FUNDAMENTALS OF SUTURING SIMULATION IN SURGICAL TRAINING ENVIRONMENT

by

Hans Fuhan Shi
B.Sc, Shandong University, China 1996

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in the School
of
Engineering Science

SIMON FRASER UNIVERSITY
Fall, 2008

APPROVAL

Name:	Hans Fuhan Shi
Degree:	Master of Applied Science
Title of thesis:	Fundamentals of Suturing Simulation in Surgical Training Environment
Examining Committee:	Dr. John Jones Chair
	Dr. Shahram Payandeh, Senior Supervisor
	Dr. Hao Zhang, Supervisor
	Dr. Ghassan Hamarneh, Examiner

Date Approved:

Abstract

This thesis presents a knotting and suturing model based on the Virtual Training Environment (VTE). We introduce a mechanics-based approach to real-time simulation of deformable linear objects (DLOs) with visual and force feedback, which can represent the mechanical properties of a real thread, such as stretching, compressing, bending, and twisting. We also present how forces propagate along the suture when the user pulls it with one or two hands.

The user can practice the basic suturing techniques on the simulator presented in this thesis. The pre-wound suturing target is modeled as a modified mass-spring system. The tools involved in the live suturing procedures are also simulated. Collisions between the soft tissue and the needle, between the soft tissue and the suture are analyzed. In addition, the tissue tearing is also studied in this thesis.

Furthermore, this thesis addresses the GPU application in simulating deformable objects.

To my wife Lucia, whose support, patience, and encouragement helped make this thesis a reality. To the family who taught and encouraged me to follow my dreams and passions. And to my adorable son Noah.

Acknowledgments

Thanks to my senior supervisor Dr. Shahram Payandeh for the advice and support throughout this thesis work. This work would not be possible without his enthusiasm and knowledge of inter-disciplinary research.

Thanks to my supervisor Dr. Hao Zhang for his support and to Dr. Ghassan Hamarneh for examining this thesis.

Also thanks to Dr. John Jones for chairing my defence.

I thank everybody who works at Experiment Robotics Lab. I would never be graduating without your support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Virtual reality (VR) has been attracting an increasing interest in computing and science research fields. It is a substantial and ubiquitous technology by which humans can interact with computer-generated virtual environments in such a way that simulates real life and engages all the senses. It comes across applications for education, learning, and training. In virtual reality, the user is placed in a three-dimensional environment. With the assistance of haptic devices, the user not only can see the virtual objects, but also can feel and manipulate them and experience the consequences. Although the entertainment industry is VR's most widely known application, the real engagement of virtual reality lies in such fields as medicine engineering, and military. By virtual reality, scientists can triple the rate of oil discovery, pilots can practice flying techniques in virtual military training environment, and surgeons can improve their surgical skills on virtual objects instead of real patients.

There are usually three main components in one typical virtual training environment: The visual interface which might be a video screen, a computer monitor, or one of a variety of head-mounted displays (HMD), depending upon the visual requirements of the task being trained. The haptic interface which can provide tactile and force feedback; and the tracking system which will detect interaction between the virtual instruments and tissues. These three components, working in conjunction, are capable of producing a compelling impression that the trainee is interacting with the real tissue.

In the medical area, developing virtual surgical training environments has become one of the most practical topics of interest. The virtual training environments offer many benefits compared to the traditional training methods, including cost reduction, decreasing patient risk, flexibility, and the possibility of self-placed learning. Surgical training is traditionally performed in a master-and-apprentice model which requires the skilled surgeons to spend much time. The novice surgeon in training watches an expert surgeon performing an operation on the real patients. After sufficient experience, he or she may perform operations under the expert guidance. After enough practice the trainee then becomes an expert surgeon. Plastic models are the common models used in the traditional training methods, which can only demonstrate a limited range of anatomy and cannot reflect the mechanical properties of the living tissue, though they are relatively less expensive and can be used many times. Further more, not only do different procedures require different models, which will increase the cost, not all the required models are available. For practicing the cutting and stitching tasks, plastic models must often be replaced. Animals such as live pigs are anatomically similar to humans, however, they do not always reflect the human anatomy and are expensive. Cadavers present the most realistic anatomy, but the tissue responses are affected by preserving technologies. In addition, there is a limited supply of cadavers for the surgical training and they are generally more expensive.

Simulations trying to mimic the entire operation frequently fail because of lacking fidelity in some of the component tasks. However, in any operation, there are always some critical procedures that the trainee must master, which will result in complications that can impair the outcome of the operation or even endanger the patients, if it is performed incorrectly. Virtual training systems can significantly lessen the probability of morbidity by exposing the trainee to the potentially critical procedure in a safe and structured environment. Just as pilots train for emergency conditions on simulators, surgeons can train for the unanticipated complication and prepare for it. First, the virtual training systems are able to provide different training scenarios easily, such as different anatomy, pathologies, and operating environment. Secondly, a virtual environment may also recreate unusual situations which seldom occur in the operating room. Thirdly, the trainee can practice on the same scenario as many times as needed without introducing any additional cost, which will likely accelerate the acquisition of basic surgical skills. In addition, the virtual training environment can objectively quantify the performance and simulate the result of an operation and will be no harm or risk to any animals or real patients. Further more, in order to solve new problems that keep popping up in real surgical procedures, new instruments are being created every day. Virtual reality systems are uniquely situated as a test bed of new instruments or procedures because they can provide immediate feedback on the effectiveness, limitations, and problems of introducing new instruments or implementing new procedures. With the improvements of computer technology, when virtual models can eventually represent the actual surgical environment with the same physical properties, texture, and complexities, computer-based simulators can be an optimal approach for the surgical training.

Recently many surgical training systems have been designed and put into market, such as, Bronchoscope Training Simulator from Fifth Dimension Technologies, Virtual Clinic from Cine-Med, CathSim and Endoscopy AccuTouch® Simulator from Immersion etc. Unfortunately, there is few such complete systems available and almost all haptic surgical simulators have their own limit applications. This thesis focuses on a training simulator for suturing, knotting, and unknotting procedures. All of these works are built on the Virtual Training Environment (VTE) which has been developed at Experimental Robotics lab, Simon Fraser University for many years.

In this thesis, we primarily focus on the laparoscopic surgery (also known as minimally invasive surgery, or MIS). MIS has been developing very rapidly in recent years and is in an intensive development phase of its life cycle. It makes surgery less traumatic to the patient. Rather than cutting large incisions in the patient to easily access the operating target, only a few small incisions are made. Instruments, such as grasping forceps, scissors, cautery hooks, and staplers, are inserted into the body through these small holes. The operating site is viewed through a laparoscope which is also inserted through a small incision. The trauma caused by the operation is small compared to open surgery, which will speed the recovery and reduce the patient discomfort. However, the surgical skill requirements are greatly increased. While performing an operation, surgeons cannot rely on traditional eye-hand coordination since they see a 2D image rather than the real operating site directly. In addition, camera views of the operating site can be unusual and unnatural compared to the open surgery, which makes the operation even more demanding.

1.2 VTE System Overview

The VTE overcomes most of those obstacles of traditional training methods by creating a software and hardware simulator where trainees may practice repeatedly with feedback from the program. It brings together visual models and force-feedback (haptic) devices to offer virtual surgical training in real-time to medical practitioners. Figure 1.1 shows a trainee using the VTE practicing knotting skills.

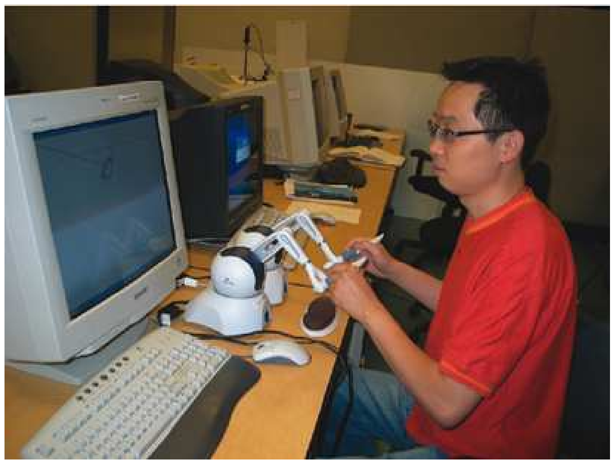


Figure 1.1: A user is practicing knotting skills with the VTE

1.2.1 VTE Tasks

Surgical procedures can be decomposed into tasks, subtasks and motions. Most computer-based surgical trainers use such component tasks rather than a whole procedure to train. The training tasks in those simulators use component tasks to train specific skills and expect that trainee to combine a set of specific skills and apply them on a more complex task. The current version of VTE provides ten tasks which can be classified into two categories, Elementary Tasks and Complex Tasks. These tasks are simple to understand and vary in the level of difficulty. The tasks have been designed in consultation with medical practitioners with the aim to develop essential skills needed to perform surgical operations.

Elementary tasks, such as touch and grasp, train basic eye-hand coordination and basic instrument manipulation, and are relatively simple to implement. However, once the trainee has gained basic skills, they need to practise on a higher-level task, which is more complex, such as dissection and suturing. Such training tasks not only require a great deal of knowledge and skill from the trainee; they are also more difficult to develop technically. The tasks of of the current VTE include:

Point and Touch Task

In this task, the user is required to touch each of the red spheres, so-called target sphere, on the surface of the textured sphere. The target spheres appear at different positions every time this task is selected. When the tip of the probe contacts any target sphere, it disappears. Once the target disappears, a message box pops up displaying the time taken to touch all the targets. The timer starts when the first target sphere is contacted, and stops when the last target sphere is contacted.

Tracing Curve Task

This task is meant to help the user practice the fine and smooth control of the probe. The user is required to slide the tip of the probe and draw a curve on the surface of mesh following the previous path. The user may use two or more probes to do the tracing task. Each tracing path of a probe has a different color so that the user may tell the tracing belongs to which probe with ease. The idea is to draw a curve with one probe and then to use the other probe to follow the previous path.

Grasp and Pluck Task

Once the user has mastered the interactions in the previous simple tasks, the relative complexity of the grasp/pluck task will be more manageable. The grasp/pluck task is more challenging. The user must grasp each of the smaller target spheres and drop them into the cylindrical receptacle in the front right corner of the visual field. It is quite difficult to align the gripper with a target sphere using the keyboard interface. In any case, it does work in the haptic version, and once all the targets have been removed from the surface of the textured sphere, a message box pops up indicating the elapsed time and the number of target spheres that missed the receptacle.

Teasing Away Task

The Teasing tasks are considered as the most challenging task among the four basic tasks, where the user is required to grasp any part of the target thread and tease it away from the surface of the deformable mesh. The use may feel a continuous force while he or she is pulling the thread away. This force is due to both the tissue between the target thread and the deformable mesh, and the tissue inside the thread. Once the tissue between target threads and the deformable mesh is broken, this force will suddenly disappear, and only the force inside the thread remains. The user may choose to use two probes to grasp the thread, in this case, the user may feel force feed back from both devices if force feedback is enabled.

Cutting Task

Cutting Task is one of the complex tasks of the current VTE. In this task, soft tissue is modeled with a surface-based mass-spring model. The surface mesh, which models the outer surface of the tissue, is composed of triangles. Collision detection between instruments and models is carried out by intersection checking of a line segment representing the instrument and triangles of the object mesh. Scalpel, cautery hook and grasper are implemented as cutting tools. Progressive cutting of a deformable object is included in this task. Users can also create a new mesh object by cutting through the objects.

1.2.2 Haptic Feed-back

Another important part of the VTE is the haptic feedback. Visual and haptic feedbacks are essential to interacting with surgery simulators. Haptic refers to manual interactions with environments and is concerned with being able to touch, feel, and manipulate virtual objects in the environment. Haptic feedback can be categorized as tactile feedback or force feedback. Tactile feedback is sensed by receptors close to the skin, especially in the fingertips. Tactile feedback is useful for presenting information about texture, local compliance, and local shape. The sense of touch is critical for example when surgeons palpate the skin to check for suspicious masses.

Haptic interface has been developing rapidly in recent years. The electro-mechanical training box is one of the earliest haptic devices, which is nothing more than a laboratory prototype. Now the haptic devices are so sophisticated that they can not only incorporate real laparoscopic instruments but also simulate the patient's external and internal anatomy.

Haptic rendering has also evolved from including only collision detection to realistic force-reflecting tissue response. Haptics is now the science of incorporating the sense of touch and control into computer applications through not only force (kinesthetic) but also tactile feedback. Collision detection provides the user with information that the virtual organs has been contacted; while force-reflecting tissue response provides tactile information about the response of the virtual organ to palpation, clamping, and suturing.

During simulations, the 3D virtual models of target tissues and the laparoscopic tools are displayed on a 2D monitor. Force feedback is provided by the haptic interface when the virtual probe encounters the virtual soft tissue. When the virtual forceps or retractors interact with the tissue in more complex maneuvers, torques are provided. Another kind of feedback - tactile feedback is based on the material properties of virtual tissues, which could produce tactile cues that the trainee can use to assess the precise location of the laparoscopic tool. The calculations of tactile feedback are computer intensive, which involves tissue parameters such as visco-elasticity, anisotropy, and nonlinearity.

Particle-based methods such as Mass-Spring System (MSS) and Finite-Element Method (FEM) are two principal approaches to developing force-reflecting organ models. In particle-based models, the organ's particles - mass nodes - are connected with springs and dampers. All the particles have their own positions, velocities, and acceleration characteristics and are moving under the influence of the implied forces of the surgical tools. In FEM, the geometric model of an organ can be divided into either surface or volumetric elements. The properties of each element are calculated and all the elements are assembled into a working model to computer the deformation of the organ under the applied forces. FEM relies on modeling the behavior of compliant biological tissues by differential equations and thus could provide more realistic tissue dynamics than MSS, although it needs more intensive computation.

A key issue in integrating force feedback into surgical simulators is the high update rate of haptic rendering required to achieve a stable feel. Real-time surgical simulation often requires computing the deformation of visco-elastic human tissue and generating both graphic and haptic feedbacks. Simulating the deformations involves calculating the tissue successive shape over time. Reaction forces result from the interactions between the virtual instruments, which are controlled by the haptic devices, and tissue models. To satisfy the requirement that virtual tissue models must look and behave realistically, the models must be based on physical laws governing the dynamic behavior of deformable objects. To update the new positions of physics-based deformable models, it usually requires solving a set of differential equations which is very computationally demanding. The visual display only needs the shape of the models to be updated at (a minimum of) 30Hz. However, the reaction forces sent to the haptic interface should be updated at a rate around 1000Hz for high fidelity.

Several commercial force feedback devices are currently available for the surgical simulation. One of the most commonly used devices is the SensAble PHANTOM Omni® from Sensable Technologies, Inc. (PHANTOM, PHANTOM Desktop, PHANTOM Omni, SensAble, and SensAble Technologies, Inc. are trademarks or registered trademarks of SensAble Technologies, Inc.), which is a point contact device. Another popular force feedback device is the Laparoscopic Impulse Engine (LapIE) from Immersion Corporation, which mimics tools used in laparoscopic surgery.

The input devices which the VTE supports are Laparoscopic Impulse Engine (see Figure 1.2) and Virtual Laparoscopic Interface (see Figure 1.3) from Immersion, SensAble PHANTOM Omni® (see Figure 1.4) and SensAble PHANTOM® Desktop™ (see Figure 1.5) from SensAble Technologies, Inc..

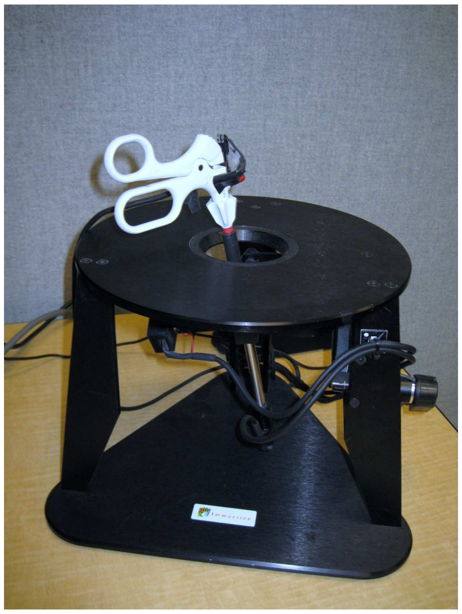


Figure 1.2: Laparoscopic Impulse Engine

The essence of the simulator operation is supporting the interactions between the models of one or more tissue objects and the virtual instruments. In order to make it easy to define various shapes for the users and experimenters, we chose the Virtual Reality Modeling Language (VRML) file format as the means of representing geometric shapes external to our program. VRML is an open standard for virtual reality and there are many free VRML models available on the Internet. Though VRML models have many properties, such as material and texture, we only use geometry and

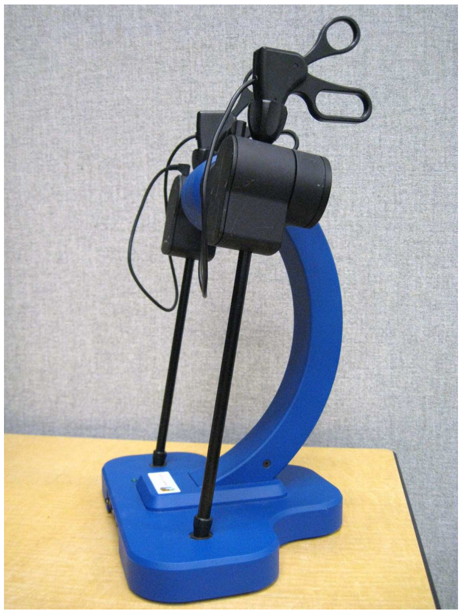


Figure 1.3: Virtual Laparoscopic Interface

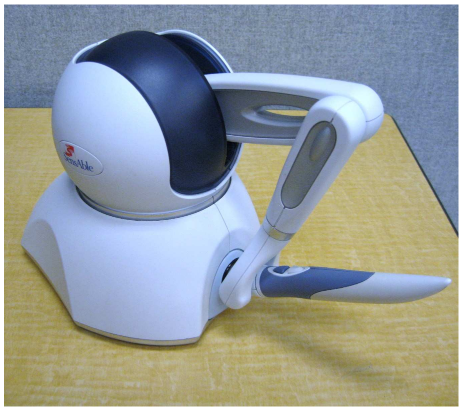


Figure 1.4: SensAble PHANTOM Omni®

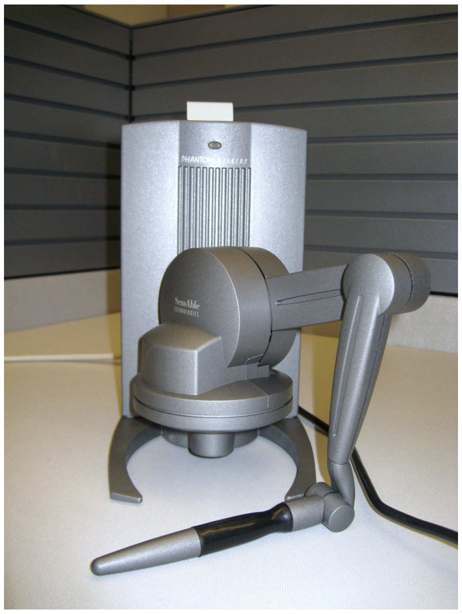


Figure 1.5: SensAble PHANTOM® Desktop™

topology information, i.e. vertices coordinates and how they are connected together. Therefore, we can use any version of VRML files, no matter VRML 1, 2 or 97. A VRML model can be composed of arbitrary planar polygons, e.g. triangle, rectangle etc. However, since our progressive subdivision algorithm to simulate cutting is based on triangle, we only use VRML models composed of triangles.

1.3 Related Work

1.3.1 Knotting and Unknotting

There are a number of works which have made some contributions to the development of Deformable Linear Object (DLO) simulations. Most of these previous models can be categorized as geometry-based models or mechanics-based models. Geometric models are slightly less accurate because they only simulate relative visual displacements. Mechanics-based models are often more accurate, although, for the virtual reality simulation, they may shift continuously until converge to equilibrium points, which makes them difficult for users to manipulate. In our training systems, because the purpose is to enable users to feel the force feedback when they manipulate the suture, especially during knotting and unknotting, to make it more realistic, we need to consider both external and internal forces to determine the force output. Thus geometric models are obviously inappropriate.

Some researchers have been focusing on knotting manipulation by robots. In [1], Wakamatsu, Arai and Hirai established a model of DLOs based on an extension of differential geometry, and proposed a planning method for knotting/unknotting of DLOs based on the knot theory. If the initial and the objective states of the linear object are given, all possible knotting/unknotting plans can be derived and be executed by their system. However, their proposed models can not simulate the DLOs dynamically in 3D space. In addition, their system does not allow any user interaction, and can not simulate the knotting/unknotting procedure in real-time. [2] describes a 2D DLOs dynamic model based on the differential geometry coordinates with the minimum number of parameters. Based on the description in this paper, the static deformation of a linear object have been formulated using the differential geometry coordinates, but the dynamic deformation has not been investigated. First, the dynamic 2D deformation of an inextensible linear object is formulated based on a differential geometry coordinate system. Second, simulation results is presented using the proposed modeling technique. Next, the proposed dynamic modeling is applied to the control of a flexible link. In [3], a knot planning from observation(KPO) system is described. First, this system observes the procedure of tying a knot by a human as a sequence of movement primitives. Then, by repeating the sequence, it can tie a similar knot. The topological information of a knot is represented in a P-data representation. A knot-tying task is converted into a sequence of movement primitives which have been defined in this paper. In [4], a topological motion planner for manipulating DLOs and tying knots using two cooperating robot arms was introduced based on Probabilistic RoadMaps (PRMs). The planner described in this paper takes a model as input, in the form of a state transition function and constructs a probabilistic roadmap in the configuration space of the DLO. The effectiveness are demonstrated by tying commonly used knots like bowline, neck-tie, bow (shoe-lace), and stun-sail.

In [5] [6] [7], Cosserat approaches of modeling DLOs based on the Cosserat theory of elastic rods have been introduced. Cosserat model is well suited for real-time applications because it needs less computation compared to finite elements models and provides a clear delineation between basic physical principles, material properties and mathematical approximations. However, in return, it yields a set of ordinary differential equations to be solved. If two end points or multiple points along the length of a suture are specified (as in the procedure of knotting or unknotting with two hands), it is significantly more difficult to solve these equations. In addition, the ”shooting” technique which is mentioned in [5] makes it very difficult to integrate external forces [8].

A particle-based model of a rope is represented in [9] by overlapping spheres representing mass-points, which are connected by simple springs. Each mass-point can collide with other mass points as in the instantaneous elastic collision model, but the author only considers the linear spring forces and does not allow any user interaction. In [10], inner bending force and the gravity are taken into consideration. In [11], the author mentioned gravity, stretch/compression force, forces from bending and twisting, dissipative friction, and contact forces with environment or to self-collision, but there is no detail about how to compute those forces.

A mass-spring model for suture in surgical training system has been built in [12]. Torsional spring, torsional damper, and viscous damper are modeled in this paper, but, the author did not use them in the simulation due to the complex computation. Further more, there is no discussion about collision detection and force propagation for haptic interaction between the user and the suture model.

1.3.2 Suturing

Suturing is one of the most fundamental tasks common to almost every surgery procedure used by surgeons worldwide. It can be used in from helping simple wound closure to complex tissue movements. Poor suturing techniques can cause traumatic negative influences on patients, such as slow healing, infection, and cosmetics. Before novice surgeons or medical school students could actually work on real patients, they must practise such skills on plastic models or virtual training systems to potentially improve their techniques.

Suturing simulations integrate various techniques from different areas, such as deformable object modeling, collision detection, and haptic rendering. Deformability of the object, difficulty of collision detection, and high demanding of haptic rendering make this topic very difficult to model and to solve. Recently many surgical simulators have been designed with or without using haptic devices.

A geometry-based suture model is presented in [13] with a outline of modeling a suture simulation. [14] describes a haptic simulation for teaching basic suturing skills for simple wound closure. During the simulation, needle holders, needle, sutures, and virtual skin are displayed and updated at real-time. An actual needle holder is attached at the stylus of a haptic device to make users feel more realistic.

[15] and [16] present a 3 DOF haptic based suturing simulator operating on mass-spring surface meshes. The suture is built as a geometry model and the method of "following the leader" is used to simulate displacements of suture nodes at each time step. A knot planner is also presented to let users tie a knot after the suturing. However, geometry models are less accurate and less realistic for dynamic simulations. Furthermore, the authors did not study the scenario about the tissue tearing when the stitches get over stretched, which is a key metric to evaluate the suturing skills of the user.

In [17] and [18], J. Berkley etc. show that real-time Finite Element Method (FEM) can be used to simulate suturing in surgical training systems. The authors present a new real-time methodology based on linear FEM analysis and prove the constraint approach is well suitable for suturing simulations. The suturing task is broken down into 9 steps in [17]. However, not all of these steps are implemented. Moreover, the computational model based on FEM is quite CPU intensive.

In [19], M. LeDuc et al present their initial works on simulating suturing using mass-spring models. Their suturing simulator shows that a wound could be closed by the suturing. However, it does not have any haptic interactions and the model does not allow the placement of arbitrary incision points around the wound. In addition, no models for the tissue reaction to excessive suture pulling forces are presented, which can result in tissue being ripped.

By comparing the performances of skillful surgeons to medical students on the same simple suturing task, [20] demonstrates that a surgical simulator can measure and develop surgical skills if that simulator has sufficient realism to emulate critical aspects of live surgical procedures. But they only focus on the penetration of the virtual needle, and the entire suturing procedure is not implemented.

A virtual suturing simulator with haptic feedback is presented in [21], in which the deformable tissue is modeled as a multi-layer mass spring system. The authors show that their simulator could suture a pre-wound soft tissue. However, there is no knotting part in their simulation, which is a key element in suturing. Once again, the study about how the suture tears the soft tissue is not conducted.

A user study about a suturing simulator is reported in [22]. A real needle holder is attached on the stylus of Phantom desktop haptic device. The user is asked to insert the needle approximately $2cm$ off the edge of the incision. Force application, time to task completion, length and straightness of the suture have been measured to discuss. Performance has been shown to improve over training. However, there is no details about how they modeled the deformable tissue and the suture, and how the suturing simulation is implemented.

Deformable objects can be built based on either physics-based model or geometry-based model. In [23], the authors have pointed out the advantages of using physics-based modeling. Among all the physics-based models, Finite Element Method (FEM) as in [17] [18] [24] [25] and Mass Spring System (MSS) as in [15] [16] [19] [26] [27] [28] are two commonly used models for representing deformable objects. In [29], S. Payandeh et al present an overview of haptic rendering using FEM and MSS. In [21], L.L. Lian has argued the benefits of using MSS in surgical simulations with haptic feedback.

1.3.3 GPU Application in Deformable Object Simulation

In the past, there have been a number of proposal to address issues related computational efficiency in the haptic rendering pipe-line. Majority of the proposed approaches can be categorized into three groups: Model simplification of deformable objects in order achieve a fast computational algorithms; Virtual coupling technique by introducing a spring-damper system between virtual objects and the end factor of the haptic device; Creating a approximate model for deformable object with which haptic interaction can be computed at high servo rates. There also have been a number of proposal for addressing the efficient utilization of Graphics Processing Unit (GPU) within the haptic rendering pipe-line. Such utilization can eventually lead to more realistic computational models which can be simulated on a standard desk-top computational environment.

[30] presents a GPU based model, in which the deformable object is simulated based on the notion of point-based mechanics. The deformation propagation of the object is computed through the use of "force fields" for both geometrically and dynamically point-based modeling. Multiple contact points deformation is also model and demonstrated in this paper.

[31] proposes a computationally inexpensive and efficient GPU based methodology to simulate complex deformable objects without compromising the haptic feedback. In this paper Gaussian function is used to compute the distribution of the deformation when the object is interacting with other objects. A GPU vertex shader is used to calculate the displacement of each vertex in parallel while the CPU computes the haptic force feedback.

[32] presents a GPU based mass-spring system for surgical simulation and gives the details of the calculation of the spring-mass system effectively in terms of the hardware accelerated features of the GPU.

An efficient method of haptic interaction with the GPU based surgical simulator is presented in [33].

[34] presents an approach for real time, progressive cutting of a complex iso surface model using a haptic device without the need to pre-compute a tetrahedrization of

the volume. A GPU-based marching tetra cutting algorithm is also discussed.

1.4 Contribution

1.4.1 Knotting and Unknotting

Knotting and unknotting are the most challenging parts of the suturing simulation which is essential to today's surgical training systems. In this thesis, we present a mechanics-based approach to the real-time simulation of deformable linear objects (DLOs) with the visual and force feedback. In our suture model which can represent the mechanical properties of a real thread such as stretching, compressing, bending, and twisting, we simulate not only the external forces, but also the internal forces including the friction force during knotting and unknotting. We also present how forces propagate along the suture when the user pulls it with one or two hands. We developed a simulator to allow users to grasp and smoothly manipulate a virtual thread, and to tie an arbitrary knot. Our Suture model is built based on all the force definition given in [12], and we provide a user-interface to allow users tie an arbitrary knot. With the virtual coupling technique [35], we can provide very smooth force feedback to the user.

1.4.2 Suturing Simulation

We also present a physics-based haptic simulation designed to teach basic suturing techniques for the simple skin or soft tissue wound closure. The pre-wound suturing target, skin or deformable tissue, is modeled as a modified mass-spring system. The suturing material is designed as a mechanics-based deformable linear object. Tools involved in the live suturing procedures are also simulated. Collisions between the soft tissue and the needle, the soft tissue and the suture are analyzed. In addition to the detail steps of one typical suturing procedure, modeling approaches on the evaluation of a stitch are also discussed. For example, if needle insertion points are too close from each other or too close to the edge of the wound, the suture will tear the soft tissue instead of suturing the incision together when the tension applied on the pierced nodes beyond a pre-set threshold. Experiment results show that our simulator can run on a standard personal computer and allow users to perform different suturing patterns with smooth haptic feedback.

1.4.3 GPU Application

In addition, we studied the GPU application in the deformable object simulation. We all know that one of the most challenging aspects of developing haptic interactive applications is to guarantee the haptic rendering rate of 1000Hz. This requirements in general can offer the user a smooth haptic sensation. With the demand of creating more complex and realistic scenes for the user interaction, there exists opportunities for the haptic system engineers to design and experiment with new computational environment which can blend the traditional computational mechanics models with more dedicated hardware utilities. In this thesis, we present some results regarding utilizing GPU processing units for computing the deformation of two experimental objects - a suture simulation model with GPU and a 2D deformable cloth model with nVidia CUDA techniques. We conducted experimental studies to compare the GPU-based suture models and with the CPU implementation. We also experimented with the implicit model of the 2D mesh which offer similar computational challenges associated with any Finite-Element modeling approaches. We proposed a method for computing the inverse of a matrix with truncated Nuemann series and nVidia CUDA technology. From the experimental results associated with the suture and the cloth, it can be seen that the GPU implementation did not effect the update rate very much with increasing the number of the segments. As such, by implementing the computational model of such deformable object at the GPU level, we can keep the update rate and be able to maintain the desired haptic frame rate.

1.5 Dissertation Road Map

The remainder of this dissertation is organized as follows: Chapter 2 of this thesis covers the description of all the models used in this research. Chapter 3 describes the knotting and unknotting procedure. Chapter 4 covers the suturing simulation. Chapter 5 describes the GPU application in the deformable object simulation, and Chapter 6 gives the conclusion and discusses about the future work.

Chapter 2

Models for Suturing

In this research, we simulate a section of skin or soft tissue, one suture, one needle, and two needle drivers. Each model is simplified to allow for faster computational environment, and to increase the haptic feedback rate in the virtual training environment. Rapid simulation of simple mechanics offers a more stable interaction in a discontinuous dynamical environment than more complex models. All the models are explained in details in the following subsections:

2.1 Suture Model

Geometry-based models and mechanics-based models are commonly used by most researches today in simulating deformable linear objects. The geometry approach directly models the motion of a suture by geometric means. "Following the leader" method is usually used to simulate the motion of the suture. Collisions are handled geometrically: if two segments overlap, they are moved apart until they do not overlap. On the other hand, mechanics-based suture models attempt to model reality by modeling each of the forces acting on a length of flexible material. External contact, twist, stiffness, and self-collision are each handled by applying forces to points along the suture.

We model our suture as a mass-spring system which consists of a sequence of mass points laying on the centreline of the suture. (see Figure 2.1).

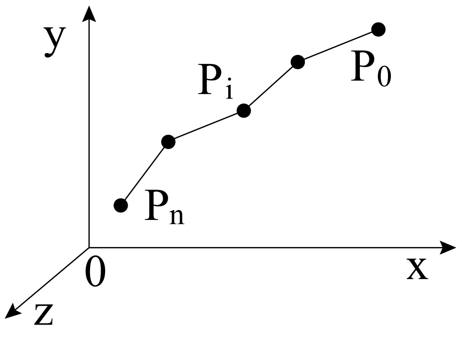


Figure 2.1: Suture model. P_i is the i^{th} mass point. All mass points are connected through segments.

In order to simulate the mechanical properties of a suture, such as stretching, compressing, bending, and twisting, we calculate not only external forces such as gravity, user input force, and contact forces with obstacles, but also internal forces including the friction force, linear spring, linear damper, torsional spring, torsional damper, and swivel damper. This model also allows the user to tie any kinds of knots with the interactions of haptic devices which will be explained in the next two chapters.

During graphical rendering, we use cylinders as suture segments connecting two successive points. We use the explicit Euler method to calculate the shape of our suture.

2.1.1 Explicit Euler Method

The differential equation of this $1D$ mass spring system is:

$$\mathbf{f} = m\mathbf{a}$$

where $\mathbf{f}$ is the force, m is the mass of the particle, a is the acceleration. Because the forces can depend on the particle's position, velocity, or time, we rewrite the above equation as:

$$\ddot{\mathbf{x}} = \frac{f(\mathbf{x}, \dot{\mathbf{x}}, t)}{m}$$

where $\mathbf{x}$ is the position of a mass point.

We add a new variable, v, to convert it to a pair of coupled 1st order equations:

$$\dot{\mathbf{x}} = \mathbf{v} \tag{2.1}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m} \tag{2.2}$$

To simulation such a system, the explicit Euler integration scheme is as follows:

$$\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \mathbf{f}_i^n \frac{dt}{m} \tag{2.3}$$

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \mathbf{v}_i^{n+1} dt \tag{2.4}$$

where i is the discrete mass point;n and $n+1$ indicate the n^{th} and the $(n+1)^{th}$ time steps.

Based on the above equations, first we compute the total force acting at each point, P_i, and then update its position based on the computed force. Once the total force at each of the nodes has been calculated, with the interval time dt, we can obtain the velocity and position of each point.

The following part of this section explains the forces we simulate in our simulator. We can use various combinations of these forces to build different models. The springs and dampers both contribute some force to the net force $\mathbf{f}$ at each point. Different springs and dampers all behave differently and we calculate their force contributions using their own particular equations.

2.1.2 External Forces

The external forces include the gravitational force, the user input forces through haptic devices, the friction forces during knotting or unknotting, as well as the contact force with obstacles:

Gravity

$\mathbf{f}_g = Gm$. where $G = 9.8N/kg$, and m is the mass of one mass point.

User Input Force

Allowing the user to provide both input and output to the simulation in the form of forces, positions, and velocity etc, a haptic device becomes a natural interface for a dynamic simulation. However, a position controlled impedance style haptic device, such as SensAble PHANTOM Omni® and SensAble PHANTOM® Desktop™ from SensAble Technologies Inc., forces are not directly available as input variables into the model. Furthermore, the mechanical characterization and digital nature of the haptic device make the operation of directly incorporating the device as part of the simulation more challenging. We use virtual coupling technique [35] to overcome these difficulties, which introduces a indirect layer of interaction between the mechanical device and the simulation by employing a spring-damper between a simulated body and the device end-effector (see Figure 2.2).

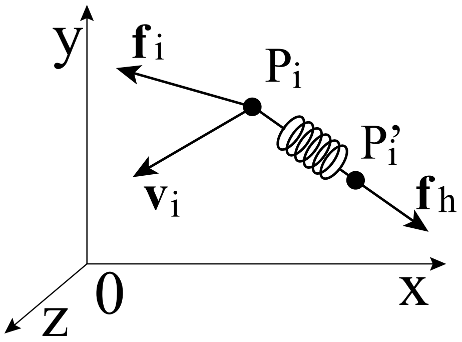


Figure 2.2: Virtual Coupling. P_i' is the point of the haptic device end-effector. P_i is virtually grasped point. $\mathbf{v}_i$ is the velocity of point P_i. $\mathbf{f}_i$ is the net force acting on point P_i. $\mathbf{f}_h$ is the user input force from virtual coupling.

All haptic devices have limits for the output forces. For example, SensAble PHANTOM Omni®, the maximum exertable force at nominal (orthogonal arms)position is 0.75 lbf (3.3 N) (See Appendix A). Before we output the force to the device, we must multiply it by an appropriate constant to make sure the output force not beyond the limits. Another advantage of virtual coupling is that we can use different constants for computing the output force for the device versus the input force for the simulated body, which makes the forces appropriate for both the haptic device and the dynamic simulation.

Friction Force

In this research, we use Coulomb model to simulate the friction forces during the procedure of knotting and unknotting. To simplify the computation, we only consider kinetic friction forces and will not compute any static frictions.

Coulomb friction is a model to describe friction forces. It is described by the following equation:

$$\mathbf{f_f} = \mu \mathbf{f_n}$$

where $\mathbf{f_f}$ is either the force exerted by friction, or, in the case of equality, the maximum possible magnitude of this force; μ is the coefficient of friction, which is an empirical property of the contacting materials; $\mathbf{f_n}$ is the normal force exerted between the surfaces

For surfaces at rest relative to each other $\mu = \mu_s$, where μ_s is the coefficient of static friction. For surfaces in relative motion $\mu = \mu_k$, where μ_k is the coefficient of kinetic friction. The Coulomb friction is equal to $\mathbf{f_f}$, and the frictional force on each surface is exerted in the direction opposite to its motion relative to the other surface.

During the simulation, we consider each suture segment as rigid body. From Coulomb's observations we know that: kinetic frictional force is approximately independent of contact area and velocity magnitude of the object; Coefficient of friction depends on pairs of materials. During knotting or unknotting procedure, suppose there are only two segments colliding with each other (see Figure 2.3).

We use linear interpolation to compute the velocity of a point on the segment. For example (see Figure 4.7), velocities of point C and E can be computed by the following equation:

$$\mathbf{v}_c = (1-a)\mathbf{v}_a + a\mathbf{v}_b \tag{2.5}$$

$$\mathbf{v}_e = (1-b)\mathbf{v}_c + b\mathbf{v}_d \tag{2.6}$$

where a is the fraction of point C along segment $\overrightarrow{P_aP_b}$; b is the fraction of point E along segment $\overrightarrow{P_cP_d}$.

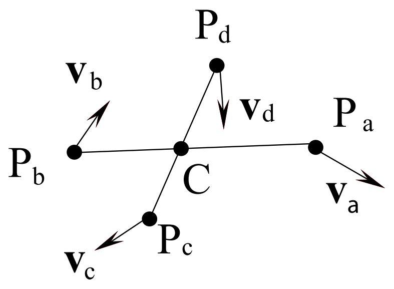


Figure 2.3: Two suture segments P_aP_b and P_cP_d are sliding on each other, where point C is the contact point; $\mathbf{v}_a$, $\mathbf{v}_b$, $\mathbf{v}_c$, and $\mathbf{v}_d$ are the velocities of points P_a, P_b, P_c, and P_d respectively.

Then the relative velocity can be obtained by:

$$\mathbf{v}_r = \mathbf{v}_c - \mathbf{v}_e \tag{2.7}$$

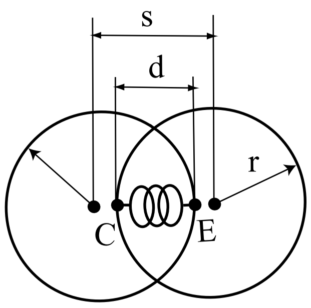


Figure 2.4: Intersection of the two contact segments. Point C and E are contact points. s is the distance between the center lines of two contact segments. r is radius of the suture segment.

Let $\hat{\mathbf{n}}$ be the unit vector from point E to point C (see Figure 2.4), then,

$$\hat{\mathbf{n}} = \frac{\overrightarrow{P_eP_c}}{||\overrightarrow{P_eP_c}||}$$

The friction direction vector $\hat{\mathbf{e}}$ is computed as follows:

$$\hat{\mathbf{e}} = \frac{(\mathbf{v}_r \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \mathbf{v}_r}{||(\mathbf{v}_r \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \mathbf{v}_r||}. \tag{2.8}$$

Let **n** be the force of repulsion. To calculate the repulsion force **n**, we introduce a spring-damper between the contact point C and the end point E.

$$\begin{aligned} \mathbf{n} &= (k_{rs}d - k_{rd}(\mathbf{v}_r \cdot \hat{\mathbf{n}}))\hat{\mathbf{n}}, && (2.9) \\ d &= 2r - s. && (2.10) \end{aligned}$$

where k_{rs} is a spring constant for the repulsion force, r is the radius of the rope model, d is the distance between contact point C to point E (of Figure 4.7), s is the distance between the two centers of the contact cylinder, k_{rd} is the damper constant for the repulsion force, $\mathbf{v}_r$ is the relative velocity of point C with respect to point E.

The friction $\mathbf{f}_f$ can be described in vector format as:

$$\mathbf{f}_f = \mu||\mathbf{n}||\hat{\mathbf{e}}. \qquad (2.11)$$

2.1.3 Internal Forces

Linear spring force

The linear spring force is computed by comparing the current segment length, l_i, between point, P_i and P_{i+1}, with the rest length of the segment l_r, and by projecting the resulting difference on the direction from point P_i to P_{i+1} (See Figure 2.5). Then,

$$\begin{aligned} l_i &= ||P_{i+1} - P_i||, && (2.12) \\ \Delta l &= \frac{l_i - l_r}{l_r}. && (2.13) \end{aligned}$$

where l_r is the rest length between point P_i and P_{i+1}.

Let $\hat{\mathbf{e}}_i$ be the unit vector from point P_i to P_{i+1}, then,

$$\begin{aligned} \hat{\mathbf{e}}_i &= \frac{P_{i+1} - P_i}{||P_{i+1} - P_i||}, && (2.14) \\ \mathbf{f}_s &= k_l \Delta l \hat{\mathbf{e}}_i. && (2.15) \end{aligned}$$

where k_l is the linear spring constant.

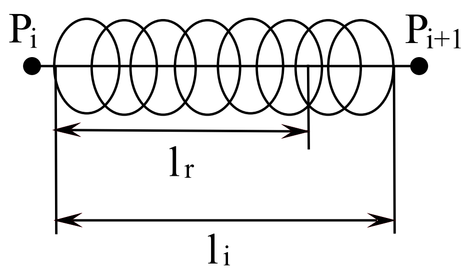


Figure 2.5: Linear Spring. Point P_i and P_{i+1} are two mass points. l_r is the rest length of the spring. l_i is the current length of the spring.

Linear damper

We simulate all the factors that try to stop the spring as it moves as one constant called the damping factor, k_d. It models all the frictions working against the motion of the moving mass points. This force opposes the direction of movement and is proportional to the velocity of the moving mass (See Figure 2.6).

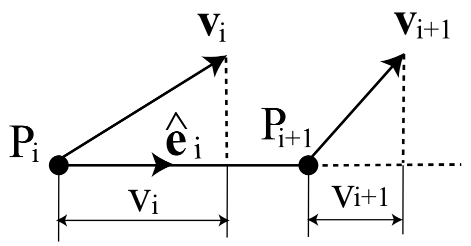


Figure 2.6: Linear Damper. Point P_i and P_{i+1} are two mass points. $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ are the velocities of Point P_i and P_{i+1} respectively. v_{i+1} and v_i are the norms of the components of the velocity of $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ on the direction of P_iP_{i+1}

When the system is at rest ($\mathbf{v} = 0$), no linear damping force is involved. The linear damper $\mathbf{f}_d$ can be computed by:

$$\mathbf{f}_d = k_d(v_{i+1} - v_i)\hat{\mathbf{e}}_i. \tag{2.16}$$

where k_d is the linear damper constant; v_{i+1} and v_i are the norms of the components of the velocity of point P_{i+1} and P_i on the direction $\hat{\mathbf{e}}_i$.

$$v_{i+1} = \mathbf{v}_{i+1} \cdot \hat{\mathbf{e}}_i, \tag{2.17}$$

$$v_i = \mathbf{v}_i \cdot \hat{\mathbf{e}}_i. \tag{2.18}$$

Torsional spring

With only the linear spring and the linear damper, tow connected suture segments can easily bend to any angles, which is not true in the real world. To overcome this problem, we introduced the third internal force - the torsional spring which is derived from the angle, α, between two connected segments of the suture (See Figure 2.7).

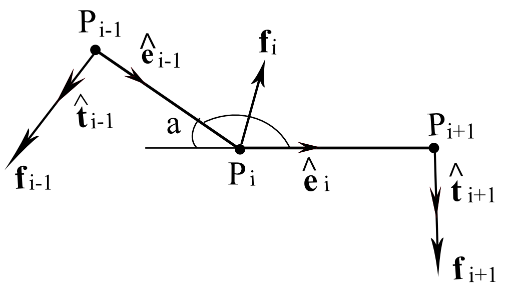


Figure 2.7: Torsional Spring. $\hat{\mathbf{e}}_{i-1}$ and $\hat{\mathbf{e}}_i$ are the unit vectors with directions from point, P_{i-1} to P_i, and from P_i to P_{i+1}, respectively. $\hat{\mathbf{t}}_{i-1}$ and $\hat{\mathbf{t}}_{i+1}$ are the unit vectors with directions the same as the torsional force applied at the two endpoints and therefore, orthogonal to $\hat{\mathbf{e}}_{i-1}$ and $\hat{\mathbf{e}}_i$ respectively and in the plane formed by segment $P_{i-1}P_i$ and P_iP_{i+1}

The basic idea is to model each two connected segments as a triangle with a spring as the hypothesis pushing the end points to the full expanded position. The length of the two connected segments remain unchanged. Only the force component orthogonal to the segments is used for the end points.

Let $\hat{\mathbf{e}}_{i-1}$ and $\hat{\mathbf{e}}_i$ be the unit vectors with directions from point, P_{i-1} to P_i, and from P_i to P_{i+1}, respectively. Let $\hat{\mathbf{t}}_{i-1}$ and $\hat{\mathbf{t}}_{i+1}$ be the unit vectors with directions the same as the torsional force applied at the two endpoints and therefore, orthogonal to $\hat{\mathbf{e}}_{i-1}$

and $\hat{\mathbf{e}}_i$ respectively and in the plane formed by segment $\overrightarrow{P_{i-1}P_i}$ and $\overrightarrow{P_iP_{i+1}}$. Then,

$$\begin{aligned} \hat{\mathbf{t}}_{i+1} &= \hat{\mathbf{e}}_i \times (\hat{\mathbf{e}}_{i-1} \times \hat{\mathbf{e}}_i), && (2.19)\\ \hat{\mathbf{t}}_{i-1} &= \hat{\mathbf{e}}_{i-1} \times (\hat{\mathbf{e}}_{i-1} \times \hat{\mathbf{e}}_i). && (2.20) \end{aligned}$$

If $\hat{\mathbf{e}}_{i-1} \cdot \hat{\mathbf{e}}_i \geqq 0$,

$$\alpha = \arcsin(||\hat{\mathbf{e}}_{i-1} \times \hat{\mathbf{e}}_i||). \tag{2.21}$$

If $\hat{\mathbf{e}}_{i-1} \cdot \hat{\mathbf{e}}_i < 0$,

$$\alpha = \pi - \arcsin(||\hat{\mathbf{e}}_{i-1} \times \hat{\mathbf{e}}_i||). \tag{2.22}$$

The common equation used to calculate the torsional spring force is:

$$P = K * \frac{Deg}{M}$$

where P is the force exerted on spring (lbs); M is the moment arm (inch); Deg is the deflection in (degrees); k is the spring constant (in-lbs/Deg).

We use the ratio of $\frac{\alpha}{\pi}$ to replace the Deg and revise the above equation. The torsional spring force then can be computed as follows:

$$\begin{aligned} \mathbf{f}_{i-1} &= k_{ts}\frac{\alpha}{\pi||P_{i-1} - P_i||}\hat{\mathbf{t}}_{i-1}, && (2.23)\\ \mathbf{f}_{i+1} &= k_{ts}\frac{\alpha}{\pi||P_{i+1} - P_i||}\hat{\mathbf{t}}_{i+1}, && (2.24)\\ \mathbf{f}_i &= -(\mathbf{f}_{i-1} + \mathbf{f}_{i+1}). && (2.25) \end{aligned}$$

where k_{ts} is the torsional spring constant.

Torsional damper

The torsional damper works against the torsional spring to prevent any harmonic motion from accumulating. Similar to the linear damper, it also models the internal friction that resists bending in regular objects. Let v_{i-1}, v_{ib}, be the norms of the velocity components of $\mathbf{v}_{i-1}$, and, $\mathbf{v}_i$, on the direction of $\hat{\mathbf{t}}_{i-1}$, and let v_{i+1}, v_{ia} be the norms of the velocity components of $\mathbf{v}_{i+1}$, and $\mathbf{v}_i$, on the direction of, $\hat{\mathbf{t}}_{i+1}$. We can get $\hat{\mathbf{t}}_{i-1}$ and $\hat{\mathbf{t}}_{i+1}$ from the previous section. Then,

$$\begin{aligned} v_{i-1} &= \mathbf{v}_{i-1} \cdot \hat{\mathbf{t}}_{i-1}, && (2.26) \\ v_{ib} &= \mathbf{v}_i \cdot \hat{\mathbf{t}}_{i-1}, && (2.27) \\ v_{i+1} &= \mathbf{v}_{i+1} \cdot \hat{\mathbf{t}}_{i+1}, && (2.28) \\ v_{ia} &= \mathbf{v}_i \cdot \hat{\mathbf{t}}_{i+1}. && (2.29) \end{aligned}$$

Then, the torsional damper on the points P_{i-1}, P_i and P_{i+1} can be computed by:

$$\begin{aligned} \mathbf{f}_{i-1} &= (\frac{(v_{i-1} - v_{ib})}{||P_{i-1} - P_i||} + \frac{(v_{i+1} - v_{ia})}{||P_{i+1} - P_i||})\frac{k_{td}\hat{\mathbf{t}}_{i-1}}{||P_{i-1} - P_i||}, && (2.30) \\ \mathbf{f}_{i+1} &= (\frac{(v_{i-1} - v_{ib})}{||P_{i-1} - P_i||} + \frac{(v_{i+1} - v_{ia})}{||P_{i+1} - P_i||})\frac{k_{td}\hat{\mathbf{t}}_{i+1}}{||P_{i+1} - P_i||}, && (2.31) \\ \mathbf{f}_i &= -(\mathbf{f}_{i-1} + \mathbf{f}_{i+1}). && (2.32) \end{aligned}$$

where k_{td} is torsional damper constant.

Swivel damper

Point P_{i-1} has a velocity relative to the center point P_i. So far, two components of that relative velocity have been dampened. There still remains a component perpendicular to those two. Without the dampening, point P_{i-1} could infinitely orbit the line formed by extending the edge connecting point P_{i+1} and point P_i (See Figure 2.8).

Let $\hat{\mathbf{s}}$ be the unit vector of the swivel dampers of point P_{i-1} and P_{i+1}, then,

$$\hat{\mathbf{s}} = \hat{\mathbf{e}}_{i-1} \times \hat{\mathbf{e}}_i. \quad (2.33)$$

The swivel dampers can be computed by:

$$\begin{aligned} \mathbf{f}_{i-1} &= k_{sw}\frac{(\mathbf{v}_{i-1} - \mathbf{v}_i) \cdot \hat{\mathbf{s}}}{||P_{i-1} - P_i||}\hat{\mathbf{s}}, && (2.34) \\ \mathbf{f}_{i+1} &= k_{sw}\frac{(\mathbf{v}_{i+1} - \mathbf{v}_i) \cdot \hat{\mathbf{s}}}{||P_{i+1} - P_i||}\hat{\mathbf{s}}, && (2.35) \\ \mathbf{f}_i &= -(\mathbf{f}_{i-1} + \mathbf{f}_{i+1}). && (2.36) \end{aligned}$$

where k_{sw} is the swivel damper constant.

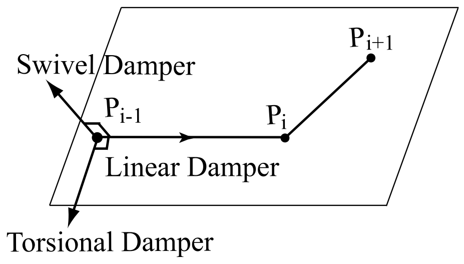


Figure 2.8: Swivel Damper. The linear damper and the torsional damper are working in the plane formed by $P_{i-1}P_i$ and P_iP_{i+1}. The swivel damper is orthogonal to the linear damper and the torsional damper.

2.2 Deformable Model

2.2.1 Mass Spring Model

A Mass Spring System (MSS) can be applied to either volumetric elements such as tetrahedral, or polygonal surface elements such as triangles. In order to reduce computation load during the real-time simulation, we choose a surface mass-spring model based on the model described in [27]. As presented in this paper, the deformable model consists of a triangular surface mesh made of nodes and springs. The vertices are mass nodes, and each edge is a simple linear spring.

Many spring-mass models provide the internal structure of the model by adding some explicit extra springs to model the interior of the model and to provide resistance when the model is compressed. We add one extra spring, home spring, to each node, connecting the node from its current position to its original position at the start of the simulation. This home spring ensures that the mesh will return to its original position if all forces are removed, and provides the illusion of an interior to the mesh with a minimum of computation. All springs are generated with a natural length that is 0.8 of the length of the edge. Combined with home-springs, this puts tension on the mesh, causing it to open naturally when it is cut.

Although a home spring has been added to each node to maintain the shape of the deformable object, the adjacent edges in small triangles still could bend to any angle if their common node is dragged by the needle or suture. We add a torsional spring and torsional damper as described in the previous sections to each pierced vertex in our model to solve this issue.

We build our 3D models of pre-wound skin or soft tissue in 3D graphics applications such as Autodesk 3ds max, and then export out the model as VRML file. Our simulator imports these VRML files to build the virtual objects. In this way, we can simulate different kinds of cuts for suturing tasks.

2.2.2 Deformation Computation

To reduce the computation load at most to satisfy the high demand of haptic rendering, we select the simple iterative explicit Euler method to update the states of the soft tissue and the suture. Although the explicit method has some drawbacks comparing to the implicit method, our experiment results show that we can make the simulator stable enough by restricting the integration time step dt to be inversely proportional to the square root of the stiffness.

If the node i is the pierced node and is also dragged by the needle or suture, in order to make the soft tissue deformation more smooth, we add torsional spring and torsional damper to it. Details about how to add such springs and dampers will be covered in the following section.

We use the method described in the previous section to calculate the torsional spring and torsional damper acting on the pierced mass node.

After we obtain the force acting on each node, we can use Equ. **??** and Equ. **??** to calculate the new velocity and then new position of each node.

2.2.3 Deformation Post-Step Constraint Enforcement

Springs are certainly not a perfect physical model for real cloth or other real deformable objects because their elongation is proportional to the force applied, which may result in implausibly large deformations. The common force-deformation curve for a material is nonlinear. So we must modify the behavior of our mass-spring system to make it more realistic. One way to achieve this is to add a post-correction phase after a time step. We choose the post-step constraint enforcement process after each time step computation to eliminate the large stretch as defined in [36].

The essential idea and implementation are the following: We set a threshold for each spring and add a post-step process after each deformation computation. We define a normalized threshold $l_{max} = \beta l_{rest}$ for each spring, where β is a constant, l_{rest} is the rest length of the spring. Each time when a spring is overstretched, we bring the two mass nodes at endpoints of the spring together along the axis while preserving the position of the center point of the spring. For the case when one of the two node is grabbed by the user, or slipping on the needle or the suture, we only move the other node to insure the appropriate elongation. We iterate over any overstretched springs and shrink them.

We can have several ways to determine when to end the iteration: we can chose to terminate it after a predefined number of loops, until convergence is reached, or time is up. The predefined number is shown to be sufficient for our case. Because this process is only about displacements and no forces are involved, stability is not an issue.

2.3 Tools model

Tools involved in a real suturing surgery include needles, needle drivers, scissors, and tweezers. We only simulate the needle and the needle driver to this stage.

2.3.1 Needle Driver Model

Figure 2.9 shows a needle hold by a needle driver and one typical needle driver used in the surgery.

The needle drivers in the simulation are represented by three line segments: one for the shaft and one for each jaw for the graspers (See Figure 2.10). As shown in the above picture, triangle AOB is the open triangle of the needle driver. The collision volume consists of lozenges about these lines. The tools are controlled by haptic

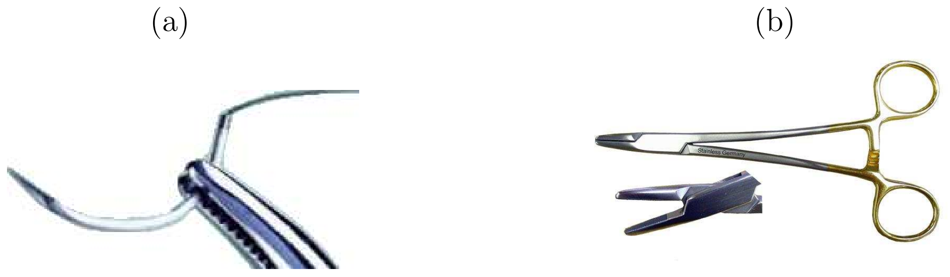


Figure 2.9: (a) A needle hold by a needle driver. (b) A needle driver commonly used in the surgery.

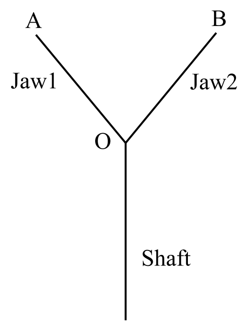


Figure 2.10: The model of the needle driver. Triangle AOB is the open triangle of the needle driver.

devices such as Virtual Laparoscopic Interface, Laparoscopic Impulse Engine (Immersion Corporation), SensAble PHANTOM Omni®, and SensAble PHANTOM® Desktop™.

2.3.2 Needle Model

In order to simulate different kinds of needles used in the real surgery, same as for the soft tissue model, we chose to use VRML to build our needle model. In this way, we can create different needle models in 3D graphics applications such as Autodesk 3ds max, and then export out the models as VRML files. Our simulator imports these

VRML files to implement the virtual objects.

Figure 2.11 shows the needle model in our simulation. We use arc TE to represent the needle. Point O is the needle arc center and also is the origin of local coordinate of the needle. Point T and E are the tip point and the end point of the needle respectively. r is the radius of the arc TE. $\hat{\mathbf{y}}$ is the unit tangent vector of arc TE at point T.

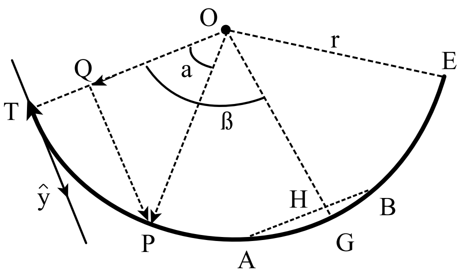


Figure 2.11: Needle model - arc TE represents the needle during simulation. Point O is the needle arc center. Point T and E are the tip point and the end point of the needle respectively. r is the radius of the arc TE. $\hat{\mathbf{y}}$ is the unit tangent vector of arc TE at point T.

For any point P on the needle, we can obtain it's position if the angle α from tip to p is given. $\overrightarrow{OP}$ can be derived from the following equation:

$$\overrightarrow{OP} = \overrightarrow{OT}\cos(\alpha) + r\sin(\alpha)\hat{\mathbf{y}}. \tag{2.37}$$

where

$$\hat{\mathbf{y}} = \frac{(\overrightarrow{OT} \times \overrightarrow{OE}) \times \overrightarrow{OT}}{||\overrightarrow{OE}||||\overrightarrow{OT}||||\overrightarrow{OT}||}$$

Figure 2.12 shows a needle is grabbed by the needle driver in the simulation.

We use the following method to determine the position and orientation of the need when grasped by the needle driver: after the needle is grabbed, we use gimbal angles of SensAble PHANTOM Omni® as explained in [37] and [38] to identify the orientation of the needle. Once we map all the gimbal angles to OpenGL coordinate angles, we use ZXY convention of fixed angle rotations to computer the needle rotation matrix

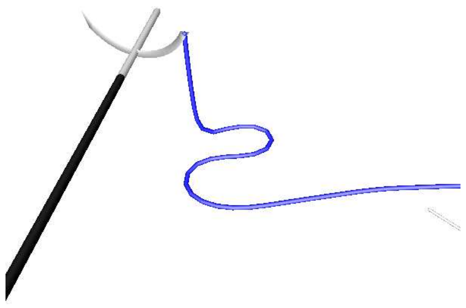


Figure 2.12: A suture is attached on the end ponit of a needle during the simulation

following the right-hand rule. rule as in Equ. 2.38.

$$
\begin{aligned}
& R_{ZXY(\gamma,\beta,\alpha)} \\
= \; & R_Y(\alpha)R_X(\beta)R_Z(\gamma) \\
= \; & \begin{bmatrix} c\alpha & 0 & s\alpha \\ 0 & 1 & 0 \\ -s\alpha & 0 & c\alpha \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\beta & -s\beta \\ 0 & s\beta & c\beta \end{bmatrix} \begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{2.38}
$$

Chapter 3

Suturing Simulation

3.1 Introduction

Suturing is the most difficult task in surgery, which requires fine manipulation and close coordination of both hands. Virtual surgery training environments offer many benefits comparing to traditional training methods, including cost reduction, decreasing patient risk, flexibility, and the possibility of self-placed learning. Unfortunately, there is few such complete systems available and almost all haptic surgical simulators have their own limit applications. Sensory feedback including visual feedback and force feedback is a crucial requirement to make surgery simulations more realistic. Mechanical knowledge of the soft tissue and the suture is required to compute feedback forces.

In this chapter, we present a suturing simulator based on the suture model described in the previous chapters. We model both the soft tissue and the suture material based on physics models. With the assistance of two Phantom Omnis, our simulator can provide smooth force feedback and allow the user to perform different suturing patterns during the training.

Section 3.2 covers the collision detection and management. Section 3.3 describes the process of suturing. Section 3.4 illustrates the stitch evaluation. Section 3.5 gives the experiment results.

3.2 Collision Detection and Management

Collision detection and collision management are two of the most important and challenging components for almost all dynamic simulations, especially for deformable objects such as the soft tissue or the suture in our case.

3.2.1 Collisions between Needle Drivers and the Needle

Haptic devices are implemented as two needle drivers in our virtual environment. To specify an object in $3D$ space, we need it's both position and orientation. The needle should follow the stylus of the haptic device when it is grabbed. To make the collision detection easier, we decompose the needle arc into six connected line segments. Suppose line segment AB is one of these needle segments (See Figure 3.1 again):

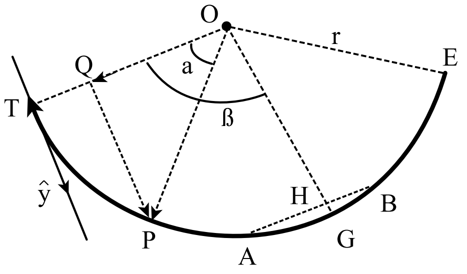


Figure 3.1: Needle model - arc TE represents the needle during simulation. Point O is the needle arc center. Point T and E are the tip point and the end point of the needle respectively. r is the radius of the arc TE. $\hat{\mathbf{y}}$ is the unit tangent vector of arc TE at point T.

First, we check needle driver's open triangle with each needle segment to see if there is an interaction. If an intersection happens, we assume the intersection point is point G on the needle instead of point H during simulation. We can easily get the point G by β which is the angle between $\overrightarrow{OT}$ and $\overrightarrow{OH}$.

3.2.2 Collisions between the Needle and the Soft Tissue

We build a bounding box around the needle tip using the length of the longest spring of the soft tissue plus a pre-set threshold as the dimension of the box. We can get the length of the longest spring when we read the model of the soft tissue from VRML file. (See Figure 3.2).

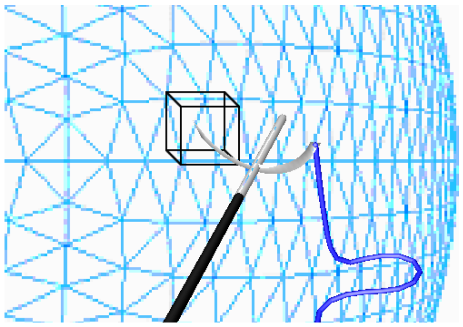


Figure 3.2: Bounding box of the needle tip

This bounding box is always following the movement of the needle. Whenever a mesh node (vertex) is found inside the bounding box, we put all it's adjacent polygons as candidates to detect if they are colliding with the needle tip. The collision detection tests the straight-line path of the needle tip during a single time step against each triangle.

3.2.3 Collisions between the Suture and the soft tissue

The pierced node on the soft tissue should follow the movements of the suture if the spring force acting on that node is less than the suture friction force (which is a pre-defined constant), otherwise, it will be sliding along the suture. We call each of these new pierced node as a soft constraint. Three kinds of soft constraints for the suture are defined based on the location of the pierced node: *Top Constraint*, *Bottom Constraint*, and *Groove Constraint*. Figure 3.3 shows the schematic of a suturing pattern. Node C and N are top constraints. Node D and E are bottom constraints.

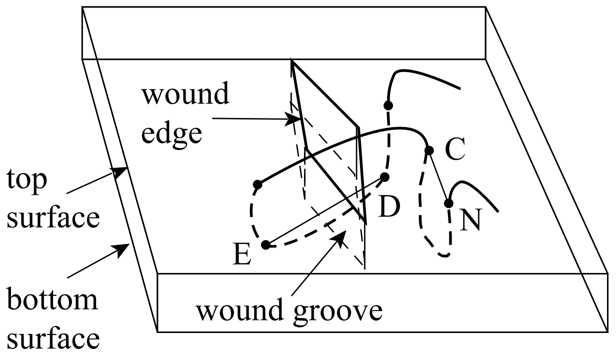


Figure 3.3: Schematic of a suturing pattern. Node C and N are top constraints. Node D and E are bottom constraints

We name the constraints in a numerical order along the suture. More than one constraints are not allowed to reside on the same suture node at the same time. To implement the collision detection between the suture and the soft tissue, first we need to find out if the suture nodes should be inside or outside of the soft tissue. We achieve this by counting how many constraints have passed this suture node because all constraints start from the needle and have to pass from the needle to the suture in order. If the number is even, the suture node should be outside the soft tissue. Otherwise, it should be inside.

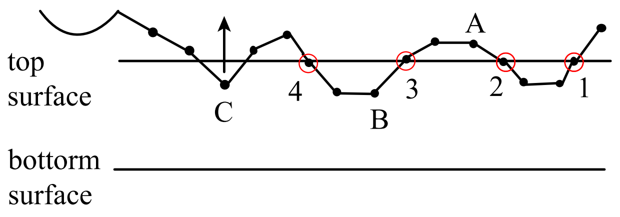


Figure 3.4: Collisions between suture and soft tissue

For example, node A and B are two mass node of a suture. Figure 3.4 shows that two constraints have passed the suture node A, therefore, A should be outside of the tissue. There are three constraints which have passed node B, so B should be inside. Using this method, we know that suture node C should be outside, therefore we must move node C to the outside of the soft tissue during the collision management.

3.3 The process of suturing

Based on what is presented in [15], we divide one whole suturing procedure into the following five steps: needle pierces the object, soft constraints slid on the needle, soft constraints slid on the suture, apply the suture tension to close the wound or incision and knotting (See Appendix B). During the first procedure, the needle pierces the object, to prevent any two connected springs from bending easily to any undesirable angle, we have added torsional springs and dampers to each of these newly created springs (See Figure 3.5).

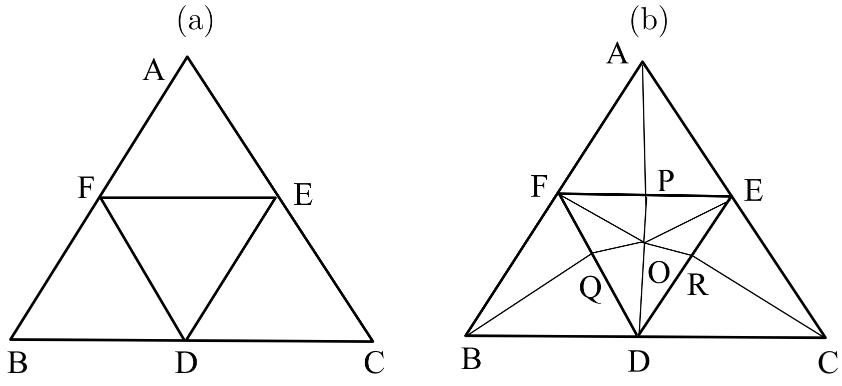


Figure 3.5: (a) Triangles before subdivisions. (b) Triangles after subdivisions

The needle is an independent object held by a needle driver under the user control as described in chapter 2. When the needle tip collides with a triangle, the pierced triangle is subdivided simply into three triangles, with a new vertex at the piercing point, and then the pierced triangles original edges are subdivided using the edge mask in the loop subdivision scheme. When the mesh is pierced, a new vertex is generated at the point where it is pierced; this new vertex is the pierced vertex. When the pierced vertex is on the needle, it cannot move except to slide on the needle. Once the vertex has been passed to the suture, the mesh will pull the suture and slide along the suture .

Because the torsional spring and the damper should be configured on two connected springs as a pair, we have to deal with two connected springs at the same time. From Figure 3.5 (b), for example, suppose point O is the pierced node on the soft tissue surface. We add torsional springs and dampers to spring FO and OR as a pair, and the same to spring DO and OP, spring EO and OQ.

3.4 Stitch Evaluation

3.4.1 Tissue Tearing

To make the suturing simulation more realistic, we need to simulate not only normal situations, but also unwanted scenarios. Tissue ripping is one of these unwanted circumstances that always happens to novice surgeons or medical students. It is also one of the key metrics to evaluate the trainee's performance during the training.

Based on the assumption that the stitch goes through from the top surface to the bottom surface of the soft tissue, we define two types of tearing: *Tear-Into* and *Tear-Through* which are similar to the cut-into and the cut-through described in [27]. We also define *Top Start Constraint* to be the constraint where the tearing starts on the top surface, and *Top End Constraint* to be the constraint where the tearing ends on the top surface. For tear-through case, we define *Bottom Start Constraint* and *Bottom End Constraint* with the same idea. The constraint on the groove or the bottom surface is called *Groove Constraint*

Figure 3.6 (a) shows that point C is a top start constraint in tear-into case, point D is a groove constraint. Figure 3.6 (b) shows that point C is a top start constraint, point D is a bottom start constraint.

Tear-Into

To simulate the tearing, we use the same cutting subdivision algorithms as described in [27] and [28].

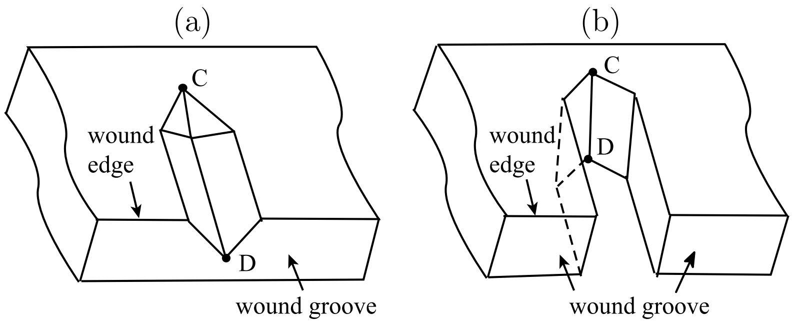


Figure 3.6: (a) Tear-into the soft tissue. C is the top start constraint and D is a groove constraint (b) Tear-through the soft tissue. C is the top start constraint and D is the bottom start constraint.

First, we divide the tearing of one polygon into two state: *Start State* and *Termination State*. Two cases are also defined to each of these two states: *Tear Through Vertex* and *Tear Through Edge* depends on the intersection point between the tearing path and the teared polygon. For our case, the start state of first tearing polygon and the termination state of the last tearing polygon are always tear through vertex (See Figure 3.7).

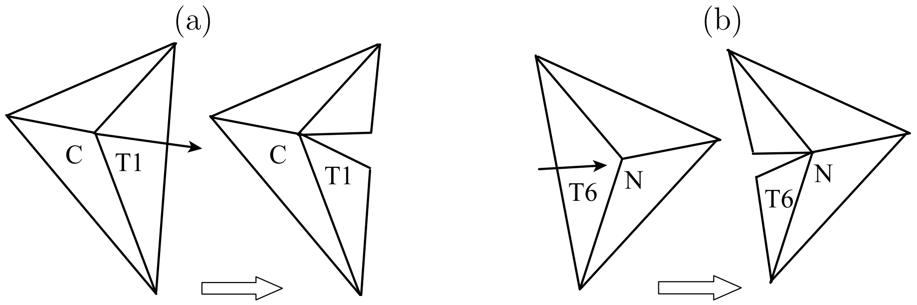


Figure 3.7: (a) Start state of the first tearing polygon. (b) Terminiation state of the last tearing polygon

Secondly, to determine the middle polygons' states along the tearing path, we need to find out the intersection points between each of these polygons and the tearing path. In order to get rid of the side affect of the deformability of soft tissue, we simply map the home positions of each mass node on the top surface to the XZ plane to find out the intersections points between the mapped tearing path and each mapped edge, then map the virtual intersection points back to the real surface. Figure 3.8 shows that C is the top start constraint and N is the top end constraint. C' and N' are the mapped nodes to C and N respectively.

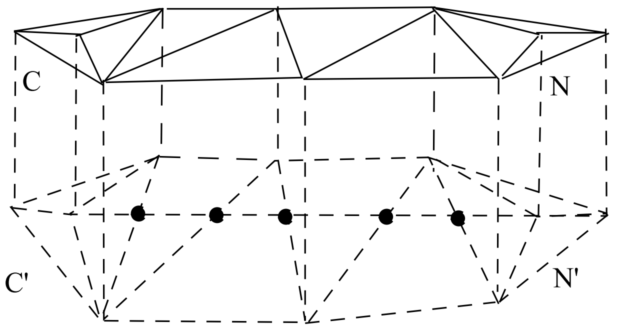


Figure 3.8: Map the surface to XZ plane to find out the intersection points

There are two cases should be considered in this condition depends on if the tearing path crosses the wound or not. Figure 3.6 (a) shows half part of a tearing across the wound. Figure3.9 shows the polygon subdivision of a tearing not across the wound. For the former case, we use the groove constraint to set the depth of the tearing groove (point D in Figure 3.6 (a)). For the case not cross the wound, we just manually set a default tearing depth.

Tear-Through

In this condition, we must find out both the bottom intersection points along the bottom tearing path and the top intersection point along the top tearing path. We use the same mapping method as in tear into case to map the top surface and the bottom surface of the soft tissue separately. Then subdivide the bottom surface and the top surface at the same time. After the subdivision, we use groove polygons to

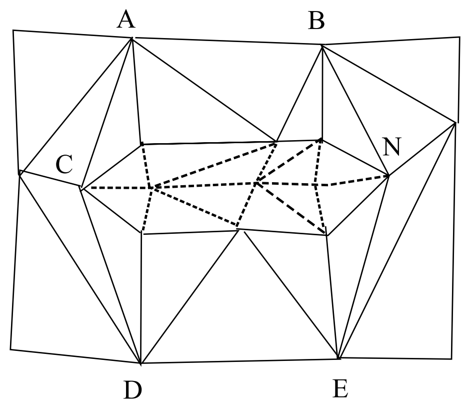


Figure 3.9: Surface polygon subdivision in tear-into condition where the tearing path is not across the wound

connect the bottom and the top up. Figure 3.6 (b) shows a tear-through case.

3.5 Haptic Force Feed-back

During the development of the high reliable surgery training environments, haptic feedback plays an very important role. In order to study the details of the forces propagation along the suture, between the suture and the soft tissue, and inside the soft tissue, we need to track the forces acting on each soft constraint in the suturing procedure. [16] has studied the forces when the needle pierces the deformable meshes. In this section, we only present force changes of the pierced mass node when the suture tears the soft tissue from this node.

Figure 3.10 is a plot of the changes of the force acting a the pierced node in different update frames. The tearing threshold is set to be $40N$. As you can see from the figure, the force steps down suddenly after reaches the threshold, which means a tearing procedure happens at that point.

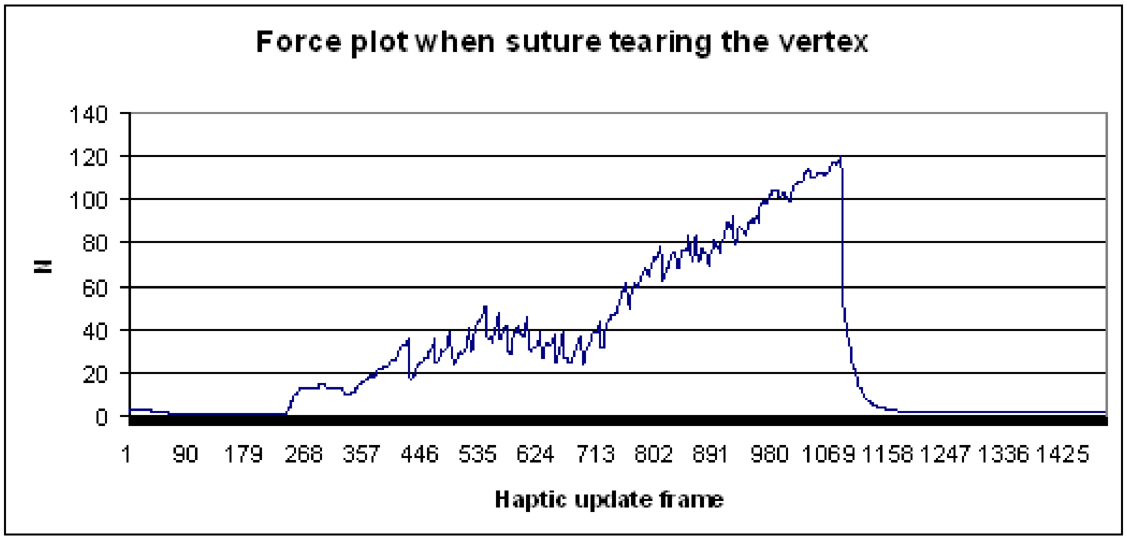


Figure 3.10: Plot about the force acting on a pierced node when the suture tears the tissue from this node

3.6 Experiment Results

We did our experiment on a standard personal computer with Intel(R) Pentium(R) 4 3.00GHz CPU and 1G Ram. Two haptic devices we were using are SensAble PHANTOM Omni®s from SensAble Technologies Inc.

3.6.1 Suturing

The following scenarios show the details of different steps in one suturing procedure. Figure 3.11 to Figure 3.12 show the changes of the soft tissue surface before the needle pierces the surface and the after. As you can see from the wireframe pictures, polygon changes show the subdivision algorithm discussed in the following sections.

Figure 3.13 to Figure 3.14 show the scenes when the constraint is sliding on the needle and on the suture.

Figure 3.15 shows a simple continuous suture pattern implemented in our simulator.

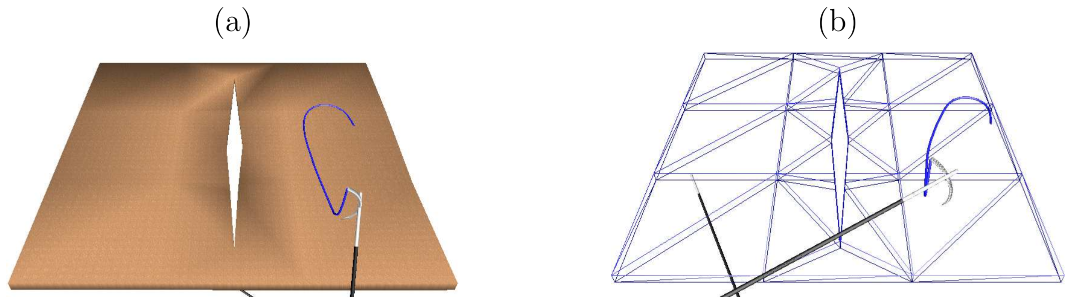


Figure 3.11: Before the needle pierces tissue. (a) Screen shot (b) Wire frame

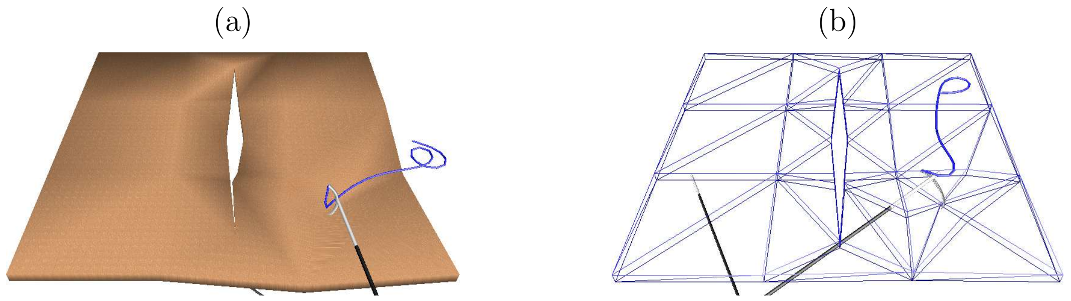


Figure 3.12: After the needle pierces the tissue. (a) Screen shot (b) Wire frame

3.6.2 Knotting

Knotting is always a key component for suturing, which will be discussed in details in Chapter 4. Figure 3.16 shows a simple suturing pattern with a knot.

3.6.3 Tearing

For the tearing experiments, we manually set the end point of the suture to be always outside of the suture to prevent the suture from sliding out the soft tissue from the insertion points.

Figure 3.17 and Figure 3.18 are screen shots of tear-into case. Figure 3.17 shows

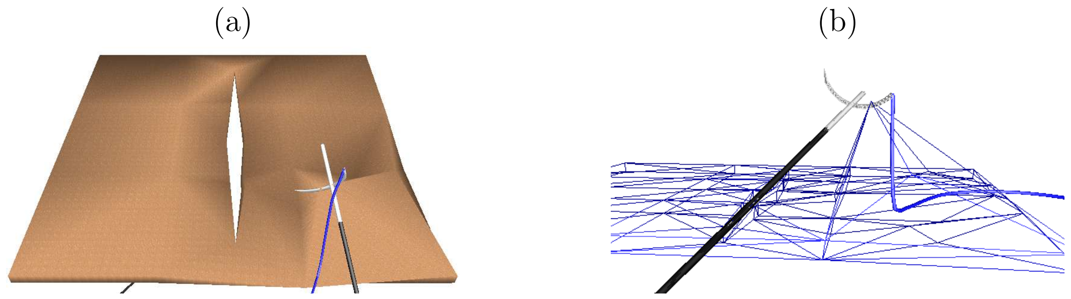


Figure 3.13: Constraint slipping on the needle. (a) Screen shot (b) Wire frame

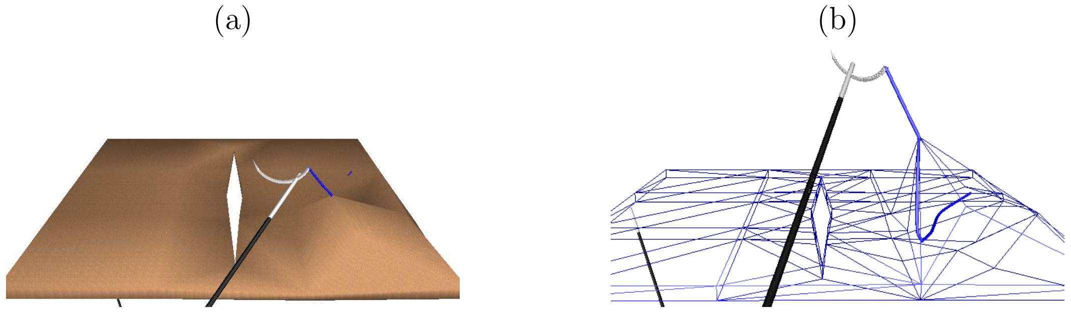


Figure 3.14: Constraint slipping on the suture. (a) Screen shot (b) Wire frame

that two needle insertion are too close from each other and the tearing path does not cross the wound. Figure 3.18 shows that the needle insertion points are too close to the edge of the wound, and the tearing path crosses the wound.

Figure3.19 and Figure3.20 are screen shots of tear-through case. Figure 3.19 shows that tearing path crosses the wound. Figure 3.20 shows that tearing path does not cross the wound.

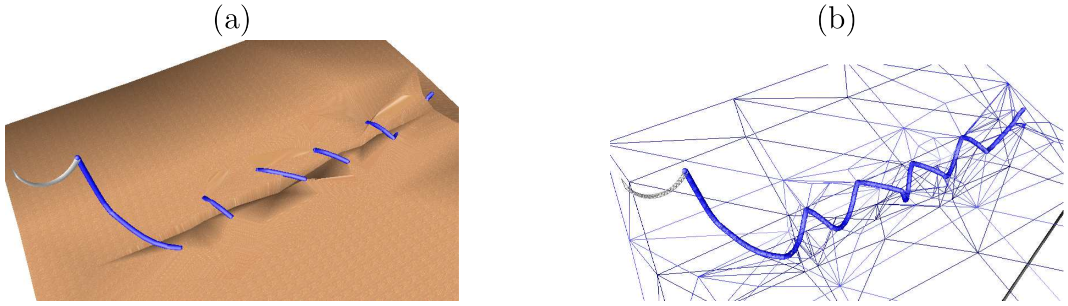


Figure 3.15: A simple continuous pattern. (a) Screen shot (b) Wire frame

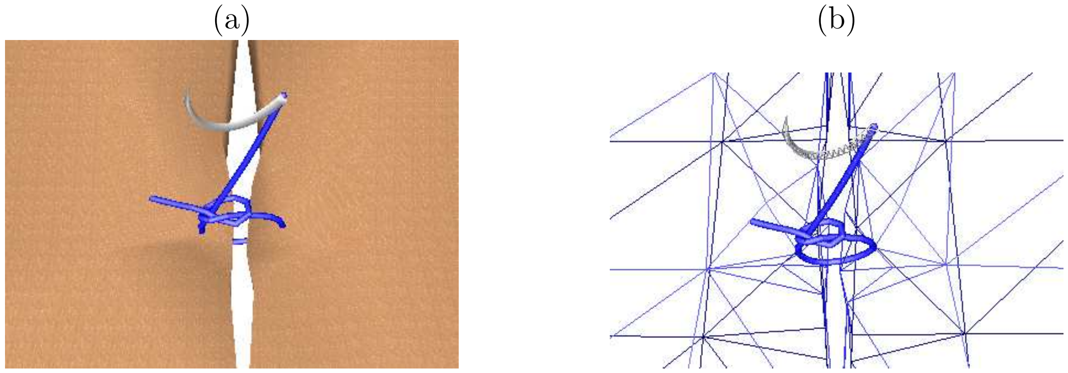


Figure 3.16: A single stitch with a knot. (a) Screen shot (b) Wire frame

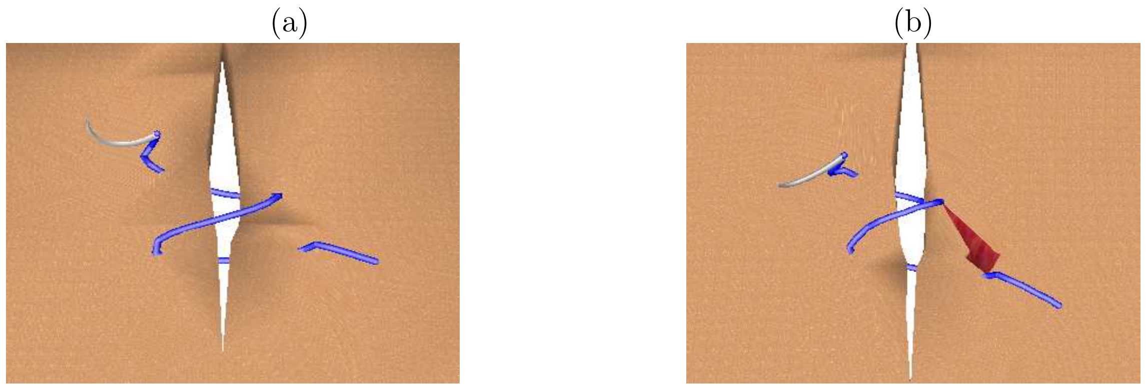


Figure 3.17: Tear-into - tearing path not across the wound. (a) Before tearing (b) After tearing

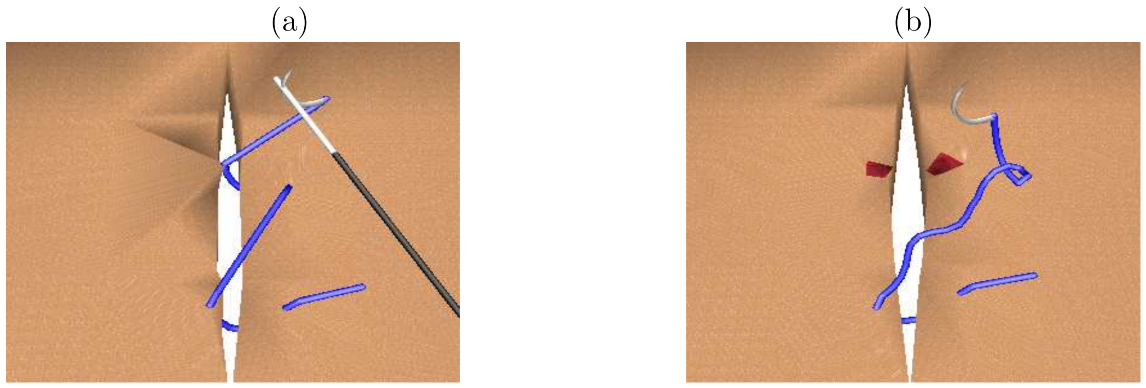


Figure 3.18: Tear-into - tearing path across the wound. (a) Before tearing (b) After tearing

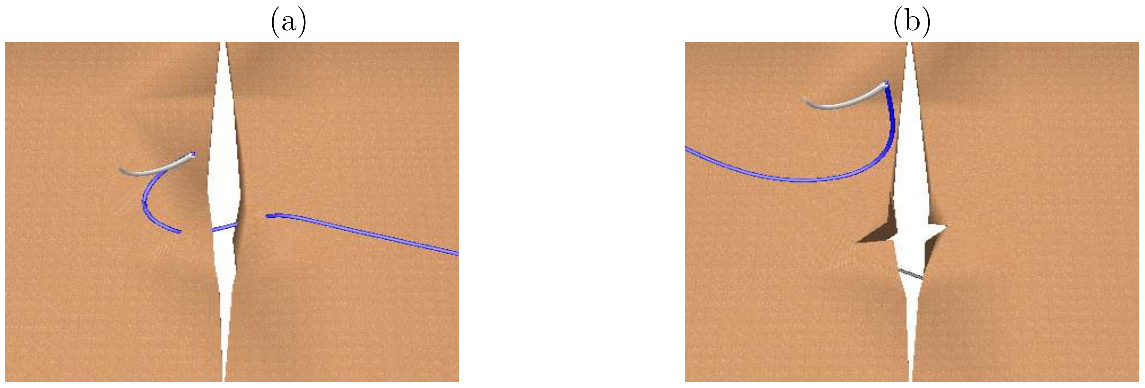


Figure 3.19: Tear-through - tearing path across the wound. (a) Before tearing (b) After tearing

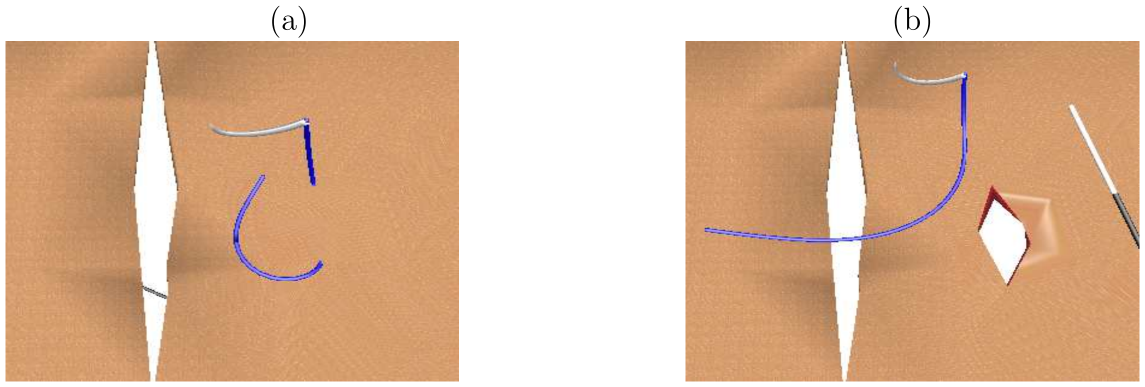


Figure 3.20: Tear-through - tearing path not across the wound. (a) Before tearing (b) After tearing

Chapter 4

Knotting and Unknotting

4.1 Introduction

The application of knots can ascend to the Paleolithic era. We use all kinds knots in our everyday lives such as fastening our shoes or clothes, wrapping gifts, animal handling, fishing, sailing, climbing, carving, and even for decoration etc. In the medical field, they are essential to the suturing in today's surgery procedures. The real-time simulation of deformable linear objects (DLOs) is required in many areas, such as surgical training systems and rock climbing or sailing training systems to teach users how to tie and untie a knot. It also related to the cloth-like deformable objects simulation, an area has attracted much attention in Computer Graphics recently.

As described in Chapter 2, first we compute the total force acting at each point, which is the sum of the friction force, the linear spring, the linear damper, the torsional spring, the torsional damper, and the swivel damper. Secondly, once the total force at each of the nodes has been calculated, with the interval time dt, we can obtain the velocity and position of each point. Knotting and unknotting procedures are all about how to deal with collision detection and collision management. The remainder of the chapter is as the follows: Section 4.2 covers how the force propagates along the suture. Section 4.3 illustrates the collision detection and managements methods used in this research. Section 4.4 describes the haptic feed-back during knotting and unknotting. Section 4.5 covers the experiments of knotting and unknotting.

4.2 Force Propagation Along the Suture

To prevent the suture from being stretched too long or compressed too short, we set l_{max} and l_{min} as the maximum and minimum length of one suture segment respectively (see Figure 4.1).

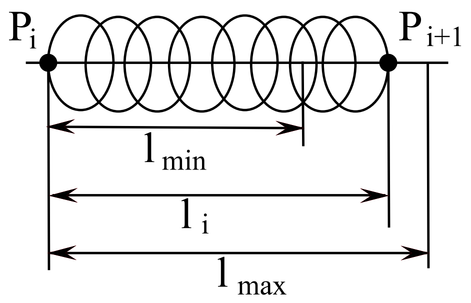


Figure 4.1: Linear Spring. Point P_i and P_{i+1} are two consecutive mass points. l_i is the current segment length between P_i and P_{i+1}. l_{min} and l_{max} are the minimum and maximum length of the spring.

Let l_i be the segment length between P_i and P_{i+1}. To analyze the force propagation when the user grasps the suture, we need to compute the forces acting at each point from the grabbed point to the start point and to the end point of the suture. We define different scenarios as follows:

4.2.1 Condition A - no propagation

Assume the user grasps point P_{i+1} with one hand. If $l_{min} < l_i < l_{max}$. There is no propagation of the user input force $\mathbf{f}_h$ from point P_{i+1} to P_i. All the user input force has been converted to the internal forces along the suture (see Figure 4.2).

4.2.2 Condition B - one-hand pulling

Assume the user grasps the suture P_{i+1} at a single point. If the expected segment length $l'_i > l_{max}$ or $l'_i < l_{min}$, we need to adjust the segment length to l_{max} or l_{min} (see Figure 4.3).

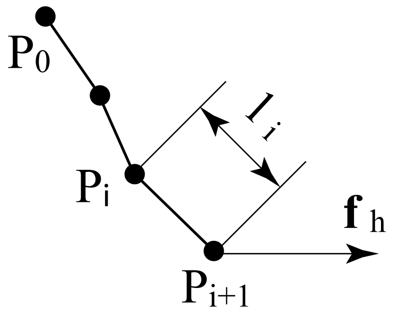


Figure 4.2: Condition A - no force propagation. Point P_i and P_{i+1} are two consecutive mass points. P_{i+1} is the grasped point. l_i is the current segment length between P_i and P_{i+1}. $\mathbf{f}_h$ is the user haptic input force.

Let $\mathbf{f}_p$ be the component of the input force $\mathbf{f}_h$ along the segment direction, and $\mathbf{f}_p$ is the input force propagated to point P_i from point P_{i+1}. $\mathbf{f}_p$ and $\mathbf{f}_m$ can be obtained from the following equations:

$$\mathbf{f}_p = (\mathbf{f}_h \cdot \hat{\mathbf{e}}_i)\hat{\mathbf{e}}_i, \tag{4.1}$$

$$\mathbf{f}_m = (\mathbf{f}_h \cdot \hat{\mathbf{e}}_m)\hat{\mathbf{e}}_m. \tag{4.2}$$

where

$$\hat{\mathbf{e}}_m = \frac{\hat{\mathbf{e}}_i \times \mathbf{f}_h}{||\hat{\mathbf{e}}_i \times \mathbf{f}_h||} \times \hat{\mathbf{e}}_i.$$

$\hat{\mathbf{e}}_i$ can be obtained from equation 2.14. Using the same method as above, we can derive the user input force propagated at each point of the suture.

4.2.3 Condition C - two-hand pulling

In this condition, we assume the user is pulling two points,P_k and P_i, of the suture. The method is almost the same as in condition B. But we need to compute twice the propagation of the input forces, first starting from point P_i, and then starting from point P_k (see Figure 4.4).

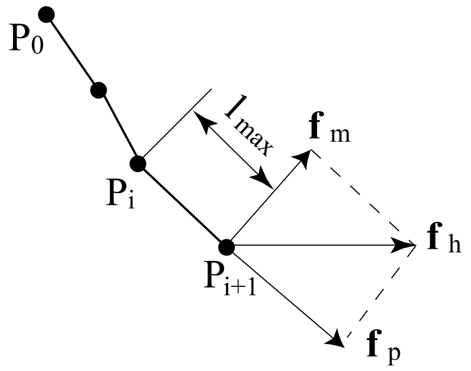


Figure 4.3: Condition B - pulling one point. Point P_i and P_{i+1} are two consecutive mass points. P_{i+1} is the grasped point. l_{max} is the max segment length between P_i and P_{i+1}. $\mathbf{f}_h$ is the user haptic input force. $\mathbf{f}_p$ is the force that be propagated. $\mathbf{f}_m$ is the force creating the movtion.

4.3 Collision Detection and Management

Collision detection and management is always the most difficult part in deformable object simulations. In this thesis, we use the similar to the one proposed in [39]. First, we build a bounding-volume hierarchy (BVH) from the bottom-up representing the shape of the suture at successive levels of detail (see Figure 4.5). To find the self-collisions of the suture, we explore two copies of the BVH from the top down. Whenever two BVHs (one from each copy) are found to not overlap, we know that they cannot contain colliding segments, and hence, we do not explore their contents. When two leaf spheres overlap, the distance between the two centers of the nodes is computed. If it is less than the node diameter, $2r$, then the two segments are reported to collide. However, no node is ever considered to be in collision with itself or its immediate neighbors along the suture chain.

To find the collisions between the suture and grippers, we consider the gripper as a triangle between two jaws which are line segments with a given radius, and check if the BHV of the suture has any overlap with this triangle. If intersection happened, compute the intersection point which will be the grab point.

For self-collison of the suture(see Figure 4.6), when two suture segments are de-

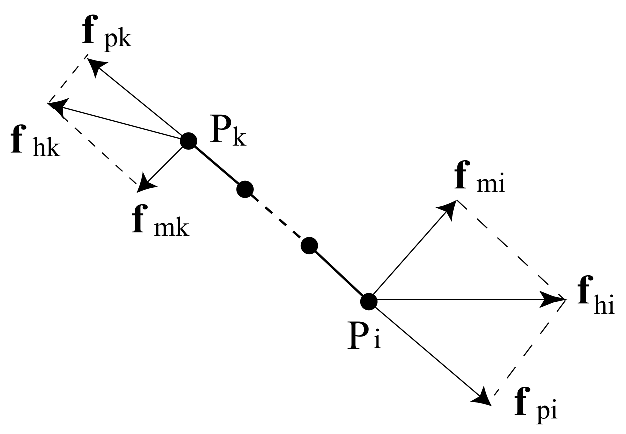


Figure 4.4: Condition C - pulling two points. P_k and P_i are the two grasped points. $\mathbf{f}_{hk}$ and $\mathbf{f}_{hi}$ are the user haptic input forces. $\mathbf{f}_{pk}$ and $\mathbf{f}_{pi}$ are the forces that be propagated. $\mathbf{f}_{mk}$ and $\mathbf{f}_{mi}$ are the forces creating the movtion.

tected to be at a distance $d < 2r$ from each other, then, an equal (but opposite) displacement vector is applied to each segment along. This displacement is just long enough to take the segments out of collision, with a slight "safety margin". Hence, each node is shifted away by $r - d/2 + \varepsilon/2$.

If a collision occurred, during real time simulation, we need to compute new velocities of mass points which are involved in the collision. Similarly to the method presented in [40], we apply impulses to the end points of these two segments. Let's take a look again at the picture of two contact segments (See Figure 4.7). Let assume point C with relative position a along the segment $\overrightarrow{P_aP_b}$ interacts with point E with relative position b along the segment $\overrightarrow{P_cP_d}$. Let $\mathbf{i}$ be the impulse, then, $\mathbf{i} = \mathbf{n}\Delta t$. where $\mathbf{n}$ is the repulsion force that we can obtain from equation 2.9. Then we can

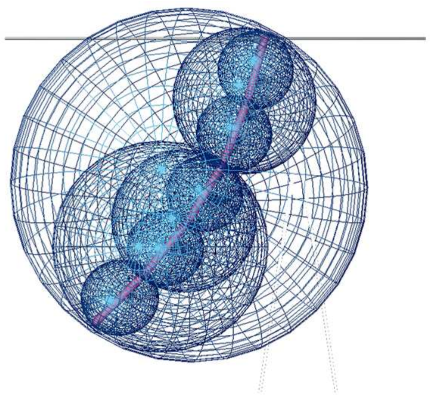


Figure 4.5: Bounding-Volume Hierarchy

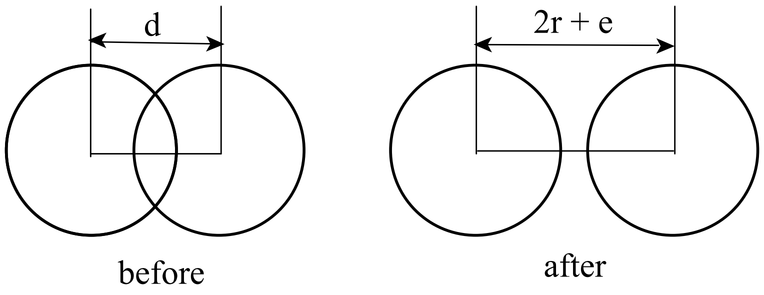


Figure 4.6: Collision management of two suture segments. d is the distance between the center lines of the two contact segments. r is the radius of the segment. ε is the adjust constant.

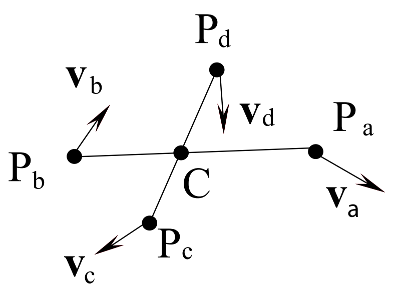


Figure 4.7: Two suture segments P_aP_b and P_cP_d are sliding on each other, where point C is the contact point; $\mathbf{v}_a$, $\mathbf{v}_b$, $\mathbf{v}_c$, and $\mathbf{v}_d$ are the velocities of points P_a, P_b, P_c, and P_d respectively.

compute the new velocities as follows:

$$i' = \frac{2||\mathbf{i}||}{(a^2 + b^2 + (1-a)^2 + (1-b)^2)}, \tag{4.3}$$

$$\mathbf{v}_a^{new} = \mathbf{v}_a + (1-a)\frac{i'}{m}\hat{\mathbf{n}}, \tag{4.4}$$

$$\mathbf{v}_b^{new} = \mathbf{v}_b + a\frac{i'}{m}\hat{\mathbf{n}}, \tag{4.5}$$

$$\mathbf{v}_c^{new} = \mathbf{v}_c - (1-b)\frac{i'}{m}\hat{\mathbf{n}}, \tag{4.6}$$

$$\mathbf{v}_d^{new} = \mathbf{v}_d - b\frac{i'}{m}\hat{\mathbf{n}}. \tag{4.7}$$

where m is the mass of each mass point P_a, P_b, P_c, and P_d; $\hat{\mathbf{n}}$ is the unit vector from point E to point C.

4.4 Haptic Force Feedback

In the previous section, we have discussed to calculate the user input force by introducing virtual coupling technique (see Figure 4.8).

Change the direction of the input force to the opposite, we can get the output force which we need to feed the haptic device. Point P is the real position of the end factor and point Q is the grabbed point. By employing a spring-damper between a simulated body and the device end-effector, we can make the output force as smooth

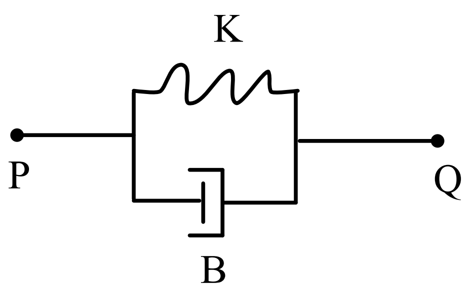


Figure 4.8: Virtual coupling. Point P is the real position of the end factor and point Q is the grabbed point. K is the spring. B is the damper working against the spring.

as possible. We can also adjust the K and B to satisfy the out put requirements for different haptic devices.

In order to study the details inside our suture model, we took the 15th node as an example and plot the spring force, spring damper, torsional spring, torsional damper, and swivel damper acting on it when the suture swings freely (see Figure 4.9 to Figure 4.13).

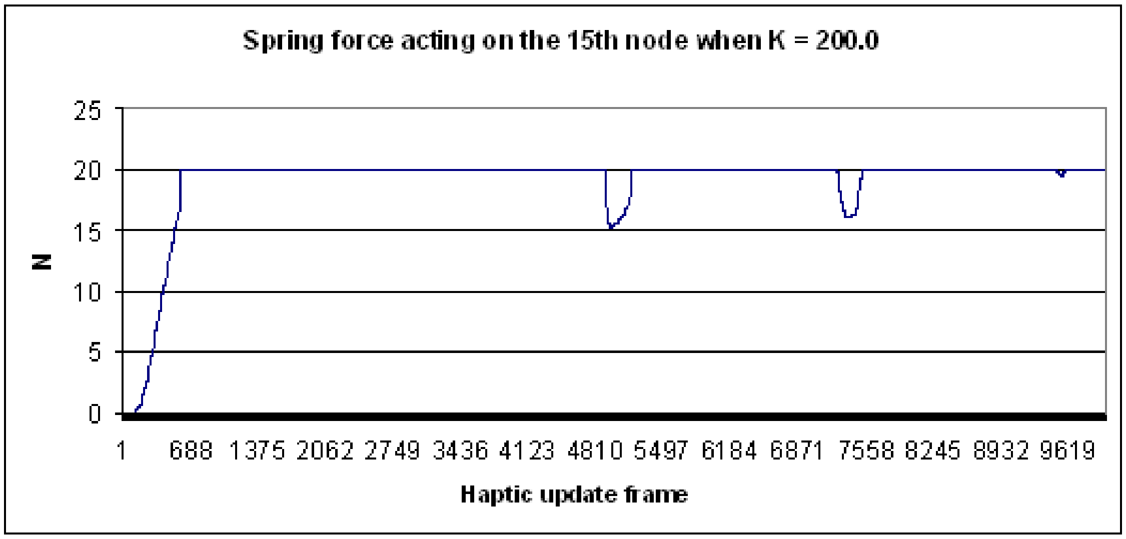


Figure 4.9: Plot of the spring force acting on one mass node

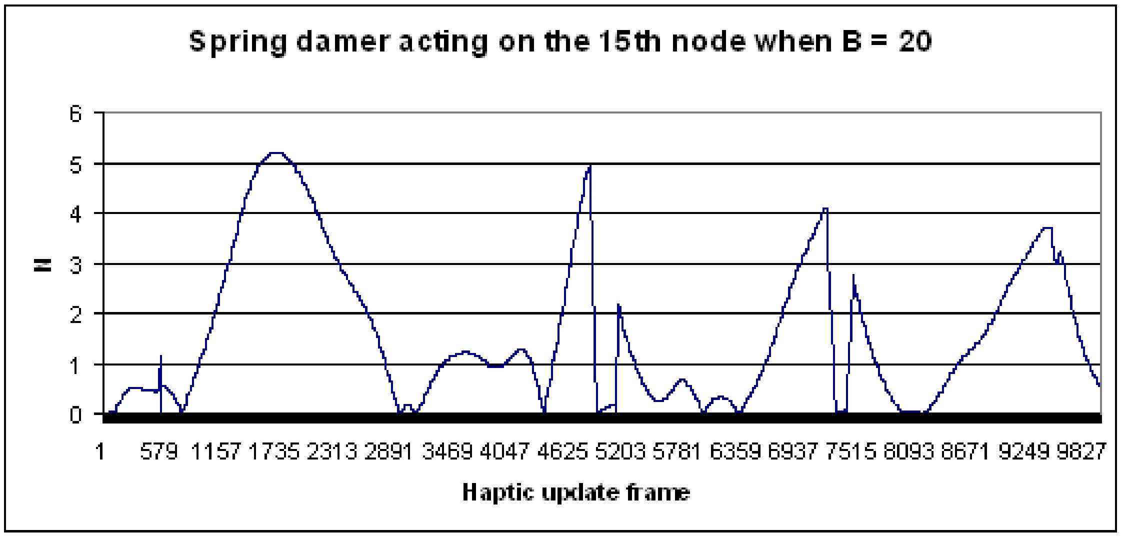


Figure 4.10: Plot of the spring damper acting on one mass node

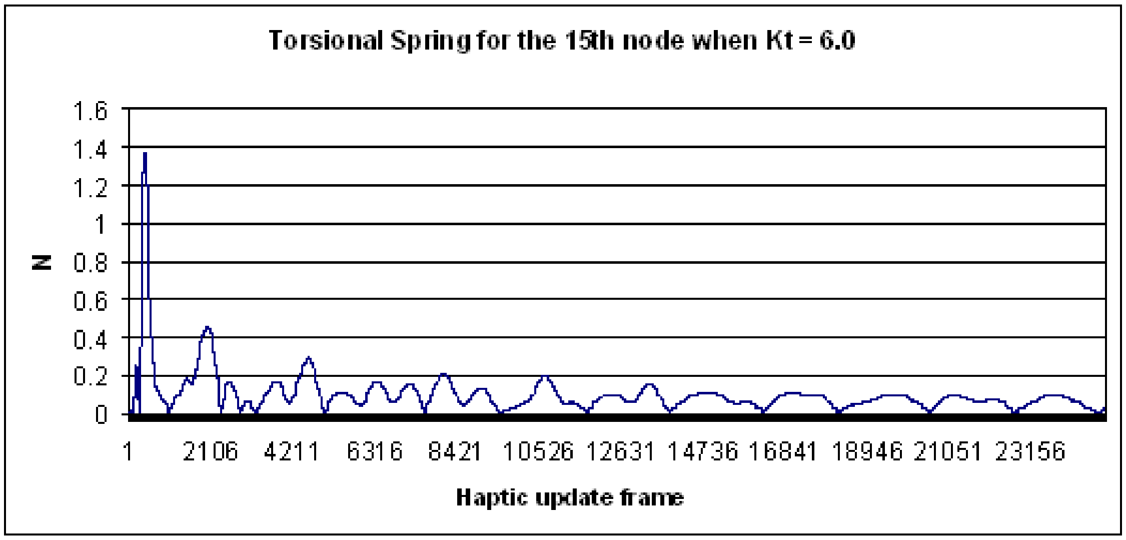


Figure 4.11: Plot of the torsional spring acting on one mass node

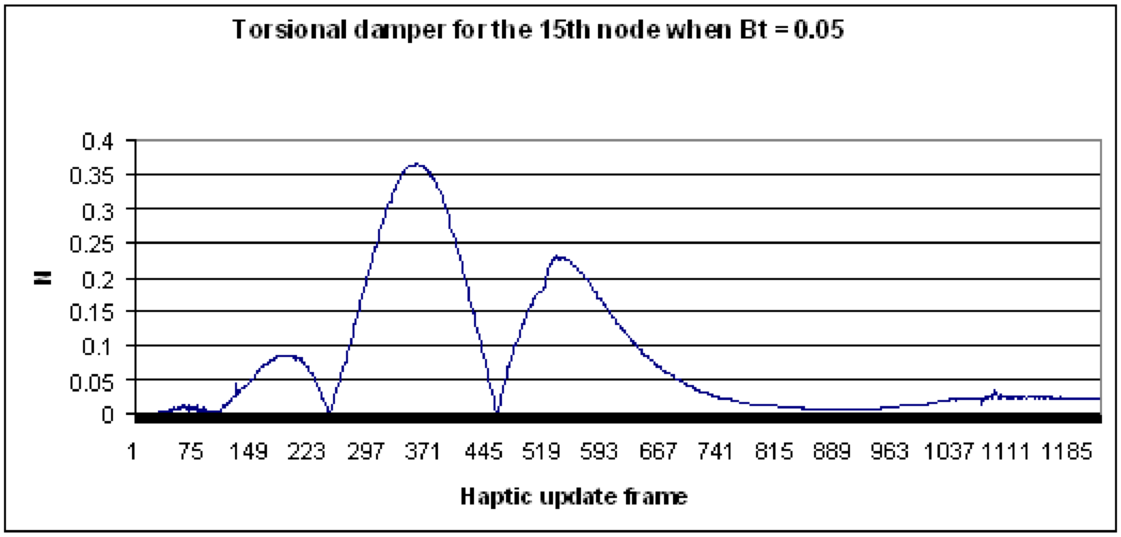


Figure 4.12: Plot of the torsional damper acting on one mass node

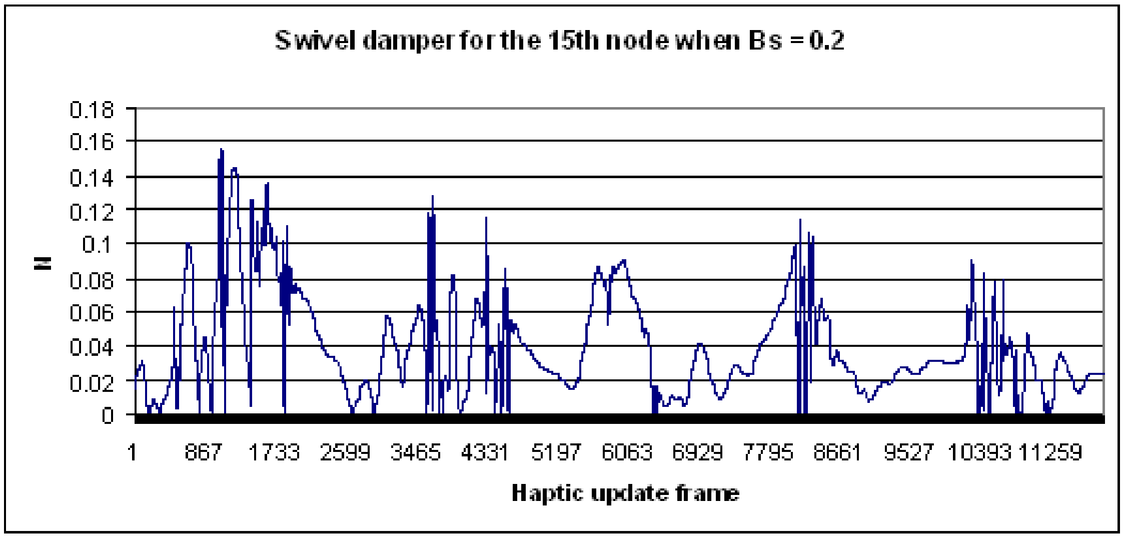
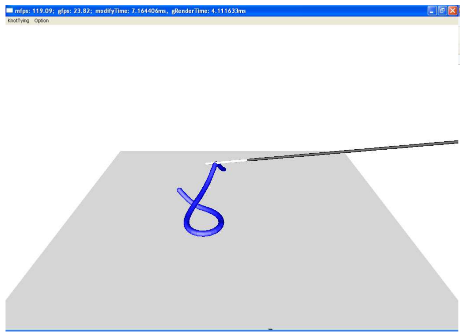


Figure 4.13: Plot of the swivel damper acting on one mass node

To demonstrate how the friction force changes when the friction constant is changed, we plot the friction forces when the suture is colliding itself and changed the friction constant (see Figure 4.14). Figure4.15 to Figure 4.18 are the friction force plots when $\mu = 0.1, 0.5, 1.0, 2.0$



```
mfps: 119.09; gfps: 23.82; modifyTime: 7.164406ms, gRenderTime: 4.111633ms
```

Figure 4.14: Screen shot of suture colliding. The user is gasping the suture and making it collide over itself. This is to demonstrate how friction force changes when the friction constant is changed.

Because the maximum exertable face for PHANTOM Omni is 0.75lbf (3.3N), we can not output the forces to the Haptic devices from virtual coupling spring directly. Therefore, we chose a constant equal to 0.003 to scale the forces before we feed them to PHANTOM Omnis. We plot the forces which we send to PHANTOM Omni during each haptic update frame for one-hand pulling and two-hand knotting cases. Taking the magnitudes of the forces as y-axis and each haptic update frame as x-axis, we obtain the forces plots as in Fig. 4.19 to Fig 4.23

4.5 Experiment

4.5.1 Setup

Our simulation was implemented on a PC with dual 3.2G Intel®Pentium®4 CPUs and 512 MB memory. For physics-based models, the most challenging part is how to determine its parameters. If parameters are inappropriate, it may impact the whole

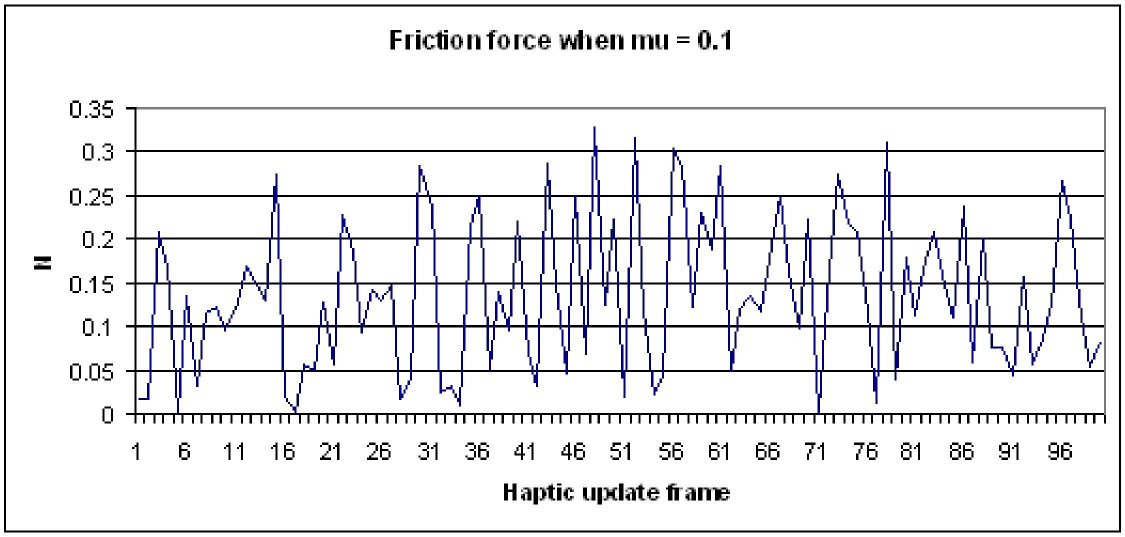


Figure 4.15: Friction plot when $\mu = 0.1$

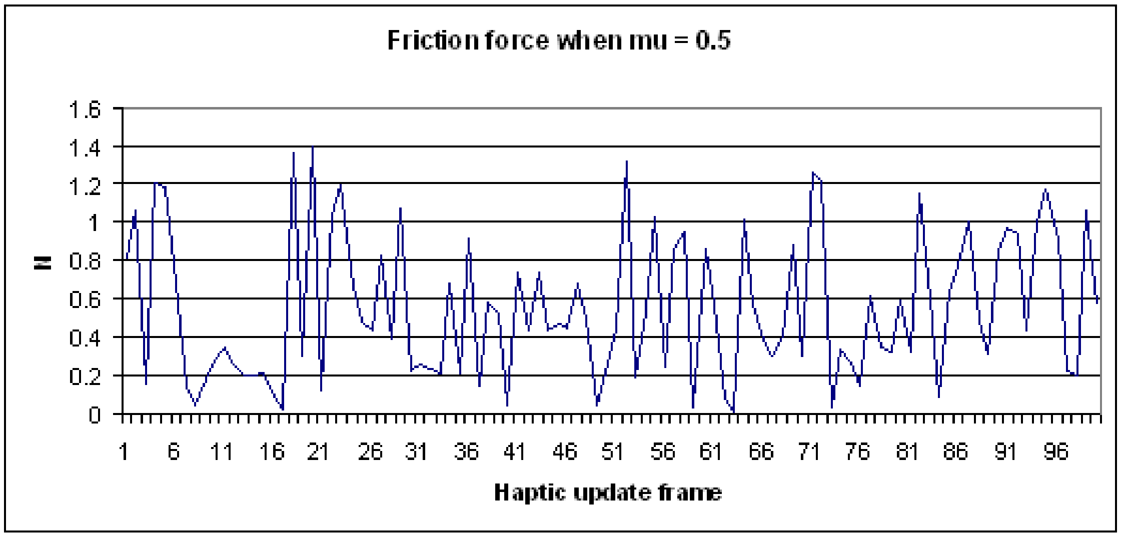


Figure 4.16: Friction plot when $\mu = 0.5$

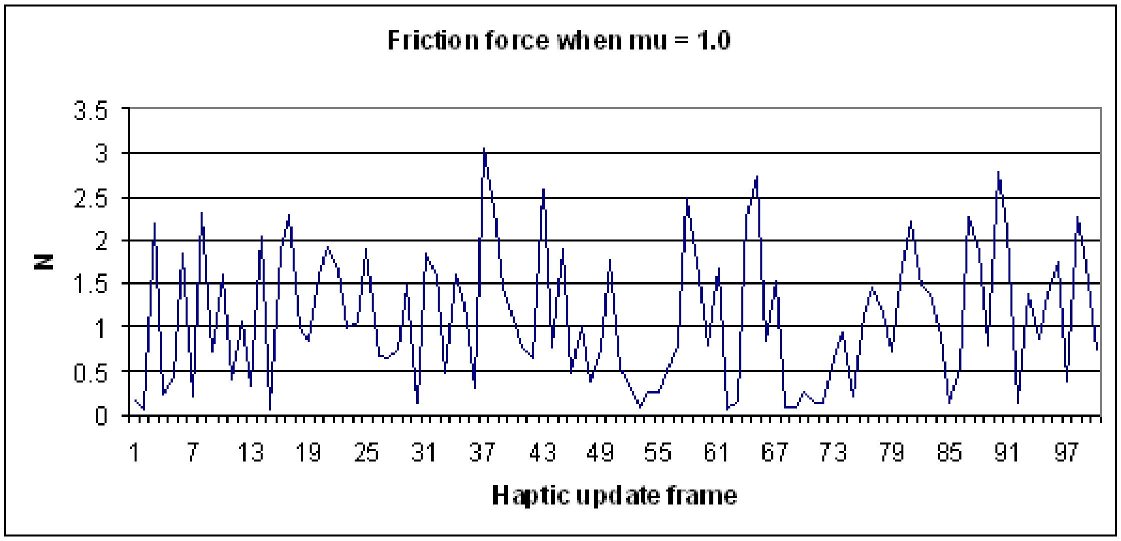


Figure 4.17: Friction plot when $\mu = 1.0$

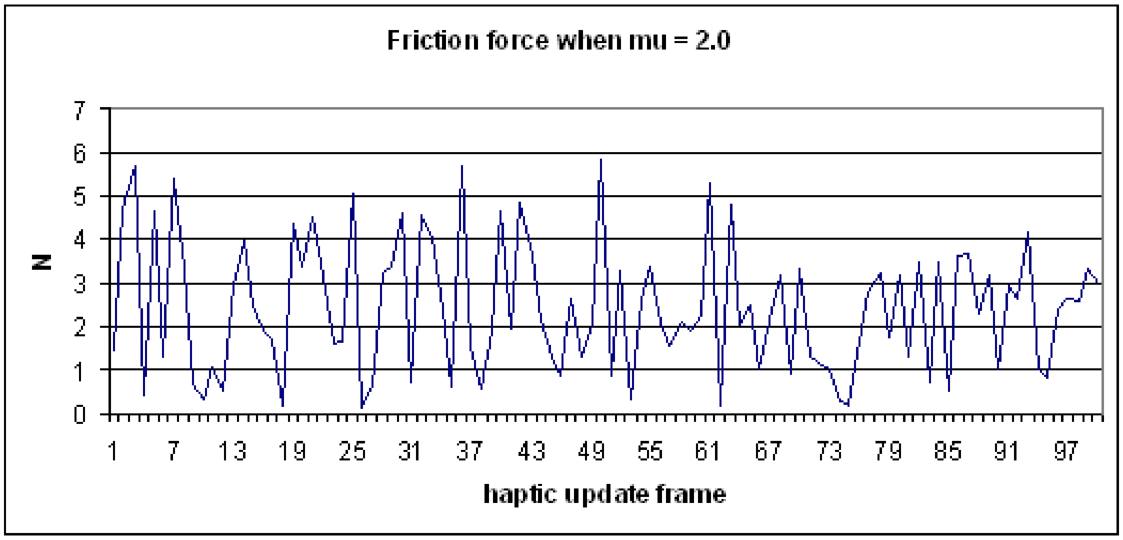


Figure 4.18: Friction plot when $\mu = 2.0$

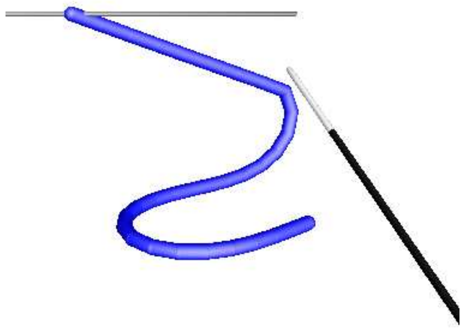


Figure 4.19: Screen shot of one-hand pulling

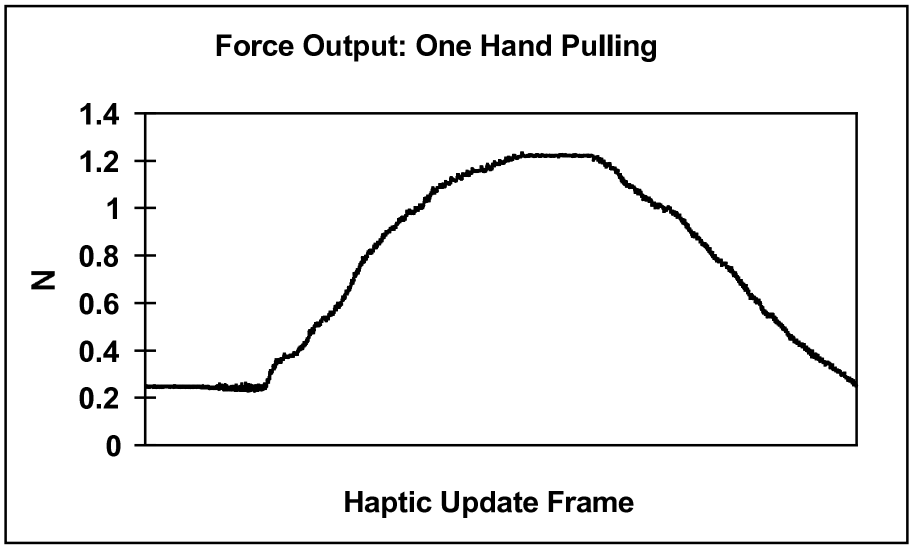


Figure 4.20: Output force plot of one-hand pulling

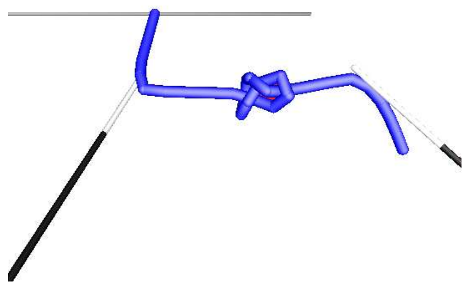


Figure 4.21: Screen shot of knotting

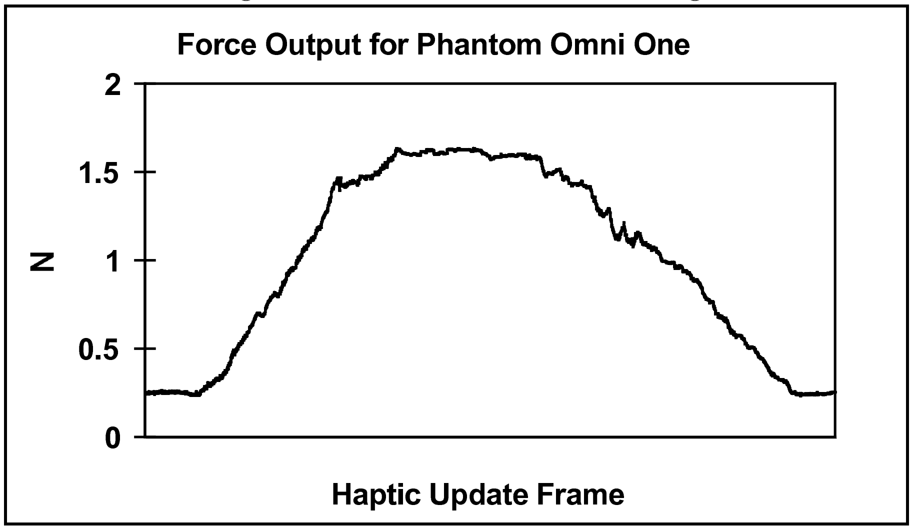


Figure 4.22: Output force plot of left hand when pulling the suture with two Phantom Omis

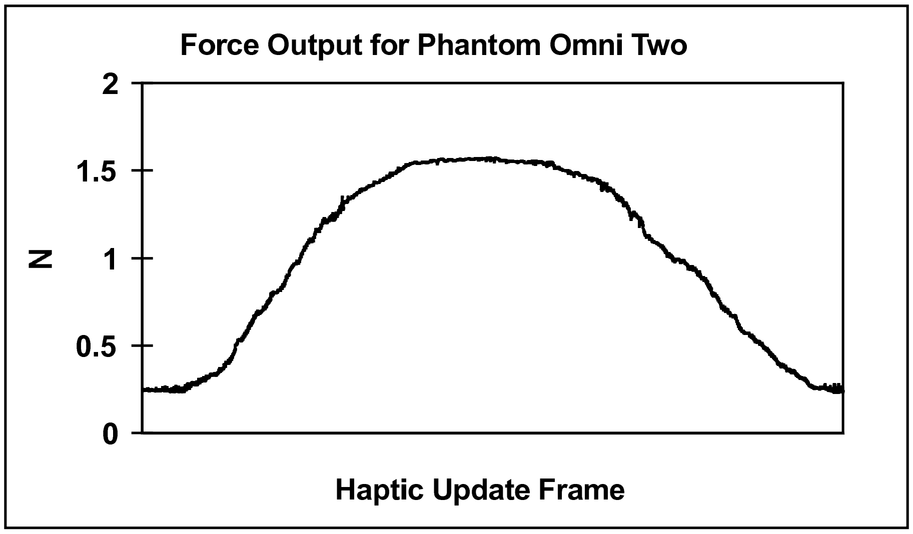


Figure 4.23: Output force plot of right hand when pulling the suture with two Phantom Omis

system's stability or even over its limits. After many experiments, we chose our suture parameters as in Table 4.1:

Table 4.1: Suture Parameter Setting

Parameter	**Value**	**Remarks**
N	$20 \sim 50$	Number of Points
l	$0.5m$	Length of the suture
r	$0.005m$	Radius of the suture
m	$0.05kg$	mass of one point
G	$9.8N/kg$	Gravity
k_h	1200	Virtual coupling spring constant
s	0.003	Scale factor for output force
μ	10	Friction constant
k_{rs}	100	Repulse spring constant
k_{rd}	5	Repulse spring damper constant
k_l	800	Linear spring constant
k_d	1	Linear damper constant
k_{ts}	10	Torsional spring constant
k_{td}	0.05	Torsional damper constant
k_{sw}	0.2	Swivel damper constant

With the parameters above, we can obtain around $500Hz \sim 1000Hz$ update rate for both Phantom Omnis. Users can feel the output forces of smooth quality.

4.5.2 Experiment of Knotting

We build five different models with various combinations of forces models described in chapter 2. With two PHANTOM Omni haptic devices, users can tie an arbitrary knot about the suture which is hung up on one fixed frame.

Model 1

This model contains only a linear spring and a linear damper. It is the least realistic model. The two connected segments can bend to any angle effortlessly (see Figure 4.24).

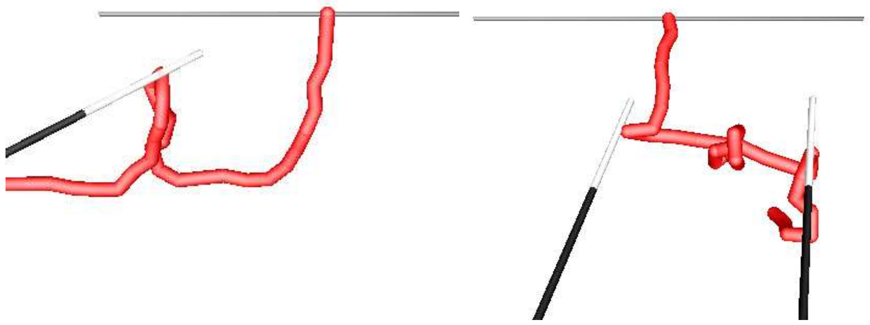


Figure 4.24: Suture model 1. Left: the user is manipulating the suture with one Phantom Omin. Right: the user is tying a know with two Phantom Omnis.

Model 2

This model is almost the same as model 1, but also contains a torsional spring. The torsional spring adds a lot more realistic behaviour to the suture, but also, because it uses a nonlinear function 'acos', it creates some harmonic wave motions (see Figure 4.25).

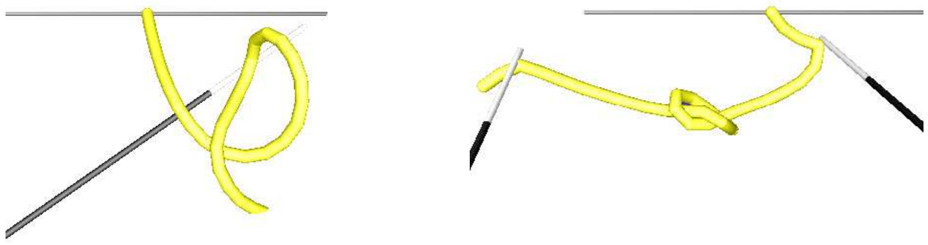


Figure 4.25: Suture model 2. Left: the user is manipulating the suture with one Phantom Omin. Right: the user is tying a knot with two Phantom Omnis.

Model 3

Compared to model 2, a torsional damper has been added to this model. This damper stops the harmonic motion presented in model 2. But this model creates another class of instability where it is very sensitive to the suture and creates a self-excitation phenomenon (see Figure 4.26).

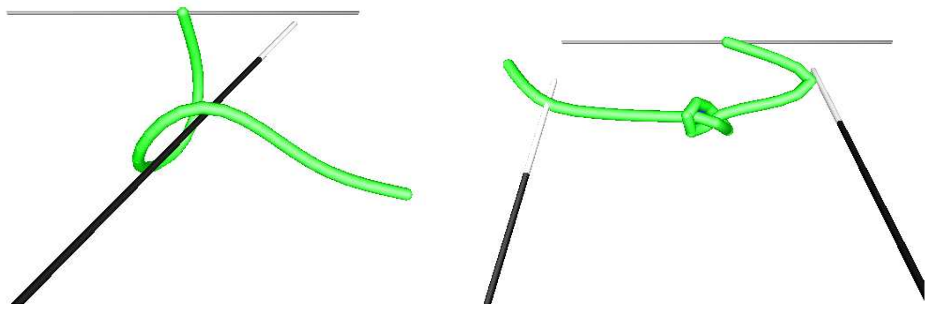


Figure 4.26: Suture model 3. Left: the user is manipulating the suture with one Phantom Omin. Right: the user is tying a know with two Phantom Omnis.

Model 4

This model includes a ‘swivel’ damper to fix the problem of perpetual orbiting (the self excitation mentioned in the above). The result is a suture that looks more like a real suture (see Figure 4.27).

Model 5

This model has all the components of model 4. The only difference is that the linear spring’s force computed quadratically on the difference between its current length and rest length, instead on linearly. This makes the suture appear a lot less stretchy, which is more realistic since the real sutures stretch very little. The suture’s non-linear response also makes it a lot more responsive to movements (see Figure 4.28).

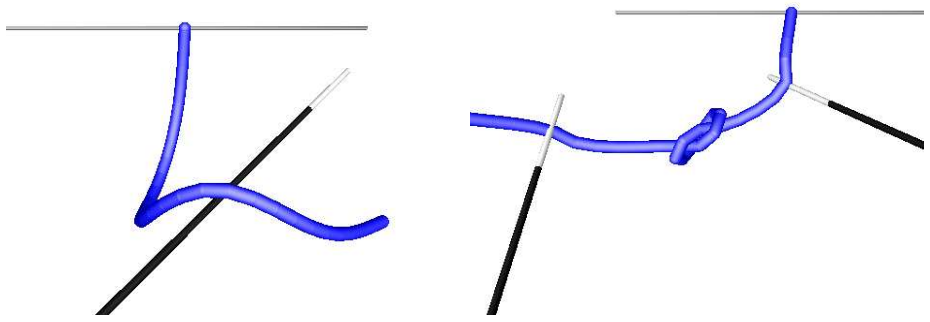


Figure 4.27: Suture model 4. Left: the user is manipulating the suture with one Phantom Omin. Right: the user is tying a know with two Phantom Omnis.

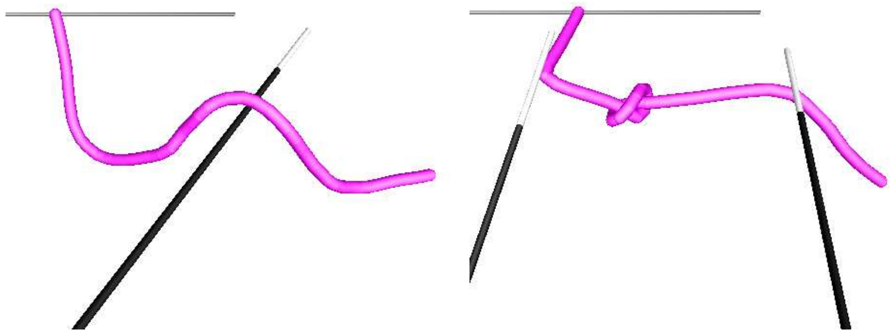


Figure 4.28: Suture model 5. Left: the user is manipulating the suture with one Phantom Omin. Right: the user is tying a know with two Phantom Omnis.

Comparing the results from above five different models, we can draw a conclusion that model 4 is the most ideal model for our surgical training environment.

4.5.3 Experiment of Unknotting

Same as the knotting experiment, the suture is hung up on one fixed frame. Also, to make knotting and unknotting easier, we set up a surface under the suture model to let part of the suture lay on the desk. In order to untie a knot successfully, we have to pick up the right point, otherwise the knot could be more tightening instead of loosening. This is part of the unknotting planning algorithm which will not be discussed here. Figure 4.29 show the successful unknotting of a over-hand knot and a figure-of-eight knot. Figure 4.30 shows if you grab the wrong point, the knot can not be untied.

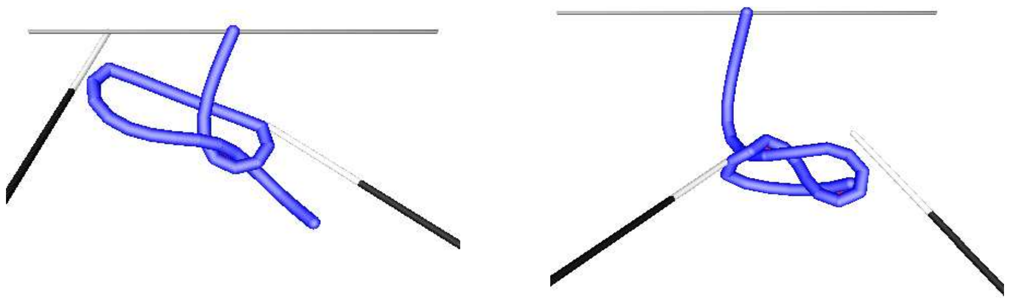


Figure 4.29: Success unknotting

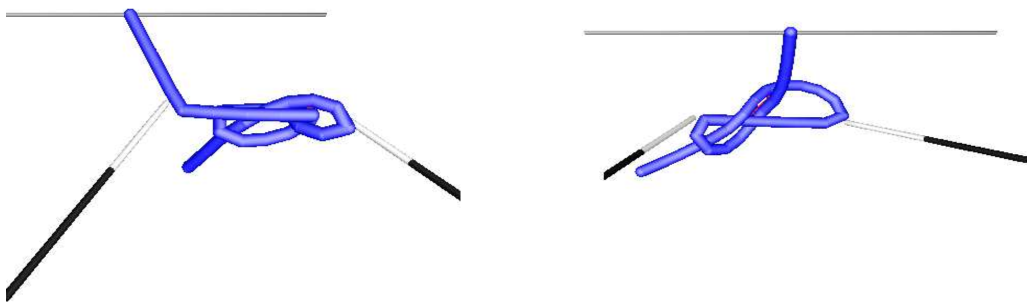


Figure 4.30: Unsuccessful unknotting

Chapter 5

GPU in Deformable Object Simulation

5.1 Introduction

In simulating complex interaction betweem deformable objects, such as the one used in a typical surgical simulators, achieving the smooth force feedback through haptic devices has been a challenging task. To offer the user a smooth haptic sensation, we have to guarantee the haptic rendering rate to be around 1000Hz. In most cases, the main issue associated with the unstable behavior is due to the computational demand associated with the model of deformable objects. Such models offer computational complexities which then render the overall system to a low servo rate and a computational time delay. Regardless of the mechanical bandwidth of the haptic device, in most cases, the computational overhead has been a key factor in defining the upper bound for the fast haptic rendering system. With the demand of creating more complex and realistic scenes for the user interaction, there exists opportunities for the haptic system engineers to design and experiment new computational environment which can blend the traditional computational mechanics models with more dedicated hardware utilities.

In this chapter, we present some results regarding utilizing GPU processing units for computing the deformation of two experimental objects. We present a suture simulation model with GPU and a 2D deformable cloth model with the nVidia CUDA techniques. In all of these models, a simplified concentrated system has been used. Both of these models were implemented and numerical solved at CPU and then GPU level. We conducted experimental studies to compare the GPU-based suture models and the CPU implementation. We experimented with the implicit model of the 2D mesh which offer similar computational challenges associated with any Finite-Element modeling appraoches. We also proposed a method for computing the inverse of a matrix with truncated Numan series and the nVidia CUDA technology. The experiments results show that we could take advantages of GPU's tremendous parallel computational power in performing vector and matrix algebra, and make it possible to achieve high sampling rate and smooth haptic feedback when considering complex interaction with deformable objects.

5.2 Deformable Linear Object (DLO) Simulation with GPU

A novel mechanics-based DLO suture model is presented in Chapter 2, which is based on the modified version of linear finite element model (See Figure 5.1).

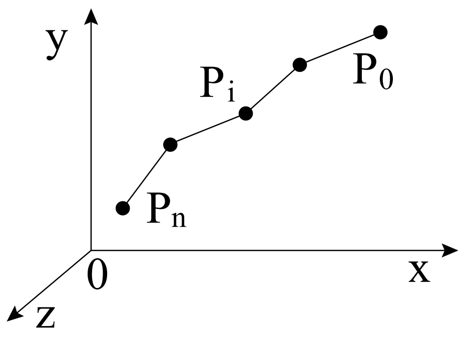


Figure 5.1: 1-D Suture Model

This model consists of a sequence of lumped points laying on the centerline of the suture. This model can represent the mechanical properties of a real object, such as stretching, compressing, bending, and twisting. Not only external forces including the friction force, the gravity, and the user input force, can be included in this model, but also the model can include the internal forces including the linear spring, the linear damper, the torsional spring, the torsional damper, and the swivel damper.

One important issue about the dynamic simulation in the context of haptic rendering is the real time efficiency. However, one of the computational challenges is the increased number of vector and scalar products to increase the mechanics-based realism in the haptic rendering. In addition, to increase the realism in the mechanics-based representation, one also needs to increase a number of the linear elements associated with the suture model. As a result, even for inclusion of a single model of the suture in the scene, it can be seen that a desirable haptic rate can not be guaranteed and the user can feel the output force feedback to be less smooth. Here, We utilize a GPU implementation for addressing the computational issues of the suture that offers a suitable parallel matrix and vector computation capabilities.

5.2.1 GPU Cg Implementation

We developed two fragment shaders using nVidia Cg shading language to simulate the suture dynamically: velocity() - to calculate the velocity of each mass point; position() - to calculate the position of each mass point.

Figure 5.2 is the flow char of the main program. It is based on the Virture Training Environment which has been developed at Experimental Robotics Lab.

The initialization of Cg and Glew and creating texture happen after the initialization of object. Activating the fragment shaders to complete the computation is inside the function Modify Objects. The detail flow charts of these two parts are as follows (See Figure 5.3):

The following experiment was conducted on a MacBook Pro (Core 2 Duo, 2G RAM, ATI Mobility Radeon X1600 256M) with Microsoft Windows XP platform. The whole program is developed with Microsoft Visual Studio VC^{++}6. Glew extension library and the Cg Toolkit are also required.

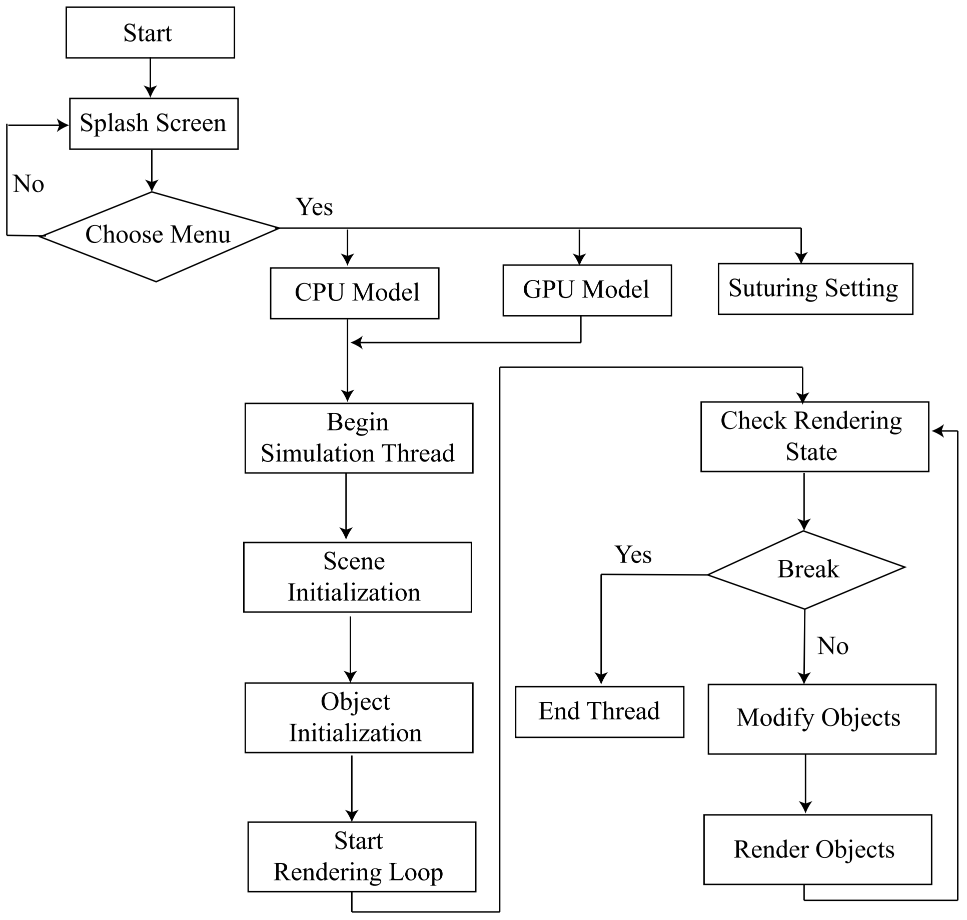


Figure 5.2: Flow chart of main program

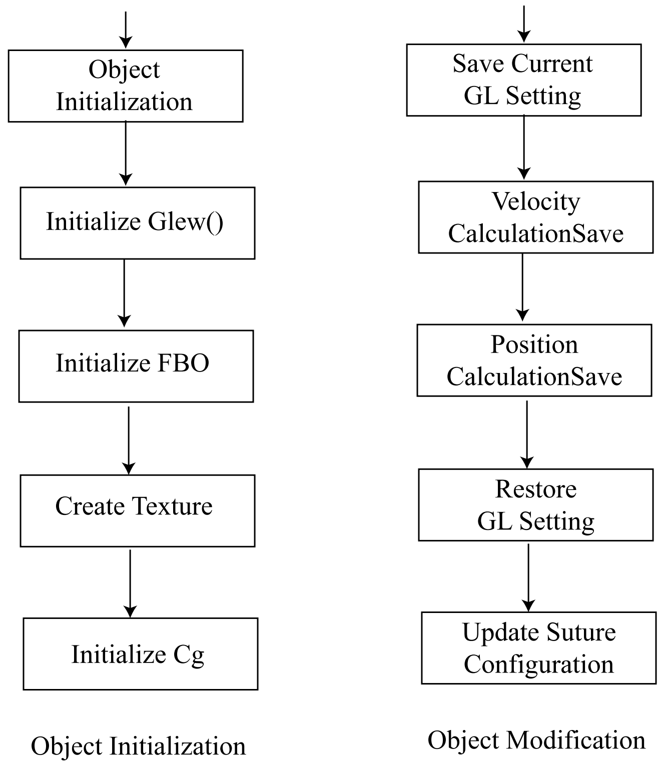


Figure 5.3: Flow chart of GPU part

By changing the number of segment (element) of the suture, we record the expected computational time. We implemented the same computational implementation on both CPU-based and GPU-based environment. To compare the performance, we recorded the time for each computation cycle. Because the variable response time at every cycle and other related implementation overheads we have recorded both the maximum and minimum update rates during a sample haptic manipulation of the suture. In all of these experiments, the suture is held and manipulated by the user with a haptic device from SensAble Technologies Inc. (Omni device). Table 5.1 demonstrate sample results of this experiment for different number of suture elements:

Table 5.1: GPU and CPU performance comparison of the suture model

Segment Number	GPU Max (ms)	GPU Min (ms)	CPU Max (ms)	CPU Min (ms)
10	0.424	0.265	0.153	0.088
20	0.453	0.276	0.284	0.167
30	0.463	0.263	0.408	0.224
40	0.435	0.262	0.534	0.296
50	0.428	0.261	0.637	0.362
60	0.427	0.263	0.765	0.445
70	0.433	0.256	0.888	0.505
80	0.436	0.258	1.017	0.575
90	0.430	0.259	1.130	0.640
100	0.436	0.258	1.670	0.717

5.3 Cloth Simulation with CUDA

Generally speaking, the big part of the dynamic simulation is about how to solve the differential equations, which describe the relationship between an unknown function and its derivatives. Most of those methods used by today's researchers could be put into two categories: explicit method and implicit method. This section will give out the introduction of explicit and implicit methods. Explicit methods, such as Euler's method or the midpoint method, face some implementation difficulties with some classes of ordinary differential equations (ODEs). In some cases, an ODE can become "stiff", in which case explicit methods can results in instability. On the other hand, the implicit method can offer stable solution to general class of ODEs but with the added computational complexities. More details please see Appendix C.

[41] presents an implicit Euler's method to simulate cloth-like objects. It proposes an approach to finding a balance between system stability and efficiency and therefore enables robust interactive animation or real-time design and seeming of cloth. Unfortunately, the method requires computation of the inverse of a large sparse matrix. If the configuration of the cloth changes during the each cycle of simulation (i.e. a time variant model), we need to compute the inverse of the matrix in real-time, which can result in a very computationally intensive procedure especially for a very large 2D model of the cloth or shell type objects. In this section, we also present a method using the truncated Nuemann series to approximate the inverse of a very large matrix, and then implement it on nVidia CUDA technology.

5.3.1 Explicit Method vs Implicit Method

In real time dynamic simulations, system stability is one of the most important issues. It is really frustrating that your simulation crashes because your simulation step size is too big. In some cases such as deformable object simulations in surgical training systems or cloth simulations in game developments, which demand very high computation, there is no place left at all because you the step size is already so small. Therefore we must find a new simulation scheme which both could guarantee system stability and is not limited by simulation step size.

The basic idea of explicit integration is that the integrators are written in a way that all unknown values (e.g., all particle velocities, positions) can be updated in a loop independently. Let's take an example as in the case of cloth simulations, the particle positions are treated as being decoupled and are not considered to affect each other. But in the real word, it is apparently not so: the particles in a cloth are coupled and connected together by the cloth. The motion of one particle is always affected by all the others. For a given time step, explicit integrators actually move the particles out-of-sync with each other, and put each particle slightly in the wrong place. However, they do become coupled again when the forces and constraints are set up prior to the next physics step. The forces and constraints are also slightly wrong because the particles are in the wrong place, which will cause a small correction during the next physics step. This small correction is to try to put the particles in the right place. But in this viscious cycle, the next physics step overcorrects and the particles again are in the wrong place. Then, the cycle repeats. Explicit method guarantees a certain order of accuracy, but loses stability. The possible size of time step is restricted, which will result in a loss of computational efficiency.

In some cases, this viscious cycle actually runs well and the simulation doesn't crash, although the results are always wrong. In other cases (e.g., stiff ODEs), the cycle doesn't work at all. The corrective forces work against the overshot positions, which will cause an unstable simulation. once particle positions and velocities grow without limit, overflow happens in just a few physics steps.

On the other hand, implicit integrators treat the unknown variables as coupled, which are solved together as a system at each time step. The big advantage is that implicit schemes are stable for any time step size. However, they require the solution of a generally non-linear set of equations at each time step.

Implicit methods is based on taking a *backward* Euler step. As it is highligted in [41] and [42] the main computational form can be written as:

$$(\mathbf{I} - \frac{dt^2}{m}H)\triangle^{n+1}\mathbf{v} = (\mathbf{F}^n + dtH\mathbf{v}^n)\frac{dt}{m}. \tag{5.1}$$

where: $\mathbf{I}$ is the identity matrix; m is the mass of each mass node; $H = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}$ is actually the negated Hessian matrix of the system.

We simulate the 2D surface model as a lumped mesh system shown in Figure 5.4 For implicit Euler's method, we need to calculate the inverse of the stiff matrix which is also a negative Hessian matrix. For example, according to the connectivity

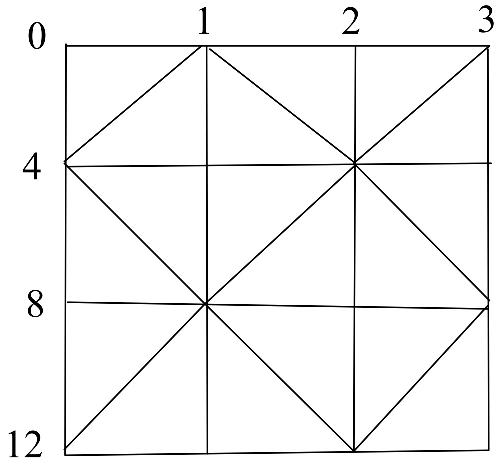


Figure 5.4: Cloth Model

relationship between nodes, the Hessian matrix of figure 5.4 can be written as:

$$H = k \begin{bmatrix} -2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & . & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -5 & 1 & 0 & 1 & 1 & 1 & 0 & . & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -3 & 1 & 0 & 0 & 1 & 0 & . & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & . & 0 & 1 & 0 & 0 & 1 & -3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & . & 0 & 1 & 1 & 1 & 0 & 1 & -5 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & . & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -2 \end{bmatrix} \quad (5.2)$$

The computationally challenging part of any implicit deformation modeling is to calculate the inverse matrix. For example, in our case the matrix is defined as shown in Eq. C.11:

$$W = (\mathbf{I} - \frac{dt^2}{m}H)^{-1}$$

Let us define $A = (\mathbf{I} - \frac{dt^2}{m}H)$, then we can show A is symmetric positive definite matrix. We write A as:

$$A = D - L - L^T \quad (5.3)$$

where D is a diagonal matrix, the minus signs before L and L^T are just a technical convenience. Then, we symmetrically scale A by its diagonal,

$$\begin{aligned} A &= D^{\frac{1}{2}}(I - D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}})D^{\frac{1}{2}} && (5.4) \\ A^{-1} &= D^{-\frac{1}{2}}(I - D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}})^{-1}D^{-\frac{1}{2}} && (5.5) \end{aligned}$$

We note that:

$$D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \quad (5.6)$$

$$\rho(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}) = \rho(I - D^{-1}A) \quad (5.7)$$

If $\rho(I - D^{-1}A) < 1$, then we can expand $(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})$ in Neumann series.

Let's define $\rho(A)$ to be the spectral radius of A, such that,

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i|$$

where λ_i are the eigenvalues of A. We note that:

$$D^{-\frac{1}{2}}(L + L^T)D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$$

and

$$\rho(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}) = \rho(I - D^{-1}A)$$

If $\rho(I - D^{-1}A) < 1$, then we can expand $(I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}})$ in Neumann series. One approach to generate a polynomial pre-conditioner is to use a few terms of the Neumann series. In this thesis, we only use the first order Neumann polynomial:

$$\begin{aligned} W &\approx D^{-1} + D^{-1}(L + L^T)D^{-1} && (5.8) \\ &= 2D^{-1} - D^{-1}AD^{-1} && (5.9) \end{aligned}$$

5.3.2 CUDA Implementation

CUDA is an extension to the C programming language. The goal of the CUDA programming interface is to provide a relatively simple interface for users familiar with the C programming language to easily write programs for execution by the device. To specify if the functions run on the device or on the host, CUDA defines

"Function Type Qualifiers" as: _device_ qualifier which declares a function is executed on the device and callable from the device only; _global_ qualifier which declares a function as being a kernel which is executed on the device and callable from the host only; _host_ qualifier which declares a function is executed on the host and callable from the host only.

The complete computation can be divided into two parts: the first part is to calculate the inverse of the Hessian matrix; second part is to compute the velocity and position of each node. Since for a given physical construction of the cloth, the construction of the Hessian matrix remains unchanged, (we discard the non-linear part of the force during the simulation), we can pre-calculate the inverse of the Hessian matrix off-line. The remaining of the computation is done by GPU at real-time.

We developed two CUDA kernels for the implicit computation: inverseMatrix() - to calculate the inverse of the Hessian matrix; implicitDeformation() - to compute the force, velocity, and position of each mass node. The flow chart of the main program is the same as Figure 5.2. The flow chart of how to call inerseMatrix() is shown as the follows:

Computation of the inverse of hessian matrix
1. Build H matrix of the cloth according the mass-node relationship 2. Computer $A = \mathbf{I} - \frac{dt^2}{m}H$ 3. Computer D^{-1} where D is the diagonal matrix of W 4. Allocate device memory for matrices H, W, and D^{-1} 5. Copy host memory to device memory 6. Set up CUDA execution parameters 7. Execute the kernel to calculate $W = A^{-1} = 2D^{-1} - D^{-1}AD^{-1}$ 8. Copy device memory to host memory 9. Clean up device memory

The flow chart of how to call implicitDeformation() is shown as the follows:

Implicit deformation at each time step
1. Load node data and spring data to GPU
2. Set barycenter to zero $\mathbf{x}_G = 0$
3. Set angular momentum to zero $\delta\tau = 0$
4. Force clear
5. Compute the force acting on each mass node
6. Filter the force with the inverse of the Hessian matrix
7. Post correction of angular momentum
8. Post-step process
9. Collision detection and management
10. Copy device memory to host memory
11. Clean up device memory

Figure 5.5 shows a typical experiment. The sphere is a obstacle placed on the floor. When the simulation starts, the cloth will fall down because gravity and collide with the sphere. Figure 5.6 shows when the cloth starts colliding with the obstacle. Figure 5.7 shows the cloth is sliding on the sphere.

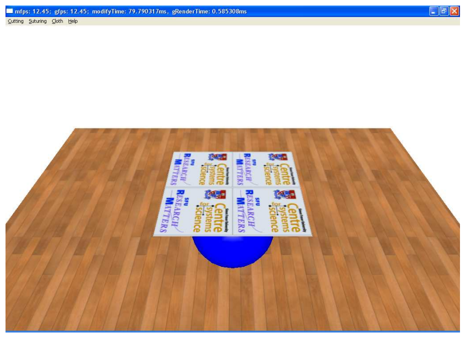

mfps: 12.45; gfps: 12.45; modifyTime: 79.790317ms, gRenderTime: 0.585308ms

Figure 5.5: Initial configuration

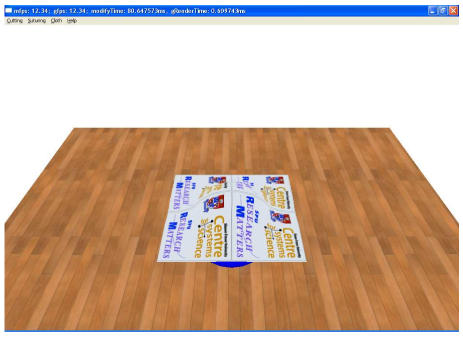

```
mfps: 12.34; gfps: 12.34; modifyTime: 80.647573ms, gRenderTime: 0.609743ms
```

Figure 5.6: the model starts colliding with the obstacle

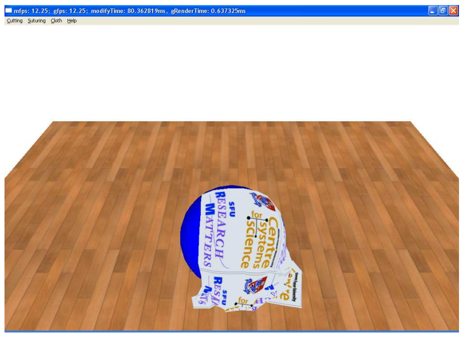

```
mfps: 12.25; gfps: 12.25; modifyTime: 80.362819ms, gRenderTime: 0.637325ms
```

Figure 5.7: The 2D model is colliding on the sphere

5.3.3 Haptic Force Feedback

Haptic rendering is still done by CPU. We use virtual coupling technique as discussed in [43] to compute the haptic force feedback when the cloth is manipulated by the user. This is due to the fact that the input variables to our cloth simulation model is a force vector which is then resolved to define the resultant cloth nodal displacements vector. In our studies, we have utilized a SensAble PHANTOM Omni® device where its position is mapped to the resultant force vector between its tip and the contact node on the cloth through the virtual coupling.

The flow chart of haptic rendering is shown as the follows:

Haptic rendering at each time step
1. Collision detection - if the cloth is grabbed by the user
2. Calculate the user input force with virtual coupling technique
3. Update the velocity and position of the grabbed node
4. Scale the user input force and output it to the haptic device

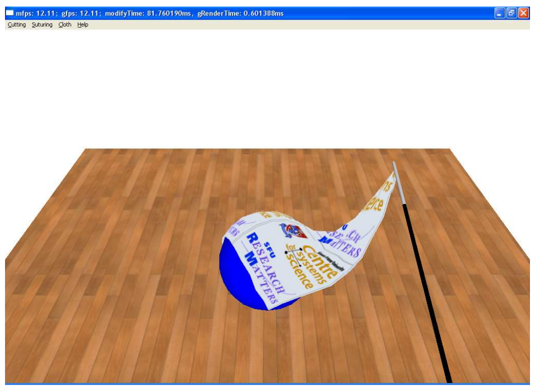
Figure 5.8 (a) shows the screen shot when the cloth is manipulated over a sphere object. Figure 5.8 (b) shows a typical plot of haptic force feedback.

5.3.4 Comparison between GPU and CPU implementation of the cloth model

Similar to the previous experiment, we are able to change the number of nodes associated with the cloth model and collect some statistical data regarding the performance of the simulation running only on CPU or GPU implementations. By changing the number of the mass nodes of the cloth, we can record the resultant time associated with various total number of nodes and compare the result between CPU and GPU. Table 5.2 shows the maximum and minimum modify time with different node number:

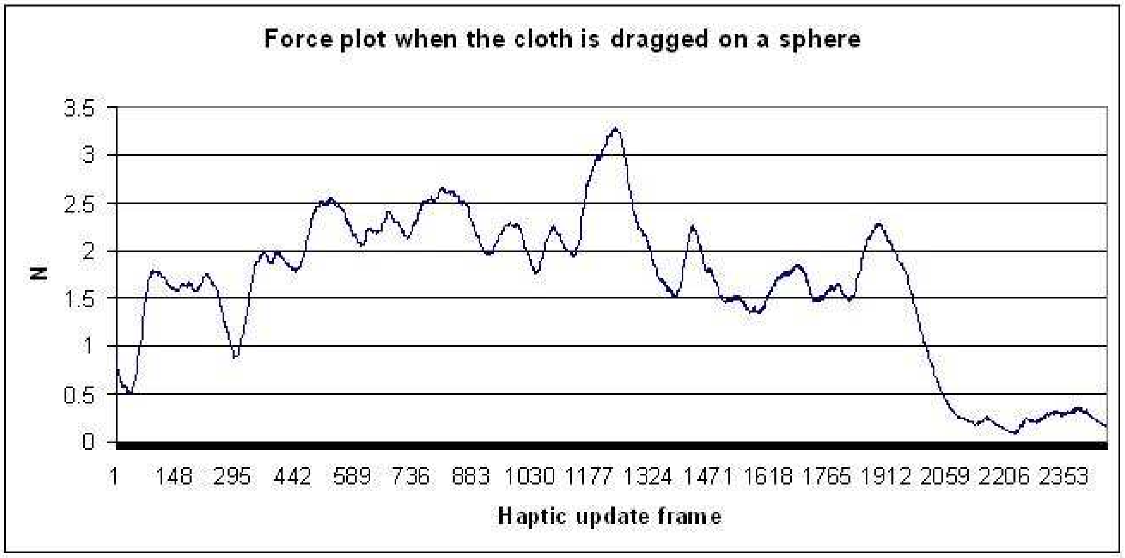
Table 5.2: GPU and CPU performance comparison of the cloth model

Node Number	GPU Max (ms)	GPU Min (ms)	CPU Max (ms)	CPU Min (ms)
16 * 16	4.01	3.94	58.10	57.37
32 * 16	5.50	5.18	121.66	119.87
32 * 32	10.19	9.83	237.50	234.68
48 * 32	16.33	15.95	370.52	369.71
48 * 48	27.49	27.21	590.31	586.52
64 * 48	42.37	41.72	750.31	742.37
64 * 64	78.47	77.05	5045.23	5042.98



```
mfps: 12.11; gfps: 12.11; modifyTime: 81.760190ms, gRenderTime: 0.601388ms
```

(a) *Screen shot*



(b) *Force Plot*

Figure 5.8: Force plot when the user grabs the cloth

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this thesis, we developed a knotting and suturing model based on the VTE. we present a physics-based approach to real-time simulation of DLOs with visual and force feedback. In our suture model, which can represent the mechanical properties of a real thread such as stretching, compressing, bending, and twisting, we simulate not only external forces, but also internal forces including the friction force during knotting and unknotting. We also present how forces propagate along the suture when the user pulls it with one or two hands. We developed a simulator to allow users to grasp and smoothly manipulate a virtual thread, and to tie an arbitrary knot.

We also present a physics-based haptic simulation designed to teach basic suturing techniques for simple skin or soft tissue wound closure. The pre-wound suturing target, skin or deformable tissue, is modeled as a modified mass-spring system. The suturing material is designed as a physics-based deformable linear object. Tools involved in the live suturing procedures are also simulated. Collisions between the soft tissue and the needle, and between the soft tissue and the suture are analyzed. In addition to the detail steps of one typical suturing procedure, modeling approaches on the evaluation of a stitch are also discussed. For example, if the needle insertion points are too close from each other or from the edge of the wound, the suture will tear the soft tissue instead of suturing the incision together when the tension applied on the pierced nodes beyond a pre-set threshold. Experiment results show that our simulator can run on a standard personal computer and allow users to perform different suturing patterns with smooth haptic feedback.

In addition, we did a study of GPU application in deformable object simulation. We all know that one of the most challenging aspects of developing haptic interactive applications is to guarantee the haptic rendering rate of 1000Hz. This requirements in general can offer the user a smooth haptic sensation. With the demand of creating more complex and realistic scenes for the user interaction, there exists opportunities for the haptic system engineers to design and experiment new computational environment which can blend the traditional computational mechanics models with more dedicated hardware utilities. In this thesis, we present some results regarding utilizing GPU processing units for computing the deformation of two experimental objects. We present a suture simulation model with GPU and a 2D deformable cloth model with nVidia CUDA techniques. We conducted experimental studies to compare the GPU-based suture models and with the CPU implementation. We experimented with the implicit model of the 2D mesh which offer similar computational challenges associated with any Finite-Element modeling approaches. Here, we have proposed a method for computing the inverse of a matrix with truncated Numan series and nVidia CUDA technology.

6.2 Future work

The experiment results show that our simulator could suture a pre-wound soft tissue together with smooth force feedback and also allow the user to tie a knot. However, there are still several issues need to be considered for the future work.

First, as mentioned in chapter 2, we did not consider the static friction for our suture model, the next step is to study the static forces when the user is trying to tie a knot tightly and untie a tight knot.

Secondly, implicit Euler's method offers many more benefits compared to the explicit methods. In the future, we might need to simulate the soft tissue using the implicit method to increase the system stability;

Thirdly, the current subdivision method can cause many unwanted small triangles if the user put the needle back and forth through the soft tissue at the same area, which will tremendously impact the stability of the system. To solve this, we may need to keep track of the topology of the movements of the needle, for example, if the user pull the needle back, the subdivided triangle should go back to its original un-subdivided configuration. Also using the edge flipping technique might be another solution.

Fourthly, although a lot of surgical training system have been designed out, how to evaluate these virtual training system is still an issue. Could virtual training system be used to measure surgical skills and could trainees really improve their surgical skill after such training are still two questions need to be answered. For the suturing simulator developed in this thesis, we still need to continue the research on how to evaluate the user's suture skills during the training and how to evaluate the correctness and robustness of this system. To solve this, a user study may need to be considered in the future.

Finally, for the GPU application in the deformable object simulation, one of the promising areas that such distributed computational power can have a greater impact is the utilization of point-based haptic rendering and the notion of Level-of-Details [44] [45]. For example, in the study of this thesis, the Hessian matrix can be very large sparse matrix. One approach for saving memory is to implement the matrix as doubly-linked lists. However, the challenge is on how to construct doubly-linked lists suitable for GPU parallel computing can be a subject of future studies. Self-collision detection for the cloth is not done in this project. As a result, the cloth behaviors can be on various cases not realistic because all the triangles could intersect with each other. Self collision detection can be another possibility where GPU may offer a benefit. In our study, we have discarded the non-linear part of the force filter and used a post correction of angular momentum. In addition, we have assumed that the matrix H does not change during simulation. However, if the 2D model of the cloth configuration changes during cutting or tearing, one needs to compute the inverse matrix calculation as the cloth configuration changes. This can be another topic of future investigation.

Appendix A

PHANTOM Omni Specifications

Table A.1: Phantom Omni Specifications

Nominal Position Resolution	>450 dpi	∽ 0.055 mm
Workspace	∼ 6.4 w x 4.8 h x 2.8 d in	> 160 w x 120 h x 70 d mm
Backdrive Friction	< 1 oz	< 0.26 N
Maximum Exertable Force	0.75 lbf	3.3 N
Continuous Exertable Force (24 hrs)	> 0.2 lbf	> 0.88 N
Stiffness	X axis > 7.3 lbs./in Y axis > 13.4 lbs./in Z axis > 5.9 lbs./in	X axis > 1.26 N/ mm Y axis > 2.31 N/mm Z axis > 1.02 N/mm
Inertia (apparent mass at tip)	∼ 0.101 lbm	∼ 45 g
Footprint	∼ 6 5/8 w x 8 d in	∼ 168 w x 203 d mm
Weight	3 lbs 15 oz	∼ 1.47 kg
Operating Temperature	50° to 95°F	10° to 35°C
Storage Temperature	−40° to 149°F	−40° to 65°C
Relative Humidity	20% to 80% (noncondensing)	
Force Feedback	3 degrees of freedom (x, y, z)	
Position Sensing	x, y, z (digital encoders)	
	Pitch, roll, yaw (3% linearity potentiometers)	Pitch, roll, yaw (5% linearity potentiometers)
Interface	IEEE-1394 FireWire port	
Input Voltage	100-240 VAC (Use supplied power supply only AD-740-1180)	
Input Frequency	50-60 Hz	
Input Current	1 A	

Appendix B

The process of suturing

B.1 The process of suturing

Paul Marshall etc. divided the whole suturing process into four steps in [15]. In this appendix, we give a brief overview for each of these steps. The only difference is, we are using mass-spring based suture model in this thesis, while Paul Marshall used geometry-based suture model.

When the mesh is pierced, a new vertex is generated at the point where it is pierced; this new vertex is the pierced vertex. When the pierced vertex is on the needle (controlled by the user), it cannot move except to slide on the needle. Once the vertex has been passed to the suture, though, the mesh will pull the suture and slide along the suture.

B.1.1 Piercing the deformable mesh

The needle held by a needle driver is an independent object under the user's control. Collision detection tests the straight-line path of the needle tip during a single timestep against each triangle. When the needle tip collides with a triangle, the pierced triangle is subdivided simply into three triangles, with a new vertex at the piercing point, and then the pierced triangles original edges are subdivided using the edge mask in the Loop subdivision scheme (See Figure B.1)

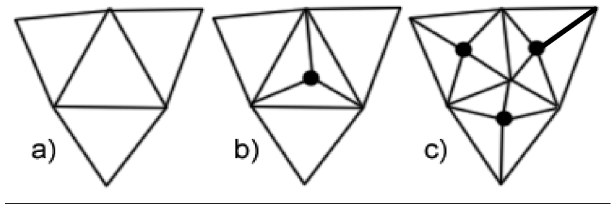


Figure B.1: Subdivision process when a portion of the mesh is pierced by the needle. a) Original mesh. b) After initial subdivision by being pierced by the needle at the highlighted point. c) After final subdivision of original triangles edges. Note the new vertices in c) are moved a short distance away from the pierced vertex.

When the edges are subdivided, the subdivision scheme ensures that the new vertices are a short distance from the edge, away from the piercing point. This prevents the newly-created triangles from being too small. Without the edge subdivision, a triangle pierced too close to one of the edges will look highly unrealistic in large deformations. The spring lengths of the new edges are 0.9 times the length of the edge at creation, to place the mesh under tension. The spring constants of the new edges are:

$$k = \frac{l_{max}}{l_{edge}} k_{mesh}$$

l_{max} is the longest edge in the mesh (computed at initialization), l_{edge} is the length of the new edge, and k_{mesh} is the spring constant of the mesh. For simulation purposes, the spring constant is limited to a range from kmesh to $4 * k_{mesh}$, for stability. The mass of each vertex in the mesh is unity, so the new vertices also have a mass of unity.

B.1.2 Slipping on the Needle

Once a newly-generated vertex has been pierced by the needle, it can slip in either direction on the needle. The needle, held by the user, is represented by displacement rather than forces so the only factors that count in slippage on the needle are the tangent of the needle at the point where the vertex is pierced, and the summed spring forces on the vertex. The slip is calculated as

$$a * (nt \cdot mf)$$

where mf is the summed spring forces of the mesh at the vertex, and nt is the tangent of the needle at that point. The term a is a constant chosen to balance the mesh force against the distance traveled on the needle. A constant friction force is applied to the slip. This equation uses the quasi-static assumption also used in the mesh: distance moved is proportional to the force applied in that timestep. Once a pierced vertex has passed the end of the needle, it is passed to a point on the suture very close to the needle.

Vertices slipping off of the tip from their initial position on the needle ignore friction: our model subdivides a triangle immediately when the needle touches the deformable model, but a normal needle would not actually pierce the surface of tissue until some force has been applied. Therefore, we allow a vertex to slip off the tip of the needle very easily if it has not yet slid farther onto the needle, in the same way that a needle that has not yet pierced tissue does not drag the tissue around.

Pierced vertices cannot slip past each other on the needle. A list of pierced vertices, sorted by location on the needle, is kept in the needle. If a vertex is about to slip past another vertex, the slipping vertex is not moved, and the interdicting vertex adds the slip value to its next slip calculation.

B.1.3 Slipping on the Suture

Once the deformable model has been pierced by the needle, and then slipped off the end of the needle onto the suture, it will slide along the suture as forces are applied.

The analysis of tissue sliding along thread shows three force components acting at the interface between the suture and the deformable model: tension in each direction of the pierced point, and the summed internal spring forces of the mesh. Sliding occurs when the net force tangential to the suture overcomes the friction at the piercing point. Force that exceeds friction follows a quasi-static model: distance slipped is force times a constant. The suture bends as it passes through the tissue, so calculating these forces precisely is difficult. A close approximation is

$$a * (pt_{length} - nt_{length}) + b * (pt_{direction} - nt_{direction}) \cdot mf$$

Where pt is the previous tension and nt is the next tension, expressed as vectors, and mf is the summed spring forces at that vertex. The first term is comparing the tension on either side of the pierced tissue: if there is greater tension on one side, then the suture is being pulled in that direction and the tissue will slide the other way. The second term is comparing the direction of the suture on each side of the constraint with the internal mesh forces at the pierced point. If the suture doubles back on itself through the pierced point, the difference between the two suture directions is small and likely in a direction nearly perpendicular to the summed mesh force, reducing the second term to almost zero. This matches with real-world experience: if a thread passes through an object and doubles back on itself, it is more difficult to pull through material than if the thread goes straight through.

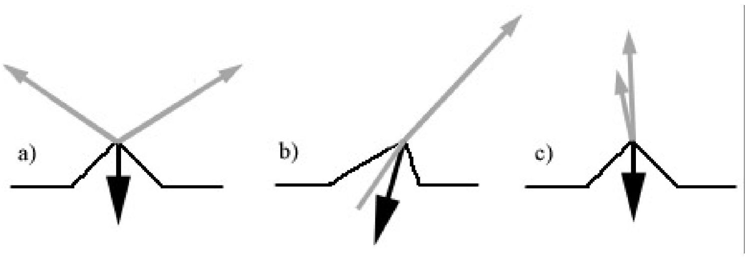


Figure B.2: Calculating the slip; gray indicates the suture, with arrows showing the magnitude and direction of the tension (a missing arrowhead indicates zero tension). Black indicates the shape of the mesh and the arrow shows the vector representing the summed mesh force. In a), the suture will not slip, because the tensions are equal in each direction, and the mesh force agrees equally with each direction of the suture. Part b) shows a normal case where one end of the suture is being pulled by the user while the other has no force; both the mesh force and the tension difference will cause the pierced vertex to slip away from the tensioned suture. Part c) shows a case where the mesh force will cause little effect on slippage, due to the directions of the suture being nearly identical; the suture will slip due to the difference in tension magnitude in each direction only

The terms a and b are constants chosen to balance the spring constants in the mesh against the tension value of the suture. A positive slip indicates sliding along the suture away from the needle. Soft constraints may not slip through each other or through hard constraints (since pierced pieces of tissue cannot slip through each other); if this happens, the moving constraint is not moved, and its slip value is added to the interdicting constraints next slip calculation, to move the interdicting constraint out of the way. Soft constraints exist at any point between the discrete nodes of the suture, so slipping is smooth in all cases.

B.1.4 Suture Tension

When a suture is stretched, tension naturally acts to restore the rest length of the suture. In most rope-like models, tension is simple to calculate due to the inherent forces within the model. The suture model used in this simulation is based in geometry. Since there is no internal model of the tension forces within the suture, a tension model is added separately. To handle interaction between the suture and the deformable model, as well as haptic rendering, we have created a quasi-spring method for calculating the tension of the suture. The links will stretch when multiple constraints are pulling the suture in different ways, and tension is calculated from this stretch in following equation:

$$T = (\sum_{i=j}^{k} l_i) - l_0(k - j)$$

Where T is tension, l_0 is the standard link length, l_i is the length of link i, j is the index of the left constraint, and k is the index of the right constraint.

Tension is calculated along each section of rope between soft or hard constraints. The tension is applied in the direction of the suture at the constraints. Each constraint can have different tension values in each direction, and the summed tension at the constraint is applied to any object the constraint is attached to, such as the needle or deformable model.

This method is similar to computing tension as a spring-like force, but the tension is not proportional to the actual strain of the suture, but rather the total distance

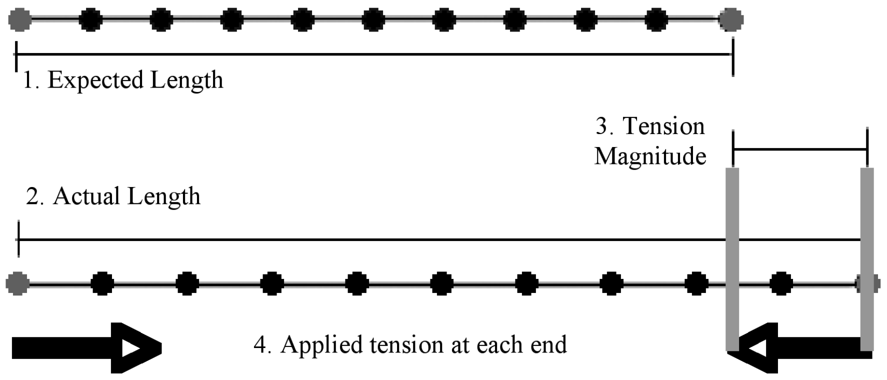


Figure B.3: Calculation of tension

it has been stretched between two contacts of interest. This is the closest the model can approach modeling stiff suture material: true suture material is quite stiff, and effectively does not stretch at all. Using this model, when the trainee pulls a suture a given amount, the suture will achieve a particular tension, which will pull in a predictable manner.

A pierced vertex on the suture is simulated the same as any other mesh node, by integrating the forces from each edge spring and the home spring, but with two additional forces: the tension force on the suture in each direction at the pierced vertex. In most cases, the tension force will be moderate or small, because the suture will slip further through the pierced vertex if the tension is large. But if the suture is blocked from slipping through the vertex or both ends of the suture are being pulled, the tension forces can pull the pierced vertex a significant distance. The user can pull a cut closed with a taut suture. The soft constraint on the suture is moved to the pierced vertexs new location after the vertex is simulated.

Appendix C

Explicit and Implicit Euler Integration

C.1 Introduction

In real time dynamic simulations, system stability is one of the most important issues. It is really frustrating that your simulation crashes because your simulation step size is too big. In some cases such as deformable object simulations in surgical training systems or cloth simulations in game developments, which demand very high computation, there is no place left at all because you the step size is already so small. Therefore we must find a new simulation scheme which both could guarantee system stability and is not limited by simulation step size.

Generally speaking, the big part of the dynamic simulation is about how to solve the differential equations, which describe the relationship between an unknown function and its derivatives. Most of those methods used by today's researchers could be put into two categories: explicit method and implicit method. This section will give out the introduction of explicit and implicit methods. More details will be discussed in the following sections.

The basic idea of explicit integration is that the integrators are written in a way that all unknown values (e.g., all particle velocities, positions) can be updated in a loop independently. Let's take an example as in the case of cloth simulations, the particle positions are treated as being decoupled and are not considered to affect each other. But in the real word, it is apparently not so: the particles in a cloth are coupled and connected together by the cloth. The motion of one particle is always affect by all of others. For a given time step, explicit integrators actually move the particles out-of-sync with each other, and put each particle slightly in the wrong place. However, they do become coupled again when the forces and constraints are set up prior to the next physics step. The forces and constraints are also slightly wrong because the particles are in wrong place, which will cause a small correction during the next physics step. This small correction is to try to put the particles in the right place. But in this viscious cycle, the next physics step overcorrects and the particles again are in the wrong place. Then, the cycle repeats. Explicit method guarantees a certain order of accuracy, but loses stability. The possible size of time step is restricted, which will result in a loss of computational efficiency.

In some cases, this viscious cycle actually runs well and the simulation doesn't crash, although the results are always wrong. In other cases (e.g., stiff ODEs), the cycle doesn't work at all. The corrective forces work against the overshot positions, which will cause an unstable simulation. once particle positions and velocities grow without limit, overflow happens in just a few physics steps.

On the other hand, implicit integrators treat the unknown variables as coupled, which are solved together as a system at each time step. The big advantage is that implicit schemes are stable for any time step size. However, they require the solution of a generally non-linear set of equations at each time step.

The equation of motion for the system can be represented by a matrix equation:

$$MA + CV + KX = F_{external}$$

where: $F_{external}$ is external forces such as the user input force, the weight of object, an explosion blast force due to pressure, etc.

M is the system mass matrix,

C is the damping matrix,

K is the stiffness matrix which represents the connection of mass points as in spring-mass cloth systems,

A is the acceleration vector for the particles,

V is the velocity vector for the particles,

X is the position vector for the particles.

The implicit solver works by computing M, C, K, then using a matrix solver or iterative (Jacobi, Gauss-Siedel, SOR) solver to compute new values of A, V, and X at the same time for all of the particles. By solving the particles as a coupled system, the implicit method gets rid of the potential for instability and the simulator almost always works without instabilities and blow-ups.

The result of implicit methods still has error (Taylor-series truncation error), so it is still imperfect.

C.2 Differential Equations

C.2.1 Initial Value Problems

A differential equation is a mathematical equation for an unknown function of one or several variables which relates the values of the function itself and of its derivatives of various orders. Differential equations play a prominent role in engineering, physics, economics, and other disciplines. In classical mechanics, the motion of a body is described by its position and velocity as the time varies. Initial value problems is one class of problems of differential equations. To solve a differential equation is to find a function that satisfies the relation and some additional conditions as well. In initial value problems, the initial condition (specified value) of the unknown function at a given point in the domain of the solution is given. The behavior of the system is described by an ordinary differential equation (ODE). In mathematics, an ordinary differential equation is a relation that contains functions of only one independent variable, and one or more of its derivatives with respect to that variable. For example:

$$\dot{\mathbf{x}} = f(\mathbf{x}, t),$$

where: $\mathbf{x}(t)$ is a moving point,

f is a known function,

$f(\mathbf{x}, t)$ is $\mathbf{x}$'s velocity,

$\mathbf{x}$ is the state of the system,

$\dot{\mathbf{x}}$ is $\mathbf{x}$'s time derivative.

The equation above defines a vector field over $\mathbf{x}$. Typically, for initial value problems, we are given $\mathbf{x}(t_0) = \mathbf{x}_0$ at some starting time t_0, and wish to follow $\mathbf{x}$ over time thereafter.

In $2D$ case, for example. $\mathbf{x}(t)$ draws out a curve that describes the motion of a point $\mathbf{p}$ in the plane. At any point $\mathbf{x}$, the function f can be evaluated to provide a 2-vector, therefore, f defines a vector field on the plane. The moving point $\mathbf{p}$ must have velocity vector $\dot{\mathbf{x}}$ when it moves through point $\mathbf{x}$.

The function f is written of both $\mathbf{x}$ and t, but the derivative function may or may not depend directly on time. If it does, then not only the point p but the the vector field itself moves, so that $\mathbf{p}$'s velocity depends not only on where it is, but on when it arrives there. In that case, the derivative $\dot{\mathbf{x}}$ depends on time in two ways: first, the derivative vectors themselves wiggle, and second, the point $\mathbf{p}$, because it moves on a trajectory $\mathbf{x}(t)$, sees different derivative vectors at different times.

C.2.2 Numerical Solutions

Standard way to solve differential equations focuses on symbolic solutions, in which the functional form for the unknown function is to be guessed. For example, the differential equation:

$$\dot{\mathbf{x}} = -kx,$$

where $\dot{\mathbf{x}}$ denotes the time derivative of $\mathbf{x}$.

This equation is satisfied by:

$$\mathbf{x} = e^{-kx},$$

Numerical solutions take discrete time steps starting from the initial value $\mathbf{x}(t_0)$. To take a step, we use the derivative function f to calculate an approximate change $\Delta\mathbf{x}$ in $\mathbf{x}$ over a time interval Δt, then increment $\mathbf{x}$ by Δx to obtain the new value. In this procedure, the derivative function f is considered as a black box: we provide numerical values for $\mathbf{x}$ and t, receiving in return a numerical value for $\dot{\mathbf{x}}$. Numerical methods operate by performing one or more of these derivative evaluations at each time step.

C.3 Explicit Method

C.3.1 Eulers Method

Eulers method is one of the simplest numerical methods for solving differential equations. First of all, let us denote the initial value for $\mathbf{x}$ by $\mathbf{x}_0 = \mathbf{x}(t_0)$ and the estimate value of $\mathbf{x}$ at a later time $t_0 + h$ by $\mathbf{x}(t_0 + h)$; where h is a step size parameter. Eulers method simply computes $\mathbf{x}(t_0 + h)$ by taking a step in the derivative direction:

$$\mathbf{x}(t_0 + h) = \mathbf{x}_0 + h\dot{\mathbf{x}}(t_0) \tag{C.1}$$

We can imagine a picture of a $2D$ vector field to visualize Euler's method. The moving point $\mathbf{p}$ will follow a polygonal path instead of the real integral curve in the vector field. Each segment of the polygonal path is determined by evaluating the function f at the beginning, and scaling by h.

The advantage of Euler's method is that it is simple and very easy to implement, but it is not accurate and will cause unstable problems in some cases. Consider the case of a $2D$ function f whose integral curves are concentric circles. The moving point $\mathbf{p}$ governed by f is supposed to orbit forever on whichever circle it started on. Instead, with each Euler step, $\mathbf{p}$ will move on a straight line to a circle of larger radius, so that its path will follow an outward spiral. Reducing the step size will slow the rate of this outward drift, but will never eliminate it.

To demonstrate its un-stability, let us take an example of a $1D$ function:

$$f = -kx,$$

which should make the moving point $\mathbf{p}$ decay exponentially to zero. For sufficiently small step sizes we can get reasonable behavior, but when $h > 1/k$, we have $|\Delta x| > |x|$, so the solution oscillates about zero. Beyond $h = 2/k$, the oscillation diverges, and the system blows up.

Finally, Euler's method has low efficiency. Derivative evaluation is pretty much CPU-time consuming, which will eat up almost all of CPU-time for most numerical solution methods. Therefore the computational cost per step is determined by the number of evaluations per step. Although Euler's method only requires one evaluation per step, in order to keep the system stability and accuracy, we must restrict the step size to very small, that means we must increase the number of simulation for a given time interval. More sophisticated methods, even some requiring as many as four or five evaluations per step, can greatly outperform Euler's method because their higher cost per step is more than offset by the larger step sizes they allow.

To improve on Euler's method, we need to look more closely at the error that it produces. The key to understanding it is the Taylor series: Assuming $\mathbf{x}(t)$ is smooth, we can approximate its value as an infinite sum involving the value and derivatives at the start point:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + ... + \frac{h^n}{n!}\frac{\partial^n \mathbf{x}}{\partial t^n} + ... \tag{C.2}$$

Comparing equation C.1 and C.2, we can get that the Euler formula is the first two terms of Taylor series. This means that Euler's method would be correct only if all derivatives beyond the first were zero, i.e. if $\mathbf{x}(t)$ were linear. The error term, the difference between the Euler step and the full, un-truncated Taylor series, is dominated by the leading term, $\frac{h^2}{2!}\ddot{\mathbf{x}}(t_0)$. Therefore, we can describe the error as $O(h^2)$. The error is proportional to the square root of time step size. If we decrease the step size to $h/2$, it will produces only about one fourth the error comparing with a step size of h. Theoretically, we can compute $\mathbf{x}$ with as little error as want by choosing a suitable small h. This could be not practical as a great many time steps night be required.

C.3.2 Midpoint Method

Let's take a look at the Taylor series again:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h\dot{\mathbf{x}}(t_0) + \frac{h^2}{2!}\ddot{\mathbf{x}}(t_0) + O(h^3) \tag{C.3}$$

If we were able to evaluate $\ddot{\mathbf{x}}$ as well as $\dot{\mathbf{x}}$, we could achieve $O(h^3)$ accuracy instead of $O(h^2)$ simply by retaining the term of $\frac{h^2}{2!}\ddot{\mathbf{x}}(t_0)$. Recall that:

$$\dot{\mathbf{x}} = f(\mathbf{x}(t), t) \tag{C.4}$$

For simplicity in what follows, we will assume that the derivative function f does depend on time only indirectly through $\mathbf{x}$, so that $\dot{\mathbf{x}} = f(\mathbf{x}(t))$. The chain rule then gives:

$$\ddot{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}}\dot{\mathbf{x}} = f'f \tag{C.5}$$

Evaluating f' would often be complicated and expensive. we only approximate the second-order term just in terms of f , and substitute the approximation into equation C.3, which will give us with $O(h^3)$ error. To do this, we perform another Taylor expansion of the function of f ,

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) = f(\mathbf{x}_0) + \Delta x f'(\mathbf{x}_0) + O(\Delta\mathbf{x}^2).$$

We first introduce $\ddot{\mathbf{x}}$ into this expression by choosing

$$\Delta\mathbf{x} = \frac{h}{2}f(x_0)$$

so that

$$f(\mathbf{x}_0 + \frac{h}{2}f(x_0)) = f(\mathbf{x}_0) + \frac{h}{2}f(x_0)f'(\mathbf{x}_0) + O(h^2) = f(\mathbf{x}_0) + \frac{h}{2}\ddot{\mathbf{x}}(t_0) + O(h^2).$$

where $\mathbf{x}_0 = \mathbf{x}(t_0)$.
Multiply both sides by h (turning the $O(h^2)$ term into $O(h^3)$) and rearrange, yielding:

$$\frac{h^2}{2}\ddot{\mathbf{x}} + O(h^3) = h(f(\mathbf{x}_0 + \frac{h}{2}f(x_0))) \approx f(\mathbf{x})$$

Substituting the right hand side into equation C.3 gives the update formula:

$$\mathbf{x}(t_0 + h) = \mathbf{x}(t_0) + h(f(\mathbf{x}_0 + \frac{h}{2}f(x_0)))$$

This formula first evaluates an Euler step, then performs a second derivative evaluation at the midpoint of the step, using the midpoint evaluation to update $\mathbf{x}$. The midpoint method is correct to within $O(h^3)$, but requires two evaluations of f.

By evaluating f a few more times, we can eliminate higher and higher orders of derivatives. The most popular procedure for doing this is a method called Runge-Kutta of order 4 and has an error per step of $O(h^5)$. (The Midpoint method could be called Runge-Kutta of order 2.) The formula for computing $\mathbf{x}(t_0 + h)$ is listed below:

$$\begin{aligned}
k_1 &= hf(\mathbf{x}_0, t_0) && \text{(C.6)}\\
k_2 &= hf(\mathbf{x}_0 + \frac{k_1}{2}, t_0 + \frac{h}{2}) && \text{(C.7)}\\
k_3 &= hf(\mathbf{x}_0 + \frac{k_2}{2}, t_0 + \frac{h}{2}) && \text{(C.8)}\\
k_4 &= hf(\mathbf{x}_0 + k_3, t_0 + h) && \text{(C.9)}\\
\mathbf{x}(t_0 + h) &= \mathbf{x}_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 && \text{(C.10)}
\end{aligned}$$

C.3.3 Adaptive Stepsizes

In the numerical integration, no matter what the underlying method you chose, a major problem is how to determine a good step size. Ideally, we want to choose h as large as possible to save CPU-time, but not so large as to give us an unreasonable amount of error, or to induce instability. If we choose a fixed step size, we can only proceed as fast as the "worst" sections of $\mathbf{x}(t)$ will allow. What we would like to do is to vary h as we march forward in time. We should increase h whenever will not incur too much error. We should decrease h if we want to avoid excessive error. This is the idea of adaptive stepsizing: varying h over the course of solving the ODE.
The basic idea of adaptive stepsizing for Euler's method is as follows: Lets assume we have a given stepsize h, and we want to know how much we can consider changing it. Suppose we compute two estimates for $\mathbf{x}(t_0 + h)$:

1. $\mathbf{x}_a$ - we take an Euler step of size h from t_0 to $t_0 + h$
2. $\mathbf{x}_b$ - we take two Euler steps of size $h/2$, from t_0 to $t_0 + h$.

Both $\mathbf{x}_a$ and $\mathbf{x}_b$ differ from the true value of $\mathbf{x}(t_0 + h)$ by $O(h^2)$. That means that $\mathbf{x}_a$ and $\mathbf{x}_b$ differ from each other by $O(h^2)$. As a result, we can write that a measure of the current error e is:

$$e = |\mathbf{x}_a - \mathbf{x}_b|$$

This gives us a convenient estimate to the error in taking an Euler step of size h. Suppose that we are willing to have an error of as much as 10^{-4} per step, and that the current error is only 10^{-8}. Since the error goes up as h^2, we can increase the stepsize to

$$(\frac{10^{-4}}{10^{-8}})^{\frac{1}{2}}h = 100h$$

Conversely, if we currently had an error of 10^{-3}, and could only tolerate an error of 10^{-4}, we would have to decrease the stepsize to

$$(\frac{10^{-4}}{10^{-3}})^{\frac{1}{2}}h = 0.316h$$

Adaptive stepsizing is a highly recommended technique.

C.4 Implicit Method

Explicit methods such as Eulers method, or the midpoint method, have a difficult time with some types of ODEs. Sometimes an ODE can become "stiff", in which case explicit methods do not do a very good job of solving them.

C.4.1 Example Stiff ODE

In more complex systems which have more than one spring, step size is limited by the largest k. One stiff spring can screw the whole system up for everyone else. Systems that have some big ks mixed in are called stiff systems.
To understand what the meaning and cause of stiff equations is, lets first consider an example that arises frequently in dynamics. Suppose that we have a particle, with position $(x(t), y(t))$, and we want the y-coordinate to always be zero. Or in other words, we have a particle p in a plane, the user is trying to pull the point with the interactive "dragging" force (f_x, f_y). And we try to keep the point p on the x coordinate (which means we want the y-coordinate to always be zero). One way of doing this is to add a component $-ky(t)$ to $\dot{y}(t)$ where k is a large positive constant. That means we add a penalty force $(0, ky)$ trying to keep p on the x-axis. If k is

large enough, then the particle will never move too far away from $y(t) = 0$, since the $-ky(t)$ term always brings $y(t)$ back towards zero.

Lets assume that there is no restriction on the x-coordinate, and that we want the user to be able to pull the particle arbitrarily along the x-axis. So lets suppose our differential equation over time interval is:

$$\dot{X}(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} -x(t) \\ -ky(t) \end{pmatrix} \tag{C.11}$$

We also assume that the particle does not start exactly with $y_0 = 0$.

Whats happening here is that the particle is strongly attracted to the line $y = 0$, and less strongly towards $x = 0$. If we solve the ODE far enough forward in time, we expect the particles location to converge towards $(0, 0)$ and then stay there once it arrives.

Now suppose we use Eulers method to solve the equation. If we take a step of size h, we get

$$X_{new} = X_0 + h\dot{X}(t_0) = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + h\begin{pmatrix} -x_0 \\ -ky_0 \end{pmatrix} \tag{C.12}$$

This yields

$$X_{new} = \begin{pmatrix} x_0 - hx_0 \\ y_0 - hky_0 \end{pmatrix} = \begin{pmatrix} (1-h)x_0 \\ (1-hk)y_0 \end{pmatrix} \tag{C.13}$$

If we look at the y component of the above equation, we see that:

$$y_{new} = (1 - hk)y_0$$

if $|1 - hk| > 1$ then,

$$|y_{new}| > |y_0|$$

In other words, if $|1 - hk| > 1$, Eulers method will not converge to an answer: each step will result in a larger value of y_{new} than the last. Technically, Eulers method is unstable for $|1 - hk| > 1$. Thus, we better have $1 - hk > -1$ or $hk < 2$ if we hope to converge. The largest step we can hope to take is less than $2/k$.

Now, if k is a large number, we will have to take very small steps. This means that the particle slides towards $(0, 0)$ excruciatingly slowly. Even though the particle may nearly satisfy $y_0 = 0$, we have to take such small steps that the particles progress along the x-axis is pretty much nonexistent. Thats the embodiment of a stiff ODE. In this case, the stiffness arises from making k very large in order to keep the particle close to the line $y = 0$. Even if we use a more sophisticated explicit method such as fourth-order Runge-Kutta, we may do a little better in the size of our steps, but we will still have major problems.

C.4.2 Solving Stiff ODEs - Backward Euler's Method

As we discussed in above sections, explicit methods have stiff ODE's problem. We have to turn to implicit methods for solutions, which is based on taking an Euler step "backwards".
Given a differential equation:

$$\frac{d}{dt}X(t) = f(X(t)), \tag{C.14}$$

The explicit Euler update would be:

$$X_{new} = X_0 + hf(X(t_0)), \tag{C.15}$$

This advances the system forward h in time.
For a stiff problem though, we change the update to be

$$X_{new} = X_0 + hf(X_{new}), \tag{C.16}$$

That is, we are going to evaluate f at the point we are aiming at, rather than where we came from. So, we are looking for a X_{new} such that f points directly back at where we came from. Unfortunately, we can not in general solve for X_{new}, unless f happens to be a linear function. To cope with this, we will replace $f(X_{new})$ with a linear approximation, again based on f's Taylor series.
Lets define ΔX by $\Delta X = X_{new} - X_0$ and rewrite equation C.16 as:

$$X_0 + \Delta X = X_0 + hf(X_0 + \Delta X). \tag{C.17}$$

or just

$$\Delta X = hf(X_0 + \Delta X). \tag{C.18}$$

Next, lets approximate $f(X_0 + \Delta X)$ by

$$f(X_0) + f'(X_0)\Delta X. \tag{C.19}$$

(Note that since $f(X_0)$ is a vector, the derivative $f'(X_0)$ is a matrix). Using this approximation, we can approximate ΔX with

$$\Delta X = h(f(X_0) + f'(X_0)\Delta X) \tag{C.20}$$

or

$$\Delta X - hf'(X_0)\Delta X = h(f(X_0) \tag{C.21}$$

Rewriting this as

$$(\frac{\mathbf{I}}{h} - f'(X_0))\Delta X = f(X_0) \tag{C.22}$$

where $\mathbf{I}$ is the identity matrix, we can solve for ΔX as

$$\Delta X = (\frac{\mathbf{I}}{h} - f'(X_0))^{-1} f(X_0) \tag{C.23}$$

Computing $X_{new} = X_0 + \Delta X$ is clearly more work than using an explicit method, since we have to solve a linear system at each step. For many types of problems, the matrix f' will be sparse for example, if we are simulating a spring-lattice, f' will have a structure which matches the connectivity of the particles. As a result, it is usually possible to solve equation C.23 in linear time (i.e. time proportional to the dimension of X).

Lets apply the implicit method to equation C.11. We have that $f(X(t))$ is

$$f(X(t)) = \begin{pmatrix} -x(t) \\ -ky(t) \end{pmatrix}. \tag{C.24}$$

Differentiating with respect to X yields

$$f'(X(t)) = \frac{\partial}{\partial X} f(X(t)) = \begin{pmatrix} -1 & 0 \\ 0 & -k \end{pmatrix}. \tag{C.25}$$

Then the matrix $\frac{\mathbf{I}}{h} - f'(X_0)$ is

$$\frac{\mathbf{I}}{h} - f'(X_0) = \begin{pmatrix} \frac{1}{h}+1 & 0 \\ 0 & \frac{1}{h}+k \end{pmatrix} = \begin{pmatrix} \frac{1+h}{h} & 0 \\ 0 & \frac{1+hk}{h} \end{pmatrix} \tag{C.26}$$

Inverting this matrix, and multiplying by $f(X_0)$ yields

$$\begin{aligned} \Delta X &= \begin{pmatrix} \frac{1+h}{h} & 0 \\ 0 & \frac{1+hk}{h} \end{pmatrix}^{-1} \begin{pmatrix} -x_0 \\ -ky_0 \end{pmatrix} && \text{(C.27)} \\ &= \begin{pmatrix} \frac{h}{1+h} & 0 \\ 0 & \frac{h}{1+hk} \end{pmatrix} \begin{pmatrix} -x_0 \\ -ky_0 \end{pmatrix} && \text{(C.28)} \\ &= -\begin{pmatrix} \frac{h}{1+h}x_0 \\ \frac{h}{1+hk}ky_0 \end{pmatrix} && \text{(C.29)} \end{aligned}$$

What is the limit on the stepsize in this case? The answer is: there is no limit! In this case, if we let h grow to infinity, we get

$$\lim_{h->\infty} \Delta X = \lim_{h->\infty} -\begin{pmatrix} \frac{h}{1+h}x_0 \\ \frac{h}{1+hk}ky_0 \end{pmatrix} = -\begin{pmatrix} x_0 \\ \frac{1}{k}ky_0 \end{pmatrix} = -\begin{pmatrix} x_0 \\ y_0 \end{pmatrix}. \tag{C.30}$$

This means that we achieve $X_{new} = X_0 + (-X_0)$ in a single step! For a general stiff ODE, we won't be able to take steps of arbitrary size, but we will be able to take much larger steps using an implicit method than using an explicit method. The extra cost of solving a linear equation is more than made up by the time saved by taking large timesteps.

C.4.3 Solving Second-Order Equations

Most dynamics problems are written in terms of a second-order differential equation as follows:

$$\ddot{\mathbf{x}}(t) = f((x)(t), \dot{\mathbf{x}}(t)). \tag{C.31}$$

This equation can be easily converted to a first-order differential equation by adding new variables. If we define $v = \dot{\mathbf{x}}$, then we can rewrite above equation as:

$$\frac{d}{dt}\begin{pmatrix}\mathbf{x}(t)\\ \mathbf{v}(t)\end{pmatrix} = \begin{pmatrix}\mathbf{v}(t)\\ f(\mathbf{x}(t),\mathbf{v}(t))\end{pmatrix}. \tag{C.32}$$

which is a first-order system. However, applying the backward Euler method to above equation results in a linear system of size $2n \times 2n$ where n is the dimension of $\mathbf{x}$. A fairly simple transformation allows us to reduce the size of the problem to solving an $n \times n$ linear system instead. We can do this simply by introducing more variables. The $n \times n$ system that needs to be solved is derived as follows. Let us simplify notation by writing $\mathbf{x}_0 = \mathbf{x}(t_0)$ and $\mathbf{v}_0 = \mathbf{v}(t_0)$. We also define $\Delta\mathbf{x} = \mathbf{x}(t_0 + h) - \mathbf{x}(t_0)$ and $\Delta\mathbf{v} = \mathbf{v}(t_0+h) - \mathbf{v}(t_0)$. The backward Euler update, applied to equation C.32, yields:

$$\begin{pmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{v}\end{pmatrix} = h\begin{pmatrix}\mathbf{v}_0 + \Delta\mathbf{v}\\ f(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v})\end{pmatrix}. \tag{C.33}$$

Applying a Taylor series expansion to f which in this context is a function of both $\mathbf{x}$ and $\mathbf{v}$ yields the first-order approximation:

$$f(\mathbf{x}_0 + \Delta\mathbf{x}, \mathbf{v}_0 + \Delta\mathbf{v}) = f_0 + \frac{\partial f}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial f}{\partial \mathbf{v}}\Delta\mathbf{v} \tag{C.34}$$

In this equation, the derivative $\frac{\partial f}{\partial \mathbf{x}}$ is evaluated for the state $(\mathbf{x}_0, \mathbf{v}_0)$ and similarly for $\frac{\partial f}{\partial \mathbf{v}}$. Substituting this approximation into equation C.33 yields the linear system:

$$\begin{pmatrix}\Delta\mathbf{x}\\ \Delta\mathbf{v}\end{pmatrix} = h\begin{pmatrix}\mathbf{v}_0 + \Delta\mathbf{v}\\ f_0 + \frac{\partial f}{\partial \mathbf{x}}\Delta\mathbf{x} + \frac{\partial f}{\partial \mathbf{v}}\Delta\mathbf{v}\end{pmatrix}. \tag{C.35}$$

Taking the bottom row of equation C.35 and substituting $\Delta\mathbf{x} = h(\mathbf{v}_0 + \Delta\mathbf{v})$yields

$$\Delta\mathbf{v} = h(f_0 + \frac{\partial f}{\partial \mathbf{x}}h(\mathbf{v}_0 + \Delta\mathbf{v}) + \frac{\partial f}{\partial \mathbf{v}}\Delta\mathbf{v}) \tag{C.36}$$

Letting $\mathbf{I}$ denote the identity matrix, and regrouping, we obtain

$$(\mathbf{I} - h\frac{\partial f}{\partial \mathbf{v}} - h^2\frac{\partial f}{\partial \mathbf{x}})\Delta\mathbf{v} = h(f_0 + h\frac{\partial f}{\partial \mathbf{x}}\mathbf{v}_0) \tag{C.37}$$

which we then solve for $\Delta\mathbf{v}$. Given $\Delta\mathbf{v}$, we trivially compute $\Delta\mathbf{x} = h(\mathbf{v}_0 + \Delta\mathbf{v})$.

The above discussion assumes that the function f has no direct dependence on time; in the case that f varies directly with time (for example, if f describes time-varying external forces, or references moving points or coordinate frames that are not variables of $\mathbf{x}$) then equation C.37 needs an additional term to account for this dependence:

$$(\mathbf{I} - h\frac{\partial f}{\partial \mathbf{v}} - h^2\frac{\partial f}{\partial \mathbf{x}})\Delta\mathbf{v} = h(f_0 + h\frac{\partial f}{\partial \mathbf{x}}\mathbf{v}_0 + \frac{\partial f}{\partial t}) \tag{C.38}$$

Bibliography

[1] H. Wakamatsu, E. Arai, and S. Hirai. Knotting/unknotting manipulation of deformable linear objects. *International Journal of Robotics Research*, 25:371–395, 2006.

[2] H. Wakamatsu, K. Takahashi, and S. Hirai. Dynamic modeling of linear object deformation based on differential geometry coordinates. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1028–1033, 2005.

[3] J. Takamatsu, T. Morita, K. Ogawara, H. Kimura, and K. Ikeuchi. Representation for knot-tying tasks. In *IEEE Transactions on Robotics*, volume 22, pages 65–78, 2006.

[4] M. Saha and P. Isto. Motion planning for robotic manipulation of deformable linear objects. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 2478–2484, 2006.

[5] D. Pai. Strands: Interactive simulation of thin solids using cosserat models. In *Proceedings of Eurographics'02*, pages 347–352, 2006.

[6] C. Wang, A.M.D. Richardson, D. Liu, R. Rosing, R. Tucker, and B. De Masi. Construction of nonlinear dynamic mems component models using cosserat theory. In *Proc. SPIE Design, Test, Integration & Packaging of MEMS Symposium*, pages 131–136, 2003.

[7] D. Q. Cao, D. Liu, and C. H.-T. Wang. Three dimensional nonlinear dynamics of slender structures: Cosserat rod element approach. *International Journal of Solids and Structures*, 43:760–783, 2006.

[8] M. Grégoire and E. Schömer. Interactive simulation of one-dimensional flexible parts. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*, pages 95–103, 2006.

[9] J. Phillips, A. Ladd, and L.E. Kavraki. Simulated knot tying. In *IEEE International Conference on Robotics and Automation*, pages 841–846, 2002.

[10] B. Kahl and D. Henrich. Manipulation of deformable linear objects: Force-based simulation approach for haptic feedback. In *12th International Conference on Advanced Robotics (ICAR 2005)*, 2005.

[11] F. Wang, E. Burdet, A. Dhanik, T. Poston, and C. L. Teo. Dynamic thread for real-time knot-tying. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 2005.

[12] M. LeDuc, S. Payandeh, and J. Dill. Toward modeling of a suturing task. In *Graphics Interface (GI)*, pages 273–279, 2003.

[13] J. Brown, K. Montgomery, J-C. Latombe, and M. Stephanides. A microsurgery simulation system. In *Medical Image Computing and Computer Aided Interventions*, 2001.

[14] R.W. Webster, D.I. Zimmerman, B.J. Mohler, M.G. Melkonian, and R.S. Haluck. A prototype haptic suturing simulator. In J.D. Westwood, H.M. Hoffman, G.T. Mogel, and D. Stredney, editors, *Medicine Meets Virtual Reality*, pages 567–569, 2001.

[15] P. Marshall, S. Payandeh, and J. Dill. Suturing for surface meshes. In *Proceedings of the 2005 IEEE International Conference on Control Applications*, pages 25–30, 2005.

[16] P. Marshall, S. Payandeh, and J. Dill. A study on haptic rendering in a simulated surgical training environment. In *Proceedings of the Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'06)*, volume 00, page 35, 2006.

[17] J. Berkley, G. Turkiyyah, D. Berg, M. Ganter, and S. Weghorst. Real-time finite element modeling for surgery simulation: An application to virtual suturing. In *IEEE Transactions on Visualization and Computer Graphics*, volume 10, pages 314–325, 2004.

[18] J. Berkley, S. Weghorst, H. Gladstone, G. Raugi, D. Berg, and M. Ganter. Banded matrix approach to finite element modeling for soft tissue simulation. *Virtual Reality: Research, Development,and Application*, 4:203–212, 1999.

[19] M. LeDuc, S. Payandeh, and J. Dill. Toward modeling of a suturing task. In *In Graphics Interface'03 Conference*, pages 273–279, 2003.

[20] R. V O'toole, R. R Playter, T. M. Krummer, W. C. Blank, N. H. Conelius, W. R Roberts, W. J Bell, and M. Raibert. Measuring and developing suturing technique with a virtual reality surgical simulator. *Journal of the American College of Surgeons*, pages 114–127, 1999.

[21] L. L. Lian and Y. H. Chen. Haptic surgical simulation: An application to virtual suture. In *Computer-Aided Design & Applications*, volume 3, pages 203–210, 2006.

[22] L. Moody, C. Baber, and T.N. Arvanitis. The role of haptic feedback in the training and assessment of surgeons using a virtual environment. In *EuroHaptics 2001*, 2001.

[23] D. Terzopoulos and K. Waters. Physically-based facial modeling, analysis, and animation. *Journal of Visualization and Computer Animation*, pages 73–80, 1990.

[24] D. T. Chen and D. Zeltzer. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. In *Computer Graphics*, volume 2, pages 89–98, 1992.

[25] M. Bro-Nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite element and condensation. In *EUROGRAPHICS'96*, volume 15, pages 57–66, 1996.

[26] D. Hutchinson, M. Preston, and T. Hewitt. Adaptive refinement for mass/spring simulation. In *In 7th Eurographics Workshop on Animation and Simulation*, pages 31–45, 1996.

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

[28] J.J. Cha and S. Payandeh. Interactive cross cutting. In *2007 IEEE International Conference on Robotics and Automation (ICRA '07)*, pages 2576–2581, 2007.

[29] S. Payandeh and N. Azouz. Finite elements, mass-spring-damper systems and haptic rendering. In *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 224–230, 2001.

[30] M. de Pascale, G. de Pascale, D. Prattichizzo, and F. Barbagli. A gpu-friendly method for haptic and graphic rendering of deformable objects. In *Proceedings of Eurohaptics 2004*, 2004.

[31] C. J. Luciano, P. P. Banerjee, and S. H. R. Rizzi. Gpu-based elastic-object deformation for enhancement of existing haptic applications. In *Proceedings of the 3rd Annual IEEE Conference on Automation Science and Engineering Scottsdale*, 2007.

[32] J. Mosegaard, P. Herborg, and T.S. Srensen. A gpu accelerated spring-mass system for surgical simulation. In *Medicine Meets Virtual Reality 13*, 2005.

[33] T. S. SÿRENSEN and J. Mosegaard. Haptic feedback for the gpu-based surgical simulator. In *Medicine Meets Virtual Reality 14*, 2006.

[34] E. Moncls, I. Navazo, and Pere-Pau Vzquez. Mtcut: Gpu-based marching tetra cuts. In Ik Soo Lim and David Duce, editors, *EG UK Theory and Practice of Computer Graphics*, 2007.

[35] J.E. Colgate, M.C. Stanley, and J.M. Brown. Issues in the haptic display of tool use. In *Intelligent Robots and Systems 95*, 1995.

[36] X. Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface '95*, pages 147–154, 1995.

[37] Inc. SensAble Technologies. Open haptics toolkit version 2.0 programmer's guide, 2005.

[38] Inc. SensAble Technologies. Openhaptics toolkit version 2.0 api reference, 2005.

[39] J.Brown, J.Latombe, and K.Montgomery. Real-time knot tying simulation. *The Visual Computer: International Journal of Computer Graphics*, pages 165–179, 2004.

[40] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 594–603, 2002.

[41] M. Meyer, G. Debunne, M. Desbrun, and A. H. Barr. Interactive animation of cloth-like objects for virtual realiy. *The Journal of Visualization and Computer Animation*, 2001.

[42] S. Payandeh, J. Dill, and Z. Cai. On interacting with physics-based models of graphical objects. In *Robotica*, volume 22, pages 223–230, 2004.

[43] H. F. Shi and S. Payandeh. Real-time knotting and unknotting. In *2007 IEEE International Conference on Robotics and Automation (ICRA '07)*, pages 2570–2575, 2007.

[44] M. Gross and H. Pfister. Point-based graphics. In *The Morgan Kaufmann Publishing*, 2005.

[45] S. Payandeh, J. Dill, and J. Zhang. A study of level-of-detail in haptic rendering. In *ACM transaction for Applied Perception*, volume 2, pages 15–34, 2005.

[46] H. F. Shi and S. Payandeh. Gpu in haptic rendering of deformable objects. In *Haptics: Perception, devices and scenarios, Proceedings of 6th International Conference, Eurohaptics 2008*, pages 163–168, 2008.

[47] H. F. Shi and S. Payandeh. On suturing simulation with haptic feedback. In *Haptics: Perception, devices and scenarios, Proceedings of 6th International Conference, Eurohaptics 2008*, pages 599–608, 2008.