A PROGRESSIVE CUTTING SCHEME FOR SURGERY SIMULATIONS

by

Jing Yi

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

Master of Applied Science

in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Faculty of Engineering

Department of System and Computer Engineering

Carleton University

Ottawa, Ontario, Canada

August, 2007

Your file Votre référence
ISBN: 978-0-494-33676-2
Our file Notre référence
ISBN: 978-0-494-33676-2

ABSTRACT

Traditional surgical training methods, such as practising on animals, cadavers or patients, have disadvantages and limits. With the advance in computer and robotics technologies, computer-aided surgery simulation provides a great potential to replace these methods.

There are many challenges to develop a successful surgery simulator: the deformable model must be both realistic and computationally efficient; the haptic model should detect the collision efficiently and render the force feedback faithfully; surgical procedures such as cutting and suturing should be simulated realistically in real-time.

This thesis addresses the simulation of cutting procedures. A tensor-mass model is used and a novel method is developed to distribute the external force applied to an arbitrary point on the model to its neighboring mesh nodes. The Fourth-order Runge-Kutta method is chosen as the numerical integrator of the model.

A new combined subdivision/separation method is developed and implemented. The actual intersection points are snapped to the element nodes or the midpoints of the edges or faces. The intersected elements will be progressively subdivided or separated based on the results of node snapping. The minimal set midpoint subdivision approach ensures that no small or badly shaped elements are created. Therefore the stability of the simulation system is increased. To improve the computational efficiency, a localized intersection detection method is introduced. The state and lookup tables improve the efficiency of re-meshing and keep the consistency of the mesh.

A separated external/feedback force computing method is used for haptic rendering. The external force is calculated by the point-based proxy method, while the feedback force is computed by the physically based method and then applied to the user through a Phantom Premium 1.5 robotic arm.

ACKNOWLEGEMENTS

I would like to express my sincere gratitude to my thesis supervisor at Carleton University, Dr. Peter X. Liu, for the invaluable advice, guidance, encouragement and financial support through this thesis effort. I would also like to thank Dr. Chang Shu at National Research Council Canada for his advice and suggestions.

I also want to thank all colleagues in my lab for sharing their ideas and knowledge with me.

A special thank goes to my family who have had to endure this process along with me. To my husband, Chunsheng, for his patience and understanding in whatever difficult situations; to my parents, especially my mother, for supporting me with every effort, your love is always so treasured; finally to my daughter, Karen, her smile lightens my dark nights.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Traditionally surgeons develop all required skills through years of surgical training on animals, cadavers and patients. There are many problems with these training methods: animals are expensive and do not always reflect human anatomy; cadavers have different mechanic properties comparing with living tissues, so that they can not provide appropriate physiological responses; training on patients is always risky. In recent years, the development and maturation of technologies in Minimally Invasive Surgery (MIS) imposed more challenges upon the traditional training methods and spawned an alternative approach: computer-based surgery simulation.

MIS is performed through keyhole incisions with the aid of some long rods, a laparoscope or an endoscope, and some other surgical instruments such as graspers, scissors and staplers. These instruments are mounted on the rods and inserted into the human body through the keyhole incisions. Surgeons can operate them from outside. The operating scene is viewed through a laparoscope or an endoscope. These operating conditions restrict surgeons' vision and mobility, and require them to have very good hand-eye coordination. Surgeons need sufficient training which usually can not be acquired by operating on patients.

With a surgery simulation system (© TriS), human body and organs are modeled as virtual objects in a virtual reality environment. Surgeons can manipulate these objects through haptic devices and do operations as if they were operating on the real patients. The operations can be repeated without limit, the results can be recorded for late study and the training cost is very low.

Surgery simulation was initially aimed at training surgeons for MIS. In recent years, it has expanded to all kinds of open surgeries and has become a research frontier in the fields of medical and computer science. It seems that surgery simulation will not only replace the traditional surgical training methods, but also play an important role in other medical procedures such as surgery planning and surgery pre-operational rehearsals.

1.1 Motivation

Computer-based surgery simulation combines several modern technologies together, which include computer science, medical science, robotics, biomechanics and virtual reality. A successful TriS should satisfy at least two essential requirements:

- **Realism:** Users should feel as if they were operating on the real patient directly when they are using a TriS. Realism includes two aspects: visual realism and haptic force feedback realism. Visual realism mainly means realistic deformation, that is, the virtual organs or tissues exhibit physically correct behaviors corresponding to the behaviors of real human organs or tissues. Haptic force feedback realism means that when the interaction (between surgical tool and organ) happens, the force which the user feels is faithful.

- **Real-time:** It means that any action from the operator should generate an

instantaneous response from the simulated organ, whatever the complexity of its geometry is. This requires the visual and force feedbacks can be reproduced at correct frequencies. To support the real time visual feedback, the deformable model should be updated in less than 33ms, while real time force feedback requires the forces to be computed with a frequency up to 1KHz.

To achieve these two goals, there are three research focuses:

- Developing a realistic soft tissue model
- Developing a realistic haptic model
- Implementing realistic simulations for some main surgical tasks such as cutting and suturing in real time

Although many research works have been done in these fields, there is a long way to go until the TriS can be used in surgeons' daily training and replace traditional methods. On the other hand, due to its great potential and advantages, more and more people are joining this research frontier to make the replacement happen as soon as possible.

1.2 Thesis Objective

Developing realistic soft tissue models is the fundamental and first step in surgery simulation research. Many of the recent efforts focus on this issue. In addition, how to realistically simulate some major surgical tasks such as cutting and suturing is another key concern. These surgical tasks are the most important procedures of all surgeries and can not be avoided. To a large degree, a TriS actually implements the realistic and real-time simulation of these tasks.

The objective of this thesis is to design and develop a TriS, which can realistically simulate soft tissue palpating and cutting in real time. To achieve such an objective, the following tasks need to be addressed:

- To implement a real-time and physically based tissue model.
- To develope and implement a volumetric soft tissue cutting scheme which can simulate cutting with high realism in real time.
- To study the haptic rendering methods, and to improve the fidelity of force feedback during simulation.
- To design a data structure which can efficiently handle real-time soft tissue simulation, topology modification and cutting simulation.

1.3 Thesis Contributions

The contributions of this thesis are:

1. A comprehensive analysis and comparison of the existing physical deformable models which can be used for soft tissue modeling in surgery simulation was presented.
2. A new force distribution method was proposed. It re-distributes the force exerted on any arbitrary point inside a surface triangle of the model to the triangle's three nodes.
3. A new volumetric finite element cutting scheme was proposed and implemented.
    - A new combined subdivision/separation method was developed to address the topology modification of the soft tissue model. This method decreases

the number of newly created elements and can handle the situation of cutting through the mesh nodes.

- A midpoint subdivision method was applied to the minimal element sets. This method makes the element subdivision easier and more efficient. In addition, it ensures that there are no very small or badly shaped elements generated such that the stability of the simulation system gets increased.
- A localized intersection detection method was proposed and used in the simulation. It can detect all intersected elements with high efficiency.
- The state and lookup tables were used in the cutting algorithm, which facilitate the topology modification and ensure the mesh be consistent.

4. A physically based haptic rendering method was used to send the faithful feedback force to the user. The force computed from the physical simulation was extrapolated over the known tool positions to achieve the 1KHz force updating rate.

5. An efficient data structure was designed and used to handle the complicated tetrahedral mesh data both geometrically and physically.

1.4 Thesis Outline

The thesis is organized as follows:

Chapter 2 briefly introduces the TriS, its classification and generic architecture. A comprehensive review of the previous research work and results, which includes soft tissue modeling, haptic rendering and cutting simulation methods, is presented. All soft tissue models are divided into two types: point-mechanics based and continuum-mechanics based models. Different models such as Mass-Spring Model(MSM), 3D

ChainMail Model, Finite Element Model (FEM), Boundary Element Model (BEM) and Finite Sphere Model (FSM) are discussed and compared. For haptic rendering, haptic interface, the problem of different frequencies and force feedback computing method are described. Different cutting methods and cutting researches on both surface models and volumetric models are presented.

Chapter 3 is the system overview and introduction to the TriS developed in this thesis. The system framework and system diagram are given and explained.

Chapter 4 first explains why the tensor-mass model is selected and then the mathematical description of the model is presented. A new force distribution method is introduced which re-distributes the force exerted on an arbitrary point on a surface triangle of the model to its three nodes rather than simply applies the force to the nearest node. The Euler and Fourth-order Runge-Kutta numerical integration methods are discussed, and their computational accuracy, efficiency and stability are compared. Then the Fourth-order Runge-Kutta method is chosen as the integrator of the soft tissue model.

Chapter 5 focuses on cutting simulation. It begins by introducing three exist cutting methods. Their advantages and disadvantages are discussed. Then a new combined subdivision/separation cutting scheme is developed. Based on the node snapping results, element splitting is implemented by mid-point minimal set element subdividing or element separating. The subdivisions of eight topologically different cases are introduced. A volumetric intersection detection method is presented. The local search by intersection propagation of this method makes the detection very efficient. During cutting, the state table is used to record the state of the intersected elements, which make element split more efficient. And the lookup table is used to keep the subdividing information and ensure the consistent of the mesh.

Chapter 6 begins by investigating three haptic rendering methods. Then a separated external/feedback force haptic rendering approach is proposed. The external forces are computed by the point-based proxy method. In order to be consistent with the explicit finite element soft tissue model used by this thesis, the physically based method is used to compute the feedback forces. The forces obtained from physical simulation loop are extrapolated over tool positions and then are sent back to the user.

Chapter 7 presents some implementation details and the experimental results. The hardware and software simulation environments are described. The tetrahedron mesh generating software and the construction of the volumetric soft tissue model are introduced. How the model parameters are chosen is also explained. The data structure of the simulation system is described. Then after the introduction of some cutting implementation details, the experimental results of palpation and cutting are presented.

Finally, chapter 8 summarizes the thesis and discusses possible future work.

Chapter 2

LITERATURE REVIEW

Although the history of surgery simulation is not very long, many research projects have been done in this area and a large amount of achievements have been published. In this chapter, a brief description of the evolution and the current state of the TriS will be given; then discussions and comparisons of previous work on soft tissue modeling, haptic rendering and cutting simulation will be presented.

2.1 TriS Overview

2.1.1 TriS Classification

According to its components and functionality, TriS can be classified into the following three categories [1].

First Generation Simulator

Organs and soft tissues are represented only by geometric models. In these simulators, the user can virtually navigate through the human body but has very limited

interactions with the modeled organs.

Second Generation Simulator

Human organs are represented by physical models, which not only include the geometric information but also integrate the biomechanics property of tissues. In addition, a haptic interface is introduced into the simulator. Users can interact with the virtual environment, which means realistic interactions between surgical tools and soft tissues, such as palpating, cutting and suturing, are simulated. There are no second-generation simulators commercially available now. Most of current researches fall into this category.

Third Generation Simulator

Organ and soft tissue models provide an anatomical, physical and physiological combined description of the human body. Besides simulating all kinds of surgery procedures, the third generation simulators even can simulate the function of some systems such as the respiratory and the digestive systems. Because it is very difficult to realistically describe the coupling between physiological and physical properties of the soft tissue, currently there are very few simulators in this category.

2.1.2 Generic Architecture of TriS

In this thesis, we focus on the second generation simulator. A TriS is based on the virtual reality (VR) environment. Users interact with this VR environment through input and output interfaces. Traditionally users can only interact with VR applications through vision and hearing. With the advance of haptics and medical robotics, touching is added into TriS. The typical input interface consists of haptic device,

mouse and keyboard. Users manipulate the virtual organs with the haptic device and interact with the VR application by mouse and keyboard, while the visual and force feedbacks are sent back to users through computer monitor and haptic device, respectively. Figure 2.1 shows the architecture of the second generation surgery simulator [2].

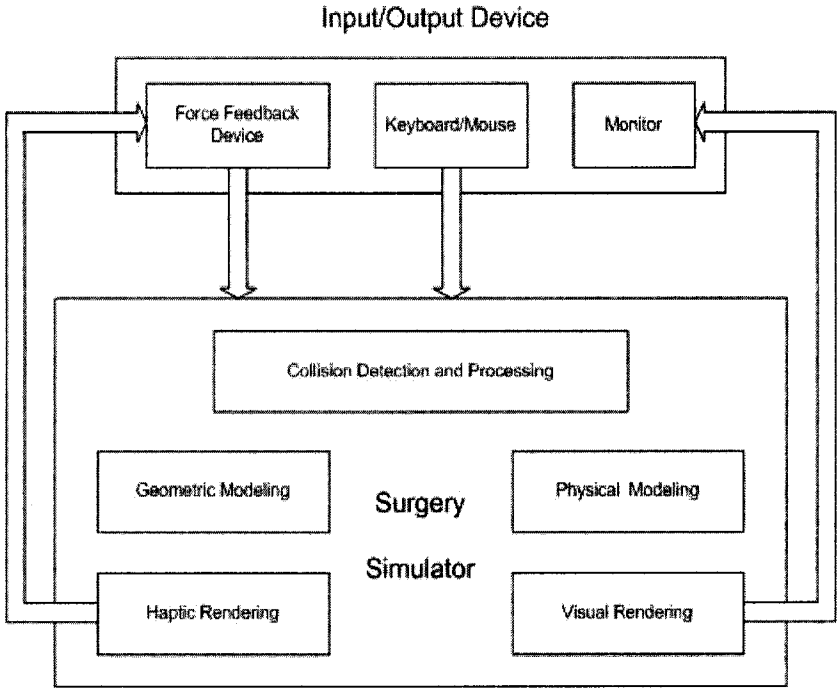


Figure 2.1: Architecture of the second generation surgery simulator (source[2])

Recently three frameworks for TriS have been proposed. The first one is the SPRING [3]. It is a general framework to create MSM based surgical simulators which can perform soft-tissue modeling, some limited rigid-body dynamics and suturing modeling. The second one is the GiPSi [4], which is a general modeling and simulation framework for developing organ level surgical simulators. GiPSi includes some modules such as modeling tool module, computational tool module, input/output

module and visualization module, and provides general APIs for these modules. The third one is the SOFA project [5]. It is an extendible, open source framework used for easy exchange of algorithmic blocks between different research groups. All these three frameworks follow the generic architecture of the second generation surgery simulator shown in figure 2.1.

2.2 Soft Tissue Modeling

Soft tissue modeling is to represent the bio-mechanical properties of living tissues by deformable models and to visualize them to the user. The quality of a soft tissue model affects directly the realism and real-time performance of the surgery simulator. It determines whether the deformation of the tissue is realistic, whether the force feedback is faithful, and whether both of them are real-time. Therefore, a good soft tissue model must satisfy the following requirements [6].

- Displays smooth deformations
- Reflects faithful and stable force to users
- Shows physically based behavior in real time
- Handles various boundary conditions and constraints

Unfortunately, human tissue is very complex and often behaves viscoelastically, anisotropily and nonlinearly. In addition, human body consists of layers of different tissues interlaced with ligaments and fascias. Thus, very complex models are needed to model these objects realistically. In addition while the complexity of the model increases the computational intensity significantly increases too, which in turn causes the real

time requirement very hard to achieve. To meet both the accuracy and speed requirements, many soft tissue models have been developed and studied. We'll discuss and compare some of them from two aspects: geometric modeling and physical modeling.

2.2.1 Geometric and Physical Modeling

A deformable model used in the second generation surgical simulation systems includes two components: geometric modeling and physical modeling. Geometric modeling provides geometric structure information of the tissue or organ. Usually it is constructed by using medical image data, such as Magnetic Resonance Imaging (MRI) or Computer Tomography (CT) images. Physical modeling adds bio-mechanical properties of soft organ/tissue to geometric model, so that the deformable model can behave like a real organ or tissue.

Geometric Modeling

According to geometric modeling method, soft tissue models can be divided into two categories: surface model and volumetric model.

- **Surface model:** Surface models represent the exterior or surface of the object by a set of polygons, typically triangles or quadrilaterals. In the early stage of surgical simulation, surface models were used by many researchers because of their simplicity and computational efficiency. Terzopoulos et al. [7] first introduced the physically-based deformable object simulation by using a surface model. Kühnapfel et al. [8] and Keeve et al. [9] also used surface models in their craniofacial surgery simulation. A surface model is a hollow shell and doesn't define the internal of the organ. Therefore surface models are only good representations for cavernous tissues such as vessels and the gallbladder

[10]. Their main disadvantage is that they cannot simulate the volumetric behavior of human organs, such as pushing on one side of an organ leads to the movement of the other side. Moreover, they cannot simulate operations that change the inner topology of the object, such as cutting. Although some simple cuts are simulated with surface models, they are impossible to display the general surgical incisions. With the quickly developed computer hardware and increased computing power, the computational bottleneck of volumetric models is gradually, although not completely, vanishing. In recent years, more and more volumetric models are adopted.

- **Volumetric model:** The entire object of the volumetric model is discretized into many volume elements such as tetrahedron, hexahedron and prism. A volumetric model can hold detailed data for the internal anatomical structure of the organs and simulate the mechanical properties of heterogeneous tissues. It is the natural way to represent the human tissues and organs, and it makes cutting and tearing simulation more realistic. Compared with surface models, the data structures of volumetric models are much more complicated. Different volumetric models have been used in many surgery simulations [11] [12] [13] [14] [15]. The main drawback of volumetric models is that they require high computational power and large storage space to handle such huge amount of data. The intensive computation will affect the rendering speed which eventually will have practical implications on the real-time requirement of TriSs.

Physical Modeling

Physical modeling is based on the following dynamic equation of the deformable object:

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{K}\mathbf{U} = \mathbf{F} \tag{2.1}$$

where the dots indicate temporal derivatives, **M** is the mass matrix of the object, **C** and **K** are the associated damping and stiffness matrix, respectively. **U** is the coordinate displacement vector of the points describing the object, and **F** is the vector of total external forces applied to the object.

By solving the above equation, the force and displacement (deformation) can be obtained. According to different ways the equation be solved, deformable models can be divided as point-mechanics based models and continuum-mechanics based models

2.2.2 Point-mechanics Based Model

Point-mechanics based models solve equation 2.1 by using point mechanical theory. With this theory, the object mass is considered to be concentrated on discrete points and linked to each other with connections which represent the internal forces arising from the deformation of the object. Typical examples in this category are mass-spring model and 3D chainmail model.

Mass-Spring Model (MSM)

A mass-spring model consists of a number of nodes connected by springs, and the mass of the modeled object is concentrated on the nodes. Each node N is represented by its own position, velocity and acceleration. Node N moves under the influence of forces exerted on N by the nodes to which N is connected. The spring-mass mesh

represents the tissue geometry and is used to discretize the equation of motion. For a node i, the equation of motion is

$$m_i\ddot{U}_i + d_i\dot{U}_i + k_iU_i = F_i \tag{2.2}$$

where m_i is the mass of the node, d_i is the damping factor, k_i is the stiffness factor and F_i is the external force, U_i is the displacement vector of node i.

- Surface MSM

The simplest mass-spring model is the surface model, which represents the surface of the object by a node-spring polygons mesh. The typical discretization ways are using triangular patches (See Figure 2.2) or $T_i - mesh$. In $T_i - mesh$, each node

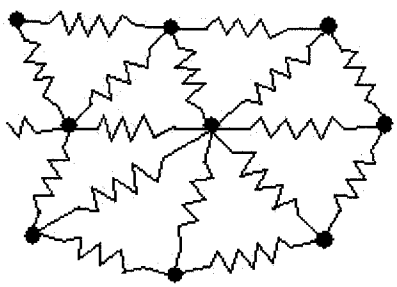


Figure 2.2: Triangular mass-spring mesh

is connected with a constant number of (i+1) adjacent nodes. Following figure is a $T_2 - mesh$, in which each node is connected with 3 adjacent nodes.

MSM is easy to program and demonstrates reasonably fast simulation speed due to its simple physical model. It is well studied and widely used for surgical simulation, especially in the early stage of the research. The real-time soft tissue model for surgery simulation was first presented by Cover et al. [16], where a simple

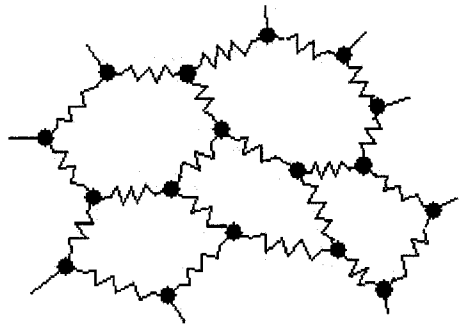


Figure 2.3: T_2-mesh

surface-based MSM was used to simulate deformation of a gallbladder. Water [17] defined a $T_3 - mesh$ (regular spring lattices) for modeling facial tissue. He also used a Marching Cubes algorithm to generate both the inner and outer surfaces of bone tissue. In a craniofacial surgery simulation, Delingette et al. [18] represented the skull, skin and muscles by using a three-simplex spring mesh model. The parameters of the springs between nodes are constant.

In order to get realistic force feedback, Zhang et al. [19] used a surface MSM to increase the computation speed, while representing the effect of the object interior with "home springs". In their model, the simulated surface is divided into small triangles and a linear spring is mounted along each triangle edge. These springs are called "mesh springs". The "home spring" connected each node to its initial position. When the soft elastic object deforms, it reflects both the shape of deformation and the force feedback which is contributed by the interior of the object. Thus, although it is a surface model, it can partially simulate the properties of volumetric objects.

Despite its easy implementation and high computational efficiency, a surface MSM is not appropriate if the interior of the object must be simulated, such as in a cutting procedure. In this situation, volumetric models must be used.

- Volumetric MSM

Typically, a volumetric MSM is represented by a tetrahedron spring mesh. Same as the surface model, object mass is concentrated on the nodes of tetrahedron, the edges of the tetrahedron is represented by the springs. The mesh complexity of the volumetric model is dramatically increased compared with the surface model. The volumetric MSM is mainly used to simulate the inner topological changing procedures such as cutting. To achieve real-time performance, some algorithms and methods are proposed to increase the computational efficiency.

Mollemans et al. [14] proposed a way to directly calculate the deformation of the tetrahedral MSM. In their model, all nodes are divided into two types: joint nodes and free nodes. Joint nodes are connected to the bone structure and are fixed. Free nodes are able to move in the space in response to the exerted force. To achieve fast calculation, a localized deformation method is employed. This method assumes the force of a node i at the start of iteration j can only have been changed when node i or a neighboring node has been moved during iteration j-1. There are two situations a node can be included in the iteration calculation: first, the resulting force of the node is larger than one tenth of maximum force; second, only nodes which are neighbors of a node that is moved in current iteration will be evaluated in the next iteration. The authors demonstrated deformation accuracy of their model by comparing with a pre-computed finite element model.

Brown et al. [20] also implemented some algorithms to speed up the calculation for volumetric MSMs. Under the assumption that the velocity of the nodes, which are subject to external forces, is small enough, the mesh achieves static equilibrium at each instance. Therefore they used a Quasi-static algorithm which neglects dynamic

inertial and damping forces. Then the motion of equation becomes:

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

Compared with equation 2.1, equation 2.3 is easy to solve. However, the computational cost is still high. To further decrease the computational intensity, a Node-ordering algorithm is proposed by the same authors. The outcome of this algorithm is a list of ordered nodes. In the list, the displaced nodes (exerted by external force) have the highest level; for other nodes, the closer to the displaced node, the higher level they have. Using this ordering result, in each calculating iteration, the system will stop propagating the deformation further automatically at a level where the displacements of the nodes are less than a small pre-specified amount. Using a $20 \times 20 \times 20$ box spring mesh (8000 nodes and 66,120 links) and a Sun Ultra 60 workstation with 1GB RAM and one 450 MHz processor, the simulation can maintain a 30Hz update rate if these two algorithms are adopted.

Except for tetrahedral MSM, the linked volume model is another volumetric extension of surface MSM. It discretizes the volume of an object into evenly spaced, cubic elements. The corresponding masses are lumped at the respective centers of these elements and interconnected with their neighbors by springs and dampers [12]. With the volumetric character, this model can simulate the complex interactions like cutting, suturing and tearing. However, the propagation of deformation is slow and real time performance is hard to achieve. In order to increase the propagation speed of deformation, the 3D ChainMail algorithm was introduced by Gibson [21].

3D ChainMail Model

This approach uses the same volumetric discretization as a linked volume model, but the elements are interconnected as links of a chain instead of links of springs. Each element is connected to six of its neighbors: top, bottom, right, left, front and back. Within a certain limit, each element can move freely without influencing its neighbors. When the object is manipulated, each element is tested to see if it violates the certain distance thresholds it shares with its neighbors. The element will stay in its original position if the threshold has not been exceeded, otherwise, it is moved in the desired direction and the deformation is propagated to its neighbors. Local deformations are generated only if distance thresholds have been exceeded.

The major advantage of this approach is that the deformation results can be obtained by applying simple calculations on elements. ChainMail method is relatively easy to be implemented compared with other volumetric MSM approaches. A knee arthroscopy simulator was developed by using this method [13].

In general, the point-mechanics based model is not considered as a good approximation of the soft tissue because the energy and mass in the model are simply concentrated into mass points and link segments, which does not agree with the natural character of the soft tissue. With this kind of models, real-time deformation of virtual anatomy can be roughly approximated, but it is impossible to accurately simulate the complex deformation and force-feedback interactions that can take place during surgery. Another major drawback is that it is difficult to extract parameters for the hundreds and thousands of individual springs, masses and dampers from experiments. In addition, how to construct an optimum three-dimensional network of springs, masses and dampers is also a difficult problem.

2.2.3 Continuum-mechanics Based Model

This kind of model is based on the law of continuum mechanics. The continuous system of mass and energy is used to model the soft tissue naturally. From this approach, more precise deformation and force feedback can be derived because equation 2.1 is obtained through the integration carried out over continuous spatial sub-domains or elements. The two well-known approaches are finite element model and boundary element model. Recently, the mesh free method was introduced into surgery simulation, and the corresponding model which adopts this method is Finite Sphere Model (FSM).

Finite Element Model

In a finite element model, the entire modeled object is partitioned into a finite number of subdomains or elements. In most cases, tetrahedron elements are used. The equations of motion are expressed on each of these elements. An implicit (traditional) finite element method merges all these equations into a large matrix system as shown in equation 2.1. Then this large matrix system is solved to get the desired deformation. Usually no analytical solution exists so that equation 2.1 must be solved by using numerical techniques. The dimensions of matrices $\mathbf{M}$, $\mathbf{D}$ and $\mathbf{K}$ are $3n \times 3n$, where n is the number of nodes in the mesh. For a volumetric mesh, n usually is quite large. To solve such a complicated matrix system with a conventional finite element method, the computation is too intensive to achieve real time performance. In order to employ finite element method in surgical simulation, which requires real-time performance with visualization update rate 30Hz and haptic update rate 1KHz, many simplified approaches over traditional finite element method are proposed and studied to decrease the computational cost. Four simplification approaches are described in

the following:

- Static FEM

The first simplification approach assumes that the model is static and doesn't depend on time. Under this assumption, equation 2.1 becomes equation 2.3. Although now it is relatively easy to solve the system, the computation is still expensive and a real-time application is still not feasible when the mesh is large if we solve this equation directly.

- Condensed FEM

The second approach is condensation [22] which was first used by Bro-Nielsen [23] in surgical simulation. During simulation, if no incision happened, we are usually only interested in the behavior of visible nodes i.e. surface nodes. The computation for the inner nodes is therefore not really necessary and can be removed by this condensation technique.

The condensation method only rearranges terms in the linear system (equation 2.3), but doesn't discard any information. The resulting matrix equation has the same size as a surface finite element model but still has the volumetric behavior. Wu *et al.* [24] used a hybrid condensed finite element model, which includes a static non-operational region and a dynamic operational region, to implement cutting and deformation. They also adopted the preprocessing method with the non-operational region. The simulation achieved an interactive rate on a current PC platform.

- Pre-computed FEM

The third approach is the pre-computed finite element model. One of the pre-computation methods is explicitly inverting the stiffness matrix $\mathbf{K}$, so that the defor-

mation of the system can be got by following matrix vector multiplication:

$$\mathbf{U} = \mathbf{K}^{-1}\mathbf{F} \tag{2.4}$$

Stiffness matrix $\mathbf{K}$ only depends on the rest shape geometry and the nature of the material. If the rest shape geometry of the deformable object keeps the same, $\mathbf{K}$ will be a constant matrix and the inverse of $\mathbf{K}$ can be computed before the simulation start. Thus this method can greatly decrease the calculation time during simulation. Bro-Nielsen [25] is the first person who proposed to use the finite element model in real-time interactive deformation simulation. He and S. Cotin implemented a real-time deformation simulation on a tetrahedral finite element leg model [26] [23]. They tested this method and demonstrated the following result: when the pre-calculation time was ignored, solution by this matrix vector multiplication was at least 10 times faster than other methods.

Another pre-computation method is computing a set of elementary deformation of the model before simulation starts. Cotin used this method in his linear elastic model [27]. Under the linear elasticity assumption, any mesh deformation can be computed from a finite set of known elementary deformations. Thus, the interactive deformation can be obtained in two steps. First, a set of elementary deformations are pre-computed, and then, the real time deformation can be computed as a linear combination of the elementary deformations. An iterative method is used to solve each linear system. With this method, the force update rate of 1KHz can be achieved even with a big mesh.

The drawback of the pre-computation approach is that it is only suitable for the situation without mesh topology change. Because stiffness matrix $\mathbf{K}$ and the elementary deformations need dynamically update under topological change, they

need to be calculated at every iteration which makes real-time interaction impossible.

- Explicit FEM (Tensor-mass Model)

The fourth approach is explicit finite element model, also called tensor-mass model (TMM). It discretizes a 3D deformable object into a tetrahedron mesh. However, instead of deriving a complex description of global interrelationships as obtained with a traditional finite element method, tensor-mass model handles each tetrahedron individually and then uses the same method as a MSM to iteratively solving the motion equation 2.1.

TMM was first introduced by Cotin et al. [27]. They reached an update frequency of 40Hz with a tensor-mass model of 760 vertices, and demonstrated that this model has the same computational complexity as a MSM. Then they constructed a hybrid volumetric liver model, which is composed of 1537 vertices and 7039 tetrahedra. About 18 percent of the liver mesh (280 vertices and 1260 tetrahedra) is modeled as a TMM, and the remaining is a precomputed linear elastic model. With this hybrid model, they achieved real-time deformation and cutting simulation. Mor [28] adopted the linearly elastic TMM in his cutting simulation. Mendoza et al. [29] used a non-linear strain tensor formulation to allow large displacements. Picinbono et al. [30] also employed a non-linear tensor-mass model in their real-time simulation of laparoscopic surgical gestures on the liver. Their model is valid for rotations and large deformations. By adding the incompressibility constraint, the model can address the problem of anisotropic behavior and volume variation.

Boundary Element Model

Boundary element model (BEM) is based on boundary element method [31] and was first introduced by Monserrat et al. [32]. Not like finite element model, in which

the domain of the modeled object is divided into volumetric tetrahedron elements, a boundary element model doesn't use volumetric tetrahedron mesh. It only uses the surface of the object. Using the Green solution of the linear elastic problem and imposing the boundary conditions in a general way, the system equation becomes:

$$\mathbf{H} \cdot \mathbf{u} = \mathbf{G} \cdot \mathbf{p} \tag{2.5}$$

where $\mathbf{H}$ and $\mathbf{G}$ are the fully populated influence matrices, $\mathbf{u}$ and $\mathbf{p}$ are the node displacement vectors and traction (i.e. surface force) vectors, respectively. In a 3D case, $\mathbf{H}$ and $\mathbf{G}$ have $3n_b \times 3n_b$ elements, where n_b is the number of nodes.

By explicitly inverting $\mathbf{H}$, the deformations of the nodes can be computed as follow:

$$\mathbf{u} = \mathbf{H^{-1}} \cdot \mathbf{G} \cdot \mathbf{p} \tag{2.6}$$

Monserrat et al. [33] tested this model with a pig liver, and it yielded quasi exact results for displacements of up to 3mm and deformation speeds between 0.04 and 0.4 mm/s. Wang et al. [34] recently employed this model in their simulation of neurosurgery. They implemented the prodding, pinching and cutting procedure on a simple triangular mesh. The simulation could provide both visual and haptic feedback in real time.

Boundary element model is well suited for the simulation of linear elastic isotropic and homogeneous materials. When the mesh topology is not modified, it is a good alternative to finite element model. But comparing to FEM, BEM has several disadvantages. First, only homogeneous and isotropic linear elastic materials can be modeled. Second, the matrices $\mathbf{H}$ and $\mathbf{G}$ are fully populated and cannot be simplified. Third, this model cannot get the displacement of any interior points. Therefore,

BEM is much less popular than FEM.

Method of Finite Sphere

The physical modeling of both FEM and BEM divide the object into element mesh, then construct system motion equations based on the mesh and numerically solve those motion equations in order to get the deformation and force feedback. The contact between tool and tissue must occur at the mesh vertices, which requires the mesh resolution be sufficiently high to ensure the calculation accuracy. This in turn increases the computational cost. Another shortcoming of FEM and BEM is that a constant expensive remeshing operation cannot be avoid when the mesh topology changes. To overcome these drawbacks, De and Bathe [35] proposed a mesh-free technique known as the method of finite sphere (MFS).

MFS discretizes the computational domain into a scattered set of points. The Galerkin formulation is used to generate the partial differential equations (system motion equations). Then the displacement is approximated using some shape functions that are nonzero over small spherical neighborhoods of the nodes. During real-time simulation, once the tool-tissue collision point is detected, MFS nodes are sprinkled in the space, including both the surface of the model and interior around the tool tip. Then the tissue deformation is computed. Kim et al. [36] proposed a prediction method to decrease the collision detection time in order to improve the computational efficiency. A stomach palpation simulator was implemented on Windows NT and the dual Pentium 900MHz processors. The geometric model of the stomach has 20,000 triangles. The simulator achieved 1KHz force update frequency and 30Hz visual frame update rate.

To meet the requirement of real time performance, a specialized version of MFS,

i.e. the point collocation-based method of finite sphere (PCMFS) was developed by Suvranu De group [37]. They presented a minimally invasive surgery simulation system, which uses this PCMFS method to simulate the tool-tissue interaction in real time [38].

Lim et al. [39] coupled a geometrically-based cutting algorithm with PCMFS to implement the deformation and cutting simulation. They proposed a localized PCMFS to speed up the computation for large organ models.

During simulation, although the mesh-free finite sphere method can avoid the expensive remeshing when doing the physical computation to get the deformation and force numerically. It cannot avoid remeshing in geometric modeling because we need render the mesh changes to the user visually if the topologic change happened. Further more, how to locate the MFS nodes and choose the proper radii of the spheres is not a trivial problem. The research on this method is not as popular as finite element method.

In summary, continuum-mechanics based models are more like the nature of soft tissue. They are more accurate than point-mechanics based models. In addition, only a few material parameters are required to do the modeling, and those parameters are coincident with characteristics of real organ tissues. Thus, they can be obtained rather conveniently from experiments. The computation problem of continuum-mechanics based models is becoming less and less important with the advance of computer technology. In the long run, continuum-mechanics based models are expected to be able to accurately simulate the deformation of human tissue in surgery simulation even without simplifications.

2.3 Haptic Rendering

Haptics broadly means the touch interactions that occur for the purpose of perception or manipulation of objects [40]. Rendering refers to the process by which desired sensory stimuli, such as vision and haptics, are imposed on the user to convey information about a virtual object. The goal of haptic rendering is to enable the users to touch, feel and manipulate virtual objects through a haptic interface. A haptic rendering subsystem of the TriS has two main components [41]:

- Haptic interface: an electro-mechanical system which is able to exert controllable force on the user
- Haptic rendering algorithm: which can join the haptic interface and the deformable object model together to compute the model-based force and feed it back to the user

2.3.1 Haptic Interface

A good surgery simulator should provide surgeons the same sensations as in real surgery. This is why the haptic interface is introduced into surgery simulation and plays an important role in the system. Haptic interface greatly enhances the realism of the surgery simulation.

Haptic interface is the input and output device between a user and a surgery simulator. It accomplishes two tasks:

- provides motion and interaction commands to the simulation system
- provides force feedback about the virtual environment to the user

In every degree of freedom, the haptic device can detect either the position (or velocity) of the tool or the force on the user's hand. According to this, there are two different types of haptic interface:

- Impedance/force feedback control: the positions or velocities imposed on the haptic device by the user are measured and the reaction forces are returned
- Admittance/position-feedback control: the forces applied by the user on the tool are sensed and the positions are fed back by the haptic device

Most of current surgery simulation systems use force-feedback control.

2.3.2 Haptic Rendering Algorithm

Typically, a haptic rendering algorithm is composed of two parts: collision detection and collision response. Based on the way the probing object is modeled, haptic rendering algorithm can be classified into three categories:

- Point-based haptic rendering
- Ray-based haptic rendering
- Object-based haptic rendering

The simulation of haptic interaction between two 3D objects is desired for many applications, but it is very complex and computationally expensive. To author's best knowledge, object-based haptic rendering haven't been used in the TriS yet since it cannot achieve real-time interaction. Therefore we will not discuss it in this thesis.

In point-based haptic rendering, the virtual representation of the haptic interface is symbolized as a single point which can probe the virtual objects. The typical methods are the "god object"[41] and the "virtual proxy"[42]. In ray-based haptic

rendering, the stylus of haptic device is modeled as a line-segment whose orientation is taken into account. Therefore the collision detection becomes much more complicated than point-based haptic rendering. Since it needs to detect the collisions of both the point and line with a 3D object, thus multiple contacts must be detected at a time. In addition, except for contact force, torque needs to be computed and rendered too in this method.

Most of current TriSs use point-based haptic rendering method because it is easy to achieve 1000Hz haptic update rate comparing to the other two methods. Zhang [43] used a virtual proxy similar to that of Ruspini [42] with modifications in the motion of the proxy, which is not guided by a local minimization method but by the physics of the system simulated by dynamic lumped finite elements. Kühnapfel [44] also used the point-based method to render the force feedback to the user in his endoscopic surgery training system.

As for the ray-based method, due to its complexity and computational intensity, it is not as popular as point-based methods. Basdogan et al. [45] used both methods in their laparoscopic surgery training system. The interaction forces between the laparoscopic forceps and the catheter are rendered by a ray-based method, while the interaction forces between the catheter and the bile duct are rendered by a point-based method.

2.3.3 Handling Different Update Frequencies

The forces in haptic systems have to be obtained at a frequency of 1KHz to give a realistic sensation of touch. However, the graphic display frames of a TriS are updated at a rate of 24-60Hz. The difference of the two update rates can cause an oscillatory behavior in the haptic device. Consequently the oscillated device may become highly

unstable and inflict bodily harm to the user.

To address this problem, many researches have been done. Mark et al. [46] proposed to make the haptic device interact with an intermediate model, which is updated at the graphical rendering rate, rather than the complete physical model. This method only works well when convexity objects are used. Unfortunately, almost all of the soft tissues are concave objects. Ellis et al. [47] used prediction and correction to obtain a force feedback between the model updates. Adams and Hannaford [48] proposed to separate the haptic interface from the virtual environment simulation to ensure stability of the haptic interface. Based on the work of Adams and Hannaford [48], Mendoza and Laugier [49] coupled the graphic and haptic loops using the local topology of the deformable object, so that the haptic interaction between the haptic device and the local topology of the model can achieve an update frequency of about 1KHz. Wang et al.[50] also used a local model-based haptic rendering in their dental preparation surgery simulation.

2.4 Cutting

Cutting is a common and critical procedure in all kinds of surgeries, especially in open surgeries. But real time soft tissue cutting simulation is still largely unexplored. No matter what type the soft tissue model is, cutting changes the topology of underlying mesh and re-meshing is necessary. The expensive re-meshing procedure makes the computation of the simulation much more intensive. This imposes a great technical challenge on real time cutting simulation.

An ideal cutting simulation should meet the following requirements:

- accurately represents and tracks arbitrary cut path

- the user can see the cutting result without delay

Both of these two requirements are very hard to achieve. In order to accurately track the arbitrary cut path, the mesh should be split following the cut path. This will inevitably create some very small and badly shaped new elements. Those elements will lead to the unstable simulation, which is definitely not allowed. To see the cutting result without delay, progressive cutting must be employed. The temporary topology modification needed by progressive cutting will slow down the computation and make this requirement harder to achieve.

According to the different ways the cutting implemented, there are three types of cutting approaches: deleting, separating and subdividing elements. Element deletion is the simplest way; it avoids expensive topology modification but leaves a big gap as the cutting incision. Furthermore, it violates the mass conservation theory. Element separation will not create new elements with the sacrifice of losing cutting realism. Element subdivision has the highest realism. Unfortunately it must deal with expensive computation and badly shaped new elements.

These three methods are used in cutting both surface and volumetric models. But cutting volumetric meshes is very different from cutting surface meshes. The intersection detection and topology re-meshing of a volumetric model are much more compilcated and difficult than those of a surface model. We will review the related work with both surface and volumetric cutting as follows.

2.4.1 Surface Model Cutting

Cutting was first applied to surface meshes because the volumetric cutting is too complicated. Song and Reddy [51] first proposed an interactive 2D finite-element template for cutting the elastic virtual surface. The nodes move as the cutting tool moves over

a flat 2D tissue surface. Later Reinig et al. [52] suggested a tissue cutting technique implemented on a 3D surface. They incrementally updated the vertices, connectivity arrays and texture maps to reveal the cut and provide a visually realistic display. But they didn't address the haptic rendering problem. Mendoza et al.[53] implemented cutting on a 2D mass-spring surface model by using the element separation method.

Basdogan et al. [54] introduced a non-progressive tissue cutting algorithm on FEM models. It maps a 3D problem into a 2D auxiliary surface to simplify the FEM calculation. Nienhuys et al. [55] also implemented the cutting on a triangulated surface mesh. They suggested using delaunary triangulation to reduce the mesh size and enhance the element quality. The delaunary flip method includes two steps: first, triangles with an angle approaching 180 degree are flip away; second, the close nodes are connected together.

The main drawback of the cutting simulations based on surface models is that they cannot represent the interior of the cut. Zhang et al. [56] proposed a novel method to address this problem. They implemented the simulation of progressive cutting on a MSM which contains mesh springs and home springs. A method was introduced to generate the interior surface, referred to as a "groove", in the opening. The groove is a function of the instrument penetration depth. They used this method with a liver model initially composed of 2466 triangles and achieved a update rate of 330Hz for haptic rendering.

2.4.2 Volumetric Model Cutting

Because cutting procedure should display the interior topology of the mesh, more cutting researches have been done on volumetric models than on surface models.

The earliest volumetric cutting with element deletion is proposed by Bro-Nielsen [23].

The cutting was performed by simply removing the intersected elements of the FE mesh. Then Cotin et al. [27] used this deletion approach on a tensor-mass model to achieve real time cutting simulation. Forest et al. [57] also implemented cutting in a real-time surgical simulator with a tetrahedra refining and removing combined method. The cutting proceeded in two steps: first the neighboring tetrahedra were refined; then a subset of these tetrahedra were removed. The local mesh refining decreases the gap and increases the cutting realism.

After implemented cutting on a 2D surface mass-spring mesh using the separation method, Mendoza extended the same method to an explicit FEM [29]. The force feedback was computed with a buffer model in the latter simulation.

The volumetric cutting simulation using element subdivision method was first introduced by Bielser et al. [58]. There are five element intersect cases while cutting through a tetrahedral mesh. The authors proposed a generic 1:17 tetrahedron split method. That means the cutting procedure always subdivides an element into 17 smaller elements. The intersected edge or face is split at the intersection, while all uncut edges and faces are split at their midpoints. It leads to a single generic subdivision for all five cases. This method can be applied to arbitrary irregular tetrahedron meshes, and simulation has high level of accuracy and topological freedom. But the universal subdivision scheme results in a rapidly increased number of tetrhedra for large cuts, this will greatly increase the computation cost and make the real time simulation become hard to achieve. In addition, it is possible to generate hanging nodes which have no connections to adjacent tetrahedra, and lead to cracks in the mesh.

Bielser and Gross [59] later extended their work by reducing the number of new elements generated for each intersected element. Instead of inserting new vertices on

uncut edges and faces, they only inserted vertices at intersection locations, plus new vertices on faces which will be split. This reduces the number of new elements, but does not minimize it. Mor [28] extended above work by demonstrating a system that generates a minimal set of new elements during cutting, with progressive updates of the cut elements while the cut is on going. With this minimal set subdivision, one vertex at the point of each face intersection and two vertices at the point of each edge intersection are generated, thus only five to nine new elements are created for each intersected element. In order to accurately present arbitrary cuts, Bielser improved his work by using a state machine to track the topology of each tetrahedron and control the progressive subdivision [60].

Ganovelli et al. [61] proposed a method to reduce the computational cost of Bielser's cutting algorithm. They used a tetrahedral mass-spring model, and coupled multiresolution and topological modification together. They suggested adopting a multiresolution mesh model and dynamically construct a optimized Level of Detail (LOD) mesh. The resolution of a given LOD has to be proportional to the proximity of the surgical focus area. The operation area has the highest resolution, and the resolution degraded smoothly with the increase of the distance from the focus. Before this work, none of proposed multiresolution frameworks supports dynamic update of the mesh topology.

Chapter 3

SYSTEM ARCHITECTURE

As we stated in previous chapter, the current TriS includes three parts: real-time simulation engine, input interface and output interface. In our system, PHANTOM Premium provided by SenAble Technologies is used as the haptic input and force output interface. The user also can use the mouse to set some global parameters and stop the simulation. The graphic output is sent to the computer monitor.

3.1 System Framework

Developing the simulation engine is the major part of this thesis work. Simulation engine includes three modules: physical simulation, haptic rendering and graphic rendering. Physical simulation module models and simulates the soft tissue, which includes solving the system equations by a numerical method and implementing the cutting algorithm. We put collision detection part into the haptic rendering module because it runs in the haptic loop and has a frequency of 1KHz. Haptic rendering module also includes force feedback computation. Graphic rendering sends the visual results of the simulation to users. Both physical simulation and graphic rendering

run in the simulation loop which has an update rate of 30Hz. Figure 3.1 showns the framework of our system.

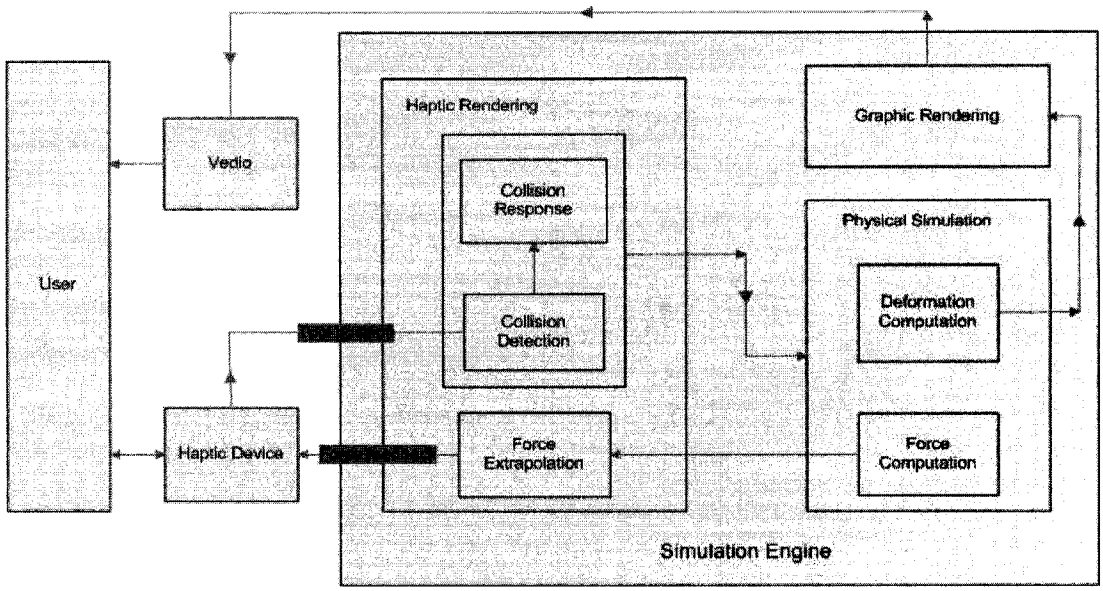


Figure 3.1: System Framework

When the simulation starts, haptic rendering module always checks the position of the tool. Once collision is detected, physical simulation module will be notified a contact event, and it then queries the position and force information from the haptic loop. By using these data the new position and force of tissue model are computed. The new position information is fed back to the user by the graphic rendering module, while the force is sent back to the haptic rendering module. After extrapolation, the force is sent to the haptic device and is felt by the user.

The implementation of the haptic rendering module is based on the haptic System Development Kit (SDK) OpenHaptics from SensAble Technology Inc. Unfortunately, its functionality is limited. Therefore, we implemented our own intersection detection algorithm and force extrapolating algorithm. As for the graphic rendering, graphic

library OpenGL is used to control, map and display the virtual scene.

3.2 System Diagram

The haptics enabled TriS is multi-threaded because physical simulation (graphic rendering also) and haptic simulation require very different update rates. These two threads run separately in the same time. They communicate with each other and share the information including the position of haptic device and the state of the soft tissue model. Figure 3.2 shows our system diagram.

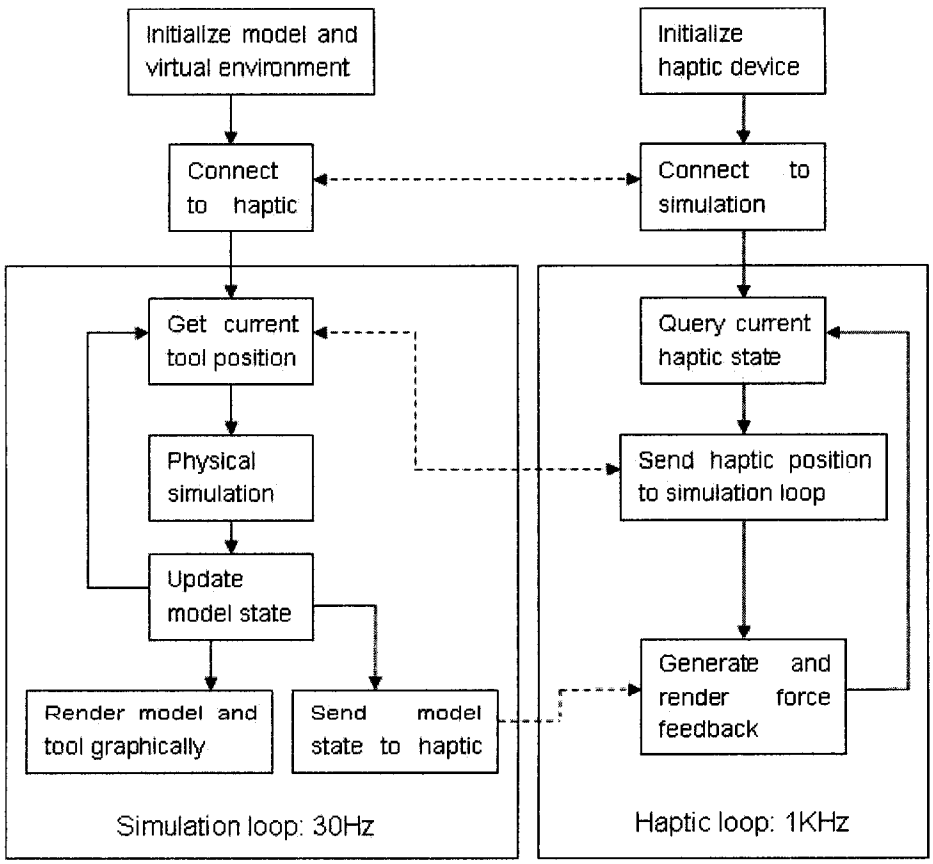


Figure 3.2: System Diagram

Our simulation loop runs in an update rate around 30Hz. After initializing system and virtual environment, it reads position data from the shared data structure repeatedly at every simulation time step (about $0.033ms$). Using these data, it computes the new state of the soft tissue model, which includes the position and force. It also implements the cutting algorithm if cutting starts. Finally, it updates the model position in the virtual environment by graphic rendering.

Once the simulation loop initializes the haptic device, the haptic loop is activated. It checks the tool position and store it into the data buffer, and in the mean time it implements collision detection by comparing the current tool position with the state of tissue model stored in the shared data structure the through collision detection algorithm. The results such as touching, untouching and haptic device moving event are sent to the simulation loop. If contact detected, it also computes the feedback force and sends it back to the device. All these actions are repeated with a frequency of 1KHz.

Chapter 4

SOFT TISSUE MODELING

Soft tissue modeling is the most fundamental and important component of a Tris. It directly affects the realism of deformation and the fidelity of feedback force. The bio-mechanical characteristics of the model also determine the speed of the modeling computation, which in turn affects whether the real time performance can be achieved or not. To develop a model for real-time deformation, the following three problems should be addressed:

- choose the geometric description of the object
- develop a mathematical model of the elastic deformation
- find a solution algorithm to the motion equations of the model

An explicit finite element model which was proposed by Cotin et al. [27], the tensor mass model (TMM), is used as the soft tissue model in this thesis.

4.1 Tensor-Mass Model (TMM)

4.1.1 Why Tensor-Mass Model

A soft tissue model should react to the applied force with a high resemblance to real one (realism) and in real time (speed). The deformable models developed up to now have prompted one characteristic to the detriment of the other. So that soft tissue modeling is still an open research field, and currently there is no any model can ideally meet all the requirements at the same time, for example, the physiological realism, speed and robustness. Therefore, we must balance the trade-off between these characteristics and use the one which is most appropriate to achieve our simulation goal.

- **Volumetric vs. Surface Model**

The main goal of this thesis is to develop a realistic real-time cutting simulation. As we stated before, cutting simulation should handle and display the interior change of the soft tissue model. This can only be represented by a volumetric model. Although the surface model is simple and easy to implement, we need to choose a volumetric model in order to achieve our goals.

- **Point-mechanics Based vs. Continuum-mechanics Based Model**

Continuum-mechanics based models are more complicated and harder to implement comparing to point-mechanics based models such as MSM, but their bio-mechanical properties are coincident with the natural characteristics of living tissues. Thus the computed deformation and force are much more accurate than those resulted from the MSM. In addition, it is a reasonable expectation that the computation bottleneck will not be a problem any more in the future. So continuum-mechanics based models have more optimistic prospective than point-mechanic based models.

Further more, another drawback of MSM is that it is difficult to determine the stiffness and damping parameters for each link, especially when the model is a volumetric mesh. In addition, inappropriate parameters are very easy to cause model oscillation. As for continuum-mechanics based models, their parameters are easily obtained from experiments. Since all bio-mechanical data related to biological soft tissue are formulated as continuum-mechanics parameters, such as Young's modulus and Poisson coefficients.

Finally, because the behavior of point-mechanics based volumetric models strongly depends on the topology of the spring network, when a spring is removed or added, the elastic behavior of the whole system may change drastically, which is not desired for the simulation. In summary, continuum-mechanics based models are more appropriate for this thesis.

- **Implicit FEM vs. TMM**

Among all types of continuum-mechanics based models, FEM is the typical and most suitable model to do the volumetric modeling. But if we use the implicit (traditional) finite element method to solve the motion equations, it would be very hard to achieve the real-time requirement with current computer technology. Solving explicit FEM doesn't need stiffness matrix inversion, and it has similar computational complexity with the MSM. Therefore tensor-mass model combines the accuracy of the FEM and the fast computational speed of MSM together. And it is very suitable to do the cutting simulation because its behavior mostly depends on the mesh resolution rather than on mesh topology. According to above characteristics, the TMM is chosen in this thesis.

4.1.2 Linear or Nonlinear Model

For a linear elastic model, any mesh deformation can be computed from the knowledge of a finite set of elementary deformations, thus the computation time can be optimized. On the other hand, the computation of nonlinear model is very complex and hard to be simplified.

Although the physical behavior of the soft tissue is nonlinear, it can be considered as linearly elastic if the displacements remain small, which means less than 10% of the mesh size [62]. Further more, it is reasonable to consider the deformation is small in surgery simulation, because when the operator deforms the virtual organ, the force applied to the hand will increase proportionally to the deformation, thus preventing large deformations. Under this situation, the linear elastic model is good enough to represent soft tissue in the simulation.

Therefore, we use a linear elastic TMM rather than a non-linear model [29] in this thesis.

4.2 Geometric Description of the Modeled Object

Tetrahedron is chosen as the element shape. So the geometry of the soft tissue is described by a set of tetrahedra. The nodal numbering of the tetrahedral elements for the model is shown in Figure 4.1. The four vertices of each tetrahedron are numbered from 0 to 3. They are ordered such that if you apply the right hand rule to the first three vertices, the resultant vector points toward the fourth vertex. The nodal numbering of all elements in the model conforms to this rule.

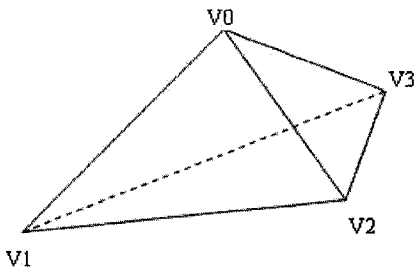


Figure 4.1: Nodal Numbering of Tetrahedral Element

4.3 Mathematical Model of TMM

Rather than merging the motion equations generated from each element into a large global one (see equation 2.1), TMM solves each element independently through a local approximation. The mass-lumping method decouples the movements of each node. Then equation 2.1 can be written as the set of independent node differential equations as follow:

$$m_i \frac{d^2\mathbf{P_i}}{dt^2} = c_i \frac{d\mathbf{P_i}}{dt} + \mathbf{F_i} \tag{4.1}$$

where m_i is the mass of node i. Node mass is proportional to the volume of the elements that the node belongs to. It is calculated with the following equation:

$$m_i = \sum_{j \in E(i)} \frac{1}{4} \rho_j V_j^e \tag{4.2}$$

In order to get node deformations, a TMM solves equation 4.1 approximately for each vertex of the mesh by an iterative approach. So first we need to compute the force applied on each vertex $\mathbf{F_i}$, which can be obtained by the derivation of the elastic energy of the model.

4.3.1 Mathematical Model of the Elastic Deformation

According to Cotin et al. [27], a TMM constructs and computes the motion equation based on tetrahedral elements. So that the total force at the nodal locations is as follow:

$$F_i = F_{i\_elastic} + F_{i\_ext} \tag{4.3}$$

$$F_{i\_elastic} = K^e U^e \tag{4.4}$$

where F_i is the total node force, which includes the elastic force and external force; $F_{i\_elastic}$ is the elastic force caused by the deformation of the model; F_{i_ext} is the external force such as the push force applied by the user on the soft tissue.

As described by Cotin, the linearly elastic force acting on each vertex can be computed by the following three steps:

1. Define the displacement vector at a point (x, y, z) inside a tetrahedron T_i as a function of the four displacement vectors at each vertex.

2. Represent the elastic energy of a tetrahedron by a function of these four displacement vectors

3. Compute the elastic force $F_{i\_elastic}$ applied to the vertex i.

This calculation decomposes the element stiffness matrix into its node and edge components, with the nodal numbering shown in figure 4.1, K^e can be presented as follow:

$$K^e = \begin{bmatrix} K^e_{00} & K^e_{01} & K^e_{02} & K^e_{03} \\ K^e_{10} & K^e_{11} & K^e_{12} & K^e_{13} \\ K^e_{20} & K^e_{21} & K^e_{22} & K^e_{23} \\ K^e_{30} & K^e_{31} & K^e_{32} & K^e_{33} \end{bmatrix} \tag{4.5}$$

Finally the ealstic force $F_{i\_elastic}$ on a given vertex P_i is:

$$F_{i\_elastic} = [K_{ii}]P_i^0 P_i + \sum_{j \in N(P_i)} [K_{ij}]P_j^0 P_j \tag{4.6}$$

where $[K_{ii}]$ is the sum of tensors $[K_{ii}^{T_j}]$ associated with the tetrahedra adjacent to vertex i; P_i is the current position of vertex i and P_i^0 is its rest position; $[K_{ij}]$ is the sum of tensors $[K_{ij}^{T_j}]$ associated with the tetrahedra adjacent to edge(i,j); T_j is the set of tetrahedra which adjacent to vertex i; $N(P_i)$ is the vertex list, those vertices are the neighbors of vertex i. Here i and j indices represent the node or edge to which the tensor matrix belongs. If the index repeat, for example, $K_{ii}^{T_j}$ related to the force felt by vertex i due to its own displacement from its rest position and we call it node tensor. If the index does not repeat, for example, $K_{ij}^{T_j}$ related to the force felt by vertex i due to the displacement of vertex j from its rest position and we call it edge tensor. Because the symmetric of the element stiffness matrix K^e (see equation 4.5), $K_{ij}^{T_j}$ equals to the transpose of $K_{ji}^{T_j}$.

All above tensor matrices depend on the Lamé coefficients: λ and μ, and the normal vectors of triangles of a tetrahedron, which we call M vectors. The tensor matrices can be computed using the following equation:

$$[K_{jk}^{T_i}] = \frac{1}{36V(T_i)} \left(\lambda_i M_k^{T_i}(M_j^{T_i})^T + \mu_i M_j^{T_i}(M_k^{T_i})^T + \mu_i (M_j^{T_i} \cdot M_k^{T_i}) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) \tag{4.7}$$

where $M_j^{T_i}$ is the normal vector of the jth triangle in tetrahedral element T_i, $V(T_i)$ is the volume of T_i.

The M vector only depends on the rest positions of the four vertices belong to

tetrahedron T_i, which can be computed as follow:

$$M_j^{T_i} = P^0_{T_i(j+1)} \wedge P^0_{T_i(j+2)} + P^0_{T_i(j+2)} \wedge P^0_{T_i(j+3)} + P^0_{T_i(j+3)} \wedge P^0_{T_i(j+1)} \tag{4.8}$$

To compute the deformation of the model and feedback force, we need to solve a set of motion equations based on each vertex of the mesh:

$$m_i \frac{d^2\mathbf{P_i}}{dt^2} + c_i \frac{d\mathbf{P_i}}{dt} + \mathbf{F_{i\_elastic}} = \mathbf{F_{i\_ext}} \tag{4.9}$$

This set of equations need to be solved approximately by using a numerical integration method. But before integration, the external force F_{i_ext} should be calculated first.

4.3.2 External Force Distribution

When the scalpel contacts with the soft tissue model, an external force is applied on the contact point, and the model is deformed in the neighborhood of the contact point. The motion equations of the TMM are based on each node of the mesh, which means the external force is applied on the nodes. But the real initial interaction point usually lies within the area of a surface triangle of the model rather than on a surface node. Therefore, this force must be re-distributed to the nodes. The simplest way is to apply this external force to the nearest surface node. We propose a method to re-distribute the force to three nodes of the triangle as follow: The location of any three dimensional points with respect to a given triangle can be represented in barycentric coordinates [63]. In barycentric form the 3D Cartesian point P is represented as a linear combination of the three vertices making up any triangle.

$$P = b_0 v_0 + b_1 v_1 + b_2 v_2 \tag{4.10}$$

$$b_0 + b_1 + b_2 = 1 \tag{4.11}$$

where v_0, v_1, v_2 are the three nodes of the tetrahedron; b_0, b_1, b_2 are barycentric coordinates of point P, which can be calculated by following equations:

$$b_0 = \frac{A_{rea}(T_{p12})}{A_{rea}(T_{012})} \tag{4.12}$$

$$b_1 = \frac{A_{rea}(T_{p02})}{A_{rea}(T_{012})} \tag{4.13}$$

$$b_2 = \frac{A_{rea}(T_{p01})}{A_{rea}(T_{012})} \tag{4.14}$$

$$\tag{4.15}$$

where T_{012} is the triangle with vertices v_0, v_1, v_2. Similar definitions are used for other sub-triangles. The area of the triangle, T_{ABC}, with vertices A, B and C is:

$$A_{rea}(T_{ABC}) = \frac{1}{2}\sqrt{(\mathbf{AB}\cdot\mathbf{AB})(\mathbf{AC}\cdot\mathbf{AC}) - (\mathbf{AB}\cdot\mathbf{AC})^2} \tag{4.16}$$

Then the force F_p acting on point P is distributed to the triangle nodes v_0, v_1, v_2 as b_0F_p, b_1F_p, b_2F_p, respectively. Figure 4.2 shows the force distribution.

4.4 Numerical Integration

When we choose a numerical integration method to solve above question, the following three aspects of the algorithm should be considered:

- computational efficiency
- stability
- accuracy

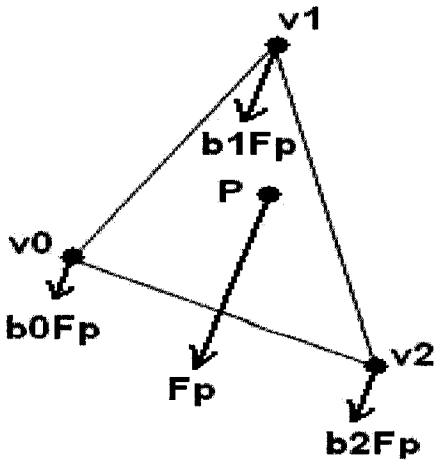


Figure 4.2: Distributing the external force on four nodes of the tetrahedron

Problems involving ordinary differential equations (ODEs) can always be reduced to the study of sets of first-order differential equations [64]. Therefore the equation 4.1 can be re-written as:

$$\frac{dP_i}{dt} = v_i(t) \tag{4.17}$$

$$\frac{dv_i}{dt} = \frac{1}{m_i}F_{i\_ext} - \frac{1}{m_i}F_{i\_int} - \frac{c_i}{m_i}v_i(t) \tag{4.18}$$

These two set equations can be easily solved by numerical integration.

4.4.1 Implicit and Explicit Integration

There are two categories of numerical integration methods: implicit and explicit integration. Implicit approaches are unconditionally stable but computationally very expensive. Explicit approaches are computationally efficient but suffer from stability problem. We briefly describe these two approaches as follows.

Implicit Integration

Implicit integrations have unknown terms at both sides of the equation. It computes the new position $x_i(t+\Delta t)$ by using the derivative of the next state of the acceleration $a_i(t+\Delta t)$:

$$v_i(t+\Delta t) = v_i(t) + \Delta t a_i(t+\Delta t) \tag{4.19}$$

$$x_i(t+\Delta t) = x_i(t) + \Delta t v_i(t+\Delta t) \tag{4.20}$$

To calculate $x_i(t+\Delta t)$, we need to know $v_i(t+\Delta t)$ first, which can be calculated from $a_i(t+\Delta t)$. We can use a first order Taylor expansion of the force to approximately get $a_i(t+\Delta t)$ from current data.

$$f_i(t+\Delta t) = f_i(t) + \frac{\partial f_i(t)}{\partial x_i(t)}\Delta x \tag{4.21}$$

This calculation involves matrix inversion. Although implicit schemes are unconditionally stable, which implies that large time steps can be used, they are still too computationally expensive. This prohibits implicit methods to be used in the real-time simulation.

Explicit Integration

Explicit schemes compute the value at time $t+1$ in terms of known values at time t. They use the forces at time t to calculate node acceleration $a_i(t)$ and velocity $v_i(t)$ and then predict the displacement at time $t+\Delta t$. They are conditionally stable, which means they require small time steps to ensure convergence. The advantage of explicit schemes is that no matrix inversion is required for updating each vertex. This is useful for cutting simulation, because when the mesh topology has been changed,

only the local tensor matrices $[K_{ii}]$ and $[K_{ij}]$ need to be updated, then the new vertex positions can be computed. Because FEM is computationally very expensive, to achieve real-time interaction, we choose the explicit integration method to solve the system. In the following two explicit integration methods will be discussed.

4.4.2 Euler Method

Euler method is a truncated Taylor series in which only two terms are taken into account. It is the simplest way to solve governing equation 4.1:

$$v_i(t + \Delta t) = v_i(t) + a_i(t)\Delta t$$
$$x_i(t + \Delta t) = x_i(t) + v_i(t)\Delta t$$

where Δt is the time step. This method can accurately solve the differential equation only when the first derivative is constant. Otherwise the computation error in the single step is proportional to the square of the time step, and the global error over all steps will be proportional to the time step Δt. So that the larger the time-step, the larger the error, then numerical divergence may appear. Therefore a very small time step is needed to ensure system stability.

4.4.3 Fourth-Order Runge-Kutta Method

Fourth-order Runge-Kutta (RK-4) method is a predictor-corrector method, which try to combine the advantages of the simplicity of the explicit method with the improved stability of the implicit method. It achieves this by using an explicit method to predict the solution $y_{n+1}^{(p)}$ at t_{n+1}, and in order to correct this prediction $f(x_{n+1}, y_{n+1}^{(p)})$ is utilized as an approximation to $f(x_{n+1}, y_{n+1})$.

As we stated before, in our system, y_{n+1} is the node velocity at time t_{n+1}, $f(x_{n+1}, y_{n+1})$ is the node acceleration at time t_{n+1}. Therefore, the RK-4 method can be described by following equations:

$$\begin{aligned}
k_1 &= \mathbf{v}_n \\
l_1 &= \mathbf{a}(t_n, \mathbf{x}_n, \mathbf{v}_n) \\
k_2 &= l_1 \Delta t \\
l_2 &= \mathbf{a}(t_n + \frac{\Delta t}{2}, \mathbf{x}_n + \frac{\Delta t}{2} k_1, \mathbf{v}_n + \frac{\Delta t}{2} l_1) \\
k_3 &= l_2 \Delta t \\
l_3 &= \mathbf{a}(t_n + \frac{\Delta t}{2}, \mathbf{x}_n + \frac{\Delta t}{2} k_2, \mathbf{v}_n + \frac{\Delta t}{2} l_2) \\
k_4 &= l_3 \Delta t \\
l_4 &= \mathbf{a}(t_n + \Delta t, \mathbf{x}_n + \Delta t k_3, \mathbf{v}_n + \Delta t l_3) \\
x_{n+1} &= x_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5)
\end{aligned}$$

Where x_n, x_{n+1} are the node position at time t and $t + \Delta t$, respectively; $\mathbf{v}$ is the node velocity which can be calculated by equation 4.17; $\mathbf{a}$ is the node acceleration which can be calculated by equation 4.18.

Thus, the next value $x(t + \Delta t)$ is determined by the present value $x(t)$ plus the product of the size of time step and a weighted derivate. The weighted derivate is calculated by four estimated derivates: one calculated at the initial point, two at trial midpoints and another one at a trial endpoint. The accuracy of RK-4 method increases dramatically comparing to Euler method. The single step error is of order Δt^5, and the global error is of order Δt^4. The following table shows the computational accuracy of these two methods.

From table 4.1, it is obvious that in these two numerical integration methods,

	Single Step Error	Global Error
Euler Method	$O(\Delta t^2)$	$O(\Delta t)$
Fourth-order Runge-Kutta	$O(\Delta t^5)$	$O(\Delta t^4)$

Table 4.1: Computational accuracy of Euler and RK-4 methods

RK-4 method has the higher computational accuracy than Euler method.

4.4.4 Computational Efficiency and Stability

Both the computational efficiency and stability of an integration method are related to the time step of the integration, and they are contradicted to each other. In order to have high efficiency, a large time step is needed to be adopted. But the integration error is directly decided by the size of time step. If the time step is too large, the error will increase rapidly, and the model will become unstable. If a small time step is used, the system become more stable but the computational load will increase, which results in the increase of the calculation time and makes the real time interaction become impossible.

Both the efficiency and stability requirements must be satisfied in order to implement a successful surgery simulator. High efficiency reduces the computational load and ensures the real-time performance. High stability ensures the model behave appropriately and do not oscillate and shoot off to infinity. System instability may cause the damage to the haptic device and inflict bodily harm to the user. We should balance the trade-off between these two requirements.

This trade-off can be approximately analyzed by examining the maximum time step that could be used without causing instability and the actual computing time. The ratio between these two time values shows, under the stable condition, the computational efficiency of the integration method. The bigger the ratio, the faster the

computation, thus the higher efficiency of the method. We compared Euler and RK-4 methods by this approach, the maximum time step and calculation time are obtained by using our Pentium 4 PC, with a model size of 100 elements and 50 vertices.

Integration Method	Max. Time Step	Calculation Time	Ratio to Real-Time
Euler	0.0016	0.003	0.53
RK-4	0.032	0.014	2.28

Table 4.2: Computational efficiency and numerical stability of integration methods

As can be seen in Table 4.2, under stable condition, the RK-4 method has the higher computational efficiency, so that it is used in our simulation.

Chapter 5

CUTTING

With a good cutting simulation, users should be able to see the realistic cutting result in real time and feel the faithful cutting force by hand through the haptic device. There are two main causes that make real-time cutting simulation a big challenge:

- All cutting manipulations change the mesh topology of the soft tissue model and re-meshing is necessary. This is a complicated procedure and it becomes extremely complex when volumetric soft tissue models are used.
- Cutting changes the stiffness matrix or the governing equation of the system. Dynamic reconstruction of the linear system is very expensive, and even worse, it invalidates the pre-computation technique which can be used to accelerate the simulation.

In addition, cutting will generate many new elements if we ask the splits follow the arbitrary cutting path well. This will dramatically increase the calculation complexity, add more work loads to the already computationally intensive simulation and make the real-time interaction much harder to achieve. Over the past ten years, many researchers have worked in this field, but the fully interactive and highly realistic

cutting simulator, which can be used in surgeons' daily training, are currently far beyond reach.

Our goal is to develop a cutting scheme which has improved realism while maintains the real time interaction. In this chapter, we will discuss and compare three existing algorithms, then describe our new cutting method and all its characteristics.

5.1 Cutting Procedure

A cutting procedure includes the following three steps:

- detect intersected elements
- handle intersected elements and modify the topology of the mesh
- reconstruct the linear system and then compute deformation and force

The cutting simulation of this thesis is based on tetrahedral soft tissue model. First, the initial contact between the scalpel and the model is detected through the intersection between the surface triangles and the scalpel, and then the external force is calculated. If the force reaches the cutting threshold, cutting procedure starts. Otherwise, only deformation will be simulated. Once the cutting starts, the simulation loop will construct the intersected element list first, then with the scalpel moving, this list will be updated and the intersected elements which the scalpel already left will be found and saved into the non-active element list. Finally all non-active intersected elements will be subdivided or separated. This procedure repeats until the scalpel totally leave the model, then the cutting ends. Figure 5.1 shows the diagram of the cutting procedure.

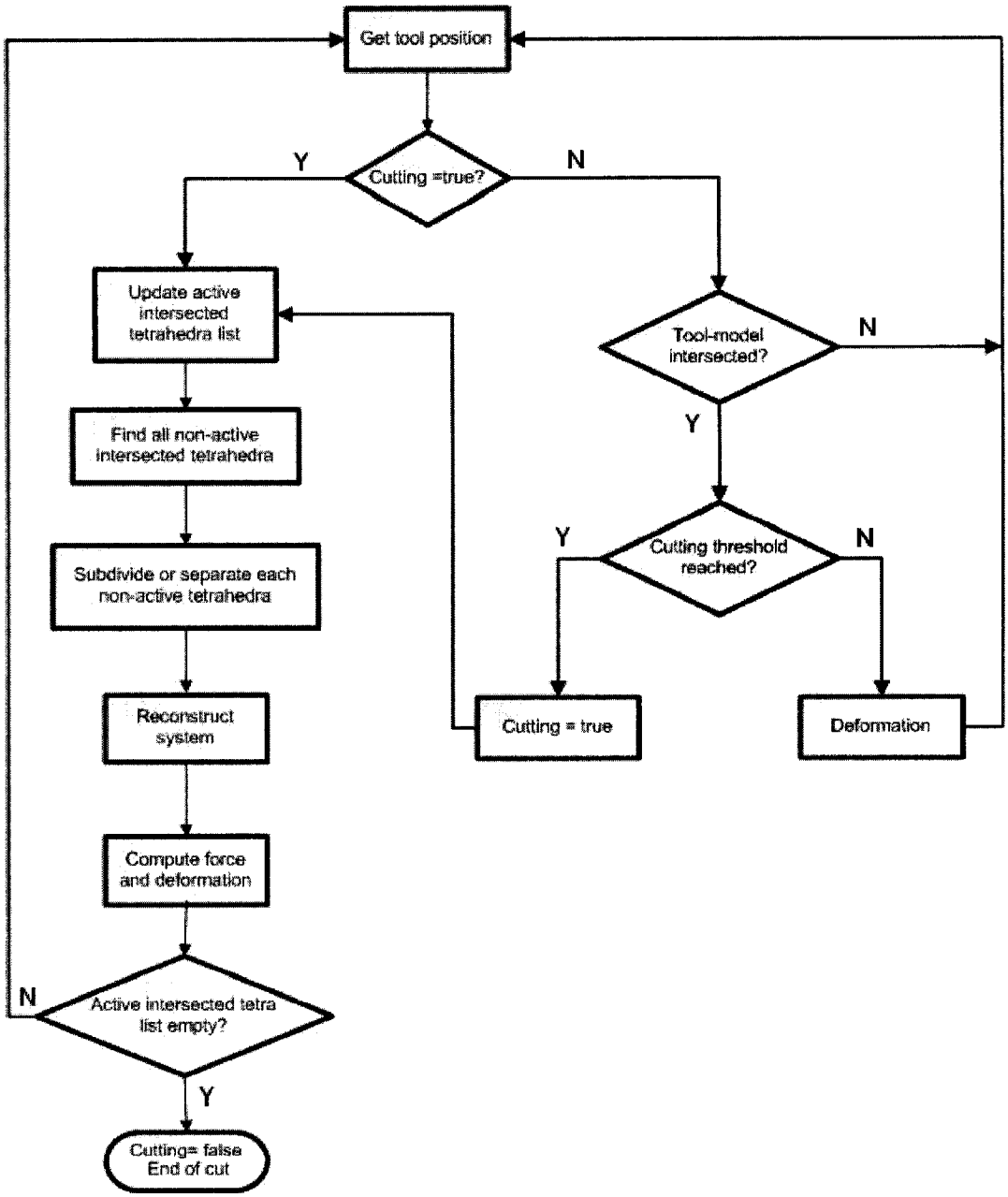


Figure 5.1: Cutting procedure

5.2 Cutting Method

According to the different handling ways of intersected elements, cutting methods fall into the following three categories: element deletion, element separation and element subdivision.

5.2.1 Element Deletion Method

This method simply removes each element which is touched by the scalpel during the cutting process (see figure 7.4 for surface mesh cutting).

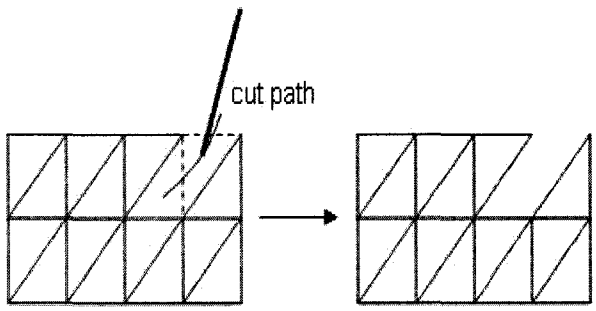


Figure 5.2: Cutting by element deletion on surface mesh

If the cutting limits to one single incision, removing elements usually will not affect the total node number of the mesh. Therefore the cutting can be simply implemented by deleting the intersected edges of a surface model or deleting intersected faces of a volumetric model. This makes geometric modification very easy and avoids the expensive re-meshing procedure. Reconstructing the system is also easier for both the point-mechanics based soft tissue models and explicit finite element models, because the stiffness matrix can be modified by simply removing the stiffness parameters or tensors associated with the deleted elements. But for an implicit finite element

model, which involves matrix inversion, computational load will increase a lot due to the cutting.

Bro-Nielsen [23] first introduced this method to cut an implicit finite element model. He removed a tetrahedron element from a simple model with 75 nodes and got the real time response. But when the number of mesh nodes increased to 1125, the modification of the inverted stiffness matrix took about one minute, which made the real time simulation impossible. Cotin et al. [27] used this method to implement a real time cutting simulation based on a pre-computed and tensor mass combined hybrid liver model. They achieved a real-time visual update rate by using a TMM of 280 vertices and 1260 tetrahedra. The element deletion method was also used in systems based on the MSM by Neumann et al. [65] and Boux de Casson et al. [66].

Although this method is simple, computationally efficient and easy to implement, it has some significant drawbacks as follow:

- This approach violates the physical principle of mass conservation.
- Both the visual effect and the simulation accuracy strictly depend on the resolution of the mesh.
- It is possible that all elements connected with a node are deleted, and this situation will cause the oscillation of the simulation system.

Only when the mesh resolution is very high, can the visual quality be acceptable. Otherwise, the cut part will leave a big gap. Unfortunately, if the resolution increases, the computational cost of the simulation will significantly increase. This is not affortable for the real-time simulation. In order to balance the trade-off between the good visual results and high computational cost, Forest et al. [57] suggested locally refine the mesh around the cutting area and then remove the intersected elements.

5.2.2 Element Separation Method

This kind of method restricts incisions to be aligned with existing edges in surface models and faces in volumetric models. Along these edges or faces, the intersected elements are separated. That means the nodes on these edges or faces, the edges and faces are duplicated. After single cut, no new element will be generated. Even after multiple cuts, only small number of new elements will be created. Figure 5.3 shows the separation cutting method with a surface mesh.

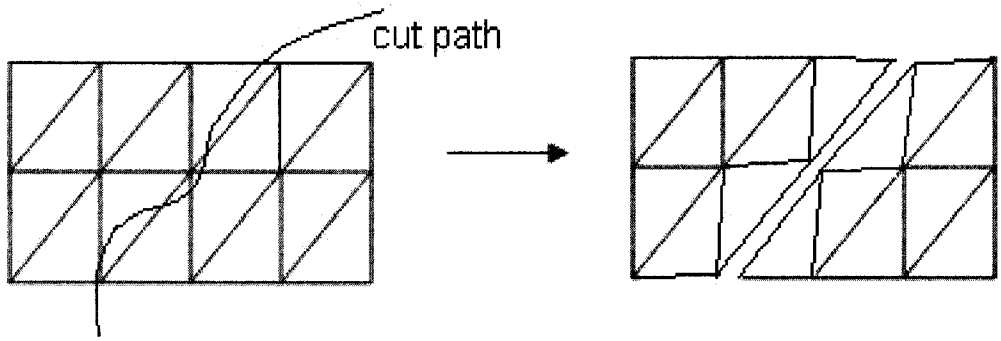


Figure 5.3: Cutting by element separation on a surface mesh

The major drawbacks of this method are as follow:

- The incision can not follow the cut path well, and the approximation quality highly depends on the resolution of the mesh. The cutting face will be jig-jagged.
- ELement separating may create singularities, which means some elements connecting to the mesh by only one node. This situation will cause the mesh and system unstable.

5.2.3 Element Subdivision Method

In this method, intersected elements are subdivided. They are replaced by a set of small elements which occupy the same volume but with no proper intersection with the trajectory of the scalpel. Comparing to volumetric models, subdividing triangle elements of the surface models is much easier. Zhang et al. [56] used this method to implement a progressive cutting on a surface MSM.

Cutting Tetrahedra

Subdividing tetrahedral element is very complicated. There are five topologically different cases in which a tetrahedron can be cut by a planar surface with one incision [58] (see Figure 5.4).

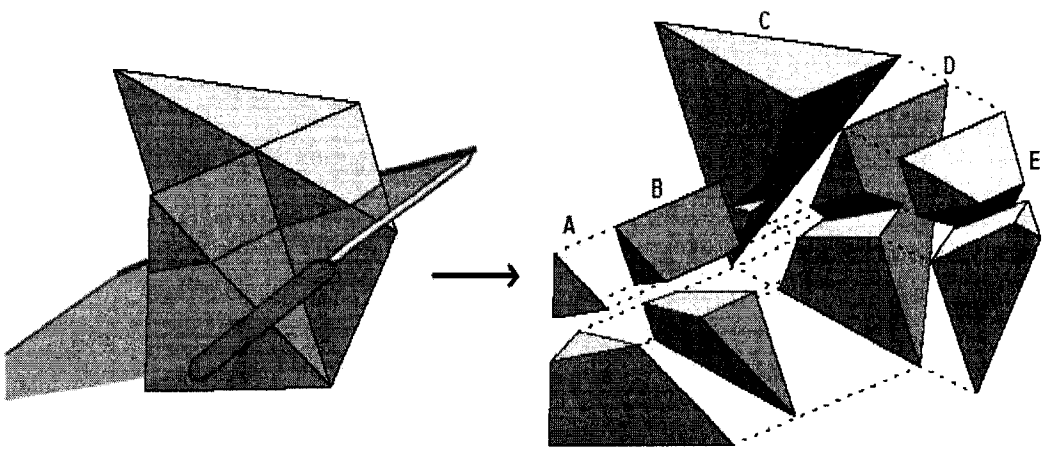


Figure 5.4: Five different cases when cutting a tetrahedral mesh (source[58])

Case A and B represent two full cut situations, in which the tetrahedron is completely cut into two pieces. Three types of partial cut can be distinguished by the number of edge and face intersections. In case C, only one edge and two faces

intersect with the scalpel. There are two face intersections in both case D and case E, but with two and three edge intersections in each case, respectively.

Subdividing Tetrahedra

There are different ways to subdivide the intersected tetrahedral elements. Bielser et al. [58] proposed a generic subdivision method, which always splits each element into 17 small tetrahedra using midpoints as shown in figure 5.5.

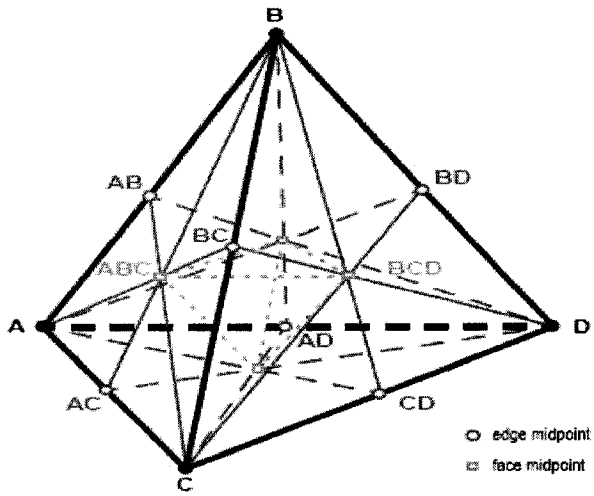


Figure 5.5: Generic subdivision of a tetrahedron (source[58])

If the edge or face has been cut, then the intersected point will substitute for the midpoint of that edge or face. Split edges are replaced by 2 edges with 2 vertices at the new location. Unintersected edges get one vertex. This method can accurately follow the arbitrary cut trajectory and has topological freedom. But the universal subdivision leads to a rapidly increased number of tetrahedra for large cuts, which subsequently makes the computation very expensive. To decrease the number of newly created elements after cutting, Mor [28] introduced an improved scheme, which

generates a minimal set of new elements during cutting. Depending on five different cases, 4 to 9 new elements will be created for each intersected tetrahedron. Our cutting scheme will base on this approach.

The advantage of element subdivision method is that its visual realism is very high. To achieve this, many small tetrahedra have to be created, which greatly increases element number and thus slow down the simulation substantially. Even worse, the small or badly shaped tetrahedra created during cut will cause simulation instability. Another problem with this method is: it doesn't handle the case which the mesh node is cut.

5.3 Combined Subdivision and Separation Cutting Method

When we develop a cutting scheme, three requirements should be always in mind:

- The resulting incision should follow the arbitrary cut path as closely as possible, so that the simulation will have very realistic visual feedbacks.
- The number of newly generated element should be as less as possible, so that the computational cost will not increase significantly, and therefore the real-time interaction can be maintained.
- There should be no very small or badly shaped elements created after cutting so that the simulation system can keep running stably.

To achieve these requirements, we introduced a midpoint minimal set subdivision and element separation combined cutting method.

5.3.1 Midpoint Minimal Set Element Subdivision

Our tetrahedron subdividing is based on the minimal set subdivision method proposed by Mor [28]. Instead of using the actual intersection points to do the splitting, we use midpoints only. That means the intersected edges are split at edge midpoint and the intersected faces are split at each triangle's centroid (Its Cartesian coordinates are the means of the coordinates of the three vertices). Therefore, two new vertices at each edge intersection and one new vertex at each face intersection will be created. Depending on the five cutting topologies stated above, 4, 6, 6, 8, and 9 new tetrahedra will replace each intersected element, respectively. The subdivisions are shown in figure 5.6.

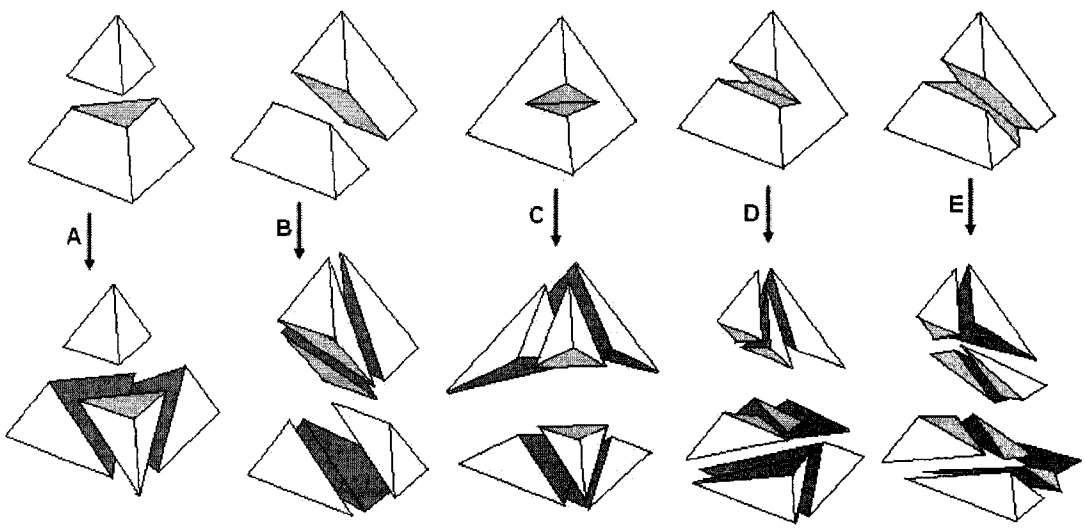


Figure 5.6: Midpoint subdivision for five different topology cases

Five different subdividing procedures are developed for these five cases. One important thing must be addressed during cutting is: keeping the consistency of the mesh. This is not a trivial problem because each vertex is neighbored by many

tetrahedra and each edge is also neighbored by many tetrahedra. When one edge is split and two new vertices created, all neighboring elements of this edge should be split into the same two vertices. By properly designing the data structure and using lookup table, we can handle this problem efficiently.

Although topologically only five different cases exit when cutting a tetrahedral mesh (here we don't consider the situation of cutting through the node of tetrahedra), each case has multiple different orientations based on the ordering of the vertices and the cut edges. We must distinguish those different orientations before we can use the subdividing procedure to do the splitting. For example, case C, in which the tetrahedron is partially cut by one edge and two face intersections, has six different permutations (see figure 5.7). They correspond to different edges being cut. We need to know which edge is the intersected edge, then the subdividing and topological re-meshing can be proceeded.

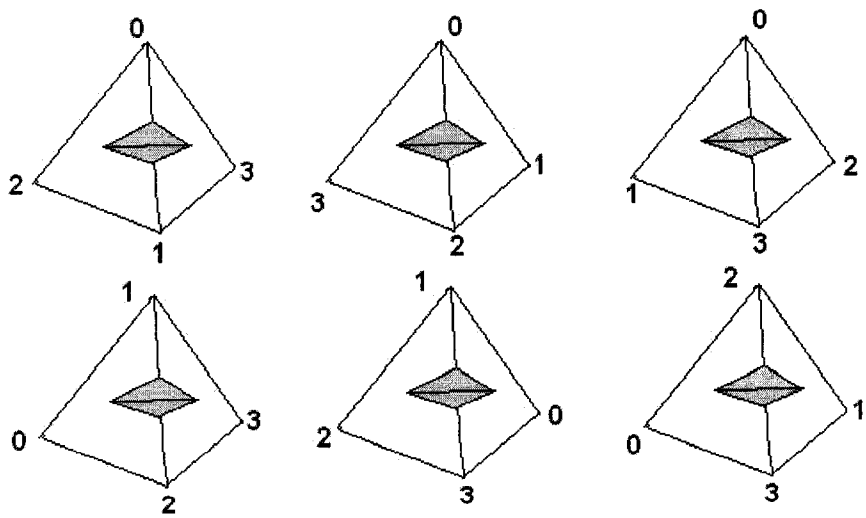


Figure 5.7: Six different orientation for cutting case C

To deal with this problem, we define a generic orientation for each case, and then

rotate the vertices of each intersected element to fit into this orientation. Therefore all subdivisions in the same type can be implemented with the same subdividing procedure. The original element is removed and the set of new tetrahedra are created.

5.3.2 Element Separation and Three New Topology Cases

The most important advantage of the element subdivision method is that the cut surface can exactly follow the cut trajectory. Unfortunately, when the scalpel passes closely to the original vertices, very small or badly shaped tetrahedra will be created. Those elements can become unstable during simulation and cause the system crash. Although our midpoint subdivision method can avoid this problem, it sacrifices the realism of visual feedback. The cut surface is approximated and jig-jagged. The approximation error becomes bigger when the intersection points are close to the original vertices. Furthermore, it is possible that the scalpel passes through one of the original vertex during cutting. The element subdivision method cannot handle this situation. To solve these problems, we introduce element separation method, which separates elements by one of the triangle face (see figure 5.8), into our cutting scheme.

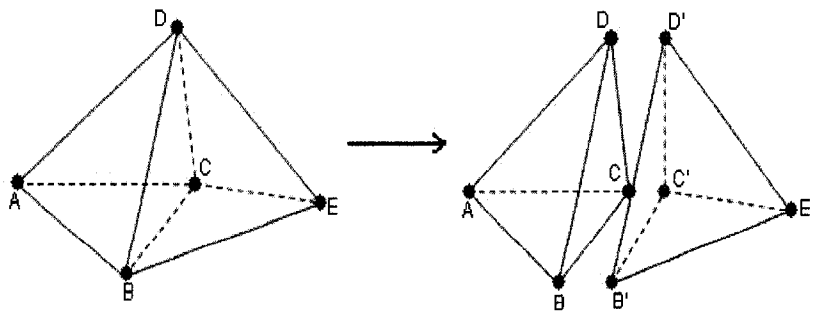


Figure 5.8: Separation of two tetrahedral elements

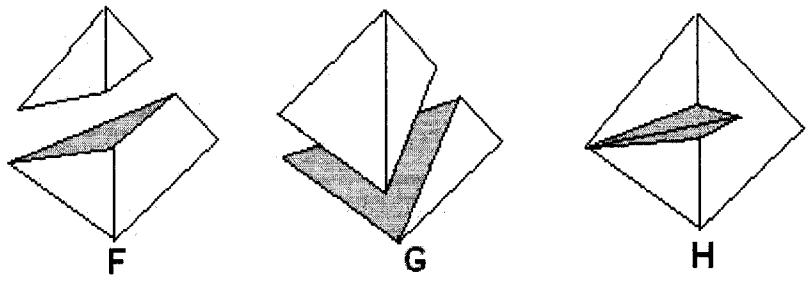


Figure 5.9: Three topology cases when cutting path passes through nodes

Due to the introduction of element separation, three new subdivision cases are needed to address the situations when one or two original vertices are separated. Figure 5.9 shows these three new cases.

Case F and G are two complete cut situations, which correspond to cutting through one node and cutting through two nodes, respectively. Case H is partially cut through one node. Figure 5.10 shows the minimal element subdivisions of these three cases.

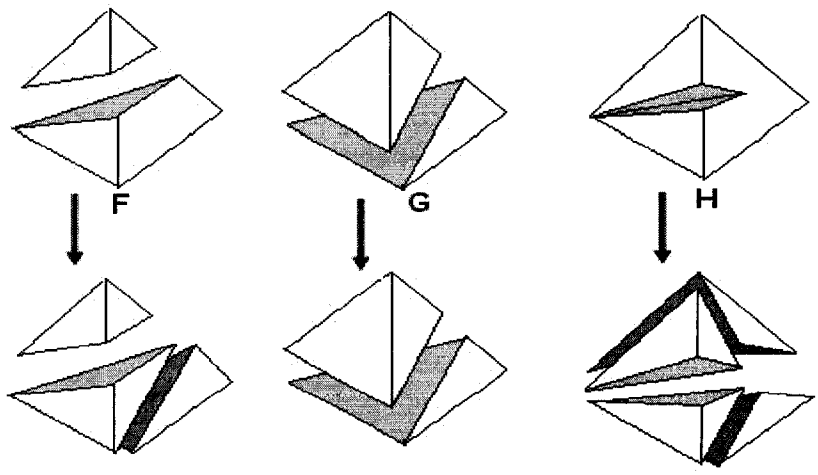


Figure 5.10: Three topology cases when cutting path passes through nodes

5.3.3 Node Snapping

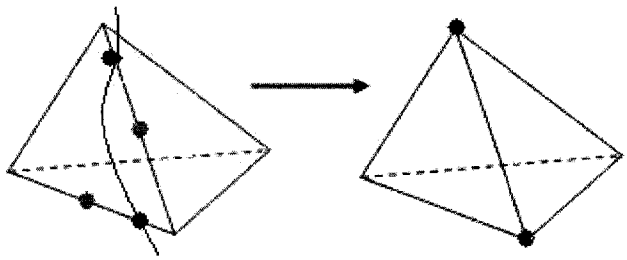
Node snapping is used to move the intersection points to the midpoints of edges and faces or to the original nodes. Once the edge intersection detected, the distances between the intersection point and two original edge nodes are calculated. Then these values are compared with a threshold, if the distance is within the threshold, the intersection point will be snapped to the corresponding original node. Otherwise it will be snapped to the midpoint of the edge. For face intersection, if the corresponding edge intersection point already snapped to one original node, the face intersection point will be snapped to one original node of the face, or the midpoint of the edge which faces to the original node to which the edge intersection point snapped. According to the node snapping results the algorithm determines whether the intersected element will be subdivided or separated. Figure 5.11 shows the snapping method of the intersection points.

5.3.4 Record Information for Intersected Element

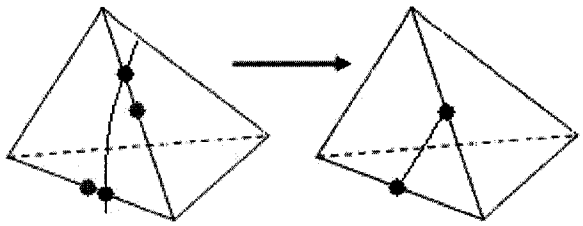
Combined subdivision and separation together, eight topologically different cases exist in our cutting scheme. For an intersected tetrahedron, how to efficiently record the cutting state and distinguish which case it belongs to is a difficult problem. We introduce a state table, which can clearly and efficiently describe the cutting topology of the tetrahedron; and use a lookup table to remember the new created nodes which must be used by future splitting the neighboring elements.

State Table

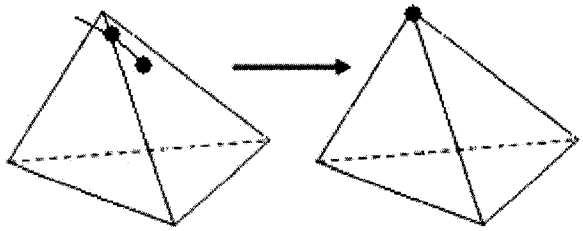
Since different cases are distinguished by different number of intersected edges, faces and nodes, the state table must show this information. Furthermore, for each inter-



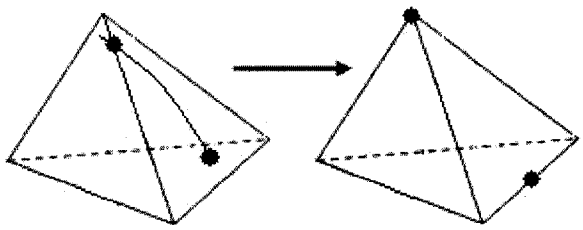
a) Intesected edge points snap to the original nodes



b) Intersected edge points snap to the midpoint of the edge



c) Intersected face point snap to the original node



d) Intersected face point snap to the midpoint of edge

Figure 5.11: Intersected node snapping

sected edge, face and node, the following two kinds of information should be recorded:

- the geometric data: which record the identification number of the intersected node, edge and face. In addition, if more than one node or edge or face are cut, the intersection order should be recorded too. This information will help us distinguish the different orientations in the same topological case.

- the state of split: whether the node, edge, or face is already split or not, use "true" or "false" to remember it. During re-meshing process, if the split state is true, we need to search the lookup table to get the newly created nodes. If the split state is false, the newly created nodes should be recorded in the lookup table for future split of the neighboring elements.

		1	2	3	4
Intersected face	face id	43	50		
	split	true	false		
Intersected edge	edge id	43	50	100	101
	split	true	false	false	false
Intersected node	node id	43	50	100	
	split	true	false	false	

Table 5.1: State table for intersected elements

Lookup Table

In a volumetric mesh, each triangle face has two neighboring tetrahedron elements, while each edge and node may have many neighboring tetrahedra (some nodes have more than thirty tetrahedra connected to it). Whenever one edge or face is subdivided, all its neighboring elements must be subdivided with the same newly created nodes in order to keep the consistency of the mesh. But not all of them are subdivided in the same time. So that once an edge or a face has been subdivided, the newly created

nodes information should be kept to be used by the future split of all its neighboring elements. In our cutting scheme, three lookup tables are used to keep this information for intersected nodes, edges and faces, respectively. The following is a lookup table for intersected edges.

index	edge id	origNd1	origNd2	newNd1	newNd2	totalNeib
1	25	43	50	100	101	6
2	40	15	20	102	102	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 5.2: Lookup table for intersected edges

In above table, origNd1 and origNd2 represent the two original nodes of the edge; newNd1 and newNd2 represent the newly created nodes due to edge split; totalNeib represents the number of elements which contain this edge but haven't been split yet. Whenever one neighboring element split, the totalNeib will decrease by one. Once the value of totalNeib becomes zero, the corresponding intersected edge entry will be deleted from the table. When a cutting procedure ends, all lookup tables should be empty. The lookup tables for intersected faces and nodes have similar structure.

5.4 Intersection Detection

In a surgery simulation system, both visual and haptic rendering are based on results of collision detection. It is necessary to know where the collisions are so that deformation and force feedback can be computed. In our simulation system, currently we don't consider the self-collisions of soft tissue. Only collisions between the surgical tool and soft tissues are addressed. Here we divide the collision information into two types:

- **Collision state:** the contact state between the haptic device and the soft tissue model, which includes three states: the scalpel touching, untouching and moving on the surface of the model.
- **Intersection state:** the intersection states between the cutting edge (scalpel edge) and the tetrahedral elements; the cutting face and the tetrahedral elements.

The first type of information is provided by the haptic SDK OpenHaptics. It only gives the touching, untouching and moving event notifications. We need to detect the intersection information by using our own algorithms.

Detecting the intersection between the scalpel and a volumetric model is much more complex than that between the scalpel and a surface model. With surface models, only single point collision detection algorithm is needed. As for volumetric models, a multiple point collision detection algorithm is required. In our simulation system, the detection of intersection is implemented in two steps:

- Detect the initial intersected tetrahedra
- If the scalpel is moving on the surface of the soft tissue model, detect consecutive intersections by using intersection propagation

5.4.1 Cutting Surface

Ideally, the interaction between the scalpel and the soft tissue model should be modeled as the interaction between two 3D objects. But this approach is too computationally expensive to implement in real-time. With our haptic device Phantom Premium, a point is used to represent the front end of the haptic device. During simulation, we can only get the point position of the haptic device. It is impossible to simulate

the cutting using the interaction between a point and a volumetric soft tissue model. In order to implement volumetric interaction, we use a line segment to represent the scalpel.

When a volumetric object is cut by a scalpel, the intersection can be obtained by sweeping a finite length of cutting edge through the object. So, the cutting surface can be defined as the planar surface swept by the cutting edge during different time steps. After the scalpel contacted with the soft tissue model, we can query haptic interface to get the proxy position and the actual haptic device position. The line between these two points is used as the cutting edge. Figure 5.12 shows the swept face of the scalpel between two time steps, which creates one edge intersection and two face intersections.

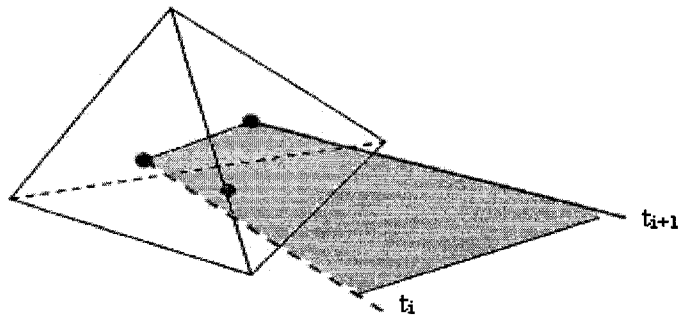


Figure 5.12: Intersections between the cutting surface and a tetrahedron

5.4.2 Initial Intersection Detection

As the haptic device only gives us the initial contact notice and the positions of the proxy and device, we need to find out the information of the initial contact element, which includes the index, the geometric and physical parameters of the element.

The following algorithm (in pseudocode) is used to implement this task. The initial intesection is obtained by global search.

```
IF (scalpel has contacted with soft tissue model)
    query the proxy position and device position;
    construct scalpel edge by proxy and device positions;
    FOR (all surface triangles)
        IF (triangle intersected with the scalpel edge)
            find the tetrahedral element which contains this
                                        surface triangle;
            return the tetrahedron element id;
        END IF
END IF
```

After finding the initially active surface tetrahedron, the next step is to find the initially active tetrahedra list by using the following pseudocode:

```
save the active surface tetra into active tetra container;
active_tetra = active_surface_tetra;
WHILE (scalpel tip is not inside the active_tetra) DO
    FOR(all other three faces of the active_tetra)
        IF (the face intersected with the edge of scalpel)
            find the neighbor tetrahedron of this face;
            save this neighbor tetra into active tetra container;
            active_tetra = this neighbor tetra;
            BREAK;
        END IF
```

When the while loop ends, all active tetrahedra will be saved into the active tetrahedra container.

5.4.3 Localized Intersection Detection

Before mesh topology modification, the tetrahedron elements which intersected with the cutting surface should be found. When we detect the intersection, two tests are required: the intersections between the scalpel edge (cutting edge) and the faces of the tetrahedron; and the intersections of the swept surface (cutting surface) and the edges of the tetrahedron. Both tests can be implemented in two ways: the global search or the local search.

Global search tests the intersection with each tetrahedron of the mesh repeatedly. Although the implementation logic is simple, it takes much longer time to get the result especially when the mesh size is big. Local search is based on the following fact: for sufficiently small time step Δt (maximum $0.033ms$) the probability of finding new active tetrahedra and new active point tetrahedron in the direct neighbors is very high.

Localized intersection detection is implemented by intersection propagation. The neighbors of initial intersected tetrahedron will be tested to find the active point tetrahedron and other active tetrahedra. At next time step, the status of all the active tetrahedra will be checked and updated. If the scalpel already passed through the tetrahedron, it will be put into the splitting tetrahedron list. And its neighboring elements will be checked for the intersection, any neighbor which intersected with the swept face will be put into the active tetrahedron list. The tetrahedra which still intersected with the cutting edge remain in the active tetrahedron list. The following is the pseudocode which implements localized intersection detection:

```
FOR (each active_tetra)
    IF (this active_tetra doesn't intersect with new scalpel edge)
        store it into split_tetra container;
        remove it from the active_tetra container;
        find the edge which intersect with the cutting face;
        find all neighboring tetra of this intersected edge;
        FOR (each of the neighboring tetra)
            IF (its face intersects with the new scalpel edge)
                save it into the active_tetra container;
            END IF;
    END IF
```

5.5 Semi-Progressive Cutting

With an ideal cutting simulation, users should be able to see the visual feedback without delay. This requires the subdividing and re-meshing be processed while the cutting is on going. As we stated before, according to this criteria, cutting simulation can be classified into two categories: non-progressive and progressive cutting. In non-progressive cutting, the model updating will not start until the whole cutting is finished.

Progressive cutting can be divided further. The first type is: once the scalpel intersects with the element, it will split by generating temporary subdivision. When the scalpel completely left the element, its subdivision will be finalized. The second type is what we used in this thesis, and we call it semi-progressive cutting. In this method, the individual element will only start splitting when the scalpel leaves it. The first method can track the tool path without any delay, but the temporary subdivision

brings more overheads to implement and increases the computational cost. Although the second method will generate a small lag between the incision and the position of the scalpel, it is more computationally efficient. Furthermore, if the mesh resolution is high, this lag is small and can be ignored.

Chapter 6

HAPTIC RENDERING

There are three kinds of haptic rendering method: point-based, ray-based and object-based method. Due to the real time requirement of the surgery simulation system, the computational cost of both ray-based and object-based methods are too high to achieve the real-time haptic update rate with current computer technology. So, the point-based haptic rendering method is used in this thesis.

6.1 Point-based Haptic Rendering

In this rendering method, the haptic device is modeled as a point, known as the haptic interface point (HIP), and only this point interacts with the virtual objects. When the user moves the haptic device and the collision detection algorithm detects that this point is inside the virtual object, the depth and indentation are calculated. Then the feedback force is computed based on the indentation and is sent back to the user. There are three main approaches used by the haptic interfaces to calculate the force in point-based haptic rendering:

- Penalty-based method

- Constraint method
- Physically based method

6.1.1 Penalty Based Method

Penalty based method is the earliest haptic rendering method. This kind of method computes the force feedback by knowing only the location of the haptic point. The history of the HIP position is not taken into account. It models surface contacts by generating a repulsive force which is proportional to the amount of penetration. That means the deeper the penetration is, the bigger the computed force will be. The force is calculated by applying a mass-spring mechanism:

$$F_{haptic} = \begin{cases} (-kx - c\dot{x})\vec{n} & \text{if } x < 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

where k and c are the rigidity and viscosity of the collision, respectively; x is the penetrated distance. The force always points to the normal direction $\vec{n}$ of the contact surface (see figure 6.1).

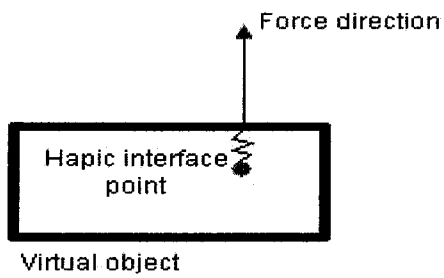


Figure 6.1: Penalty based haptic rendering

Penalty based methods behave poorly when the virtual environment contains

thin or overlapping objects. Therefore in 1995, Zilles et al. [41] first introduced the constraint based method.

6.1.2 Constraint Method

Constraint methods describe the interaction between objects based on non-penetration constraints in the virtual environment. Two points are defined in this method. One is HIP, which represents the real position of the haptic device and can penetrate into virtual objects. The other one is the virtual point, called god-object or proxy, which traces the position of HIP but will remain on the surface of the virtual object. Similar to penalty based method, the reaction force of constraint method is computed depending on the penetration. The force calculation equation is:

$$F_{haptic} = f(d) \tag{6.2}$$

where $f(d)$ is the function of the penetration d, which can be linear or nonlinear; F_{haptic} is the feedback force which is sent to the user.

God-object algorithm and proxy algorithm are the typical constraint methods. The differences between these two algorithms are:

- The god-object is a point and the proxy is a sphere. Using a sphere as a virtual representation of the haptic point avoids some rounding-off problems such as falling into small gaps between the primitives of the virtual object.
- Only proxy algorithm supports dynamic environment or deformable models

The proxy algorithm is used in our simulation.

Proxy Algorithm

Proxy algorithm was proposed by Ruspini et al. [67], it is a constraint method which can be used by a deformable object to render feedback forces. The proxy is a finite sized, massless substitution of the haptic device in the virtual environment. When the proxy contacts with a virtual object, its motion is obstructed by the object and it quickly moves to a position which minimizes its distance to the haptic interface point along the constraint surface, see figure 6.2.

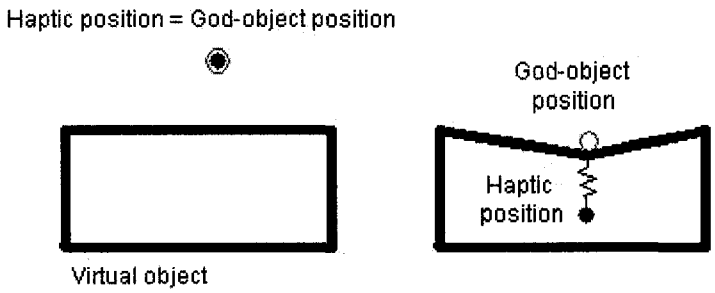


Figure 6.2: Proxy haptic rendering method

The proxy algorithm introduces some modifications in order to take into account of the moving primitives due to deformations. After the soft organ deformed, the proxy will be constrained by the deformed surface. The new proxy position is obtained by interpolation from the old proxy's position, see Figure 6.3. This method will give the users a smoothly transition between discrete low-frequency model updates so that they can get the realistic sensation of soft tissue deformation.

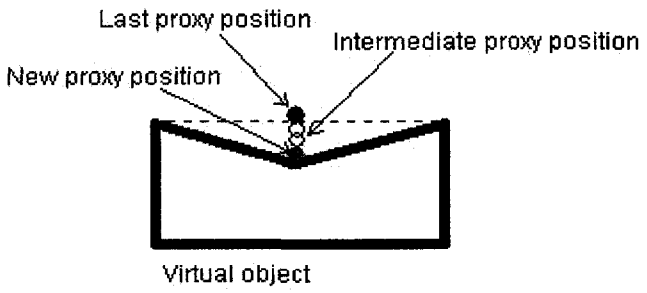


Figure 6.3: Simulate deformation by using intermediate proxy position

The proxy algorithm computes the force using a mass-spring mechanism, and takes into account the history of the haptic position. It can solve the corner ambiguity problem (see figure 6.4) which cannot be avoided by the penalty method.

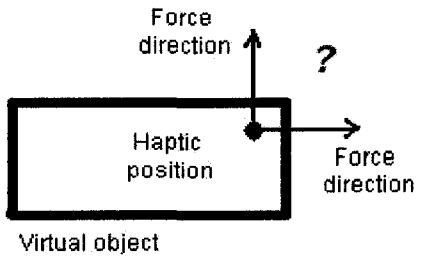


Figure 6.4: Ambiguity of the force direction

6.1.3 Physically Based Methods

Both penalty and constraint based methods compute the feedback force by using a force proportional to the penetration depth of the tool in the soft tissue model. Thus the force is only calculated from the geometric constraint instead of the physical deformation. Deformable models based on the biomechanics properties of living tissues

lead to physically meaningful forces and are very likely to provide a realistic feeling to the user. Physically based methods calculate the force according to the physical deformation, which is depending on the physical properties of the object such as superficial tension, rigidity, stresses, etc. Thus different soft tissue models have different force computing methods.

6.2 Separated External and Feedback Force Computation

Usually, the external force and feedback force are calculated using the same haptic rendering method. These two forces have equal magnitude but with opposite directions. In this thesis, two different methods are employed to compute these two forces.

6.2.1 External Force Computation

PHAMTON Premium is used as the haptic device in this thesis, therefore, OpenHaptic SDK is adopted to handle part of haptic rendering problem. OpenHaptic SDK uses proxy algorithm to implement the haptic rendering. In our simulation, after the tool-tissue contact has been detected, we use proxy algorithm to calculate the external force which is applied to the soft tissue model.

6.2.2 Feedback Force Computation

The integration of force feedback to surgery simulation is almost as important as visual feedback. With precisely computed force feedback, it is possible for the user to feel haptic sensations close to reality. On the other hand, adding force feedback to

the simulation brings additional constrains, make soft tissue modeling more difficult. Because force computation depends on the chosen deformable model, and requires soft tissue models to have high accuracy and computational efficiency.

Only since the second generation surgery simulator, can the force feedback be added to the system. Until now, how to render the faithful force to the user is still a problem far to be solved. The deformable object modeling technology hasn't been mature yet; current models cannot represent the living tissue accurately. Moreover, the feedback force is different during different simulation procedures such as palpating, cutting and suturing. It's hard to simulate those different forces properly with one of the current soft tissue model.

As we described before, the force calculation of proxy algorithm is based on mass-spring mechanism. But the soft tissue model in this simulation is a finite element model; the force computed from the proxy algorithm cannot faithfully represent the force produced by the model. In order to improve the fidelity of the feedback force, the physically based haptic rendering method is used to compute the feedback force.

6.3 Physically Based Force Feedback

The simulation loop provides us a discrete time series of parameters (t_i, P_i, F_i), which represents the force F_i applied to the tool in position P_i at time t_i. These parameters are updated with the simulation rate, i.e. around 30 Hz. The computed force cannot be sent to the haptic device directly and feedback to the user because the haptic force should be updated with a rate of 1KHz in order to make sure that the user has a realistic sensation.

In order to achieve the 1KHz force update rate, the extrapolation method is used to generate intermediate forces. This method estimates the current force by

extrapolating the previously computed force from last simulation time step.

6.3.1 Force Extrapolation

To solve the different rates problem between visual and haptic display, Ellis et al. [47] first proposed to extrapolate the force obtained from the physical simulation. They suggested using the equal time sampling to numerically predict the force and adopt a correction term to take into account the error incurred at the previous time steps. Picinbono et al. [68] introduced and compared three force extrapolation methods: constant extrapolation, time linear extrapolation and position linear extrapolation. Later they implemented these force extrapolation methods in a liver surgery simulation [69]. The linearly elastic finite element liver model has 8000 tetrahedra, and a 500Hz haptic update rate was achieved based on a 25Hz visual update rate.

As stated above, for each physical simulation step, a set of forces is computed. The time between two successive force sets is about $0.033ms$. Good quality force feedback requires 1KHz update rate, which means the time between two successive forces is $0.001ms$. Force extrapolation uses an extrapolation function $F(t)$ to calculate the estimation of the force at time $t(t_n < t < t_{n+1})$ to achieve 1KHz force update rate based on the known data $(t_i, P_i, F_i), i = 0 \ldots n$.

6.3.2 Extrapolation Methods

There are three typical force extrapolation methods which were proposed by Picinbono et al [68]: constant extrapolation, linear extrapolation over time and linear extrapolation over position.

Constant Extrapolation

Constant extrapolation uses the previous force F_i as the estimated force during time $t_i < t < t_{i+1}$. The extrapolation function is as follow:

$$F_{est}(t) = F_i \qquad t_i \leq t \leq t_{i+1}, i = 0 \ldots n \tag{6.3}$$

This is the easiest way to produce the estimated force. In addition, all applied forces are valid because they come from the physical simulation. The disadvantage of this method is that the obtained force is discontinuous (see figure 6.5).

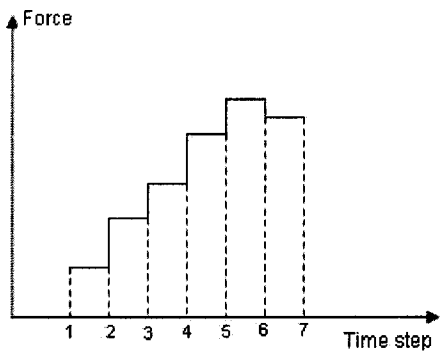


Figure 6.5: Constant force extrapolation

Linear Extrapolation over Time

This method estimates the current force value by extrapolating over time. Let's define F_{i-1}, F_i as previously obtained force from physical simulation at two previous time step t_{i-1} and t_i, respectively. Now, we want to estimate the force at time t during the current simulation time step, i.e. $t_i < t < t_{i+1}$. We can get the following extrapolation

function based on linear extrapolating over time:

$$F^t_{est}(t) = F_i + \frac{t - t_i}{t_i - t_{i-1}}(F_i - F_{i-1}) \qquad t_i \leq t < t_{i+1}, i = 0 \ldots n \tag{6.4}$$

Figure 6.6 shows the extrapolating.

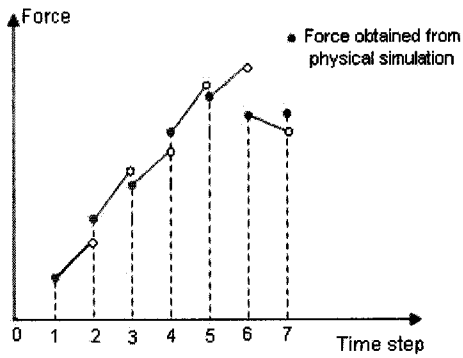


Figure 6.6: Force linear extrapolation over time

Linear Extrapolation over Position

In surgery simulation, the tool position determines the deformation or model position, and then the model position determines the force applied to the tool. Since the haptic loop detects the haptic device position with a rate of 1KHz, therefore, we can query the tool position with a frequency of 1KHz. So it is more reasonable to estimate the force over tool position.

Because the tool position is a vector, we need to project it on the line defined by two previous positions P_i and P_{i-1} (see figure 6.7). Then we can write the linear extrapolation function as follow:

$$F^p_{est}(t) = F_i + \frac{\|P_{proj} - P_i\|}{\|P_i - P_{i-1}\|}(F_i - F_{i-1}) \qquad t_i \leq t < t_{i+1}, i = 0 \ldots n \tag{6.5}$$

From the above equation, we know when P_{i-1}, P_i and P are in a line, the position projection will not introduce extrapolation errors.

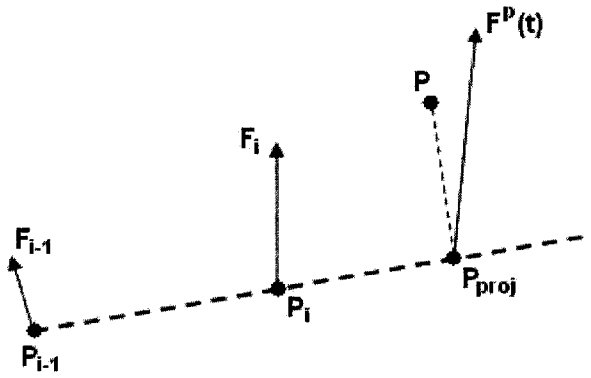


Figure 6.7: Force linear extrapolation over tool position

Chapter 7

IMPLEMENTATION AND RESULTS

Even though the general theories of the surgery simulation system are well understood, implementing a surgery simulator is still full of challenges. As the developer need to combine several different technologies, such as multi-threaded programming, computer graphics, numerical methods, haptic devices and their developing environments, together. Many of the implementation details directly affect the performance and quality of the system. In this chapter, some of them will be introduced and the experimental results will be presented.

7.1 Environment of the System

The simulation system was implemented on a PC with a Pentium IV 3.60 GHz processor by using programming language C++. The haptic device, which connects to the PC through a parallel port, is PHANTOM Premium 1.5 from SensAble Technologies Inc. It has six degree of freedoms (DOF) for force input and three DOFs for force

feedback. The source code was developed with MS Visual C++ 6.0, using OpenGL library for graphic rendering and OpenHaptics SDK (from SensAble Technology Inc.) for haptic rendering.

7.2 Implementation of Soft Tissue Model

Most of the object models in virtual environment and 3D graphics are surface models since showing the interior of the objects usually is not required. Thus it is easy to find the surface geometric models of some soft tissues/organs such as heart, liver and stomach, but the volumetric models are hard to obtain. Therefore we must construct a volumetric model first.

7.2.1 Constructing a Volumetric Soft Tissue Model

There are many mesh generators both in public and commercial domains [70]. We selected several tetrahedron mesh generators from public domain: DistMesh, Geompack++, LBIE-Mesher and GiD. After comparing their functionalities, we chose GiD as our tetrahedron mesh generator.

Giving boundary and volume, GiD can generate a consistent tetrahedron mesh. The mesh data can be exported as a .msh file. Besides tetrahedron information, the triangle mesh of the selected surface can be easily obtained.

By using some commercial software, for example Simal, realistic soft tissue/organ models can be generated from real medical data, such as magnetic resonance imaging (MRI) and computer tomography (CT) images. Due to lack of real data and the proper software, currently in our simulation, a hexahedron shaped model is used to represent the soft tissue. This mesh is composed of 50 nodes and 100 tetrahedra.

7.2.2 Tissue Parameters

Tissue parameters greatly affect the realism of soft tissue simulation. Unfortunately, due to the complexity of human tissues, currently it is very hard to get the accurate experimental data about the parameters of different tissues/organs. So we aimed to find tissue parameters that were roughly correct and appeared appropriate.

In our tensor mass model, the elastic force at a node is calculated by adding together all elastic force contributions from the adjacent tetrahedra (see equation 4.6). Thus for each tetrahedron, we need to calculate its four node tensors and six edge tensors first by using equation 4.7. This equation shows that only Lamé coefficients λ and μ are needed besides the rest position of the tetrahedron. These two coefficients can be calculated from two continuum mechanics parameters of the material: Young's modulus and Poisson's ratio, by using equation 7.1 and equation 7.2. In addition, we need to know tissue density to compute the node mass. Finally to construct the governing equation of the system, the damping parameters are also needed.

$$\lambda = \frac{\nu E}{(1-2\nu)(1+\nu)} \tag{7.1}$$

$$\mu = \frac{E}{2(1+\nu)} \tag{7.2}$$

where E is the Young's modulus and ν is the Poisson's ratio.

We adopted human liver's parameters as our model's parameters. According to the study and experimental results of [71], [72] and [73], we chose the Young's modulus as $E = 642.55 Pasca$ and the Poisson's ratio as 0.47 in our soft tissue model. By using above equations 7.1 and 7.2, the Lamé coefficients of the tissue model can be computed. The same value will be used for the whole model.

We used uniform tissue density and set the value as $1.05 g/cm^3$, which was ob-

tained from [74]. As for the damping parameters, we used the same value as that suggested by Mor [28]. A value of 1e-5 $1/sec$ is used for α, and 2e-5 $m * sec$ is used for β. The damping coefficient can be calculated from the following equation:

$$[C] = \alpha[M] + \beta[K] \tag{7.3}$$

where $[M]$ and $[K]$ are the mass matrix and stiffness matrix, respectively.

7.3 Data Structure Design

The data structure of a volumetric finite element model is very complex. There are two sets of data need to be stored and manipulated. One set represents the geometric information of the model, such as 4 vertices, 6 edges and 4 triangle faces of a tetrahedron; tetrahedral neighbors by the vertex and tetrahedral neighbors by the edge. The other set represents the physical information of the model, for example the mass of node, the node and edge tensors of the element. These two sets of data should be easily referenced by each other.

In order to implement real-time cutting simulation, changing the state of the model should be finished on-the-fly. To help facilitate the modification of the model, we need a memory structure that would be easily and quickly updated, for both adding and removing elements to the model. Another character of our data is that they are dynamically increased. Therefore, we cannot predict how many elements the model will have after the cutting occurs.

There are two types of structures which are suitable to store unknown number of elements in C++: the vector and the linked list. We will compare these two structures as follow:

- Vector: A vector represents a contiguous area of memory in which each element is stored in turn. Random access to a vector, which means accessing to any one of the elements, is very efficient. Insertion and deletion of an element at any position other than the back of the vector are inefficient.

- Linked list: A list represents noncontiguous memory doubly linked through a pair of pointers that address the elements to the front and back allowing for both forward and backward traversal. Insertion or deletion of elements at any position within the list is efficient. But random access, on the other hand, is not well supported.

The most important requirement to the data structure used in our simulation system is random access to any element. Random deletion is desired too, since we need to remove the original tetrahedron when it has been subdivided. But neither vector nor linked list can meet these two requirements in the same time.

We use vector containers to store our objects such as nodes, triangles, and tetrahedra. We can avoid deletion in the middle of the container by replacing the subdivided tetrahedron with a newly created tetrahedron. The main components of data structure in our simulation system are shown in Figure 7.1.

FEM_Object represents the soft tissue model, which is in the highest position of our object hierarchy. It includes three categories of data:

- Geometric data of the soft tissue model, which are stored in the TetVolume object. FEM_Object contains a pointer to this object.

- Physical data of the soft tissue model, which are stored in the vector container of TetFEMElement. FEM_Object also has a pointer to this container.

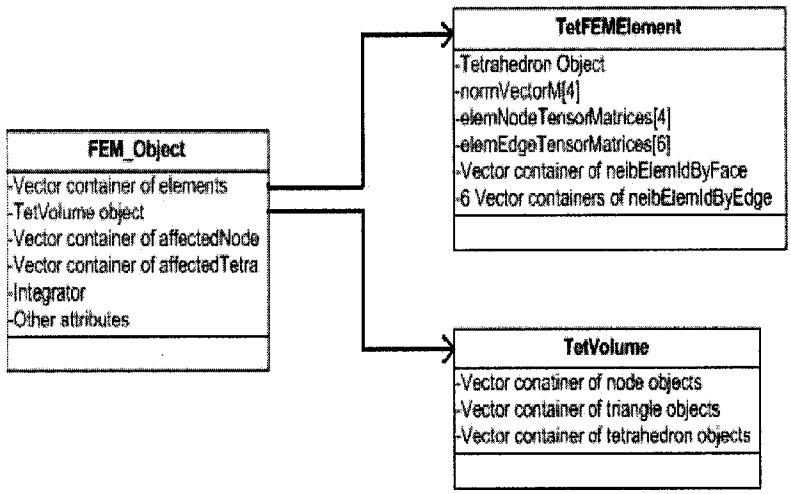


Figure 7.1: Data structure at simulation object level

- Other data related to the system simulation, such as affectedNode, affectedTetra and integrator, ect.

Geometrically, the soft tissue model is composed of node, edge, triangle and tetrahedron. Figure 7.2 shows the structure of those objects.

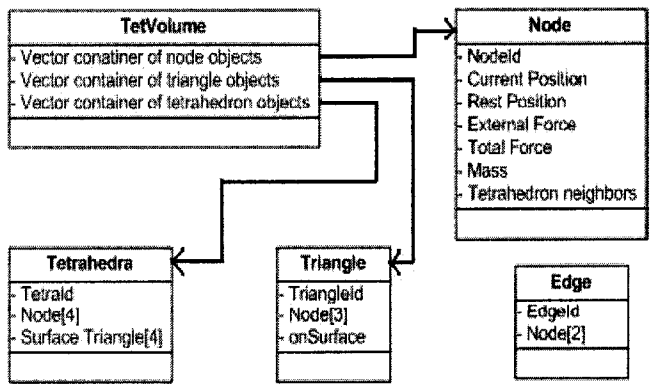


Figure 7.2: Geometric data structure of the tissue model

Node is the most basic data structure. It not only contains the current and rest positions, but also contains the physical information such as force and mass. Tetrahedron, triangle and edge objects all contain the pointers to the nodes which they have. TetFEMElement object can obtain the node information through the pointer to corresponding tetrahedron.

7.4 Implementation of Cutting

Due to the complexity of topology modification and system reconstruction, volumetric cutting simulation is very hard to implement. In Chapter 5, the details of our cutting scheme had been introduced. Here we only briefly describe the major functionality of the most important three procedures in the simulation: AdvanceSimulation(), geoMeshReconstruct() and reconstructFEMObject().

7.4.1 Advance the Simulation

When the simulation starts, the simulation loop will do three types of initialization. The first is to setup the OpenGL graphic environment; the second is to initialize the haptic device and start the haptic thread; the third is to load the geometric soft tissue model and construct the FEM_Oject (physical model) system. During the last step, the physical data of the soft tissue model will be built and stored, which means the node masses, node and edge tensors will be calculated, and the neighbor elements of the node, edge and face also will be found and stored. After these initialization, the AdvanceSimulation() function will be called each time the simulation loop advance the system time and simulation will keep going. Following is the pseudocode of function AdvanceSimulation().

```
AdvanceSimulation()

 IF (cutting == true)
     IF (active_tetra_list == empty)
         find the initial contact surface tetrahedron;
         find initial active_tetra_list;
     ELSE
         update active_tetra_list;
     IF (non-active_tetra_list_empty == false)
         call function geoMeshReconstruct(non-active_tetra_list);
         call function reconstructFEMObject();
     compute the deformation and force;
     update geometric model with new position;
 ELSE
     IF (non-active_tetra_list_empty == false)
         call function geoMeshReconstruct(non-active_tetra_list);
         call function reconstructFEMObject();
         non-active_tetra_list_empty = true;
     compute the deformation and force;
     update geometric model with new position;
```

7.4.2 Reconstructing Geometric Mesh

This procedure reconstructs the geometric mesh of the soft tissue model. Which includes following steps:

- subdivide or separate each non-active tetrahedron based on the intersection type (case A to H) it belongs to; add newly created nodes, edges and faces into their

container and the lookup tables; replace the old tetrahedron by a set of newly created tetrahedra.

- find all uncut neighboring tetrahedra of this non-active tetrahedron, update their information about edge and face neighboring tetrahedra
- construct affacted TetFEMElement list.
- construct affacted node list.

Following is the pseudocode for this procedure:

```
geoMeshReconstruct()
 FOR (each non-active tetra)
     find all uncut neighbor tetrahedra of this non-active tetra;
     FOR (each of uncut neighbor tetrahedron)
         update the information of face neighbor;
         update the information of edge neighbor;
     split this tetrahedron based on the intersection case;
     construct affacted tetra element list;
     construct affacted node list;
     FOR (each affacted node);
         update the neighbor information of the node
```

7.4.3 Reconstructing FEM System

After geometric mesh modification, the FEM system is reconstructed by calling function reconstructFEMObject(). This function implements following tasks:

- Update physical parameters of each affacted TetFEMElement, which include: the volume, 4 node tensors and 6 edge tensors.

- Update the node mass and node tensor sum of each affacted node

Following is the pseudocode for this procedure:

```
reconstructFEMObject()

  get all affacted TetFEMElements;
  FOR (each affacted element)
       update the volume of the element;
       update its 4 node tensors;
       update its 6 edge tensors;
  get all affacted nodes;
  FOR (each affacted node);
       update node mass;
       update the sum of node tensors from all neighbor elements
     at this node
```

7.5 Palpation Simulation Result

With the volumetric soft tissue model described in 7.2.1, we can perform the real time palpating simulation. When the virtual scalpel touches the model, it starts to deform. Because palpation doesn't change the topology of the model, all the physical parameters of the soft tissue model, such as the node masses, node tensors and edge tensors, will keep the same. With the advance of the simulation, the integration method will be called continuously and the node positions and forces will be updated repeatedly. The deformation becomes bigger while the palpating force increases but is still within the cutting threshold. Figure 7.3 shows the result.

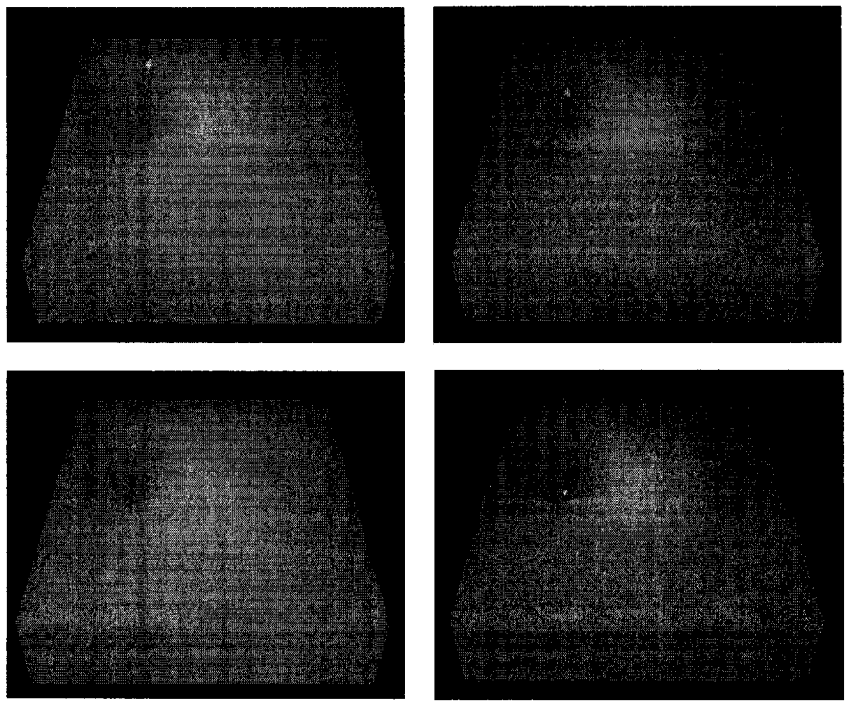


Figure 7.3: Deformed soft tissue model

7.6 Cutting Simulation Result

When the system detects that the external force exceeds the cutting threshold, the cutting simulation starts. We implemented the cutting simulation using two different cutting approaches: element deletion method and combined subdivision/separation method. The two simulations perform on different hexahedron shaped soft tissue models. The mesh resolution used for deletion cutting is two times of the mesh resolution used for implementing our cutting method.

7.6.1 Cutting Result by Element Deletion Method

The cutting simulation by element deletion method uses a hexahedron shaped model with 85 nodes and 258 tetrahedra. The average mesh resolution is 0.5 cm. Figure 7.4

shows the cutting results by element deletion. The experimental results show that

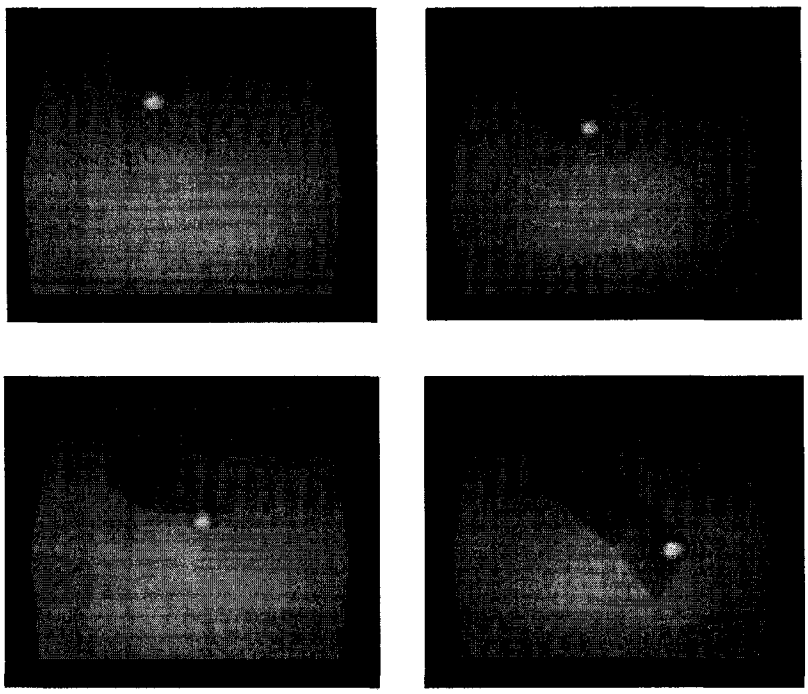


Figure 7.4: Cutting result with element deletion method

during cutting, all the tetrahedron elements which contact with the cutting tool will be deleted. There is no lag between the incision and the tool. With the cutting tool moving on the model, a big gap with the size of mesh resolution will be displayed as the cutting incision.

7.6.2 Cutting Result by Combined Subdivision and Separation Method

The cutting simulation with combined subdivision/separation method uses a hexahedron shaped model with 50 nodes and 100 tetrahedra. The average mesh resolution is 1.0 cm. Figure 7.5 shows the result.

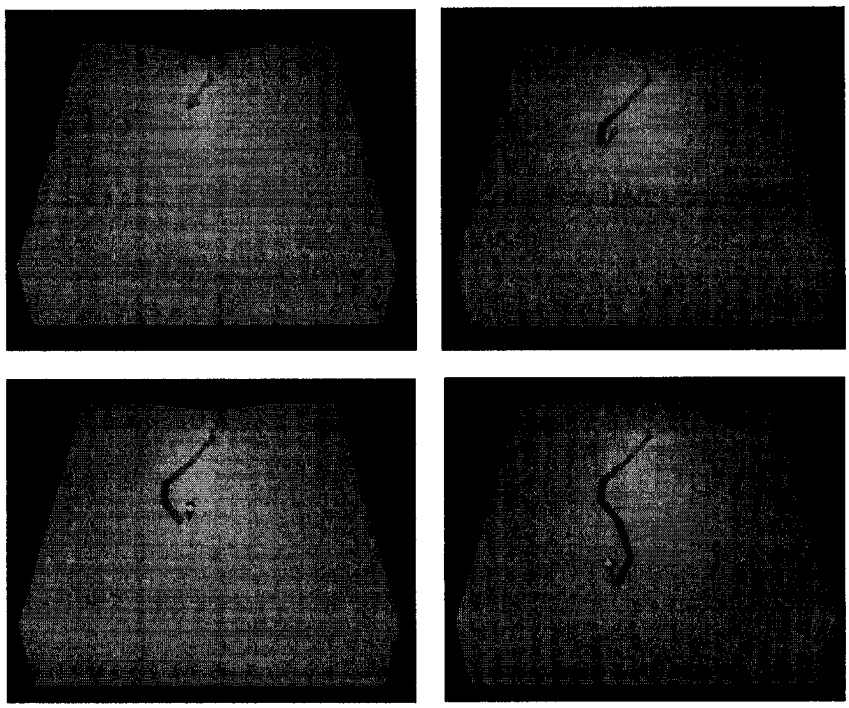


Figure 7.5: Cutting result by combined subdivision and separation method

From the above result, we see, although the mesh resolution is one time higher than the mesh used in our method, the cutting by element deletion still creates a big gap. While our cutting algorithm can trace the tool path well and has a realistic visual feedback. The calculation time for an entire loop is 0.032 ms, that means the cutting simulation has a real time update rate of around 30Hz.

In addition, with the force extrapolation, we get a physically based force feedback with an update rate around 1KHz. The figures also show that there is a small lag between the scalpel and the cutting incision, because the combined subdivision/separation method is a semi-progressive cutting scheme.

Chapter 8

SUMMARY

The goal of this thesis was to develop a surgery simulation system which can simulate some common surgical procedures. In particular, we aimed to simulate soft tissue cutting and tried to improve its realism while maintaining real-time interactions. In order to achieve this goal, first, a volumetric finite element soft tissue model was created; second, a novel combined subdivision/separation cutting method was developed; third, a realistic force, which was calculated based on the physical deformation, was sent back to the user. This force feedback allows the user to have the sensations of touching and cutting the virtual soft tissue.

To build the soft tissue model, first the existing deformable object models were investigated. After compared the advantages and disadvantages of those models, the tensor mass model was chosen as our soft tissue model because it has finite element model's accuracy and mass spring model's computational efficiency. More important, the tensor mass model supports topology modification well. Then the explicit fourth-order Runge-Kutta integration method was used to solve the motion equations of the model. This method has high accuracy. Although the computational cost of each time step is higher than Euler method, RK-4 method can handle much larger time

steps comparing to Euler methods. Therefore, the overall computational efficiency and stability of RK-4 method is higher than that of Euler method.

After the construction of the soft tissue model, more efforts were put on the cutting simulation. A new combined subdivision/separation method was proposed to simulate the soft tissue cutting. This volumetric cutting approach well balances the trade-off between simulation realism and computational efficiency. On one hand, the element subdivision and separation combined method ensures the incision following the arbitrary cut path and showing realistic visual feedback. On the other hand, the localized intersection detection and the efficient topology modification method, all contributes to the improvement of the computational efficiency. In addition, the proposed midpoint minimal set subdivision greatly increases the stability of the simulation system.

Faithful force feedback is another way to improve the realism of surgery simulation. In this thesis, we separated the feedback force computation from the traditional haptic rendering method. The proxy algorithm was used to compute the external force, while a physically based method was used to compute the feedback force. The force obtained from physical simulation was extrapolated based on the known haptic positions in order to achieve a force update rate of 1KHz.

The experimental results showed that the system can simulate soft tissue palpating and cutting with high realism in a real-time rate. The visual feedback is realistic and the haptic force feedback is faithful and smooth.

8.1 Contributions

The main contributions of this thesis are:

1. A comprehensive analysis and comparison of the existing physical deformable models which can be used for soft tissue modeling in surgery simulation was presented.

2. A new force distribution method was proposed. It re-distributes the force exerted on any arbitrary point inside a surface triangle of the model to the triangle's three nodes.

3. A new volumetric finite element cutting scheme was proposed and implemented.

   - A new combined subdivision/separation method was developed to address the topology modification of the soft tissue model. This method decreases the number of newly created elements and can handle the situation of cutting through the mesh nodes.
   - A midpoint subdivision method was applied to the minimal element sets. This method makes the element subdivision easier and more efficient. In addition, it ensures that there are no very small or badly shaped elements generated such that the stability of the simulation system gets increased.
   - A localized intersection detection method was proposed and used in the simulation. It can detect all intersected elements with high efficiency.
   - The state and lookup tables were used in the cutting algorithm, which facilitate the topology modification and ensure the mesh be consistent.

4. A physically based haptic rendering method was used to send the faithful feedback force to the user. The force computed from the physical simulation was extrapolated over the known tool positions to achieve the 1KHz force updating rate.

5. An efficient data structure was designed and used to handle the complicated tetrahedral mesh data both geometrically and physically.

8.2 Future Work

We had developed and implemented a surgery simulation system which can simulate surgical palpating and cutting with improved realism in real-time. However, there are some areas within this research and the experimental simulator which can be explored further.

1. This thesis used a linearly elastic soft tissue model which is only valid when the deformation is small. The soft tissue can be more accurately modeled, for example, some other living tissue characteristics such as non-linear elasticity, anistroscopics can be included into the soft tissue model in the future. Some other efficient explicit or implicit numerical integration methods can be studied to replace the Fourth-order Runge-Kutta method used by this thesis.

2. In the experiment, the current hexahedron shaped model can be replaced by a real soft organ model such as the liver or the heart model. How to generate the volumetric organ mesh from real medical data need further investigation.

3. A good collision detection algorithm is very important to the surgery simulation system. More research effort can be put into this part to develop and apply more efficient 3D object collision detection algorithms.

4. How to model the force feedback during cutting is still an open topic, more real life experimental data are needed to support this research.

Appendix A

GLOSSARY OF TERMS

TriS — Surgery Simulation System

MIS — Minimally Invasive Surgery

SSS — Surgery Simulation System

MSM — Mass-Spring Model

FEM — Finite Element Model

BEM — Boundary Element Model

FSM — Finite Sphere Model

MRI — Magnetic Resonance Imaging

CT — Computer Tomography

VR — Virtual Reality

API — Application Programming Interface

RAM — Random Access Memory

TMM — Tensor-Mass Model

FE — Finite Element

PCMFS — Point Collocation-based Method of Finite Sphere

LOD — Level of Detail

ODE — Ordinary Differential Equation

RK-4 — Fourth-order Runge-Kutta

HIP — Haptic interface point

DOF — Degree Of Freedom

SDK — Software Development Kit

OpenGL — Open Graphics Library

Bibliography

[1] H. Delingette and N.Ayache. Hepatic surgery simulation. *Communications of the ACM*, 48(2):31–36, Feb 2005.

[2] H. Delingette and N. Ayache. Soft tissue modeling for surgery simulation. URL http://citeseer.ist.psu.edu/662892.html.

[3] K. Montgomery, C. Bruyns, J. Brown, S. Sorkin, F. Mazzela, G. Thonier, A. Tellier, B. Lerman, and A. Menon. Spring: A general framework for collaborative, realtime surgical simulation. In *Medicine Meets Virtual Reality (MMVR)*, pages 23–26, 2002.

[4] M.C. Cavusoglu, T.G. Goktekin, F. Tendick, and S. Sastry. Gipsi: An open source/open architecture software development framework for surgical simulation. In *Medicine Meets Virtual Reality (MMVR)*, pages 46–48, January 2004.

[5] J. Allard, S. Cotin, F. Faure, P. J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni. Sofa - an open source framework for medical simulation. In *Medicine Meets Virtual Reality (MMVR)*, 2007.

[6] C. Basdogan, S. De, J. Kim, M. Muniyandi, and H. Kim amd M.A. Srinivasan. Haptics in minimally invasive surgical simulation and training. *Computer Graphics and Applications, IEEE*, 24:56–64, March-April 2004.

[7] D. Terzopoulos, J. Platt, A. Barr, and K. Fleisher. Elastically deformable models. *Computer Graphics (SIGGRAPH'87)*, 21(4):205–214, July 1987.

[8] U. Kühnapfel, C. Kuhn, H.G. Krumm, and B. Neisius. Cad-based simulation and modeling for endoscopic surgery. In *Proc. Med. Tech.*, Berlin, 1994.

[9] E. Keeve, S. Girod, and B. Girod. Cranofacial surgery simulation. In *Proceedings of the Visualization in Biomedical Computing, Lecture Notes in Computer Science*, pages 541–546, Berlin, 1996. Springer.

[10] M.C. Cavusoglu, T.G. Goktekin, F. Tendick, and S. Sastry. Towards realistic soft-tissue modeling in medical simulation. In *Proceedings of IEEE*, volume 86, pages 512–523, March 1998.

[11] A.Q. Frank, I.A. Twombly, T.J. Barth, and J.D. Smith. Finite element methods for real-time haptic feedback of soft-tissue models in virtual reality simulators. In *Proceedings IEEE Virtual Reality 2001*, pages 257–263, 2001.

[12] S.F. Frisken-Gibson. Using linked volumes to model object collisions, deformation, cutting, carving, and joining. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):333–348, October-December 1999.

[13] S. Gibson, J. Samosky, A. Mor, C. Fyock, E. Grimson, T. Kanade, R. Kikinis, H. Lauer, N. McKenzie, S. Nakajima, H. Ohkami, R. Osborne, and A. Sawada. Simulating arthroscopic knee surgery using volumetric object representations, real-time volume rendering and haptic feedback. *Medical Image Analysis*, 2(2):121–132, 1998.

[14] W. Mollemans, F. Schutyser, J. Van Cleynenbreugel, and P. Suetens. Tetrahedral mass spring model for fast soft tissue deformation. In *International Symposium*

of Surgery Simulation and Soft Tissue Modeling (IS4TM 2003), volume 2673 of *Lecture Notes in Comput. Sci.*, pages 145–154. Springer, 2003.

[15] Q.H. Zhu, Y. Chen, and A. Kaufman. Real-time biomechanically-based muscle volume deformation using fem. In *Computer Graphics Forum*, volume 17, pages C275–84, C387–9, 1998.

[16] S.A. Cover, N.F. Ezquerra, J.F. O'Brien, R. Rowe, T. Gadacz, and E. Palm. Interactively deformable models for surgery simulation. *IEEE Comput. Graphics Applicat. Mag.*, pages 68–75, November 1993.

[17] K. Water. A physical model of facial tissue and muscle articulation derived from computer tomography data. In *Proc. Visualization in Biomedical Computing (VBC'92)*, volume 574, pages 574–584. Chapel Hill, 1992.

[18] H. Delingette, G. Subsol, S. Cotin, and J. Pignon. A craniofacial surgery simulation testbed. In *Proc. Visualization in Biomedical Computing (VBC'94)*, pages 607–618, Rochester, NY, October 1994.

[19] J. Zhang, S. Payandeh, and J. Dill. Haptic subdivision: an aproach to defining level-of-detail in haptic rendering. In *Proceedings of the 10th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'02)*, pages 201–208, 2002.

[20] J. Brown, S. Sorkin, J.-C. Latombe, K. Montgomery, and M. Stephanides. Algorithm tool for real time microsurgery simulation. *Medical Image Analysis*, 6(3):289–300, 2002.

[21] S.F.F. Gibson. 3d chainmail: a fast algorithm for deforming volumetric objects. In *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 149–154, 1997.

[22] H. Kardestuncer. *Finite Element Handbook.* McGraw-Hill, New York, 1987.

[23] M. Bro-Nielsen. Finite element modeling in surgery simulation. In *Proceedings of the IEEE: Special Issue on Surgery Simulation*, volume 86 of *3*, pages 490–503, March 1998.

[24] W. Wu and P.A. Heng. An improved scheme of an interactive finite element model for 3d soft-tissue cutting and deformation. *The Visual Computer*, 16:437–452, 2005.

[25] Morten Bro-Nielsen. *Medicl Image Registration and Surgery Simulation.* PhD thesis, IMM Technical University of Denmark, Lingby, Denmark, March 1996.

[26] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Computer Graphics Forum*, volume 15 of *3*, pages C57–66, 1996.

[27] S. Cotin, H. Delingette, and N. Ayache. A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. *The Visual Computer*, 16:437–452, 2000.

[28] Andrew B. Mor. *Progressive Cutting with Minimal New Element Creation of Soft Tissue Models for Interactice Surgical Simulation.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, October 2001.

[29] C. Mendoza and C. Laugier. Simulating soft tissue cutting using finite element models. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, volume 1, pages 1109–1114, Taipei, Taiwan, September 2003.

[30] G. Picinbono, H. Delingette, and N. Ayache. Non-linear and anisotropic elastic soft tissue models for medical simulation. In *Proceedings of the 2001 IEEE*

International Conference on Robotics and Automation, volume 2, pages 1370–1375, May 2001.

[31] P.K. Banerjee. *The boundary element methods in engineering.* McGraw-Hill, New York, 1994.

[32] C. Monserrat, U. Meier, F. Chinesta, M. Alcaniz, and V. Grau. A fast real time deformation algorithm for surgery simulation. In *Proceedings of the Computer Aided Radiology and Surgery*, pages 812–817, 1998.

[33] C. Monserrat, U. Meier, M. Alcaniz, F. Chinesta, and M.C. Juan. A new approach for the real-time simulation of tissue deformation in surgery simulation. *Computer Method and Programs in Biomedicine*, 64:77–85, 2001.

[34] P. Wang, A.A. Becker, I.A. Jones, A.T. Glover, S.D. Benford, C.M. Greenhalgh, and M. Vloeberghs. Virtual reality simulation of surgery with haptic feedback based on the boundary element method. *Computers and Structures*, 85:331–339, April 2007.

[35] S. De and K.J. Bathe. The method of finite spheres. *Computat. Mech.*, 25:329–345, 2000.

[36] J. Kim, S. De, and M.A. Mandayam. Computational efficient techniques for real time surgical simulation with force feedback. In *Proceedings of the 10th International Symp. On Haptic Interfaces For Virtual Envir. & Teleoperator Systs. (HAPTICS'02)*, pages 51–57, 2002.

[37] S. De, J. Kim, and M.A. Mandayam. A meshless numerical technique for physically based real time medical simulation. In *Proceedings of MMMR '9 Conference*, pages 113–118, 2001.

[38] S. De, J. Kim, Y.-J. Lim, and M.A. Mandayam. The point collocation-based method of finite spheres (pcmfs) for real time surgeru simulation. *Computer and Structures*, 83:1515–1525, 2005.

[39] J. Kim, S. De, and M.A. Mandayam. On the use of meshfree methods and a geometry based surgical cutting algorithm in multimodal medical simulations. In *Proceedings of the 12th International Symp. On Haptic Interfaces For Virtual Envir. & Teleoperator Systs. (HAPTICS'04)*, pages 295–301, 2004.

[40] K. Salisbury, F. Conti, and F. Barbagli. Haptic rendering: introductory concepts. *IEEE Computer Graphics and Applications*, 24:24–32, 2004.

[41] C.B. Zilles and J.K. Salisbury. Constraint-based god-object method for haptic display. In *IEEE International Conference on Intelligent Robots and Systems*, volume 3, pages 146–151, 1995.

[42] D.C. Ruspini, K. Kolarov, and O. Khatib. The haptic display of complex graphical environment. In *SIGGRAPH 97 conference proceedings*, volume 1, pages 295–301, August 1997.

[43] Y. Zhuang and J. Canny. Haptic interaction with global deformations. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 2428–2433, 2000.

[44] U. Kühnapfel, H. Akmak, and H. Maa. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers and Graphics*, 24:671–682, 2000.

[45] C. Dasdogan, S. De, J. Kim, M. Nuniyandi, H. Kim, and M.A. Sirnivasan. Haptics in minimally invasive surgical simulation and training. *IEEE Computer Graphics and Applicaitons*, 24(2):56–64, March-April 2004.

[46] W. Mark, S. Randolph, M. Finch, J. Van Verth, and R. Taylor. Adding force feedback to graphics systems: Issues and solutions. In *Computer Graphics Proc., Annual Conference Series, ACM SIGGRAPH*, pages 447–452, New Orleans, Lousiana, August 1996.

[47] E. Ellis, N. Sarkar, and M. Jenkins. Numerical methods for the force reflection of contact. *ASME Transactions of Dynamic Systems Measurement and Control*, 119:768–774, 1997.

[48] R. Adams and B. Hannaford. Stable haptic interface with virtual environments. *IEEE Transactions on Robotics and Automation*, 15(3):465–474, June 1999.

[49] C.A. Mendoza and C. Laugier. Realistic haptic rendering for highly deformable virtual objects. In *Proceedings IEEE Virtual Reality 2001*, pages 264–269, 2001.

[50] D. Wang, Y.Zhang, Y.H. Wang, Y.S. Lee, P. Lu, and Y. Wang. Cutting on triangle mesh: local model-based haptic display for dental preparation surgery simulation. *IEEE Transactions on visualization and computer graphics*, 11(6):671–682, November-December 2005.

[51] G. Song and N. Reddy. Tissue cutting in virtual environments. In *Proceedings of Medicine Meets Virtual Reality (MMVR'3) Conference*, pages 359–364, San Diego, CA, 1995.

[52] K.D. Reinig, C.G. Rush, H.L. Pelster, V.M. Spitzer, and J.A. Heath. Realtimevisually and haptically accurate surgical simulation. In *Proceedings of*

Medicine Meets Virtual Reality (MMVR'4) Conference, pages 542–545, San Diego, CA, 1995.

[53] C.A. Mendoza, C. Laugier, and F. Boux de Casson. Virtual reality cutting phenomena using force feedback for surgery simulations. In *Proceedings of Interactive Medical Image Visualization and Analysis, MICCAI*, Holland, 2001.

[54] C. Basdogan, C. Ho, and M.A. Srinivasan. Simulation of tissue cutting and bleeding for laparoscopic surgery using auxiliary surface. In *Proceedings of Medicine Meets Virtual Reality (MMVR'7) Conference*, pages 38–44, 1999.

[55] Han-Wen Nienhuys and A. Frank van der Stappen. A delaunay approach to interactive cutting in triangulated surfaces. In *Proc. 5th Int. Worksh. Algorithmic Foundations of Robotics (WAFR 2002)*, pages 113–130, 2002.

[56] H. Zhang, S. Payandeh, and J. Dill. On cutting and dissection of virtual deformable objects. In *Proceedings of IEEE International Conference on Robotics and Automation (ICR'04)*, volume 4, pages 3908–3913, April 2004.

[57] C. Forest, H. Delingette, and N. Ayache. Cutting simulation of manifold volumetric meshes. In *Medical Image Computing and Computer-Assisted Intervention-MICCAI'2002. 5th International Conference. Proceedings*, pages 235–244, 2002.

[58] D. Bielser, V. Maiwald, and M.H. Gross. Interactive cuts through 3-dimensional soft tissue. In *Proceedings of the Eurographics'99, Computer Graphics Forum*, volume 18 of *3*, pages C31–C38, 1999.

[59] D. Bielser and M.H. Gross. Interactive simulation of surgical cuts. In *Proceedings of Pacific Graphics, IEEE Computer Society*, pages 116–125, 2000.

[60] D. Bielser, P. Glardon, M. Teschner, and M. Gross. A state machine for real-time cutting of tetrahedral meshes. In *Proceedings of 11th Pacific Conference on Computer Graphics and Applications*, pages 377–386, October 2003.

[61] F. Ganovelli, P. Cignoni, C. Montani, and R. Scopigno. A multiresolution model for soft objects supporting interactive cuts and laceration. In *Proceedings of the Eurographics 2000, Computer Graphics Forum*, volume 19 of *3*, pages C271–C281, 2000.

[62] Water Maurel. *Biomechanical models for soft tissue simulation.* Springer-Verlag, New York, 1998.

[63] Gregory M. Nielson, Hans Hagen, and Heinrich Müller. *Scientific Visualization: Overviews, Methoddologies, and Techniques.* Matt Loeb, Cambridge, UK, 1997.

[64] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing.* Cambridge University Press, Cambridge, UK, 2002.

[65] P.F. Neumann, L.L. Sadler, and Jon Gieser M.D. Virtual reality vitrectomy simulator. In *Medical Image Computing and Computer-Assisted Intervention-MICCAI'98. First International Conference. Proceedings*, pages 910–917, 1998.

[66] F.B. de Casson and C. Laugier. Modelling the dynamics of a human liver for a minimally invasive surgery simulator. In *Medical Image Computing and Computer-Assisted Intervention-MICCAI'99. Second International Conference. Proceedings*, pages 1156–1165, 1999.

[67] D. Ruspini and O. Khatib. Dynamic model for haptic rendering systems. *Advances in Robot Kinematics*, pages 523–532, June 1998.

[68] G. Picinbono and J-C. Lombardo. Extrapolation: a solution for force feedback? In *International Scientific Workshop on Virtual Reality and Prototyping*, pages 117–125, Laval, France, June 1999.

[69] G. Picinbono, J-C. Lombardo, H. Delingette, and N. Ayache. Improving realism of a surgery simulator: linear anisotropic elasticity, complex interactions and force extrapolation. *The Journal of Visualization And Computer Animation*, pages 147–167, 1998.

[70] Robert Schneiders. Mesh generation software. URL http://www-users.informatik.rwth-aachen.de/roberts/software.html.

[71] E.J. Chen, J. Norvokovski, W.K. Jenkins, and W.D. OBrien Jr. Young's modulus measurements of soft tissue with application to elasticity imaging. *IEEE Trans. Ultrason. Ferroelect. Freq. Control*, pages 191–194, January 1996.

[72] W.C. Yeh, Y.M. Jeng, H.C. Hsu, P.L. Kuo, M.L. Li, P.M. Yang, P.H. Lee, and P.C. Li. Young's modulus measurements of human liver and correlation with pathological findings. In *Ultrasonics Symposium, 2001 IEEE*, volume 2, pages 1233–1236, October 2001.

[73] H.M. Yin, L.Z. Sun, G. Wang, and M.W. Vannier. Modeling of elastic modulus evolutioin of cirrhotic human liver. *Biomedical Engineering, IEEE Transactions on*, pages 1854–1857, October 2004.

[74] H. Gray. Anatomy of the human body. URL http://www.bartleby.com/107/, 2001.