The fact that the ACL graft exhibits negligible viscoelastic behavior, prompts us to use a static resolution method instead of a dynamic approach. The resulting system is maintained numerically stable throughout the simulation. The algorithms for this approach has been presented. Furthermore, we have presented interactive-time collision detection between the deformable ACL graft and the rigid generic bones.

Chapter 7

Conclusion

7.1 Summary of Findings

Soft Tissue Model. We have presented and examined various physical models that have been used for soft tissue simulations. In particular, we have been interested in the complexity of the models and realism. It is clear that there exists a trade-off between speed and accuracy. By recognizing that soft tissue obeys the law of conservation of volume, we proposed a new physical model; the Volume Distribution Method (VDM), for soft tissue simulation. This new model is derived from bulk variables like pressure and volume. In terms of complexity, this new model is one order of magnitude lower than classical volumic models like the finite element method (FEM). We have also shown that VDM can be used to model properties like anisotropic and large deformations. Three different techniques for static resolution have been examined. We believe that a static resolution is sufficient for soft tissue simulation because this type of material is well-damped and hence viscoelastic effects can be neglected.

Collision Detection. Medical simulators require interaction with the practitioner. Hence, we studied collision detection between complex polygonal models and a virtual tool. In general, this is an expensive and time-consuming task. Since the objects being simulated is deformable and the motion of the surgeon is not known *a priori*, we examined different static interference tests (SIT) for collision detection. In these form of tests, an underlying data structure is often used to lower the computational overhead. Thus, we proposed a single underlying data structure for various types of interference queries. We find that the computational overhead cost of updating a single underlying data structure will significantly improve the interactive-time compliance of the application. We also proposed two original algorithms for distance

computation and contact localization. The performance, importance and benefits of these algorithms have been presented. We have further integrated these algorithms into a single library which is independent of the 3D rendering engine and the operating system platform. The codes written for this library is available for download and has been integrated and tested on various 3D simulators.

Application to Medical Simulators. We have integrated our findings in two prototype medical simulators; an echographic thigh exam (ETE) simulator and an arthroscopy knee reconstruction (AKR) simulator. In the earlier, we were interested in global small deformations and faithful haptic feedback while in the latter we were interested in global large deformations. Collision detection was investigated in both cases.

In the ETE simulator, based on assumption that the thigh exhibits volume conservation, we examined the feasibility of using the VDM model to represent the human thigh. In this context, we tried to identify and match physical parameters for our VDM model. This was done using the data obtained from the literature. The simulation was then done using a quasi-linear static analysis, owing to the fact that this procedure normally consists of very delicate motions and that the human thigh is well-damped. Validation was done by comparing the results against previously measured forces due to the deformation of the thigh using a force sensor. The interaction of the virtual probe and the simulated thigh was handled by our collision detection library, ColDetect. This library was also responsible for localizing the contact elements that would be part of the collision treatment process. Within this context, we investigated the suitability of the VDM model to produce faithful haptic feedback.

The details for a prototype CT-Free AKR simulator has also been presented. In this application, the surgeon needs to know the optimal configuration to position the anterior cruciate ligament (ACL) graft. Typically, a navigationless procedure is executed. In this case, the surgeon places the graft in the original insertion sites of the torn ACL. If navigation is present, a purely geometrical reasoning is used to best position the ACL graft. But since the shape of the ACL graft is unlike the original ACL, the graft will be subjected to additional stress during flexion. Hence, the aim of this simulator is to help the surgeon decide the best placement such that failure due to too high a stress is avoided. We have contributed to this procedure by adding a physical constraint to the geometrical one. The VDM model of the ACL graft allows the surgeon to inquire online the stress state of the graft given a *planned* configuration. With this model, the surgeon has the possibility to visualize deformation and stress distribution of the ACL graft. The model is realistic, obtained

preoperatively and is simulated in real-time. A user friendly graphical user interface has also been designed. This interface is easy to understand and easy to use because the surgeons do not have any free hands to touch the computer during surgery. In fact, our first prototype is very simple in the sense that the model is hidden, because the surgeon only cares about the result: "*Is the ACL graft well placed or not?*" and not about the underlying models used by the system.

7.2 Analysis

The ultimate issue that we would like to address is the feasibility of an interactive-time medical simulator. Two major distinct problems is envisaged; realism and rapidity. Realism cannot be compromised because surgeons are going to be trained on these simulators or assisted by these simulators. If their hands-on experience is far from reality, then training becomes useless. Rapidity is required to ensure no sense of discomfort; visually or tactically. Deformation and force feedback has to be computed fast enough such that the practitioner gets the impression that the virtual environment is real.

In this context, this thesis has concentrated on physical models for soft tissue and collision detection for medical simulators. The choice of the used physical model must be based on the targeted application. Do we require speed or accuracy? Ideally of course, we would want both. However, if this is impossible to achieve because of the computational overheads, we must carefully examine the tradeoffs between accuracy and speed. For example, in the echographic simulator, the points of concern were interactive-time global deformation and force feedback. Accuracy in the deformation was compromised by using a quasi-linear analysis supported by the fact that only small deformations were foreseen. On the other hand, in the arthroscopy knee simulator, global large deformations were expected. Hence a complete nonlinear analysis was carried out. But, speed was not entirely compromised. The VDM model being less complex, allowed us to retain the speed required for interactive-time compliance.

But medical simulators are not limited to purely simulation. In many cases, interaction with the practitioner is inevitable. With regards to this, collision detection becomes a significant issue. The problem here is that, very often, algorithms dedicated to this problem are too restrictive. These algorithms often break down when complex situations are encountered. The reason for this is the overhead cost in maintaining the underlying data structure used for collision detection. Thus, the goal in this case would be to have maximum flexibility with minimum computa-

tional cost. The state of the art in this field suggests that a single underlying data structure must be exploited to the maximum to achieve optimal cost. This has been the aim of this thesis. An efficient data structure that optimizes memory and an efficient method of manipulating hierarchies is required. We have presented several algorithms to this end.

In conclusion, we have proposed an alternative physical model for soft tissue simulation and implemented a collision detection library suitable for medical simulators. Our findings have allowed us to investigate the feasibility of developing prototype simulators for relevant medical procedures. It is here that our main contributions lies; the analysis of these methods in terms of accuracy and computational complexity are essential towards the building of virtual medical simulator. Hence, we hope that future work involving feedback from medical professionals and clinical tests will permit us to extend this work in order to eventually provide an efficient training tool for practitioners.

7.3 Perspectives

There is still a lot of work that can be done to contribute to the development of medical simulators. We list a few future directions here:

Algorithms. Various algorithms that are part of a medical simulator can be improved. The conception of new numerical resolution methods and collision detection algorithms are still mandatory. In particular multi-grid solvers or successive over-relaxation techniques could offer an interesting alternative to the conjugate gradients solvers used in this work for nonlinear analysis. While exponential improvements in computing power have contributed to the development of today's processing capabilities, computing power alone does not account for the dramatic expansion of the field, nor will future improvements in computer hardware be a sufficient springboard to enable the development of the medical simulators described in this thesis. Development will require continued research in new algorithms and the mathematical sciences of physical models of soft tissue, fields that have contributed greatly to the biomedical domain and will continue to do so.

Parallelism. A possible direction of future work would be to advance in the development of low-cost, high performance computation systems based on PC clusters, specifically for the medical VR area. We foresee that many of the procedures for simulating physical systems can be parallelized. In particular the linear solvers and the

calculation of the internal forces would be suitable candidates. The purpose of this research would be to reduce the computational costs associated to various processes by developing methods to simplify mathematical formulations and the distribution of computing resources, taking into consideration simultaneously issues like accuracy and realism. Another purpose would be to develop mechanisms for algorithm parallelization which will allow using a computer cluster for tasks like real-time collision detection in surgical simulation environments, with possible changes in object geometry depending on the operation of the surgical instruments or real-time haptic response computation in arthroscopy environments.

Medical Imaging. Many of the envisioned innovations in this thesis are fundamentally dependent on medical imaging. Equations that link imaging measurements to quantities of interest must be sufficiently complex to be realistic and accurate and yet simple enough to be interactive-time compliant. The development of mathematical methods for producing images from projections thus also requires a capability for overcoming errors or artifacts of the reconstruction method that arise from different sources, and much remains to be done. The result is the need for approximate reconstruction strategies or the use of accurate generic models. In addition, mathematical models and computer simulation of deforming images plays an essential role in allowing the mathematician and physicist to critically evaluate new ideas in the emerging field of dynamic biomedical imaging.

Software Development. A realistic medical simulator is still far from reality. But nevertheless, some commercial products and prototypes have been made for certain procedures. Needless to say, there is still much work to be done. An area of paramount interest is the identification of parameters of the different variables of the physical models for soft tissue simulation. This requires collaboration with medical professionals and researchers from the biomechanical field. The expertise of the graphics and computer science community cannot be spared because complex environments will consist of millions of polygons and will require real-time rendering, physically realistic behavior, numerically stable simulation and fast collision detection. It is clear that the knowledge required to produce a realistic medical simulator is very broad; it needs the integration of several disciplines. Within this context, the increase in collaboration amongst the surgical simulation community is a favorable sign that medical simulation will indeed not remain as a myth but become reality in the near future.

APPENDICES

Appendix A

Collision Detection Libraries

A.1 Convex Based Packages

Convex packages rely on objects being convex or composed of convex pieces. There are two main classes of algorithms that are used for convex polyhedral proximity query. One is the Voronoi region based Lin-Canny (LC) algorithm and its derivatives. Another one is the simplex-based Gilbert-Johnson-Keerthi (GJK) algorithm and its derivatives.

1. **EGJK** - *http://www.comlab.ox.ac.uk/cameron/distances.html*
   This is an implementation of the GJK algorithm by S. Cameron which calculates the distance between convex polyhedra. Interference is detected when the distance is zero. This implementation also gives an approximation of the negative distance which is a measure of interpenetration.

2. **I-Collide** - *http://www.cs.unc.edu/ geom/I_COLLIDE/index.html*
   This is the original LC implementation. I-Collide is an interactive and exact collision detection library for large environments composed of convex polyhedra . Many non-convex polyhedra may be decomposed into a set of convex polyhedra, which may then be used with this library. I-Collide uses the closest features tracking algorithm and exploits coherence (the property of a simulation to change very little between consecutive time steps) and the properties of convexity to achieve very fast collision detection which is exact to the accuracy of the input models.

3. **V-Clip** - *http://www.merl.com/projects/vclip/*
   The Voronoi Clip, or V-Clip, algorithm is a low-level collision detection algorithm for polyhedral objects. The V-Clip library is a C++ implementation of

this algorithm, with facilities for constructing and manipulating geometries. The source code is freely distributed for educational, research and non-profit purposes. V-Clip operates on polyhedral objects which may be nonconvex or even disconnected. It returns the closest points between objects and the distances between them. If the objects penetrate, it returns a penetration depth. V-Clip requires the application to specify nonconvex or disconnected objects as hierarchies of convex pieces. When called on disjoint nonconvex objects, V-Clip returns a lower bound on the actual distance between the objects.

4. **SOLID** - *http://www.win.tue.nl/ gino/solid/*
   This is a library for collision detection of $3D$ objects undergoing rigid motion and deformation. SOLID is designed to be used in interactive $3D$ graphics applications, and is especially suited for collision detection of objects and worlds described in VRML. It is an implementation of GJK. Frame coherence is exploited by maintaining a set of pairs of proximate objects and caching separating axes for these pairs. SOLID also computes penetration depth. Deformation is allowed but only for transformations that keep the objects convex. It also requires QHULL to compute the convex hull of an object.

5. **Q-Collide** - *http://www.stanford.edu/ kelchung/collision_library.html*
   This library presents a simple and exact collision detection algorithm for convex polytopes. The algorithm finds quickly a separating plane between two polytopes if they are non-colliding, or else reports collision and the pair of closest points between them if it cannot possibly find a separating plane. In the case of non-collision, the separating plane found for one time frame is cached as a witness for the next time frame; this use of time coherence further speeds up the algorithm in dynamic applications. Both temporal and geometric coherence are exploited to make this algorithm run in expected constant time empirically.

A.2 Polygon Soup Based Packages

Polygon soup packages operate on lists of triangles. They typically create a hierarchy of bounding volumes whose leaf nodes are the triangles themselves. The hierarchies are composed of spheres, AABB's (Axis-Aligned Bounding Boxes), OBB's (Oriented Bounding Boxes), SSV's (Sphere-Swept Volumes), or k-DOP's (k-Discrete Oriented Polytopes). These packages are designed to handle rigid motion but some may be adapted to work based upon deformable objects as well.

1. **SWIFT++** - *http://www.cs.unc.edu/ geom/SWIFT++/download.shtml*
   This algorithm is an improved LC implementation. Provides sweep and prune bounding box support and employs outer bounding hierarchies to speedup walking. Performance can be made independent of motion coherence. SWIFT++ allows objects composed of convex pieces and has been shown to be very robust.

2. **ColDet** - *http://photoneffect.com/coldet/*
   This library is an effort to provide a free collision detection library for generic polyhedra. Its purpose is mainly for $3D$ games where accurate detection is needed between two non-simple objects. It provides exact point of collision, plus the pair of triangles that collided and also supports timeout setting, to limit detection time.

3. **PQP** - *http://www.cs.unc.edu/ geom/SSV/*
   The Proximity Query Package is a library for performing proximity queries on a pair of geometric models composed of triangles. This package uses SSV bounding volumes and computes three types of interference; intersection detection, tolerance verification, and exact and approximate minimum distance computation. PQP takes some advantage of coherence.

4. **RAPID** - *http://www.cs.unc.edu/ geom/OBB/OBBT.html*
   This library is a robust and accurate polygon interference detection library for large environments composed of unstructured models. It uses OBB bounding volumes. It is applicable to polygon soups - models which contain no adjacency information, and obey no topological constraints. The models may contain cracks, holes, self-intersections, and nongeneric (e.g. coplanar and collinear) configurations. It is numerically robust - the algorithm is not subject to conditioning problems, and requires no special handling of nongeneric cases (such as parallel faces).

5. **V-Collide** - *http://www.cs.unc.edu/ geom/V_COLLIDE/*
   This package used the RAPID library but unlike RAPID, it supports many simultaneous objects. V-Collide keeps track of where objects are, so that if objects do not move between queries their locations need not be resupplied to the collision detection system On the other hand, V-Collide only reports when pairs of objects collide and not the distance between them.

6. **QuickCD** - *http://www.ams.sunysb.edu/ jklosow/quickcd/QuickCD.html*
   QuickCD is a general-purpose collision detection library, capable of performing fast and exact collision detection on highly complex models. No assumption

is made about the structure of the input. QuickCD robustly handles unstructured inputs consisting of a polygon soups. No adjacency information is needed; the models are only specified as a collection of triangles. The library is based upon constructing hierarchies of discrete orientation polytopes, or k-dops, which are convex polytopes whose facets have normals from a given discrete set of k vectors, to approximate the input models.

7. **H-Collide** - *http://www.cs.unc.edu/ geom/H_COLLIDE*
   H-Collide is a framework for fast and accurate collision detection for haptic interaction. It consists of a number of algorithms and a system specialized for computing contact(s) between the probe of the force-feedback device and objects in the virtual environment. To meet the stringent performance requirements for haptic interaction, we use an approach that specializes many earlier algorithms for this application. this library utilizes spatial decomposition, bounding volume hierarchy based on OBB-Trees and frame-to-frame coherence.

Appendix B

ColDetect - Reference Manual

B.1 Introduction

ColDetect is a library for collision detection, exact distance computation, and contact localization of three-dimensional polygonal objects. These objects can be concave or convex, rigid or deformable. It is numerically robust - the algorithm is not subject to conditioning problems, and requires no special handling of nongeneric cases. ColDetect has been implemented in standard C++ and relies heavily on STL in order to be as fast and memory efficient. Currently it compiles under GNU g++ version 2.95 and 3.2. It provides a very simple API. The main features of ColDetect is as follows:

- Computing distance between convex objects
- Computing distance between concave objects
- Computing distance between convex and concave objects
- Collision detection between concave and convex rigid objects
- Collision detection between concave and convex deformable objects
- Contact localization between concave and convex rigid objects
- Contact localization between concave and convex deformable objects
- Frame coherence is exploited
- Real time algorithms
- C++ implementation
- Portable library

B.2 Installation

These installation instructions apply to UNIX/Linux systems and assumes that you have root access. Upon installation, the ColDetect library for collision detection for medical simulators will be obtained. If you have to alter or modify any steps in order to install on your computer configuration, or if these instructions are not clear or if these instructions do not work , please email `Kenneth.Sundaraj@inrialpes.fr` with the details. To install, do the following:

- Download the file $coldetection- < VERSION > .tar.bz2$ from the site `http://www.inrialpes.fr/sharp/coldetection` and substitute the ColDetect version number for $< VERSION >$
- Expand the above files into the directory that will be used for compiling, *e.g.* $bzip2 \ -dc \ coldetection- < VERSION > .tar.bz2 \mid tar \ xf \ -$
- Change to the directory where you have expanded the files, *e.g.* $cd \ /usr/src/coldetection- < VERSION >$
- This step does not apply to a native Windows OS build. After reading about all the available options in $configure.usage$, type: $./configure \ [--OPTION[= VALUE]...][CONFIGURATION]$
- Type *make*
- Switch to the root user and type: *make install*

If you installed the library in a specific directory, you have to set the include and library paths accordingly.

Index of cited Authors

J

K

L

M

N

P

Q

S

T

U

V

W

Z

Bibliography

[Aulignac, 2001] Aulignac, D. (2001). *Modelisation de l'interaction avec des objets déformables en temps-réel pour des simulateurs medicaux*. PhD thesis, Institute Nationale Polytechnique de Grenoble, France.

[Aulignac et al., 1999] Aulignac, D., Laugier, C., and Cavusoglu, M. C. (1999). Towards a realistic echographic simulator with force feedback. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Baciu et al., 1998] Baciu, G., Wong, W., and Sun, H. (1998). Hardware assisted virtual collisions. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*.

[Baraff and Witkin, 1992] Baraff, D. and Witkin, A. (1992). Dynamic simulation of non-penetrating flexible bodies. In *Computer Graphics*.

[Baraff and Witkin, 1998] Baraff, D. and Witkin, A. (1998). Large steps in cloth simulation. In *Computer Graphics*.

[Bathe, 1996] Bathe, K. J. (1996). *Finite Element Procedures*. Prentice Hall.

[Bourguignon and Cani, 2000] Bourguignon, D. and Cani, M. P. (2000). Controlling anisotropy in mass-spring systems. In *Proceedings of EACG Conference on Eurographics*.

[Boux-de-Casson, 2000] Boux-de-Casson, F. (2000). *Simulation Dynamique de Corps Biologiques et Changements de Topologie Interactifs (in French)*. PhD thesis, Université de Savoie, France.

[Bro-Nielsen and Cotin, 1996] Bro-Nielsen, M. and Cotin, S. (1996). Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Proceedings of EACG Conference on Eurographics*.

[Brown and Montgomery, 2001] Brown, J. and Montgomery, K. (2001). A microsurgery simulation system. In *Proceedings of Medical Image Computing and Computer Assisted Intervention - MICCAI*.

[Cameron, 1997] Cameron, S. (1997). Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Cohen et al., 1995] Cohen, J. D., Lin, M. C., Manocha, D., and Ponamgi, M. K. (1995). I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics*.

[Costa and Balaniuk, 2001a] Costa, I. F. and Balaniuk, R. (2001a). LEM - An approach for real time physically based soft tissue simulation. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Costa and Balaniuk, 2001b] Costa, I. F. and Balaniuk, R. (2001b). Static solution for real time deformable objects with fluid inside. In *ERCIM News*.

[Cotin, 1997] Cotin, S. (1997). *Modèles anatomiques déformables en temps réel (in French)*. PhD thesis, Université de Nice, France.

[Cotin et al., 1999] Cotin, S., Delingette, H., and Ayache, N. (1999). Real-time elastic deformations of soft tissues for surgery simulation. In *IEEE Transactions on Visualizations and Computer Graphics*.

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

[Davanne et al., 2002] Davanne, J., Messure, P., and Chaillou, C. (2002). Stable haptic interaction in a dynamic virtual environment. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[De and Bathe, 2001] De, S. and Bathe, K. J. (2001). The method of finite spheres: A summary of recent developments. In *Proceedings of MIT Conference on Computational Fluid and Solid Mechanics*.

[Debunne et al., 1999] Debunne, G., Desbrun, M., Barr, A., and Cani, M. P. (1999). Interactive multiresolution animation of deformable models. In *Proceedings of EACG Conference on Eurographics*.

[Debunne et al., 2001] Debunne, G., Desbrun, M., Cani, M. P., and Barr, A. (2001). Dynamic real-time deformations using space and time adaptive sampling. In *Computer Graphics*.

[Deguet et al., 1998a] Deguet, A., Joukhadar, A., and Laugier, C. (1998a). A collision model for deformable bodies. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Deguet et al., 1998b] Deguet, A., Joukhadar, A., and Laugier, C. (1998b). Models and algorithms for the collision of rigid and deformable bodies. In *Proceedings of the Workshop on the Algorithmic Foundations of Robotics*.

[Delingette et al., 1999] Delingette, H., Cotin, S., and Ayache, N. (1999). A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In *Proceedings of CGS Conference on Computer Animation*.

[Derek and Gupta, 1996] Derek, J. and Gupta, K. (1996). Octree-based hierarchical distance maps for collision detection. *Journal of Robotics Systems*.

[Desbrun et al., 1999] Desbrun, M., Schröder, P., and Barr, A. (1999). Interactive animation of structured deformable objects. In *Proceedings of Graphics Interface*.

[Dessenne, 1996] Dessenne, V. (1996). *Gestes Médicaux-Chirurgicaux Assistés par Ordinateur: Applications à la Ligamentoplastie du Genou et la Chirurgie Orthognastique (in French)*. PhD thesis, Université Joseph Fourier, France.

[Dimaio and Salcudean, 2002] Dimaio, S. P. and Salcudean, S. E. (2002). Simulated interactive needle insertion. In *Proceedings of International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator System*.

[Dobkin and Kirkpatrick, 1990] Dobkin, D. and Kirkpatrick, D. (1990). Determining the separation of preprocessed polyhedra - A unified approach. In *Lecture Notes in Computer Science*, volume 443. Springer-Verlag.

[Ehmann and Lin, 2001] Ehmann, S. A. and Lin, M. C. (2001). Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Proceedings of the EACG Conference on Eurographics*.

[Eichhorn, 2002] Eichhorn, J. (2002). *Navigation und Robotik in der Gelenk und Wirbelsäulenchirurgie*. Konermann.

[Garcia-Alonso et al., 1994] Garcia-Alonso, A., Serrano, N., and Flaquer, J. (1994). Solving the collision detection problem. In *IEEE Transactions on Computer Graphics and Applications*.

[Gilbert et al., 1988] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between objects in three-dimentional space. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Gottschalk et al., 1996] Gottschalk, S., Lin, M. C., and Manocha, D. (1996). OBB-Tree: A hierarchical structure for rapid interference detection. In *Proceedings of ACM SIGGRAPH*.

[Hamada and Hori, 1996] Hamada, K. and Hori, Y. (1996). Octree based approach to real-time collision-free path planning for robot manipulator. In *Proceedings of IEEE 4th International Workshop on Advanced Motion Control (AMC)*.

[Hilde et al., 2001] Hilde, L., Meseure, P., and Chaillou, C. (2001). A fast implicit integration method for solving dynamic equations of movement. In *Proceedings of ACM Symposium on Virtual Reality Software & Technology (VRST) Conference*.

[Hoff et al., 2001] Hoff, K., Zaferakis, A., Lin, M. C., and Manocha, D. (2001). Fast and simple 2D geometric proximity queries using graphics hardware. In *Procceedings of ACM Symposium on Interactive 3D Graphics*.

[Hoff et al., 2002] Hoff, K., Zaferakis, A., Lin, M. C., and Manocha, D. (2002). Fast 3D geometric proximity queries between rigid and deformable models using graphics hardware acceleration. Technical report, University of North Carolina.

[Hubbard, 1996] Hubbard, P. M. (1996). Approximating polyhedra with spheres for time-critical collision detection. In *ACM Transactions on Graphics*.

[Hutchinson et al., 1996] Hutchinson, D., Preston, M., and Hewitt, T. (1996). Adaptive refinement for mass/spring simulation. In *Proceedings of EACG Conference on Eurographics*.

[James and Pai, 1999] James, D. and Pai, D. (1999). Accurate real-time deformable objects. In *Proceedings of ACM SIGGRAPH*.

[James and Pai, 2001] James, D. and Pai, D. (2001). A unified treatment of elastostatic contact simulation for real-time haptics. *Journal of Haptics-e*.

[Jimenez et al., 2001] Jimenez, P., Thomas, F., and Torras, C. (2001). 3D Collision Detection: A Survey. *Journal of Computers and Graphics*.

[Johnson and Cohen, 1998] Johnson, D. E. and Cohen, E. (1998). A framework for efficient minimum distance computations. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Johnson and Cohen, 1999] Johnson, D. E. and Cohen, E. (1999). Bound coherence for minimum distance computations. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Johnson, 1987] Johnson, D. W. (1987). *The optimisation of robot motion in the presence of obstacles*. PhD thesis, University of Michigan, USA.

[Joukhadar et al., 1996] Joukhadar, A., Wabbi, A., and Laugier, C. (1996). Fast contact localisation between deformable polyhedra in motion. In *Proceedings of CGS Conference on Computer Animation*.

[Kim et al., 2002a] Kim, J., De, S., and Srinivasan, M. A. (2002a). Computationally efficient techniques for real time surgical simulation with force feedback. In *Proceedings of International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator System*.

[Kim et al., 2002b] Kim, Y. J., Lin, M. C., and Manocha, D. (2002b). DEEP: dual space expansion for estimating penetration depth between convex polytopes. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Larsson and Moller, 2001] Larsson, T. and Moller, T. (2001). Collision detection for continuously deforming bodies. In *Proceedings of EACG Conference on Eurographics*.

[Laugier et al., 2003] Laugier, C., Mendoza, C., and Sundaraj, K. (2003). Towards a realistic medical simulator using virtual environments and haptic interaction. In Jarvis, R. A. and Zelinsky, A., editors, *Robotics Research*, volume 6 of *Springer Tracts in Advanced Robotics (STAR)*. Springer-Verlag.

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

[Lombardo et al., 1999] Lombardo, J., Cani, M. P., and Neyret, F. (1999). Real-time collision detection for virtual surgery. In *Proceedings of CGS Conference on Computer Animation*.

[Mendoza and Laugier, 2003] Mendoza, C. and Laugier, C. (2003). Simulating soft tissue cutting using finite element models. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Mendoza et al., 2003] Mendoza, C., Sundaraj, K., and Laugier, C. (2003). Faithfull haptic feedback in medical simulators. In Siciliano, B. and Dario, P., editors, *Experimental Robotics*, volume 5 of *Springer Tracts in Advanced Robotics (STAR)*. Springer-Verlag.

[Meseure and Chaillou, 1997] Meseure, P. and Chaillou, C. (1997). Deformable body simulation with adaptive subdivision and cutting. In *Proccedings of Winter School of Computer Graphics (WSCG) Conference*.

[Meseure and Chaillou, 2000] Meseure, P. and Chaillou, C. (2000). A deformable body model for surgical simulation. *Journal of Visualization and Computer Animation*.

[Meseure et al., 2003] Meseure, P., Davanne, J., Hilde, L., Lenoir, J., France, L., Triquet, F., and Chaillou, C. (2003). A physically-based virtual environment dedicated to surgical simulation. In *Proceedings of International Symposium on Surgery Simulation and Soft Tissue Modeling*.

[Mirtich, 1998] Mirtich, B. (1998). V-Clip: Fast and robust polyhedral collision detection. In *ACM Transactions on Graphics*.

[Mirtich and Canny, 1995] Mirtich, B. and Canny, J. F. (1995). Impulse based simulation of rigid bodies. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics*.

[Moller, 1997] Moller, T. (1997). A fast triangle-triangle intersection test. *Journal of Graphics Tools*.

[Moore and Wilhelms, 1988] Moore, M. and Wilhelms, J. (1988). Collision detection and response for computer animation. In *Computer Graphics*.

[Naylor et al., 1990] Naylor, B. F., Amantides, J. A., and Thibault, W. C. (1990). Merging BSP trees yields polyhedral set operations. In *Proceedings of ACM SIGGRAPH*.

[Picinbono et al., 2002] Picinbono, G., Delingette, H., and Ayache, N. (2002). Nonlinear and anisotropic elastic soft tissue models for medical simulation. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Pioletti, 1998] Pioletti, D. (1998). *Viscoelastic Properties Of Soft Tissues: Application to Knee Ligaments and Tendons*. PhD thesis, Ecole Polytecnique Federale de Lausanne, Switzerland.

[Ponamgi et al., 1995] Ponamgi, M. K., Manocha, D., and Lin, M. C. (1995). Incremental algorithms for collision detection between general solid models. In *Proceedings of ACM Symposium on Solid Modeling*.

[Press et al., 1992] Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1992). *Numerical Recipes in C*. Cambridge University Press.

[Quilan, 1994] Quilan, S. (1994). Efficient distance computation between non-convex objects. In *Proceedings of IEEE International Conference on Robotics and Automation*.

[Saragaglia et al., 2003] Saragaglia, D., Sauteron, D., Chaussard, C., Boux-de-Casson, F., and Liss, P. (2003). Orthopilot assisted anterior cruciate ligament reconstruction analysis of tunnel positioning in 12 cases. In *Proceedings of Internal Conference on Computer Assisted Orthopaedic Surgery*.

[Sundaraj et al., 2000] Sundaraj, K., Aulignac, D., and Mazer, E. (2000). A new algorithm for computing minimum distance. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Sundaraj and Laugier, 2000] Sundaraj, K. and Laugier, C. (2000). Fast contact localisation of moving deformable polyhedras. In *Proceedings of IEEE International Conference on Automation, Robotics, Control and Vision*.

[Sundaraj and Laugier, 2002] Sundaraj, K. and Laugier, C. (2002). Physically realistic simulation of large deformations using LEM for interactive applications. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Sundaraj et al., 2003] Sundaraj, K., Laugier, C., and Boux-de-Casson, F. (2003). Intra-Operative CT-Free examination system for anterior cruciate ligament reconstruction. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Sundaraj et al., 2001] Sundaraj, K., Laugier, C., and Costa, I. F. (2001). An approach to LEM modelling: Construction, collision detection and dynamic simulation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[Terzopoulos and Waters, 1990] Terzopoulos, D. and Waters, K. (1990). Physically-based facial modeling analysis and animation. *Journal of Visualization and Computer Animation*.

[Thibault and Naylor, 1987] Thibault, W. C. and Naylor, B. F. (1987). Set operations on polyhedra using Binary Space Partioning trees. In *Proceedings of ACM Computer Graphics*.

[Tseng and Lin, 2000] Tseng, D. C. and Lin, J. Y. (2000). A hybrid physical deformation modeling for laparoscopic surgery simulation. In *Proceedings of IEEE International Conference of the Engineering on Medicine and Biology Society*.

[Tumer and Engin, 1993] Tumer, S. and Engin, A. (1993). Three body segment dynamic model of the human knee. *Journal Of Biomechanical Engineering*.

[Ulrich, 2000] Ulrich, T. (2000). *Game Programming Gems*, chapter Loose Octrees. Charles River Media.

[Van der Bergen, 1997] Van der Bergen, G. (1997). Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphic Tools*.

[Van der Bergen, 1999] Van der Bergen, G. (1999). A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphic Tools*.

[Van der Bergen, 2001] Van der Bergen, G. (2001). Proximity queries and penetration depth computation on 3D game objects. In *Proceeding of Game Developers Conference*.

[Vepstas, 1996] Vepstas, L. (1996). High performance graphics hardware design requirements. Technical report, http://linas.org/linux/graphics.html.

[Waters, 1987] Waters, K. (1987). A muscle model for animating three dimensional facial expression. In *Proceedings of ACM SIGGRAPH*.

Real-Time Dynamic Simulation and 3D Interaction of Biological Tissue : Application to Medical Simulators

The advent of medical imaging and new operating techniques has revolutionized the working methods of medical professionals. This change requires physicians and surgeons to undergo additional training. This is why the development of appropriate tools, like medical simulators, is of paramount importance. Within this context, we focus our work on physical modeling of soft tissue deformations and collision detection in virtual environments. First, we present various physical models and the numerical resolution methods associated with deformable objects. We then propose a new model developed for soft tissue simulation, by successively presenting the aspects related to the formulation of the model, the resolution of the model, and the treatment of the physical interactions. This model, based on Pascal's principle, allows us in a relatively simple way to represent biological tissue, thus making it possible for interactive simulation. Next, we present various existing algorithms for collision detection, as well as the difficulty in adapting these algorithms in medical simulators where complex deformable objects form the base of the simulated environment. We then propose the algorithms developed in our work to deal with this problem within the framework of the medical simulators. These algorithms have better numerical robustness and are optimized, allowing us to treat deformable bodies effectively. We apply our results within the framework of an echographic simulator of the human thigh and a simulator for the arthroscopic reconstruction of the ACL (anterior cruciate ligament of the knee).

Keywords : Medical Simulation, Deformable Models, Collision Detection.

Simulation Dynamique en Temps-Réel et Interaction 3D de Tissu Biologique : Application aux Simulateurs Médicaux

L'avènement de l'imagerie médicale et de nouvelles techniques opératoires a bouleversé les méthodes de travail des médecins. Mais ce changement nécessitera une formation renforcée des praticiens et chirurgiens. C'est pourquoi le développement d'outils appropriés comme les simulateurs médico-chirurgicaux se fait de plus en plus ressentir. Dans ce cadre, nous nous sommes intéressés au problème de la modélisation des phénomènes de déformation de tissu biologique et à la détection des collisions dans un environment virtuel. Dans un premier temps, nous présentons les différents modèles physiques existants et les différentes méthodes de résolution numérique associées aux objets déformable. Nous proposons ensuite un modèle développé pour la simulation de tissu biologique, en présentant successivement les aspects liés à la formulation du modèle, à la résolution du modèle, et au traitement des interactions physiques. Ce modèle, basé sur l'utilisation du principe de Pascal, permet de modéliser de manière relativement satisfaisante des corps biologiques, tout en permettant une simulation interactive. Dans un deuxième temps, nous présentons les différents algorithmes existants pour la détection de collision, ainsi que la difficulté d'adapter ces algorithmes aux simulateurs médicaux où les objets déformables complexes forment la base du modèle. Nous proposons ensuite les algorithmes développés pour traiter ce problème dans le cadre des simulateurs médicaux. Ces algorithmes présentent des caractéristiques de robustesse numérique et d'efficacité supérieures á l'existant, et permettent de traiter des corps déformables. Nous appliquons ces résultats dans le cadre d'un simulateur échographique de la cuisse humaine et d'une simulateur de chirurgie arthroscopique du LCA (ligament croisé antérieur du genou).

Mots Clés : Simulation Médicale, Modèles Déformables, Détection de Collision.