Computer Haptics: Rendering Techniques for Force-Feedback in Virtual Environments

By

Chih-Hao Ho

M.S. Mechanical Engineering
Massachusetts Institute of Technology, 1996

SUBMITTED TO THE
DEPARTMENT OF MECHANICAL ENGINEERING
IN PARTICAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOISOPHY IN MECHANICAL ENGINEERING
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

FEBRUARY 2000

© 2000 Massachusetts Institute of Technology. All rights reserved.

Author ________ ________________________________
Department of Mechanical Engineering
December 17, 1999

Certified by ____ ________________
Dr. Mandayam A. Srinivasan
Thesis Supervisor, Department of Mechanical Engineering

Accepted by _______ ________________________________
Professor Ain A. Sonin
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
SEP 2 0 2000
LIBRARIES

ENG

Computer Haptics: Rendering Techniques for Force-Feedback in Virtual Environments

By
Chih-Hao Ho
Submitted to the Department of Mechanical Engineering on December 17, 1999, in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Mechanical Engineering

Abstract

Haptic virtual environments (VEs) are computer-generated environments within which human users can touch, feel, and manipulate virtual objects in real time through force or tactile feedback. Integration of force-feedback into VEs with graphical and auditory displays is expected to have many applications in the area of medical training, CAD/CAM, entertainment, graphic arts, and education. The development of force-feedback system for VEs is in the research area of haptics, which can be divided into three sub-categories: human haptics, machine haptics, and computer haptics. Human haptics focuses on the understanding of human hand brain system. Machine haptics focuses on the design and development of force-feedback devices (also called haptic interfaces). Computer haptics focuses on the algorithm and software for the creation of virtual objects and how to efficiently *display* these objects to the users.

The research of this thesis focuses on computer haptics. First, a point-based interaction paradigm called *Neighborhood Watch* is developed to allow human users to use a point probe to manually explore and manipulate virtual objects. A major feature of *Neighborhood Watch* is that the computational time is independent of the number of polygons of the virtual objects. Second, another interaction paradigm, called *ray-based rendering*, which allows human users to use a line probe to interact with virtual objects is also described. The computational time with ray-based rendering is also essentially independent of the number of polygons of the virtual objects. In addition to the two haptic interaction paradigms, various object property display algorithms have also been developed. Using these algorithms, we can add friction and textures to the surfaces of arbitrarily shaped virtual objects and also add compliant and dynamic behavior to the virtual objects. All of the techniques developed for haptic rendering were finally incorporated into a surgical simulator to demonstrate their usefulness.

Thesis Supervisor: Mandayam A. Srinivasan
Title: Principal Research Scientist, Department of Mechanical Engineering
Thesis Committee: Harry H. Asada
Title: Professor, Department of Mechanical Engineering
Thesis Committee: David C. Gossard
Title: Professor, Department of Mechanical Engineering

Acknowledgements

There are many people to whom I would like to express my thanks and appreciation. First of all, I want to thank my advisor Dr. Srinivasan for giving me the opportunity to work in such a great project and for his constant encouragement, understanding, patience, and guidance throughout the study.

Thanks to Cagatay Basdogan for his help and guidance. Cagatay, no word can express my great thanks to you.

I would also like to thank my Touch Lab colleagues for their help and for making the lab a pleasurable place to work.

Love to my wife Rachel for her encouragement and help and also to my parents for their concern and love.

Thanks to everyone who has helped make this research possible.

Contents

Chapter 1

Introduction

Virtual environments (VEs) are computer-generated environments with which a human user can interact in real time. In particular, a multi-modal virtual reality (VR) system will enable the humans to interact with the computer and obtain visual, auditory, and haptic sensations, as they would encounter in real life. A VE system typically consists of several interface devices such as a head mounted display and a position tracker that send and receive signals of the interactions between the computer and a human user. The development of VEs is a very popular research topic for the past decades. The techniques in creating visual and auditory VEs have already been broadly applied to entertainment such as the creation of video games and movies. In addition to the visual and auditory VEs, the development of haptic VEs that can provide human users with force-feedback interaction is a relatively new research topic in the area of virtual reality simulation. With the addition of force-feedback interaction to the VEs, the users will be able to touch, feel, and manipulate virtual objects in addition to see and hear them. The haptic VEs, with the feature of force feedback, are particularly helpful when manual actions such as exploration and manipulation are essential in an application. For example, surgical residents would benefit from training in a haptic VE as it offers a consistent environment and unlimited opportunity to practice surgical procedures. Aside from medical training, other applications of the haptic VEs may include CAD/CAM, entertainment, graphic arts, and education.

In creating haptic VEs, a device called " *haptic interface*" is used to provide force-feedback interactions via "*haptic rendering*" techniques. The typical functions of a haptic interface are to sense the position of the user's hand and reflect forces back to the user. The term "*haptic rendering*" refers to the process that compares the position information of user's hand with the data of virtual objects and calculates an appropriate interaction force to send back to the user. A computer is the main hardware that performs the haptic rendering. For a given position information, a computer checks whether the hand is inside the virtual objects or not. If the hand is inside the virtual objects, it sends force commands to the haptic interface to prevent the hand from further penetrating into the objects. It should be noted that, with the technology available today, the user unfortunately couldn't get the complete sensation of touching the virtual objects directly as in real life. The user can touch and manipulate objects in virtual environments only through an end-effector of a haptic device (see Figure 1-1). This end-effector could be a thimble in which the fingertip could be inserted, a stylus or a mechanical tool that could be held in the hand, or an actuated glove or exoskeleton that the user could wear. During the simulations, the user manipulates the end-effector and feels the reaction forces when the end-effector contacts the virtual objects. Although the end-effector can have different physical shapes and designs, they can be modeled as a point, a line, or a 3D object in virtual environments for simulating haptic interactions.

Most of the techniques for creating visual VEs are covered in the research area of computer graphics, which is mainly concerned with the generation and rendering of graphic images. Some of the techniques in creating haptic VEs are similar to those in computer graphics. However, there are still many differences between them. To differentiate the difference between the techniques in creating visual and haptic VEs, researchers have proposed the term "*Computer Haptics*" to represent the field of research that is concerned with the generation and rendering of haptic virtual objects (Srinivasan and Basdogan, 1997 and Ho, Basdogan, and Srinivasan, 1999).

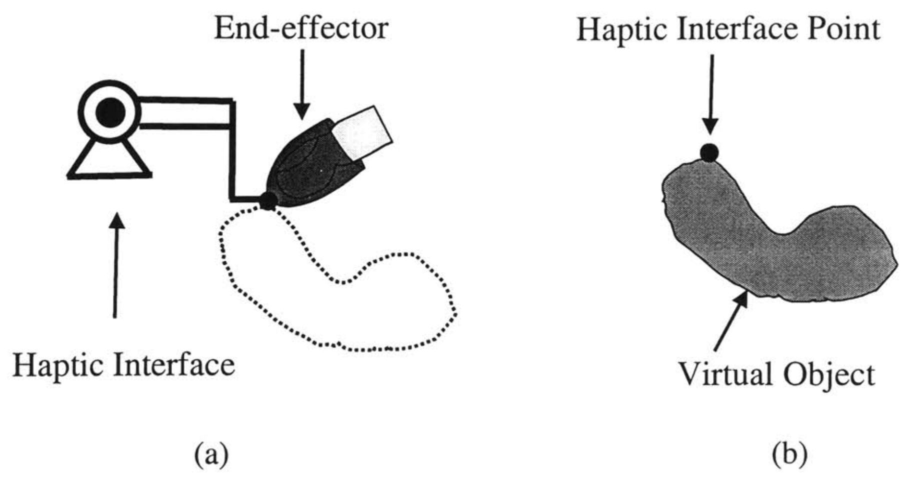


Figure 1-1. The concept about a human user interacting with virtual objects: (a) The user puts the index finger into the end-effector of the haptic interface. When the user moves the index finger, the end-effector will follow the movement, too. (b) The computer reads the position information of the end-effector and uses a model to represent the end-effector. In this example, the model of the end-effector is simplified to a point. The computer then checks whether the point has a collision with the virtual object or not. If there is no collision, the computer will not send any force to the haptic interface. However, if the point penetrates the object, the computer will send forces to the haptic interface to prevent the end-effector from further penetrating into the object.

Haptic display of 3D objects in virtual environments, with various applications in many areas, has been an exciting and challenging research topic for scientists and engineers in the last few years (Srinivasan, 1995, Burdea, 1996, Srinivasan and Basdogan, 1997, Salisbury et al, 1995, and Salisbury and Srinivasan, 1997). It is generally accepted that for haptic objects in VEs to appear natural, the haptic servo rate should be of the order of 1000 Hz. In other words, the computational time for each haptic servo loop (where a reaction force is computed for one position input) should take about a millisecond or less. If the 3D objects in VEs are simple primitives (e.g. cube, cone, cylinder, sphere, etc.), this goal could be easily achieved with the computer technology available today. However, realistic synthetic environments usually contain multiple 3D objects that have complex surface as well as material properties. Therefore, the development of efficient haptic interaction techniques that can render arbitrary 3D objects in a time-critical manner becomes essential.

The research of this thesis focuses on two fundamental fields of computer haptics. The first one is a *haptic interaction paradigm* that defines the nature of the "haptic cursor" and its interaction with object surfaces. The second one is an *object property display algorithm* that renders surface and material properties of objects. Finally, various techniques developed in this research were incorporated into a surgical simulator to demonstrate the usefulness of these techniques.

A literature review is first described in the next chapter to give an overview about computer haptics. The hardware and the software architecture for creating haptic VEs are described in chapter 3. After that, I describe a *point-based* haptic interaction paradigm in chapter 4. By using the point-based rendering technique, we can create VEs that allow users to use a point probe to interact with arbitrary 3D polyhedral objects. The second haptic interaction paradigm that is called *ray-based rendering technique* is presented in chapter 5. The ray-based rendering techniques allow users to use a line probe to interact with virtual objects. In chapter 6, I present various object property display algorithms. These techniques could add different material properties to the virtual objects. The details of the surgical simulator developed in this research are described in chapter 7. The suggestions of future work are summarized in chapter 8.

Chapter 2

Background

2.1 Force-Feedback Devices

In haptic VEs, the typical functions of a force-feedback haptic interface are to sense the position of the user's hand and reflect forces back to the user. In the past few years, different types of haptic interfaces have been developed for different purposes. An important distinction among haptic interfaces is whether they are tactile displays or net force displays. The corresponding difference in interactions with VEs is whether the direct touch and feel of objects contacting the skin is simulated or the interactions are felt through a tool. Simulation of interactions through a tool, such as feeling the virtual world through a rigid stick, requires only net force (and torque) display. Simulation of direct contact with objects is much more difficult since it requires a tactile display capable of distributing the net forces and torque appropriately over the region of contact between the object and the skin (see Ikei, Wakamatsu, and Fukuda, 1997 for a recent example of such devices). Since human tactile sensation is very complex and delicate, the performance of the currently available tactile displays is inadequate in comparison to the human sensory capabilities.

At present, net-force display devices that can match at least some of the capabilities of the human haptic system are available. Depending on how the interfaces are set up, they can be categorized as ground-based or body-based. The ground-based interface has devices attached to a fixed object. Whereas, the body-based interface is a freestanding device attached directly to human hands. Since the body-based devices usually provide larger workspaces, they require more complex calculation in rendering. Current available software technologies don't provide good enough support for such computation. In addition, human hand is a very versatile organ, the body-based devices developed today provides only a limited simulation. Therefore, general speaking, ground-based devices provide better performance in the creation of haptic VEs than the one simulated by body-based devices.

The following figures are examples of force-feedback haptic interfaces. The devices shown in Figure 2-1 to 2-6 are all capable of sending ground-based net force to the users through an end-effector. They could be used to simulate indirect contact. The algorithms developed in this research are for this type of haptic interfaces. The devices shown in Figure 2-7 and 2-8 are force-feedback gloves. These gloves are body-based devices. These devices could usually provide larger workspace compared to those shown in Figure 2-1 to 2-6. More haptic devices and the developing history of haptic interfaces could be found in Minsky, 1995 and Burdea, 1996.

Figure 2-1. The "PHANToM" by SensAble Technology, Inc.

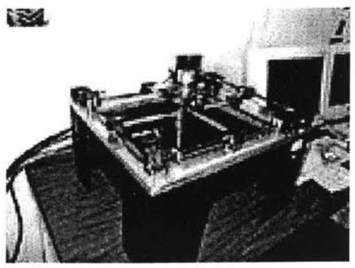


Figure 2-2. 3 DOF "Linear Haptic Display" by haptic technologies Inc.

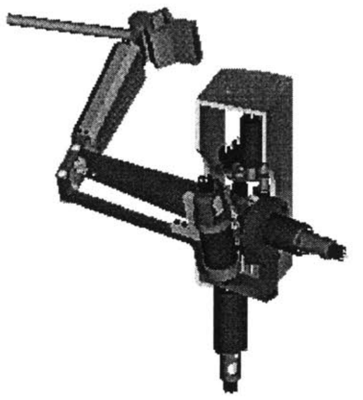


Figure 2-3. "Freedom-7" Haptic Device by Vincent Hayward at McGill University.

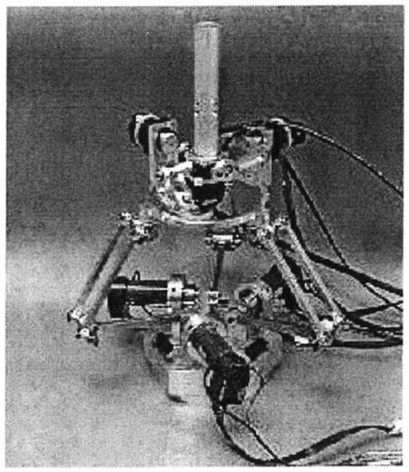


Figure 2-5. 6-DOF Haptic Interface by Tsumaki at Tohoku University

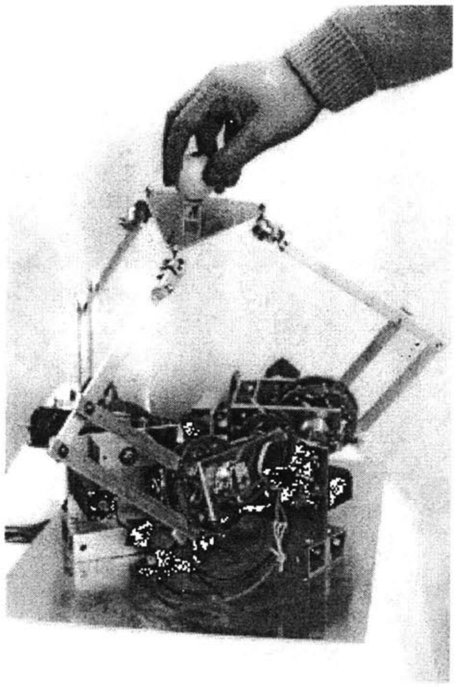


Figure 2-4. "Haptic Master" by Iwata at University of Tsukuba.

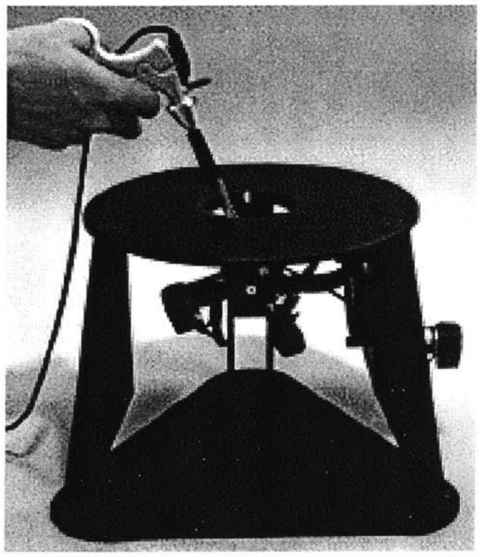


Figure 2-6. "Laparoscopic Impulse Engine" by Immersion Corporation.

Figure 2-7. "RM II" Force Feedback Device developed in Rutgers University.

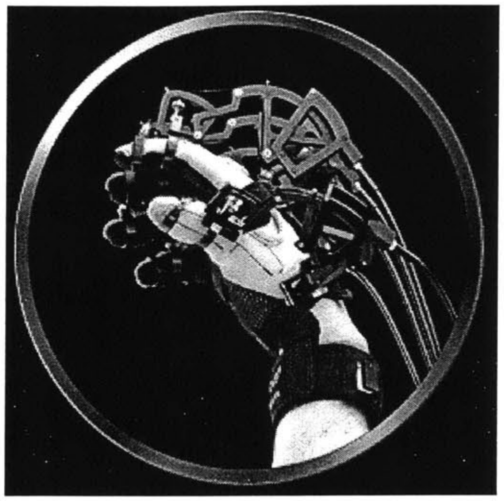


Figure 2-8. "CyberGrasp" by Virtual Technologies, Inc.

2.2 Rendering Algorithms

The goal of haptic rendering is to display the haptic attributes of surface and material properties of virtual objects in real time via a haptic interface device. As mentioned earlier, the user can touch and manipulate virtual objects only through an end-effector of a haptic device. This end-effector could be a thimble, a stylus, a mechanical tool, or an actuated glove. Since the users use the end-effector to explore the virtual environments, we use the general term "*probe*" to represent these end-effectors. In order to detect the collisions between the probe and the virtual objects, we have to create a mathematical model in the VEs to represent the probe. Although the probes can have different physical shapes and designs, they can be modeled as a point, a line, or a 3D object in virtual environments for simulating haptic interactions.

In order to help readers understand the process of haptic rendering, I give a simple example here. The probe is modeled as a point in this example. The virtual object we are going to create is a sphere (see Figure 2-9). The radius of the sphere is ***R*** and its center is located in **C** whose coordinate is (Cx, Cy, Cz). In each servo loop, we can get the location of the probe from the sensors of the haptic interface. Let's say we read the sensors and find that the probe is located at position **P** whose coordinate is (Px, Py, Pz) for the current loop. Next step is to do collision detection to see whether the probe is inside the sphere or not. To do this, we calculate the distance ***D*** between the probe and the center, which will be the square root of ((Cx-Px)*(Cx-Px)+(Cy-Py)*(Cy-Py)+(Cz-Pz)*(Cz-Pz)). If ***D*** is larger than ***R***, we know that the probe is outside the sphere. On the other side, if ***D*** is smaller than ***R***, the probe is inside the sphere and, therefore, we need a collision-response phase to calculate the interaction force. To decide the force, we need to find out the penetration first. The penetration direction *V* will be from the probe pointing to the center: (Cx-Px, Cy-Py, Cz-Pz). We normalize the vector *V* to make it a unit vector. After that, we can calculate the penetration vector *PV,* which is equal to (*R*-*D*)**V*. To calculate the interaction force, we use a simple mechanistic model $\bar{F} = -k\bar{x}$, where $\bar{x}$ is the penetration vector and k is the spring constant representing the material stiffness. So, the force that will be sent to the user will be (-*k* * *PV*). In each servo loop, the haptic rendering basically repeats the same procedure to compute the interaction force.

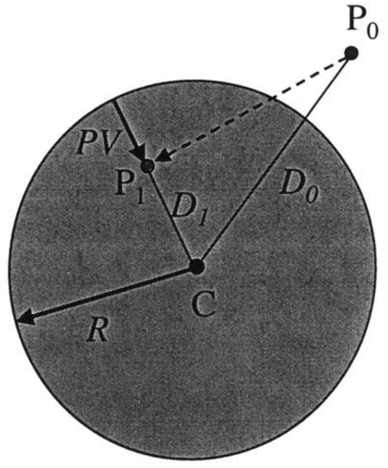


Figure 2-9. The procedure of haptic rendering between a point probe and a virtual spherical object. The center of the spherical object is located in **C** and its radius is **R**. At time $\mathbf{t_0}$, the point probe is located at $\mathbf{P_0}$ and, at time $\mathbf{t_1}$, it moves to $\mathbf{P_1}$. At time $\mathbf{t_0}$, the distance between the $\mathbf{P_0}$ and **C** is larger than **R.** Therefore, the point probe is outside the spherical object. At time $\mathbf{t_1}$, the distance between the $\mathbf{P_1}$ and **C** is smaller than **R,** which means that the point probe is inside the spherical object. We can calculate and find out that the penetration is the vector *PV*. By using a simple mechanistic model, we send a force $(-k * PV)$ to the user to prevent the point probe from further penetrating into the object.

Initial haptic rendering methods modeled the probe as a single point since it is computationally less expensive. In this haptic interaction model, the user could interact with 3D objects in VEs through the end point of the haptic device, which is defined as the Haptic Interface Point (HIP). Massie and Salisbury (1994) developed the PHANToM haptic interface device and proposed a simple method to render 3D geometrical primitives. Later, Zilles and Salisbury (1995) developed a more sophisticated, constraint-based method to render polyhedral objects. They defined a "god-object" point, to represent the location of a point that is constrained to stay on a particular facet of the object when the HIP penetrates that object. Lagrange multipliers are used to compute the new location of the god-object point such that the distance between the god-object and the haptic interface point is minimized. Adachi, Kumano, and Ogino (1995) and Mark et al. (1996) suggested an intermediate representation (a tangent plane) to render virtual surfaces. Although the forces are updated frequently (~1 kHz), the tangent plane is updated more slowly. Ruspini, Kolarov, and Khatib (1997) proposed an approach similar to the god-object technique and improved the collision detection algorithm by constructing a bounding sphere hierarchy and configuration space. Gregroy et al. (1999) proposed a framework for fast and accurate collision detection by combining many of the techniques in computer graphics. Ho, Basdogan, Srinivasan (1999) have also proposed a rendering method called "*Neighborhood Watch*" to render polyhedral objects. The local connectivity information is used to reduce the computation and make the computational time independent of the number of polygons of the objects in the VEs. In addition to the techniques that render polyhedral objects, Avila and Sobierajski (1996) proposed techniques to render 3D volumetric objects. Salisbury and Tarr (1997) proposed a method to render implicit surfaces. Thompson, Johnson, and Cohen (1997) proposed a method for rendering NURBS surfaces. Since the probe is modeled as a point for these methods, I call them *point-based* haptic rendering.

In a departure from the point-based methods described above, we have proposed a *ray-based* interaction technique where the probe is modeled as a line segment rather than a point (Ho, Basdogan, and Srinivasan, 1997 and Basdogan, Ho, and Srinivasan, 1997). Using ray-based interaction technique, we can simulate the contact between the tip as well as side of the probe with several convex objects at the same time. We can then compute the associated forces and torques to be displayed to the user.

One further step for improving haptic interaction paradigm might be to model the probe as a 3D object. However, the detection of collision between the 3D probe and 3D objects is computational too expensive for haptic rendering since the required update rate is about 1000 Hz. Instead of directly detecting collision between 3D probe and 3D objects, researchers have proposed alternative methods to approach this goal. For example, McNeely, Puterbaugh, and Troy (1999) have presented a voxel-based approach. In their approach, static objects in the scene are divided into voxels and the probe is modeled as a set of surface points. Then multiple collisions are detected between the surface points of the probe and the voxels of the static object. The calculation of interacting forces is based on a tangent-plane force model. A tangent plane whose normal is along the direction of the collided surface point is constructed at the center of each collided voxel. Then, the net force and torque acting on the probing object is obtained by summing up all of the force/torque contributions from such point-voxel intersections. Although this approach enables 3D probe to interact with static rigid objects, its extension to dynamical and deformable objects would significantly reduce the haptic update rate because of the computational load. Moreover, rendering of thin or small objects will also have problems with this approach.

In addition to the interaction paradigms described above, various techniques have been proposed for displaying surface properties such as shape, friction, and texture of virtual objects. Most of these techniques are developed for polyhedral objects. One of the advantages for polyhedral representation of 3D objects is that any type of objects can be represented in polyhedral format within an arbitrary tolerance. However, due to the sensitive perception of our haptic system, the users can easily feel the discontinuity at the connection of two faces that may be undesirable for some applications. To solve this problem, Morgenbesser and Srinivasan (1996) have proposed *force-shading* methods to smooth the feel of polyhedral objects by eliminating the force discontinuities at polygonal boundaries. In addition to smooth the object surfaces, researchers have also proposed techniques to add roughness to the surfaces. Salcudean and Vlaar (1994), Salisbury et al. (1995), Chen et al. (1997) and Green and Salisbury (1997) have proposed different techniques to simulate friction on smooth surfaces. Minsky et al. (1990 and 1995) proposed algorithms to simulate textures on 2D surfaces. Siira and Pai (1996) and Fritz and Barner (1996) have also proposed methods to generate stochastic textures on well-defined surfaces. Basdogan, Ho, and Srinivasan (1997) presented techniques to add haptic textures to arbitrary 3D polyhedral objects.

Some readers may think that the interaction paradigms for computer haptics are identical to those collision detection and collision response techniques in computer graphics. Indeed, many of the techniques in these two areas are identical. However, the basic concept is different between the two research areas. The main difference is that the program can fully control the behavior of the objects in graphic VEs, not in haptic VEs. For example, in graphic VEs, when an object moves toward another object and, finally, a collision occurs (Figure 2-10), the program could send forces to the two collided objects and change their velocity to make the two objects separated. Since the program can control the behavior of the objects, the overlap between the two objects usually occurs only in a small amount of time. Therefore, it is usually good enough for the collision response algorithms to make decision based on objects' current positions.

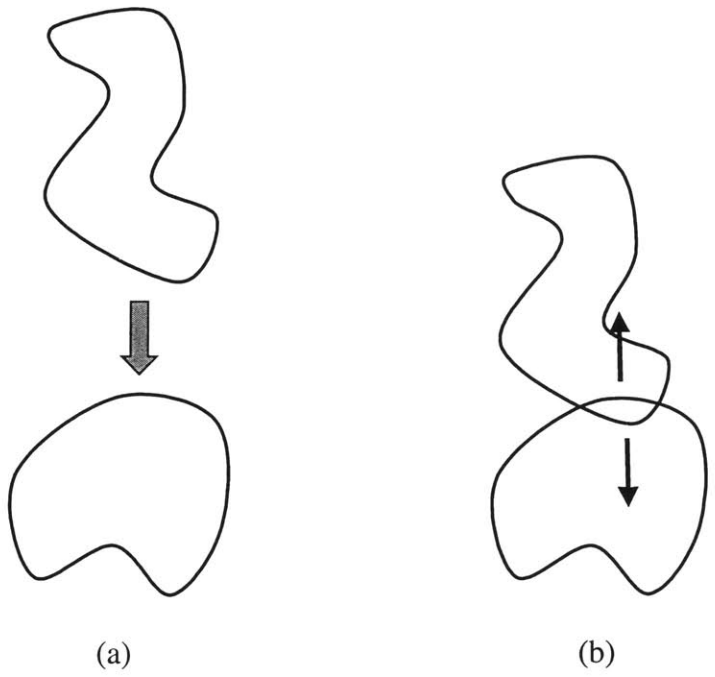


Figure 2-10. A simulation of dynamic behavior in computer graphics. (a) One object moves toward another object. (b) The moving object contacts the other object. The collision detection algorithm could detect the overlap between the two objects. The collision response algorithm is then used to change the velocity of the two objects.

Compared to graphic VE, the programs for haptic VE do not have full control on the probe. The user is the one that controls the behavior of the probe. Therefore, the contact time between the probe and virtual objects is typically very long. This makes most of the algorithms in computer graphics inappropriate for computer haptics. One example is the simulation of a probe contacting a cube (see Figure 2-11). The probe approaches from the top to contact the cube (Figure 2-11(a)). Based on our daily experiences, we know that the probe should contact the top surface of the cube and the direction of the interacting force should always be up (Figure 2-11(b)). If we calculate the collision based on the current configuration, we will get a wrong answer telling us that the probe is contacting the right surface of the cube (Figure 2-11(c)). And the direction of interacting force will be towards right side, which is not correct.

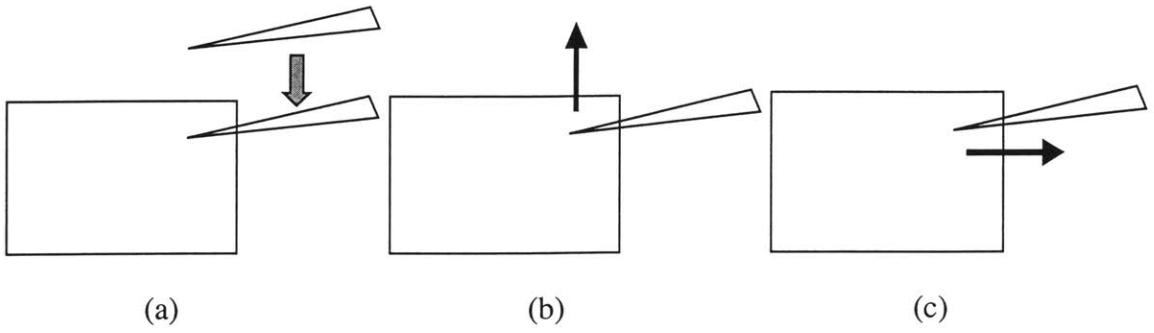


Figure 2-11. A simulation of a probe contacting a cube. (a) The probe approaches from the top to contact the cube. (b) Based on our daily experiences, we know that the direction of the interacting force should always be up. (c) If we calculate the collision based only on current configuration, we will get a wrong answer telling us that the direction of interacting force is towards the right side.

Other examples showing that the algorithms in computer graphics might give wrong answers could be found in Figures 2-12 to 2-14. In Figure 2-12, a long rod moves from the right side to contact a thin object (Figure 2-12(a)). The correct direction of interacting force should be towards right side (Figure 2-12(b)). However, most of the algorithms will suggest a direction either up or down (Figure 2-12(c)). In Figure 2-13, a thin paper moves towards another thin paper. The two papers are perfectly parallel. Since it is a discrete-time system, the program can update the position of the moving paper only in certain times. It is very likely that the moving paper passes the other paper between a time interval without contacting it. Therefore, no collision could be detected. In Figure 2-14, we try to simulate that an object moves towards a hole of another object (Figure 2-14(a)).

The ideal interacting force should be the one shown in Figure 2-14(b) since the object should be in the position indicated by the dash line. Using the existing algorithms, we will get wrong answers such as the one shown in Figure 2-14(c).

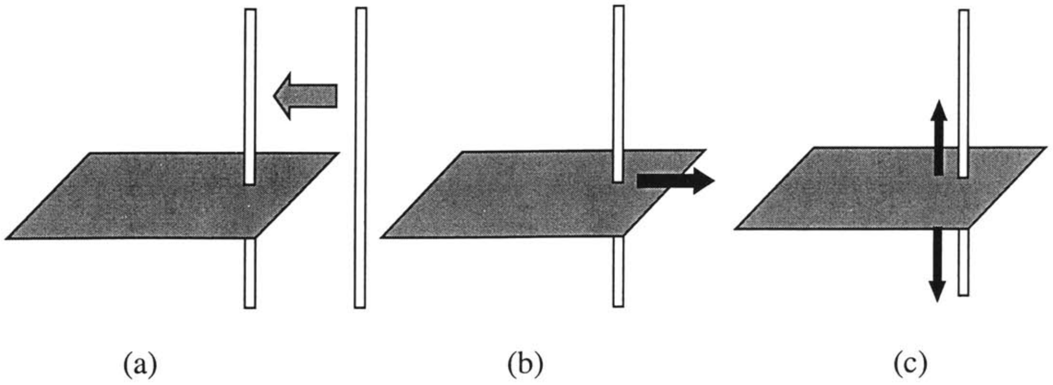


Figure 2-12. A simulation of a long rod contacting a thin object. (a) The long rod moves from the right side to contact a thin object. (b) The correct direction of interacting force should be towards the right side. (c) Most of the algorithms will suggest a wrong direction that is either up or down.

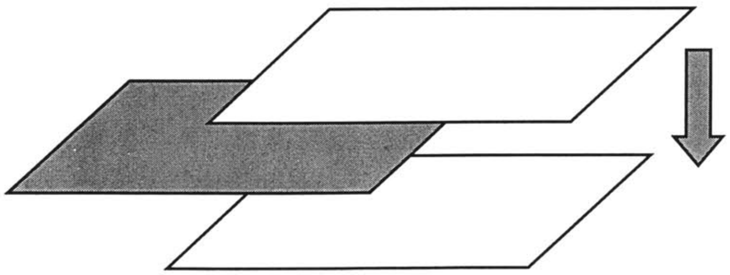


Figure 2-13. A thin paper moves towards another thin paper. The two papers are perfectly parallel. It is very likely that the moving paper passes the other paper between a time interval without contacting it. Since no collision could be detected in either previous or current configurations, no interacting force would be sent to the moving paper.

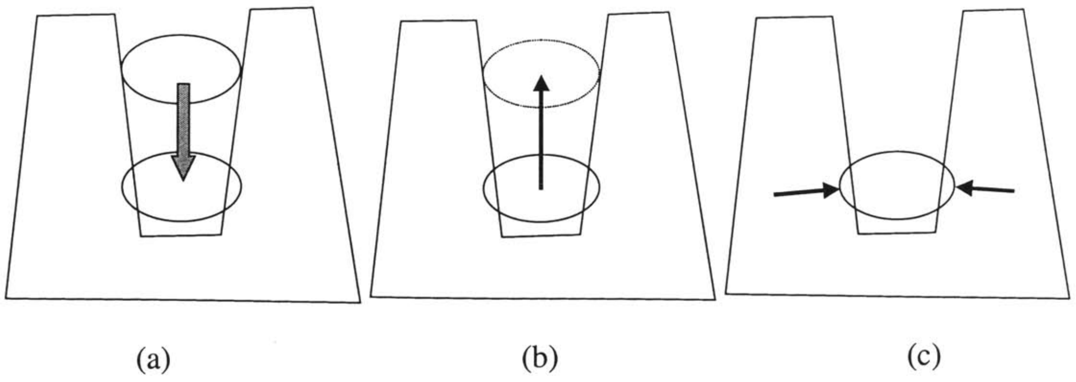


Figure 2-14. A simulation that an object moves towards a hole of another object. (a) The moving object moves from up towards down. (b) The ideal interacting force is towards up since the object should be in a position indicated by the dash line.. (c) With the existing algorithms, we will get wrong answers such as the one shown in the figure.

Chapter 3

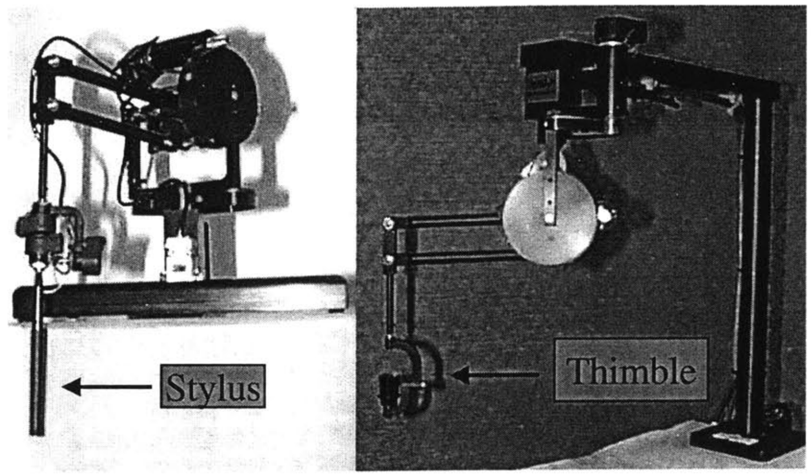
Hardware Set-up and Software Architecture

3.1 Hardware Set-up

The minimum hardware requirement for creating a haptic VE includes a force-feedback haptic interface and a computer. The haptic interface could sense the position of the probe that is held by the user and send interaction force to users. The computer performs the process of haptic rendering. That is, for a given position information, a computer checks whether the probe is inside the virtual objects or not. If the probe is inside the virtual objects, it sends force commands to the haptic interface to make the user feel the objects. Usually the VE system will also includes a graphic display device (such as a monitor or a head-mounted display device) to let the users visually see the virtual objects. The hardware set-up for the VR systems used in this research includes a computer, a monitor, and a force-feedback haptic interface.

3.1.1 Haptic Interface for Point-Based Interaction

In order to apply the point-based rendering techniques, the haptic interface should have the ability to sense 3D positional information and send 3D force back to users. The haptic interface device that I have used for the point-based rendering is a commercial product available in the market (called PHANToM from SensAble Technology, Inc.). The PHANToM can reflect forces along three axes and sense 3 or 6 degrees of position information (depending on the models). The probe attached to the end of the PHANToM could be a thimble in which the fingertip could be inserted (Figure 3-1(b)) or a stylus that could be held in the hand (Figure 3-1(a)).



(a) (b)

Figure 3-1. The PHANToM by SensAble Technology, Inc. Different probes could be attached to the end of the PHANToM.

3.1.2 Haptic Interface for Ray-Based Interaction

To implement the ray-based rendering techniques, the haptic interface device needs to have the ability to reflect back at least 5 DOF (degree of freedom) force. The haptic devices designed by Millman and Colgate (1991), Iwata (1993), and Buttolo and Hannaford (1995) are examples of such devices. To implement the ray-based algorithm, we have put together a haptic device set-up that is capable of displaying torques as well as forces. As mentioned earlier, the PHANToMs can reflect forces along three axes only. However, if two PHANToMs are connected to each other through a rigid probe, then a 5-dof force/torque display can be obtained (see Figure 3-2 and Figure 5-2). This configuration is the one we used to implement ray-based rendering and to conduct experiments described in Chapter 5.

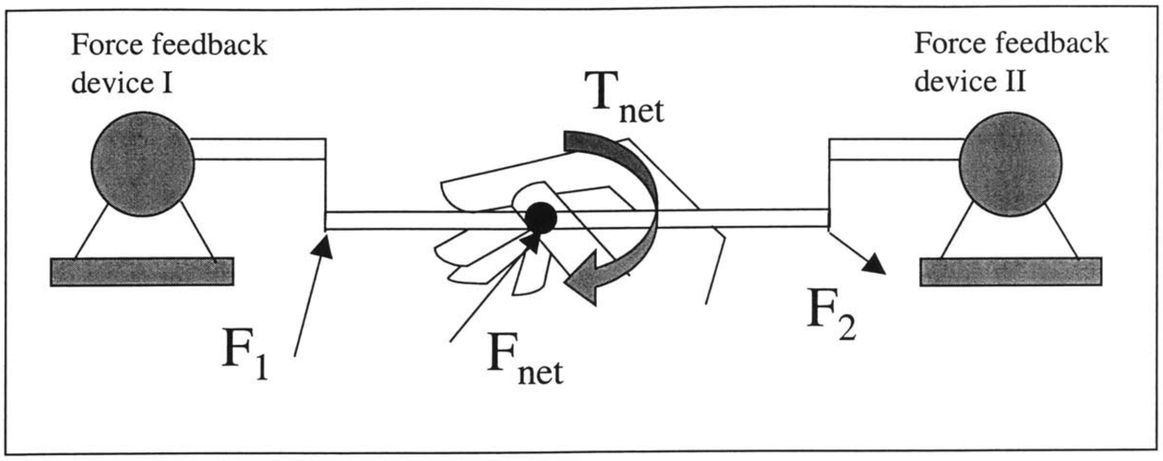


Figure 3-2. Schematic description of our force-feedback haptic interface. In order to display forces and torques to the user, we connected 2 force-feedback devices using a rigid probe. This enables us to display forces in 3 axes and torques about 2 axes (The torsion about the long axis of the probe cannot be displayed with this design).

3.2 Software Architecture

It is generally accepted that for the objects in VEs to appear natural, the graphic and haptic update rates need to be maintained at around 30 Hz and 1000 Hz, respectively. Therefore, to have optimal performance for a fixed computer power, the graphic and haptic loops have to be separated. There are two types of software architectures that can achieve this goal: multi-threading and multi-processing techniques (see Figure 3-3). In the multi-threading structure, both the graphic and haptic loops are processed in the same computer and share the same database. In this structure, the synchronization of the two loops in accessing to the data is important. In the multi-processing structure, the haptic and the graphic loops have their own copies of databases that are not shared. The two processes could run on the same machine or on different machines. The communication protocol between the two loops that ensure consistent between the graphic and haptic databases is important in this structure.

Based on our experience, both multi-threading and multi-processing techniques are quite useful in achieving stable haptic interactions and high graphic and haptic rendering rates. The choice of multi-processing or multi-threading structures should depend on the application. If the application requires large amount of data to be transferred between the two loops, we recommend multi-threading technique since it requires less programming effort and can achieve faster and more efficient communication. On the other side, if the application requires more computation or needs higher update rates, the multi-processing technique is recommended since it enables the user to process the two loops in two different machines.

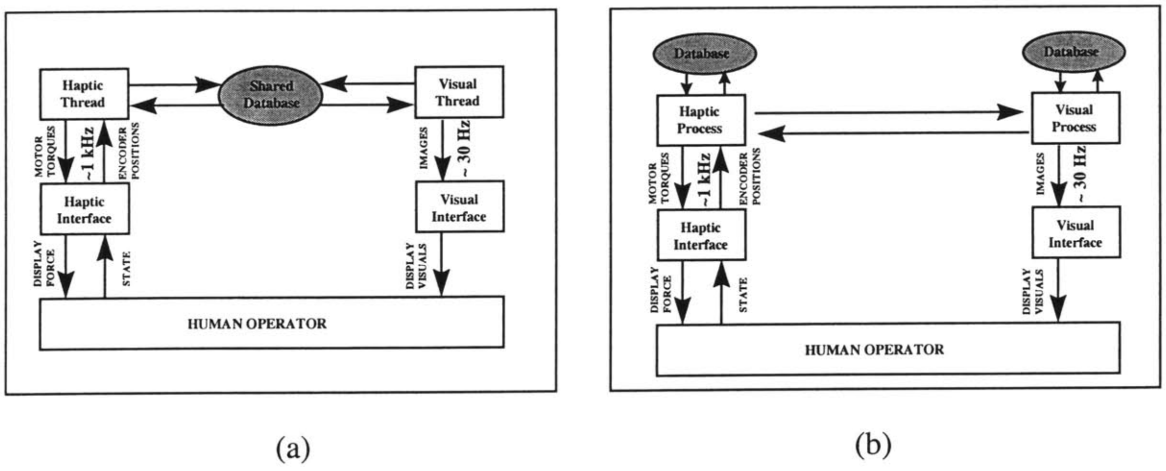


Figure 3-3. Software architectures for (a) multi-threading and (b) multi-processing. In the multi-threading structure, both the haptic and the graphic loops share the same data. In this structure, the synchronization of the two loops in accessing the data is important. In the multi-processing structure, the haptic and the graphic loops have their own copies of data that are not shared. The two processes could run on the same machine or on different machines. The communication protocol between the two loops that ensure consistent update of both graphics and haptics data is important in this structure.

Chapter 4

Point-Based Interaction

One of the major concerns for haptic rendering is that the computational time usually increases when the number of polygons in the VE increases. In such a case, the quality of the haptic interaction will depend on the complexity of the environment. If the interacting forces cannot be updated at a sufficiently high rate, contact instabilities occur. In order to have stable haptic interactions, a haptic rendering algorithm that makes the computational time essentially independent of the number of polygons of an object would be quite beneficial. In this chapter, I present a haptic interaction paradigm called "*Neighborhood Watch*" that utilizes a hierarchical database and a local search technique geared towards achieving this goal. In this rendering algorithm, the computational model of the probe is simplified to a point and the virtual objects are represented as polyhedrons. The reason I focused on polyhedral objects is that any type of objects can be represented in polyhedral format within an arbitrary tolerance. Also, most of the digitized models of real objects are in polyhedral format.

4.1 Introduction

There are two important issues any haptic interaction paradigm has to specify. The first one is the *collision detection* that detects the collisions between the probe and the objects in the scene. The second one is the *collision response* that computes the response to collision in terms of how the forces reflected to the user are calculated. A good collision detection algorithm not only reduces the computational time, but also helps in correctly displaying interaction forces to the human operator to make the haptic sensing of virtual objects more realistic. The collision detection and collision response techniques for haptics and graphics are slightly different. In computer graphics, collision detection (Cohen et al., 1995, Lin, 1993, Gottschalk, Lin, and Manocha, 1996, Smith et al., 1995, Hubbard, 1995) techniques are used to detect if two objects overlap. When the collision is detected, objects are separated from each other using collision response methods (Moore and Wilhelms, 1988, Baraff, 1994, Mirtich, 1995 and 1996). In general, the purpose of the collision detection and response in graphics is to avoid the overlap between objects and to simulate the behavior of objects following the overlap.

In contrast, the purpose of collision detection in haptic rendering is to check collisions between the probe and virtual objects to compute the interaction forces. When simulating interactions between the probe and the objects, the reflected force typically increases with penetration distance such that it resists the probe from further penetrating the object. Thus, the probe will always be inside the object during the collision response phase. This is a major difference between the collision response techniques developed for haptic and graphic interactions. In graphics, typically the existence of the overlap needs to be detected, followed by collision response algorithms to separate the overlapped objects. In haptics, the main goal is to compute the reaction force, instead of pushing the probe out of the objects. Hence the depth of penetration and how the penetration evolves are important. After detecting a collision between the probe and the objects, a simple mechanistic model such as the Hooke's law ($\vec{F} = -k\vec{x}$, where $\vec{x}$ is the penetration vector) can be used to calculate the force.

One simple way to determine the depth of penetration is to use the shortest distance between the probe and the object's surface (Massie, 1993, Massie and Salisbury, 1994). This approach works well for primitive objects such as a cube, sphere, cylinder, etc. However, the drawbacks of this technique are that it cannot display the objects that are small, thin, or polyhedral (Massie, 1993, Ruspini et al., 1996, 1997). Another approach to decide the depth of penetration is to use a constraint-based method (Zilles and Salisbury, 1995). This method defines an imaginary point, called the god-object point, which is constrained by the facets of the object. The penetration depth could then be defined as the distance from the god-object to the probe. This approach can be applied to polyhedral objects, even when the objects are thin or small. However, their proposed method requires different sets of rules to handle concave and convex objects. The "Neighborhood Watch" rendering algorithm presented in this chapter can handle both convex and concave objects uniformly. This algorithm is conceptually similar to the god-object algorithm (Zilles and Salisbury, 1995) but follows a different approach. The new approach reduces the computational time, makes the haptic servo rate independent of the number of polygons of the object, and results in more stable haptic interactions with complex objects.

In order to describe the concept more clearly for this approach, we need to define three types of probe. The first one is called *real probe* that is the physical piece held by the user and is the real 3D probe. The second one is called *haptic interface point* (HIP) that is the computational model of the real probe. It should be noted here that the probe is modeled as a point in this algorithm. The HIP could be the coordinate of the probe tip, the coordinate of the center of the probe, or the coordinate of a specific part of the probe. In the mathematical computation, the HIP is used to represent the real probe. The last one is called *ideal haptic interface point* (IHIP) that represents the ideal location of the HIP. The IHIP is similar to the god-object proposed by Zilles and Salisbury, 1995 or the proxy by Ruspini et al., 1997). Ideally, a probe cannot penetrate into any objects in a real world. Therefore, the IHIP is constrained so that it cannot penetrate into objects. If the HIP is outside the objects, the positions of IHIP and HIP will be the same. However, it the HIP moves into an object, the IHIP will be constrained to stay on the surface of the object.

The positions of HIP and IHIP play an important role in computing the interacting force (see section 4.4 for more details).

4.2 Hierarchical Database

In order to develop efficient collision detection and collision response algorithms, a processing stage called "pre-processing phase" is needed. The function of the pre-processing phase is to arrange the data in a way such that it can help in later computation. In our pre-processing phase, we build two types of data to achieve faster collision detection: the hierarchical bounding-box tree and the connectivity information between primitives. The bounding-box tree is useful in determining the contact situation between the probe and the objects when there is no contact in the previous loop. Once a contact occurs, the connectivity information could be used to perform local search for the up-coming loops since we know the probe cannot move too far in a small amount of time (since the servo rate is around 1 kHz, and human motions are relatively slow).

The first step in the pre-processing phase is to load/create the data for each object. The object data include the coordinates of each vertex and how the polygons are made from these vertices. After that, we construct another type of 2D geometrical primitive, namely, the lines that are simply the edges of the triangular polygons. As a result, the polyhedral objects in our own database are made of three types of 2D geometrical primitives: polygons, lines, and vertices. In order to implement a fast search technique for detecting collisions between the probe and 3D objects, we extend our database such that each primitive has a list of its neighboring primitives. Each polygon has neighboring primitives of lines and vertices, each line has neighboring primitives of polygons and vertices, and each vertex has neighboring primitives of polygons and lines (see Figure 4-1 and Figure 4-2). In addition to the connectivity information, we compute the normal vector of each primitive. For a polygon, the normal is the vector that is perpendicular to its surface and points outwards. For a line, its normal is the average of the normals of its neighboring polygons. For a vertex, its normal is the average of the normals of its neighboring polygons, linearly weighted by the associated angle subtended at the vertex.

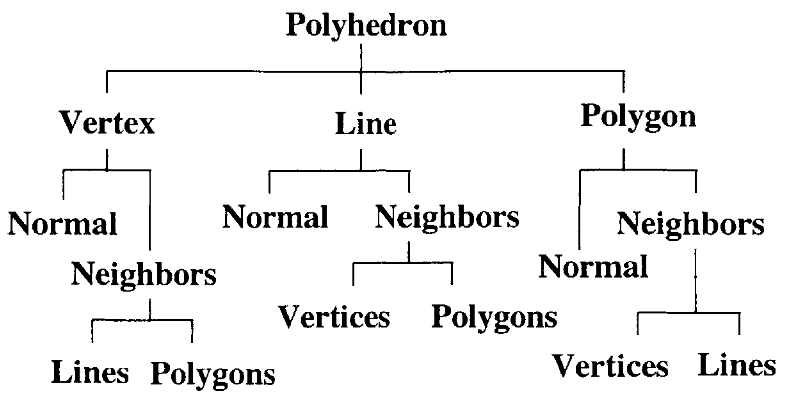


Figure 4-1. The connectivity information for primitives. The polyhedron representing the object is composed of three primitives: vertex, line, and polygon. Each primitive is associated with a normal vector and a list of its neighbors.

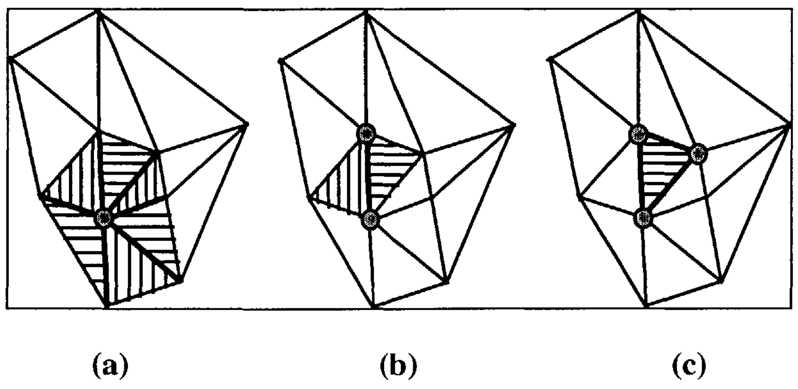


Figure 4-2. Illustration of how the neighbors of a vertex, line, and polygon are defined. (a) The vertex has six neighboring lines and six neighboring polygons. (b) The line has two neighboring vertices and two neighboring polygons. (c) The polygon has three neighboring vertices and three neighboring lines.

Another task performed in the pre-processing phase is creating the hierarchical bounding-box tree for each object (Gottschalk, Lin, and Manocha, 1996). At the top of the hierarchy is a bounding box that covers all of the polygons of the object. The polygons are then separated into two groups based on their geometric centers. We then create a bounding box for each group. These two new bounding boxes are placed under the first bounding box in the hierarchical tree. We repeat the same process to create two children bounding boxes for each parent bounding box at each hierarchical level of the tree until there is only one polygon left in each bounding box (see Figure 4-3).

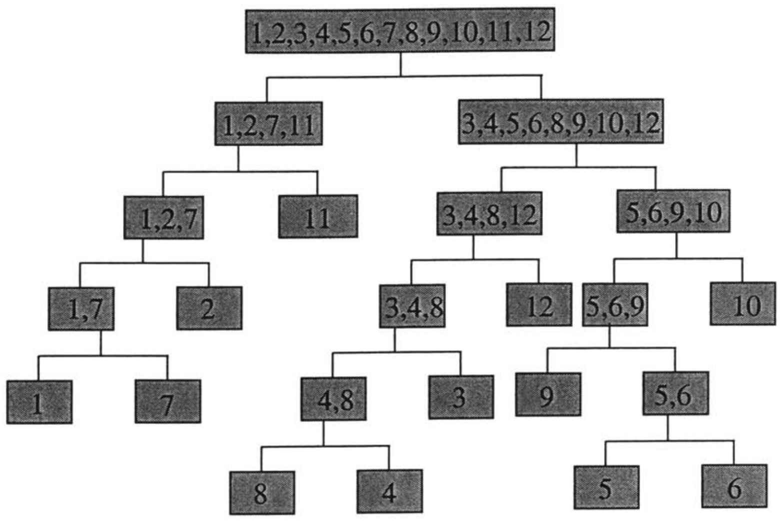


Figure 4-3. An example of hierarchical bounding-box tree. At the top of the tree is a bounding box that contains all 12 polygons. The top bounding box is separated into two bounding boxes at the second level. The separation of bounding box continues until there is only one polygon in each of the box.

4.3 Collision Detection

When exploring virtual environments, we interact with virtual objects through the real probe, which is computationally modeled as the haptic interface point (HIP) in the VEs. As mentioned earlier, we create another point called ideal haptic interface point (IHIP) in our program to represent the ideal location of the HIP. The HIP is not constrained and, consequently, it can penetrate the surface of virtual objects. However, we constrain the IHIP such that it cannot penetrate any objects. When the HIP is in the free space of the virtual environments, the location of the IHIP will be coincident with the HIP. If the movement of the HIP does not penetrate any object, the IHIP will keep following the path of the HIP. However, if the HIP moves into a virtual object, the IHIP will stay on the surface of the object. To computationally achieve this concept, we keep track of the path of HIP and check if this path penetrates any polygon. We construct a line segment between the coordinates of the HIP in the previous and current loops. We then detect whether this line segment (since the servo rate is around 1 kHz, and human motions are relatively slow, this line segment is very short) has an intersection with the polygons of the 3D objects or not. To achieve fast checking, we utilize "hierarchical bounding boxes" approach (Gottschalk et al., 1996). We first check whether the line segment is inside the bounding box of the objects or not. If it is outside of the bounding box, the line segment cannot have an intersection with the polygons. If it is inside the bounding box, we then check if the line segment is inside any of the two children bounding boxes in the next level of the hierarchical tree. If the line segment is outside the bounding box, the check is stopped. However, if the line segment is still inside one of the bounding boxes, we check with the lower level of that bounding box in the hierarchical tree again. As intersections are detected with successive bounding boxes along one particular branch of the tree, the last intersection is checked between the line segment and the polygon that is inside the lowest level of bounding box of the tree. If the line segment penetrates a polygon, we set this polygon as the *contacted geometric primitive*. The IHIP will then be constrained to stay on the surface of this polygon. The nearest point from this polygon to the current HIP is set as the IHIP and the distance from the IHIP to the current HIP is set as the depth of penetration. Although the first contacted geometric primitive is always a polygon and the IHIP is assigned to be on the surface of this polygon, it can easily be a line or a vertex in subsequent iterations.

Once the first contact is detected, we can use the local search techniques to reduce the computational time for the subsequent loops. The concept to support the local search techniques is that the probe cannot move too far away from the current position in just one loop. (Since the typical servo rate is around 1 kHz and human motions are relatively slow, the movement of the probe in one loop is usually less than one millimeter.) Therefore, if we know the contacted geometric primitive in the previous loop, we can start from that primitive and check collision only with its neighbors. The connectivity information described in the previous section is used to perform this local search. The local search approach significantly reduces the number of computations and also makes them essentially independent of the number of polygons that represent the objects. So, in the next iteration, we calculate the nearest distances from the current HIP to the contacted geometric primitive and its neighboring primitives. For example, if the contacted primitive is a polygon, then we check the distance from the current HIP to the neighboring lines and vertices. If the contacted primitive is a line, we check the distance from the current HIP to the neighboring polygons and vertices. Then, we set the primitive that has the shortest distance to the current HIP as the new contacted geometric primitive and move the IHIP to a point that is on this primitive and nearest to the current HIP (see Figure 4-4). This rule-based algorithm is repeatedly applied for the ensuing interactions. From the description in Figure 4-5 and Figure 4-6, we can see that this rule-based algorithm works with both non-convex and thin objects

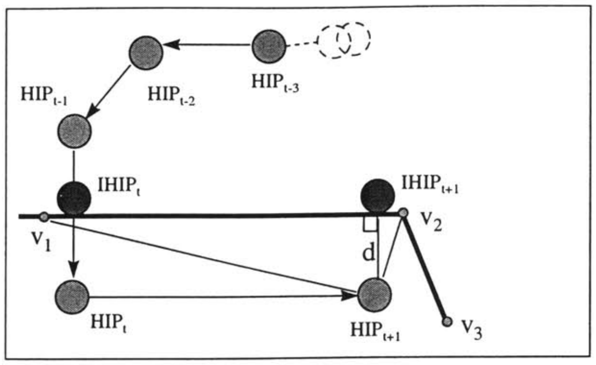


Figure 4-4. Haptic interactions between the probe and 3D objects in VEs. Before the collision occurs, HIP is outside the object surface and is identical with IHIP (see HIP$_{t-3}$, HIP$_{t-2}$, and HIP$_{t-1}$). When the HIP penetrates into object at time t, the IHIP is constrained to stay on the surface. At time t+1, HIP moves to a new location (HIP$_{t+1}$) and the new location of IHIP is determined by the current HIP and the neighboring primitives based on the nearest distance criterion.

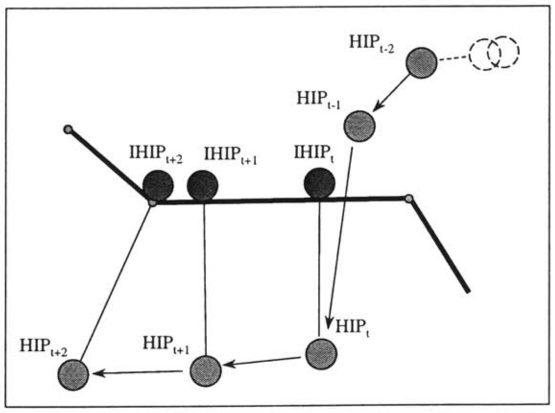


Figure 4-5. Haptic interactions between the probe and a non-convex object in VEs. Before the collision occurs, HIP is outside the object surface and is identical with IHIP (see HIP$_{t-2}$ and HIP$_{t-1}$). When the HIP penetrates into object at time t, the IHIP is constrained to stay on the surface. At time t+1 and t+2, HIP moves to new locations (HIP$_{t+1}$ and HIP$_{t+2}$). The nearest distance criterion could be used to find the new locations of IHIP.

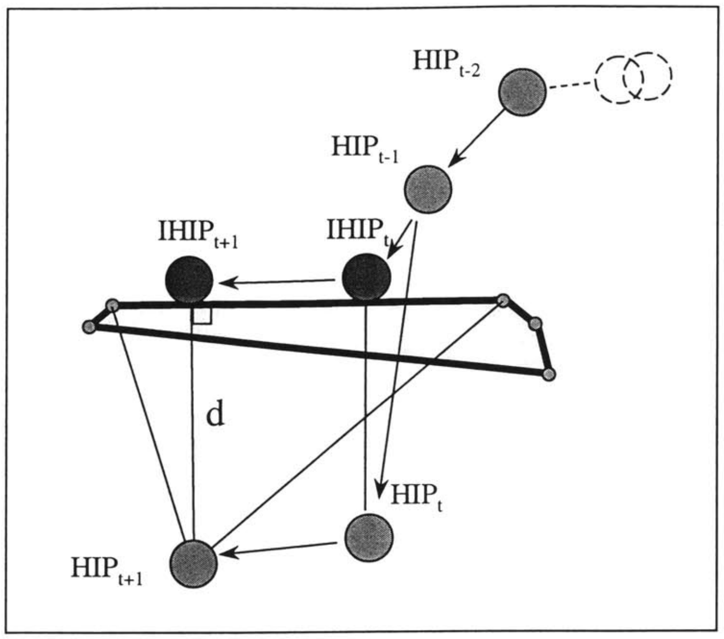


Figure 4-6. Haptic interactions between the probe and a thin object in VEs. Before the collision occurs, HIP is outside the object surface and is identical with IHIP (see HIP_{t-2} and HIP_{t-1}). When the HIP penetrates the object at time t, the IHIP is constrained to stay on the surface ($IHIP_t$). At time t+1, HIP moves to a new location (HIP_{t+1}). The nearest distance criterion will still work in finding the new location of IHIP.

In each cycle, we also need to check if the current HIP is still inside the virtual object. For this purpose, we construct a vector from the current HIP to the IHIP. If the dot product of this vector and the normal of the contacted primitive is negative, the current HIP is no longer inside the object and there is no penetration any more. If the dot product is positive, then the current HIP is still inside the object. The pseudo-code for this "Neighborhood Watch" haptic interaction algorithm is given below.

```
if (collision == FALSE)
{
    if (the path of HIP penetrates a polygon)
    {
        Set the polygon as the contacted geometric primitive;
        Move the IHIP to a point on this polygon that is closest to the HIP;
        collision ← TRUE;
    }
}
else
{
    contacted geometric primitive ← the contacted geometric primitive in the previous loop;
    primitive1 ← contacted geometric primitive ;
    distance1 ← closest distance from current HIP to primitive1;
    repeat {
            primitive1 = contacted geometric primitive;
            for ( i = 1 : number of neighboring primitives of primitive1)
            {
                primitive2 ← the i^th neighboring primitive of primitive1;
                distance2 ← distance from current HIP to primitive2;
                if (distance2 < distance1)
                {
                    contacted geometric primitive ← primitive2;
                    distance1 ← distance2;
                }
            }
    } while (primitive1 != contacted geometric primitive)

    Move IHIP to a point that is nearest from the contacted geometric primitive to current HIP
    vector1 ← vector from current HIP to current IHIP;
    normal1 ← normal of the contacted geometric primitive;

    if (dot product of vector1 and normal1 < 0)
        collision ← FALSE;
}
```

Using this algorithm, we can render both convex and concave objects in an efficient manner. The computational time for detecting the first collision will be in the order of log(N) for a single object, where N is the number of polygons (Gottschalk et al., 1996). After the first collision, we only calculate the distances from the current HIP to the contacted primitive and its neighbors to determine the new location of IHIP. Therefore, the servo rate will be fast since it only depends on the number of neighbors of the contacted geometric primitive. For a homogeneously tessellated polyhedron, as N increase, because the number of computational operations for searching the neighbors of each primitive is about the same, the servo rate will continue to be independent of the total number of polygons after the first collision.

4.4 Collision Response

In haptic rendering, the collision response focuses on the computation of reaction force that arises from the interactions between the probe and the 3D objects. Although the collision response phase has been studied in computer graphics (Baraff, 1994; Mirtch and Canny, 1995; Moore and Wilhelms, 1988), its implementation to haptics shows some differences.

The first step in collision response phase is to find the penetration depth. In this "Neighborhood Watch" algorithm, the penetration depth is simply the position difference between the HIP and the IHIP and the penetration direction is from the IHIP to HIP. Then, the interacting force could be computed based on the penetration depth. One way to calculate the force is to use a simple mechanistic model such as the Hooke's law ($\bar{F} = -k\bar{x}$, where k is the spring constant and represents the material stiffness, $\bar{x}$ is the penetration vector starting from IHIP to HIP, see Figure 4-7). We can choose a larger k to simulate a stiff object or choose a smaller k to simulate a soft object. A damping term ($-b\dot{\bar{x}}$) could be added to the model to add the damping effect. We can also change the model (such as: $\bar{F} = -k_1\sqrt{\bar{x}} - k_2\bar{x} - k_3\bar{x}^2$) to simulate non-linear material properties.

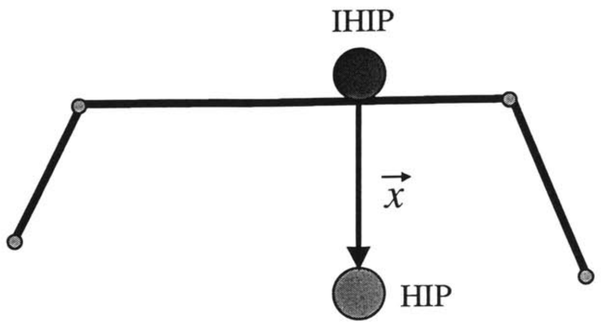


Figure 4-7. When the HIP penetrates into an object, the IHIP stays on the surface. The penetration vector $\vec{x}$ is defined such that it points from the IHIP to HIP.

4.5 Results

I have successfully applied the rendering techniques described in this chapter to render various objects both on a Windows NT platform and a Unix platform (see Figure 4-8 and 4-9). In order to demonstrate the efficiency of our haptic interaction technique, we did a test to compare the rendering rate for different objects. The structure of the programs includes two separate loops, one for updating graphics and the other for displaying force. The graphics update rate is around 30. The polygon number of the objects we tested ranges from hundreds to thousands (see Table 4-1). It can be seen from table 4-1 that the haptic servo rate is approximately constant even if the number of polygons is increased by approximately 5000 times.

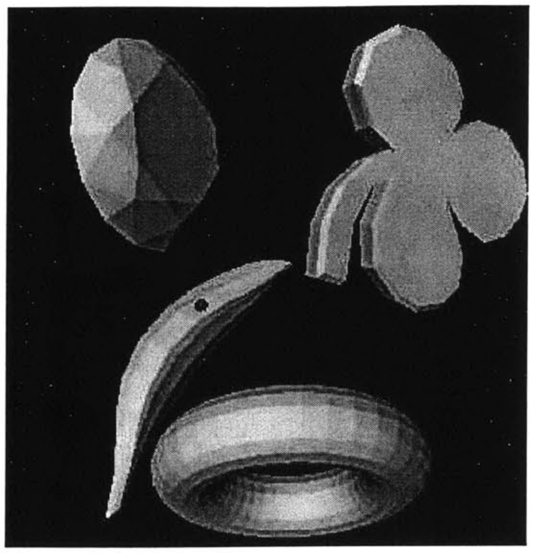
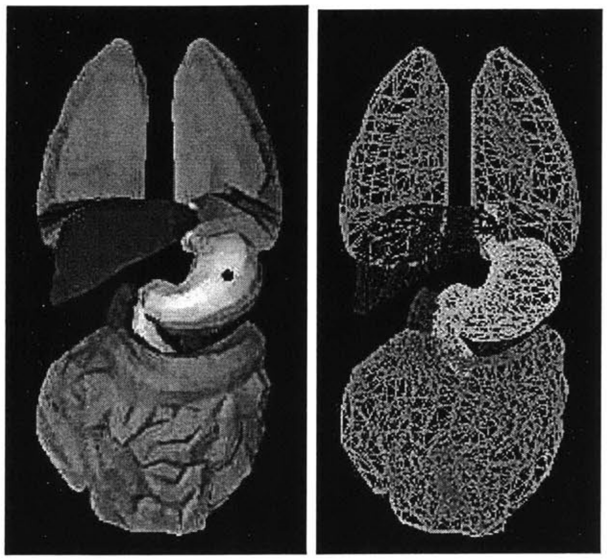


Figure 4-8. Rendering of various 3D virtual objects. The small dark dot represents the haptic probe. The user uses this haptic probe to interact with the virtual objects.



(a) (b)

Figure 4-8. Rendering of 3D virtual organs. (a) The small dark dot represents the haptic probe. The user uses this haptic probe to interact with the virtual organs. (b) The virtual organs are all polyhedrons composed of triangular polygons.

Table 4-1. Haptic rendering rates for various 3D rigid objects. The results were obtained with a Pentium II, 300 MHz PC running two threads and equipped with an advanced graphics card (AccelECLIPSE from AccelGraphics). Servo rate results are based on rendering 3D objects for at least 3 minutes. Rendering test is repeated at least three times for each object.

	Object 1	Object 2	Object 3	Object 4
Number of vertices	8	239	902	32402
Number of lines	18	695	2750	97200
Number of polygons	12	456	1800	64800
Haptic servo rate (kHz)	~ 12 to 13	~ 11 to 12	~ 11 to 12	~ 9 to 10

Chapter 5

Ray-Based Interaction

Most of the haptic rendering techniques developed so far are point-based in which the probe is simplified to a point. This approach is computationally less expensive, but it only enables the user to feel interaction forces, not the torques. Since the probe is modeled as a point, the net force is the only haptic feedback that could be sent to the user. For exploring the shape and surface properties of objects in VEs, these methods are probably sufficient and could provide the users with similar force feedback as what they would get when exploring the objects in real environments with the tip of a stick. However, point-based methods are not sufficient to simulate *tool-object* interactions that involve multiple constraints. The computational model of the simulated tool cannot be reduced to a single point since the simulated tool can easily contact multiple objects and/or different points of the same object simultaneously. Moreover, the resulting reaction torque has to be computed and reflected to the user to make the simulation of haptic interactions more realistic.

Modeling haptic interactions between a probe and objects using ray-based technique has several advantages over the existing point-based techniques. First of all, side collisions between the simulated tool and the 3D objects can be detected. User can rotate the haptic probe around the corner of the object in continuous contact and get a better sense of the object's shape. In point-based methods, one of the common unrealistic feelings is that the user's hand could go inside the objects although the point probe is still outside (see Figure

5-1). This situation could be eliminated using ray-based rendering. Second, ray-based rendering provides a basis for displaying torques to the user. Using the ray-based rendering algorithm, we can compute the contact points, depth of penetration, and the distances from the contact points to both ends of the probe. Then, we use this information to determine the forces and torques that will be displayed to the user. Third, the ray that represents the probe can be extended to detect the collisions with multiple layers of an object (in fact, this is the reason why we name the technique as ray-based). This is especially useful in haptic rendering of compliant objects (e.g. soft tissue) or layered surfaces (e.g. earth's soil) where each layer has a different material properties and the forces/torques depend on the probe orientation. Fourth, it enables the user to touch and feel multiple objects at the same time. If the task involves the simulation of haptic interactions between a tool and an object, ray-based rendering provides a more natural way of interacting with objects. Fifth, the reachable haptic workspace can potentially be extended using this technique since we have the full control of forces and torques that are displayed to the user. This means that it may be possible to create an illusion of touching distant objects by virtually extending the length of the probe and appropriately changing the direction and magnitude of the reflected forces (similar to seeing distant objects with a flash light, see Figure 5-2).

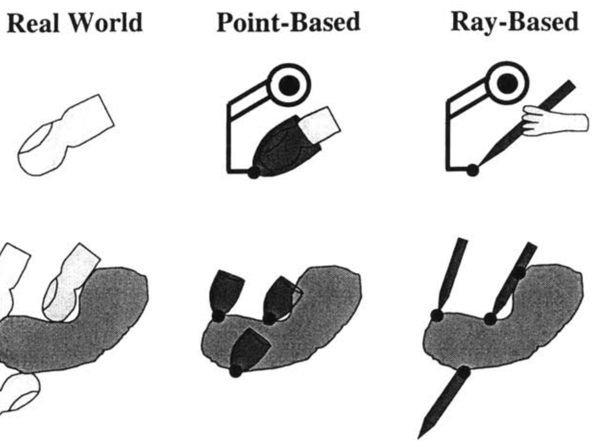


Figure 5-1. The difference in point- and ray-based rendering techniques: In point-based rendering, the haptic probe is modeled as a single point leading to artifacts such as feeling the bottom surface of the object by passing through the object. In ray-based rendering, end point as well as side collisions are detected between a line segment and virtual objects.

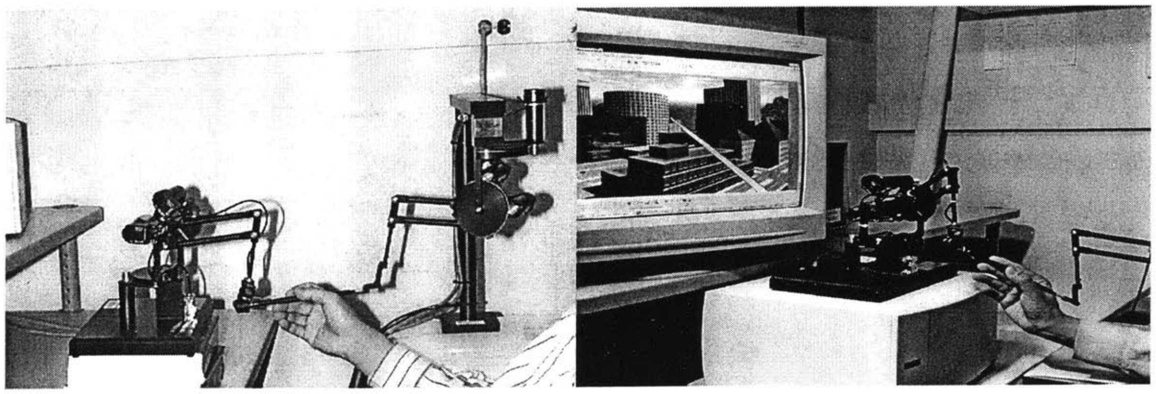


Figure 5-2. (a) Our two-PHANToM set-up for displaying forces and torques to the user using ray-based haptic rendering algorithm. (b) The reachable haptic workspace can potentially be extended using this technique since we have the full control of forces and torques that are displayed to the user.

For example, in performing minimally invasive surgeries, the surgeon inserts thin long rigid tubes into the body of the patient through several ports. Small size instruments attached to these tubes are used for manipulating internal organs. During the surgery, surgeon accesses the targeted area by pushing the organs and surrounding tissue to aside using the instruments and feels both the interaction forces and torques. A point-based technique is inadequate to fully simulate such haptic interactions between surgical instruments and virtual organs. If the instrument is modeled as a single point, the side collisions of an instrument with organs will not be detected and the instrument will pass through any organ other than the one touching the tip. We have observed that simulation of haptic interactions between the line segment models of laparoscopic instruments and organs using the ray-based technique could significantly improve the realism (Basdogan et al., 1998). In addition, multi-layered and damaged tissues whose reaction forces depend on the tool orientation can be simulated better using the ray-based technique since the ray can be extended along the contacted surface and the contacts in multiple layers could be detected to compute interaction forces.

Another example where the ray-based rendering is preferable would be the simulation of assembly line in car manufacturing. A scenario may involve a mechanic to go under a virtual car and turn the nuts of an engine block. Some of these procedures are done through mechanical instruments attached to a long and rigid shaft that enables the mechanic to reach difficult areas of the engine. Typically, the vision is limited and the mechanic finds his way around using haptic cues only. Moreover, the path to the nuts is usually blocked by several other mechanical components that make the haptic task even more challenging. The simulation of this procedure in virtual environments will certainly involve the modeling of torques and detection of multiple collisions simultaneously since a long rigid shaft is used to reach the targeted areas.

The details of the ray-based rendering techniques are presented in this chapter. In section 5.1, a general introduction of ray-based rendering is presented first. Following the introduction is the hierarchical database in section 5.2. The hierarchical database for ray-based is extended from the one for point-based. The collision detection and the collision response methods are presented in section 5.3 and section 5.4, respectively. In section 5.5, I present a haptic perceptual experiment to show that the ray-based rendering is better than the point-based one in human perception. The conclusions are presented in section 5.6.

5.1 Introduction

In ray-based rendering, we model the generic probe of the haptic device as a line segment and then detect collisions with 3D objects in the scene to compute the interaction forces/torques. In the point-based rendering, the point probe and virtual object can only have point-vertex, point-edge, or point-polygon contacts (see Chapter 4). However, the type of contacts between the line segment model of the haptic probe and virtual object can, in addition, be line segment-edge and line segment-polygon. There can also be multiple contacts composed of a combination of the above cases.

In order to reflect the forces properly to the user in haptic rendering, we has to consider the history of probe's movements, which is one of the main differences from graphical rendering. Although this means the tracking of probe's tip position in the point-based interactions, the tracking of probe's orientation has to be considered as well in the ray-based interactions. Therefore, the detection of collisions between a line-segment model of a haptic probe and arbitrary shaped 3D objects (i.e. convex and concave) is more complicated and computationally more expensive than that of point-based interactions. However, the advantages of the ray-based rendering over the point-based techniques that are mentioned earlier were still quite appealing. I have developed a rule-based algorithm that can successfully handle the collision conditions between a line segment model of a haptic probe and triangular convex objects. Although the algorithm is developed to render only convex objects at this stage, I do not consider this as a major limitation since the concave objects can always be divided into several convex objects. The proposed algorithm works well with convexified objects.

In general, three types of collisions occur when a convex object is explored with a line probe: (1) either end of the probe contacts with a polygon (we use triangles in our implementation) of the polyhedron and the intersection is a single point (*Point-Polygon,* see Figure 5-3a); (2) the probe collides with an edge of the polyhedron and the intersection is a single point that is in between the end points of the probe (*Line Segment-Edge,* see Figure 5-3b); (3) The probe comes to a position in which it is perfectly parallel to a polygon or a set of polygons of the object and the intersection is a portion of the probe (*Line Segment-Face,* see Figure 5-3c). Other types of contacts such as point-vertex, point-edge, and line segment-vertex are very unlikely to happen and will be covered by the three contact types mentioned above because the boundaries are included in the definition of edge and face.

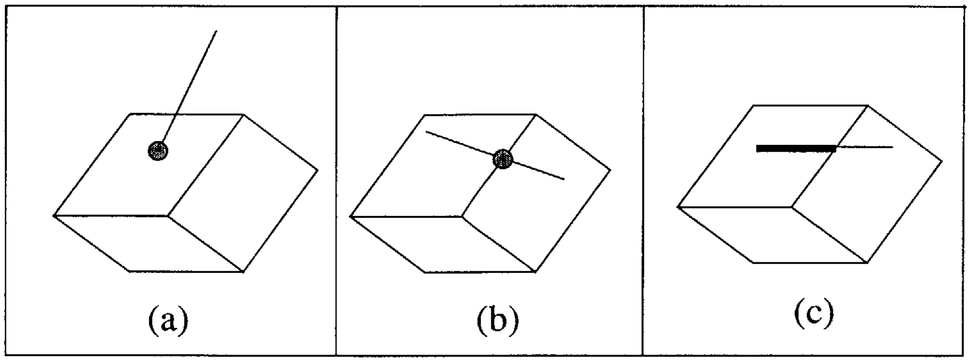


Figure 5-3. Possible contact conditions for the line segment-convex object interactions: (a) the probe contacts a polygon of the object at a single point (*Point-Polygon*), (b) the probe intersects an edge of the convex object at a single point (*Line Segment-Edge*), (c) the probe stays on the surface of a plane constructed from a single or multiple polygons and the intersection between the probe and the plane is either a portion or all of it (*Line Segment-Face*).

In order to describe the concept more clearly for this rendering algorithm, we need to define three types of probe: (1) *real probe*: the physical piece that is held by the user, (2) *virtual probe*: the computational model of the probe (i.e. line segment model) that is defined by the tip and tail coordinates of the real probe (3) *ideal probe*: the ideal location of the virtual probe that is constrained to stay on the surface when the virtual one penetrates into an object. The ideal probe is similar to the IHIP in point-based rendering. As we manipulate the real probe of the haptic device in the real environment, the line segment model of the probe (i.e. virtual probe) that is defined by the end points of the real probe is updated at each servo loop. The collisions of this line segment with the virtual objects are then detected. Although the line segment model of the virtual probe can be anywhere in 3D space, the movements of the ideal probe are restricted such that it cannot penetrate into objects. The location of the ideal probe relative to the virtual one is displayed in Figure 5-4 for each contact condition.

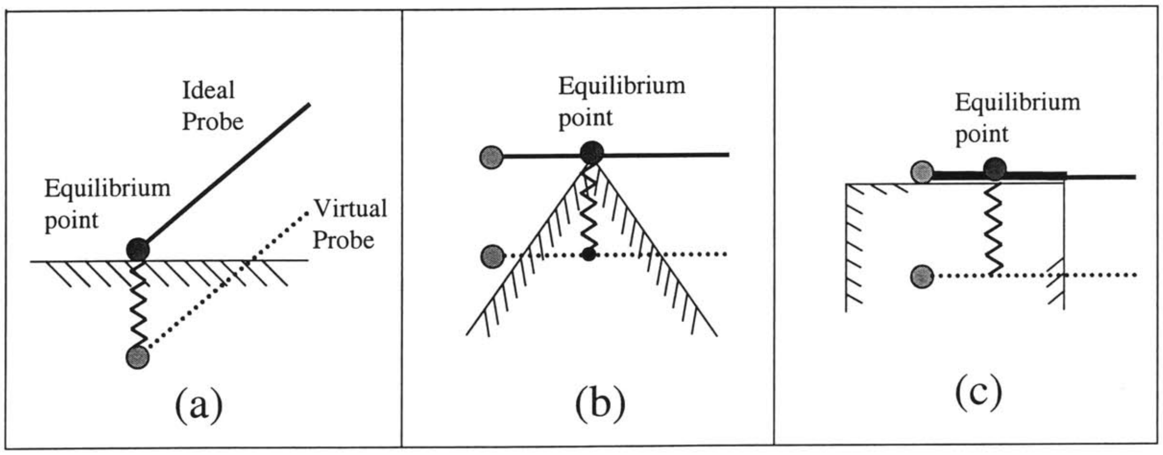


Figure 5-4. Location of ideal probe relative to the virtual one for the contact conditions shown in Figure 4. Although the virtual probe can penetrate into objects, the ideal one is restricted to stay outside the objects. During the interactions, the collisions of the virtual probe with objects are detected and the collision information is used to calculate the location of the ideal probe. The stretch of the spring in between the ideal and virtual probes in the figure illustrates a method for computing force interactions between the probe and the convex object for three different contact conditions.

In ray-based rendering, the collision-response phase is more complicate than that in the point-based one since we have to calculate the torques as well as the reaction forces. To properly distribute the forces to each force feedback device, we need to know the point of equilibrium on the ideal probe (see Figure 5-4). The net moment is computed according to this point to distribute the forces. The location of this point on the ideal probe changes with respect to the contact type. For example, in point-polygon collision (Figure 5-4a), the equilibrium point coincides with the tip point of the ideal probe. On the other hand, in line segment-face contact (Figure 5-4c), the location of this point depends on the portion of the probe that is in contact with the surface. The details of the collision detection and the responsc phases are described in the following sections.

5.2 Hierarchical Database

Similar to the point-based algorithm, the pre-processing phase is needed to develop efficient collision detection and collision response algorithms. The function of the pre-processing phase is to arrange the data in a way such that it can help in later computation. In the pre-processing phase of ray-based rendering, we build, again, two types of data to achieve faster collision detection: the hierarchical bounding-box tree and the connectivity information between primitives. The bounding-box tree is used to determine the contact situation between the probe and the objects when there is no contact in the previous loop. Once a contact occurs, the connectivity information could be used to perform local search for the up-coming loops.

The first step in the pre-processing phase is to load/create the data for each object. The object data include the coordinates of each vertex and how the polygons are made from these vertices. After that, we construct another type of 2D geometrical primitive: the edges of the triangular polygons. After the addition, the polyhedral objects in our database are made of three types of 2D geometrical primitives: polygons, edges, and vertices. Same as the one in point-based rendering, we extend our database such that each primitive has a list of its neighboring primitives. Each vertex has neighboring primitives of polygons and edges. The difference from the one in point-based rendering is that each polygon has neighboring primitives of vertices, edges, and polygons. And each edge has neighboring primitives of vertices, edges, and polygons. (Figures 5-5). We also compute the normal vector of each primitive (refer to section 4.2 for details).

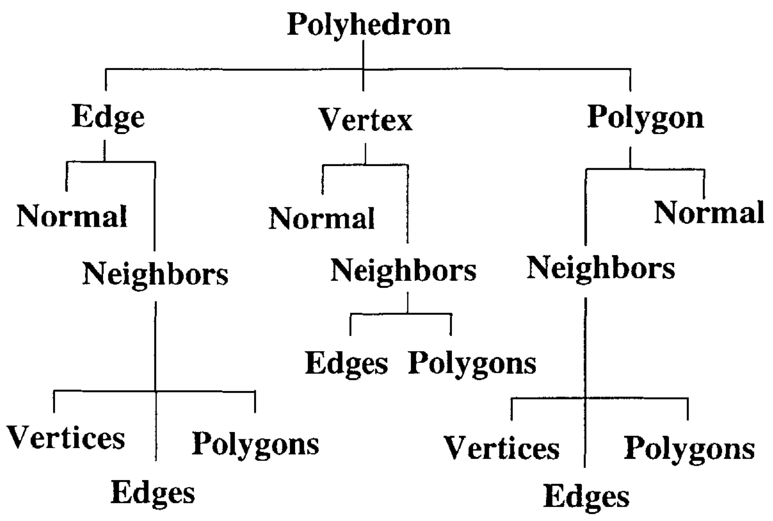


Figure 5-5. The connectivity information for primitives. The polyhedron representing the object is composed of three primitives: vertex, line, and polygon. Each primitive is associated with a normal vector and a list of its neighbors.

Another set of neighboring primitives we created in the pre-processing phase is the possible contacted primitives (PCPs). The PCPs of a primitive are those primitives that are likely to be contacted by the probe in the next loop when the primitive is currently contacted. For example, if a polygon is contacted in the current loop, its neighboring edges are likely to be contacted in the next loop. Therefore, the neighboring edges of the polygon are in the list of PCPs. If that polygon and other polygons together form a face, those polygons and edges that are in the same face will be in the PCPs list of the polygon, too. As mentioned earlier, there are only three types of contact when the user uses a line probe to explore a convex object in real environment (see section 5.1). The vertex is not one of the contacted primitives in the three contact situations. Therefore, the vertex is not in the PCPs list. After adding the PCPs list to the primitives, the database will look like the one in Figure 5-6.

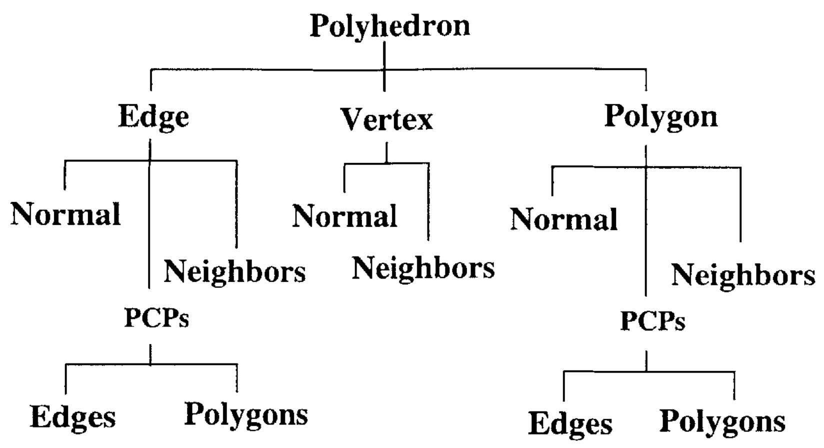


Figure 5-6. The connectivity information for primitives. The polyhedron representing the object is composed of three primitives: vertex, line, and polygon. Each primitive is associated with a normal vector, a list of its neighbors, and a list of possible contacted primitives.

Another task performed in the pre-processing phase is creating the hierarchical bounding-box tree for each object (Gottschalk et al., 1996). There will be two bounding-box trees for each object in ray-based rendering. The first bounding-box tree is for polygon primitives (refer to section 4.2 for details). The second bounding-box tree is for edge primitives. At the top of the hierarchy is a bounding box that covers all of the edges of the object. The edges are then separated into two groups based on their length centers. We then create a bounding box for each group. These two new bounding boxes are placed under the first bounding box in the hierarchical tree. We repeat the same process to create two children for each parent at each hierarchical level of the tree until there is only one edge left in each bounding box. This bounding-box tree is, in fact, similar to the one for polygons except that the primitives inside the bounding boxes are the edges.

5.3 Collision Detection

I have constrained the ray-based rendering method to handle only convex objects in order to simplify the computations. In fact, this simplification permits us to use the local search technique (called "Neighborhood Watch") that is described in Chapter 4. Finding the constrained position and orientation of the ideal probe using a localized neighborhood search makes the rendering rate independent of the number of polygons.

In general, the collision detection phase between the virtual probe and a virtual object is composed of two states: the probe did or did not have a collision with the object in the previous loop. Once this state in the previous loop is known, the following steps are followed to detect the subsequent contacts:

If there was no contact with the object in the previous cycle, we first check (1) if the movement of end points of the virtual probe contact with any polygon of the object (see Appendix A.5 for details), or (2) if the movement of the virtual probe crosses any edge of the object (see Appendix A.8 for details). To speed up the collision detection calculations, we first check if the virtual probe is inside the top most bounding box of the hierarchical tree. If no, the checking is finished since the probe has no chance to contact the object if it does not contact the bounding box. If so, we then check the collisions with the bounding boxes at the second level. This process is repeated until the lowest level is reached by progressing along one or multiple particular branches of the tree. Finally, we check if the virtual probe collides with the primitive (i.e. polygon or edge) that is inside the lowest level of the bounding-box tree. If either of the end points of the virtual probe penetrates a polygon, we have a point-polygon collision. If the virtual probe crosses any edge, we encounter line segment-edge collision.

If there was a certain type of contact with the object in the previous cycle (i.e. continuation of the previous contact) such as point-polygon, line segment-edge, or line segment-face, we then study all the possible contact conditions for the upcoming loops. For example, if the contact type in the previous cycle was point-polygon, then the possible contact conditions for the current loop can be point-polygon, line segment-edge, or line segment-face. Each contact condition and the possible contact conditions that may follow it are discussed in detail below.

1. <u>If there was a point-polygon collision in the previous loop</u>: The first step is to update the vector that defines the virtual probe. A line segment that connects the end points of the real probe defines this vector (V) and its direction is from the end that was in contact with the object in the previous loop to the other end point (see Figure 5-7). We then calculate the dot product of V and the normal (N) of the polygon that was contacted in the previous loop.

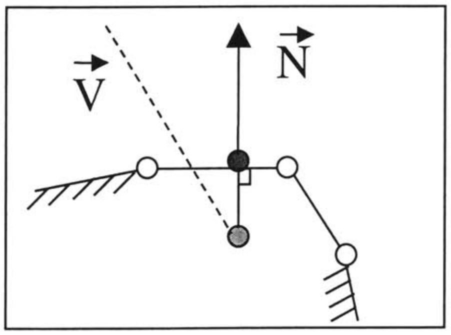


Figure 5-7. A point-polygon collision: In this figure, one end of the probe is already inside the object whereas the other end point is outside the object.

If the dot product of the collided polygon normal (N) and the vector (V) is larger than zero (i.e. if the angle between these two vectors is less than 90 degrees), we first project the collided end point of the probe to the plane of the previously collided polygon (see Appendix A.1) and then check whether the projected point is still inside the same polygon or not. If so, the type of contact condition in the current servo loop is a point-polygon collision again (Figure 5-8a). If not, the probe could have a point-polygon collision with a neighboring polygon (Figure 5-8b) or have a line segment-edge collision with a neighboring edge (Figure 5-8c). If the collided end point is above the surface containing the polygon, then there is no collision in the current cycle.

If the dot product of (N) and (V) is smaller than zero, we encounter a line segment-face contact (Figure 5-8d).

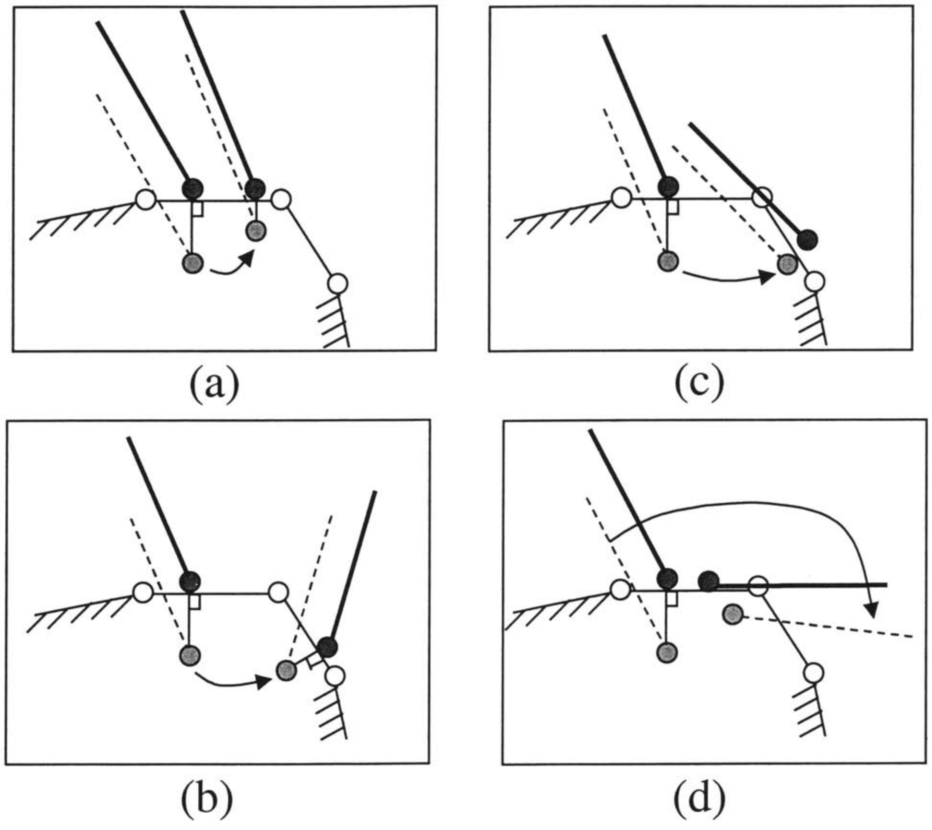


Figure 5-8. Possible contact types following a point-polygon collision: (a) point-polygon, (b) point-neighboring polygon, (c) line segment-edge, and (d) line segment-face.

2. If there is a line segment-edge collision in the previous loop: First, we find the projection of the virtual probe on a plane which contains the previously contacted edge and is parallel to the virtual probe (see Appendix A.4 for details). Then, we check whether the projected probe has an intersection with the previously collided edge or not. In addition to this check, we also define two angles (α and β) to describe the collision status of probe with the edge (see Figure 5-9a). Each edge primitive in the database has two neighboring polygons. The angle β is the angle between the *first* polygon (arbitrarily chosen) and the extension of the second polygon (see Figure 5-9a). Similarly, the angle α is the angle between the *first* polygon and the probe.

   If the value of α is larger than zero and smaller than the value of β and the projection of the probe (see Appendix A.4) has an intersection with the edge of the object, the probe should still be in contact with the same edge. If the probe is above the edge, then there is no contact.

   If (1) the value of α is larger than zero and smaller than the value of β, and (2) the projection of the probe (see Appendix A.4) does not have an intersection with the edge, the probe should either have a line segment-edge collision (see Figure 5-9b), a point-polygon collision (see Figure 5-9c), or no collision at all in the upcoming loop.

   If the value of α is smaller than zero or larger than the value of β, we infer that the probe has a line segment-face contact with a neighboring face.

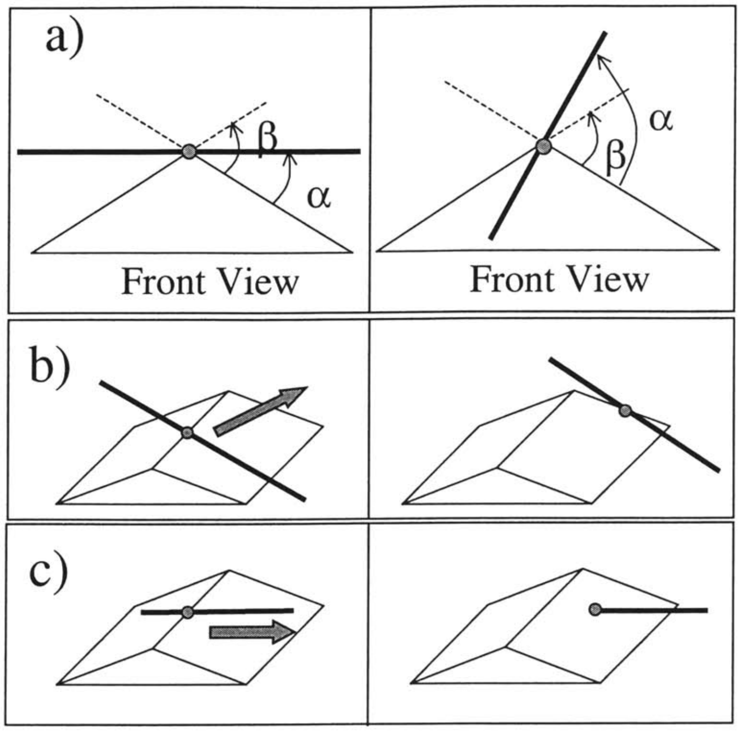


Figure 5-9. Possible collision situations following a line segment-edge contact.

3. <u>If there is a line segment-face collision in the previous loop</u>: We have already discussed the cases where the tip of the probe penetrates into object (point-polygon) and the probe collides with the edge of the object (line segment-edge). However, there is a situation, for example, where a part of the probe may be on the surface of the object. Or, when we touch the surface of a convex object with the tip of a probe and then rotate the probe around the contacted polygon in continuous contact: first, a single-point contact occurs (point-polygon), then the probe becomes parallel to the contacted surface of the object. We call the phase described in these examples as line segment-face where the term "face" refers to the face that is constructed from the collided polygon (i.e. since the probe lies on the surface of the object, it could be in contact with multiple polygons). For the detection of line segment-face contact, we

first define an angle (θ) that is between the probe and the face (see Figure 5-10). We, then check whether the projection of the probe on the plane of the face (see Appendix A. 3) still has a collision with the face or not.

If the value of θ is smaller than a user defined small angle epsilon (set at, say, 1°) and the projection of the probe (see Appendix A. 3) has a collision with the face, the probe should be in contact with the same face. If the probe is above face then there is no contact.

If the value of θ is larger than epsilon and the projection of the line probe (see Appendix A. 3) has a collision with the face, the type of contacts can be a point-polygon (see Figure 5-11a), line segment-edge (see Figure 5-11b), or no collision at all.

If the projection of the probe (see Appendix A. 3) does not have a collision with the face, we trace the path of the probe and find out which direction the probe moves out of face. Based on this direction, we determine whether the probe has a point-polygon collision, a line segment-edge collision, or no collision at all. To determine the type of contact, we use the direction of the movement in the following manner: (a) if the probe moves out of the face from an edge, we check if the end points of the probe is below the neighboring polygon of this edge. If so, then there is a point-polygon collision (see Figure 5-12), otherwise there is no collision. (b) if the probe moves out of the face through a vertex, we check if the probe is below any of the neighboring edges of the vertex (see Figure 5-12). If so, then there is a line segment-edge collision. To quickly determine the unqualified edges (i.e. a vertex can have multiple number of neighboring edges) in which the probe cannot possibly collide, we consider an imaginary plane. This plane contains the vertex in consideration and its normal is determined using the closest vector from the vertex to the projected probe (i.e. the probe is projected to the plane of face). If the neighboring edges of the vertex are behind this imaginary plane, then we do not consider them for a possible collision.

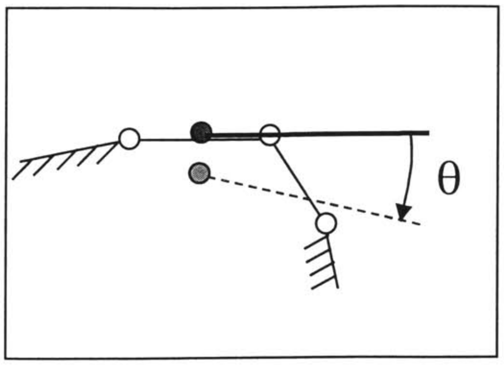


Figure 5-10. Line segment-face collision. The definition of the angle (θ) that is between the probe and the face is as shown on the figure.

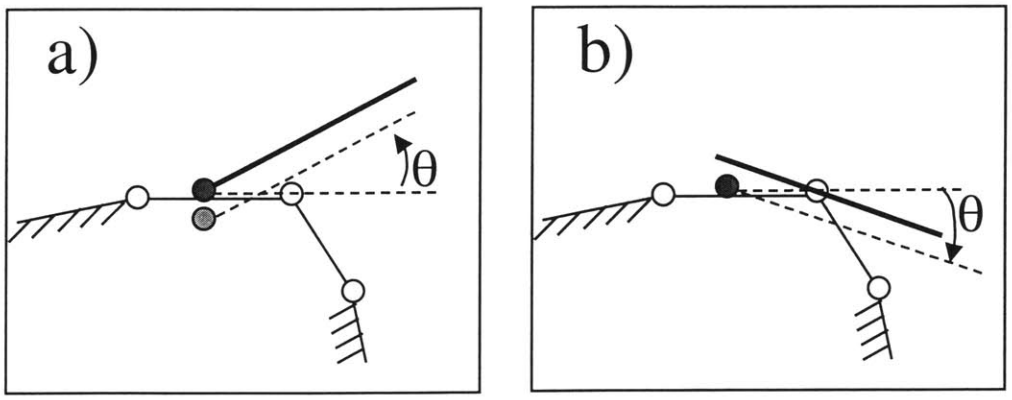


Figure 5-11. Possible future contacts if θ becomes larger than epsilon and the projection of the line probe has a collision with the face in line segment-face contact: (a) point-polygon, (b) line segment-edge.

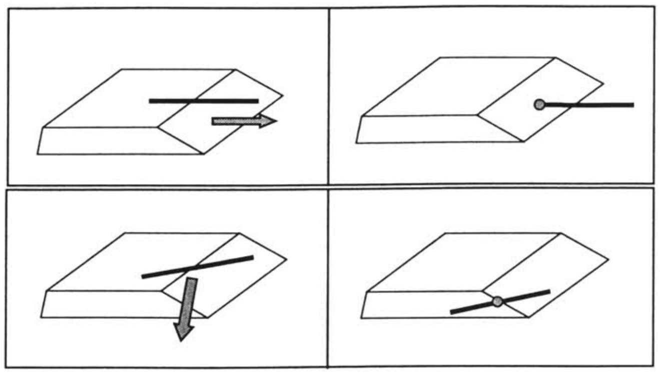


Figure 5-12. Possible contact types if the projection of the probe (see Appendix A.3) does not have a collision with the face when there is a line segment-face contact in the previous cycle: (a) point-polygon and (b) line segment-edge.

5.4 Collision Response

In haptic rendering, the collision response involves the computation of the ideal probe relative to its virtual counterpart and the reaction forces/torques that arise from the interactions with 3D objects. Although the collision response phase has been studied in computer graphics (Baraff, 1994; Mirtch and Canny, 1995; Moore and Wilhelms, 1988), its implementation to haptics shows some differences (Ho et al., 1999).

- The location of the ideal probe relative to the current location of the virtual probe: The probe that is held by the user is free to move anywhere in 3D space until its motion is constrained by force feedback or workspace boundary of the device. Since the virtual probe is allowed to penetrate into objects for the purpose of detecting collisions, we have to compute the location and orientation of the ideal probe for the calculation of forces/torques (see Figure 5-13)
- Computation of forces and torques that will be displayed to the user: During haptic interactions with virtual objects, the computer sends force commands to the haptic

device. This prevents the user from further penetrating into the objects. The forces and torques are computed based on the differences in the locations of the virtual and ideal probes.

The computation of ideal probe location and the interaction force depends on the type of contact. Each contact case is studied in detail below:

1. In point-polygon collision, we first determine the surface point on the collided polygon that is nearest to the end point of the virtual probe. Then, the virtual probe is projected to this nearest surface point to define the new position of the ideal probe while keeping its orientation the same as the virtual probe. Following the projection, the nearest surface point and the end point of the ideal probe coincides. This nearest surface point is called as the equilibrium point in Figure 5-4a since it enables us to compute interaction forces and torques. We assume that there is a virtual spring between the contacted end points of the virtual and ideal probes (see Figure 5-13a). The forces and torques are then computed based on the spring constant and the position difference between the virtual and ideal probes (see section 4-4 for details).

2. In line segment-edge collision, we first determine the plane that contains the collided edge and is parallel to the virtual probe. The virtual probe is projected to this plane (see Appendix A.4) to define the new position of the ideal probe. We then compute the intersection point of the collided edge and the ideal probe, which is called the equilibrium point in Figure 5-4b. To compute the net interaction force, we assume that there is a virtual spring between the virtual and ideal probes (see Figure 5-13b). The net force is then distributed to two Phantoms to display them to the user. The force reflected from each PHANToM is inversely proportional with the distance from the PHANToM in consideration to the collision point:

$$F_1 = \frac{L_2}{L} F_{net} \qquad F_2 = \frac{L_1}{L} F_{net} \qquad (5\text{-}1)$$

where, L is the total length of the stylus ($L_1 + L_2$), F_1 and F_2 are the forces reflected from PHANToM devices, and F_{net} is the force coming from the virtual spring.

3. In line segment-face collision, a part of the ideal probe lies on the surface of the object while the virtual one penetrates into the object. If the user rotates the probe even slightly, the type of contact may quickly change to point-polygon or line segment-edge. This is undesirable since it can cause instability. For this reason, the orientation of the probe relative to the object surface (angle θ in the Figure 5-10) plays an important role in computing the equilibrium point where the net force is acting. We first determine the contour points of the face that collides with the probe (note that the probe intersects the contours of the face at two points which are marked as A and B in Figure 5-13c). We then compute the distances from one end of the probe to these points (x_1 and x_2 in the Figure 5-13c). Now, we can compute the location of the collision point where the net force is acting as follows:

$$L_1 = (\frac{x_1 + x_2}{2}) + \left(\frac{\theta}{\theta_{epsilon}}\right)(\frac{x_1 - x_2}{2}) \qquad (5\text{-}2)$$

where, $L_2 = L - L_1$ and the angle θ is defined as the current orientation of the probe relative to its ideal orientation and it varies between $-\theta_{epsilon}$ and $\theta_{epsilon}$. In our simulations, $\theta_{epsilon}$ was chosen as one degree. For example, observe the collision response phases in Figure 5-13c: If θ is equal to $\theta_{epsilon}$, then L_1 becomes equal to x_1 and equilibrium point moves to the point A and the contact phase switches to line segment-edge. Similarly, if θ is equal to $-\theta_{epsilon}$, then L_1 becomes equal to x_2. The equilibrium point moves to the point B and contact phase switches to line segment-edge. For θ between $-\theta_{epsilon}$ and $\theta_{epsilon}$, L_1 will have a value between x_1 and x_2.

Following the computation of L_1 and L_2, the force that will be reflected from each PHANToM can be easily determined using Eq. (5-1).

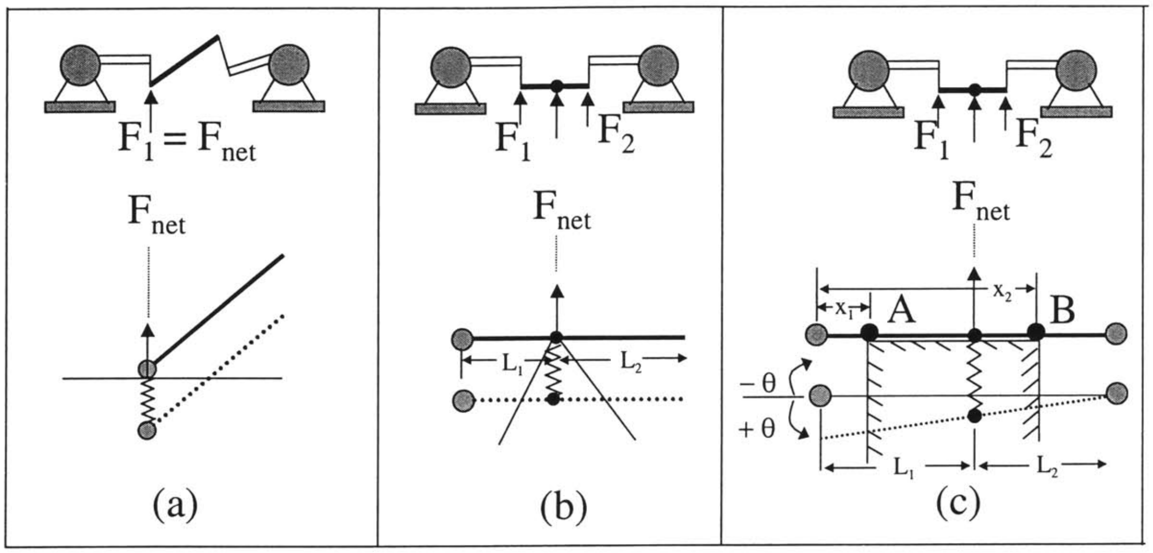


Figure 5-13. The collision response for ray-based haptic rendering: The ideal location of the probe is determined based on the type of collision (i.e. point-polygon, line segment-edge, or line segment-face). In the figure, dashed and solid lines represent the virtual and ideal probes respectively

In the computations above, the net torque from the two PHANToMs is equal to zero since the collision is described for a single object. However, the user may feel torques as well as the forces when there are multiple objects to interact in the scene and the movement of the probe is constrained. For example, if there are two convex objects and the user attempts to touch both of them at the same time as it is schematically described in Figure

5-14 and illustrated in an example in Figure 5-15, certain amount of torque and force will be felt. The net force (F_{net}) that will be reflected to the user's hand under this situation will be the vector summation of the forces F^a and F^b (see Figure 5-14). The net torque acting on the user's hand is computed as

$$T_{hand} = F^a r^a + F^b r^b \qquad (5\text{-}3)$$

The component of the force F^a that will be reflected from PHANToM-1 and PHANToM-2 can be computed using the Equation (5-1) as:

$$F_1^a = \frac{L_2 + r^a}{L} F^a \qquad F_2^a = \frac{L_1 - r^a}{L} F^a \qquad (5\text{-}4)$$

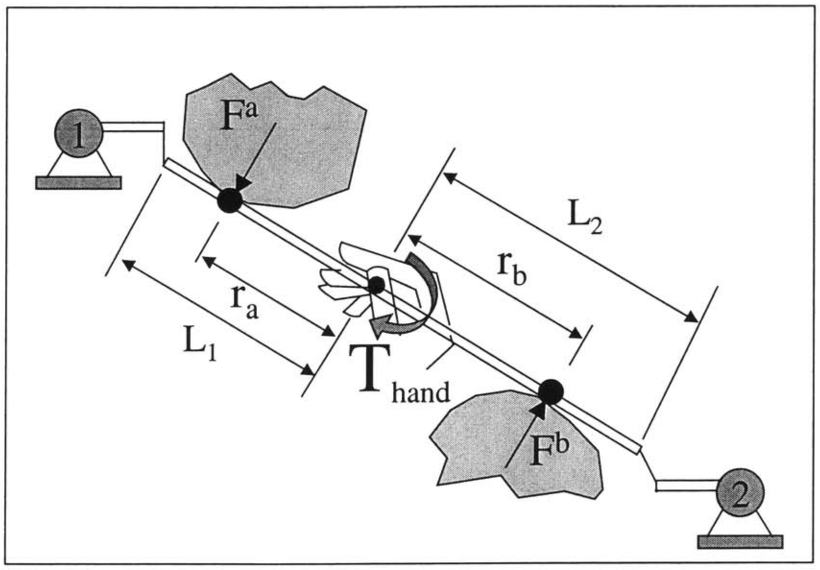


Figure 5-14. Computation of interaction forces/torques when the line probe interacts with two virtual objects: The forces are distributed to the PHANToMs based on the net force and moment principles.

Similarly, the force F^b is distributed between the two Phantom devices as:

$$F_1^b = \frac{L_2 - r^b}{L} F^b \qquad F_2^b = \frac{L_1 + r^b}{L} F^b \qquad (5\text{-}5)$$

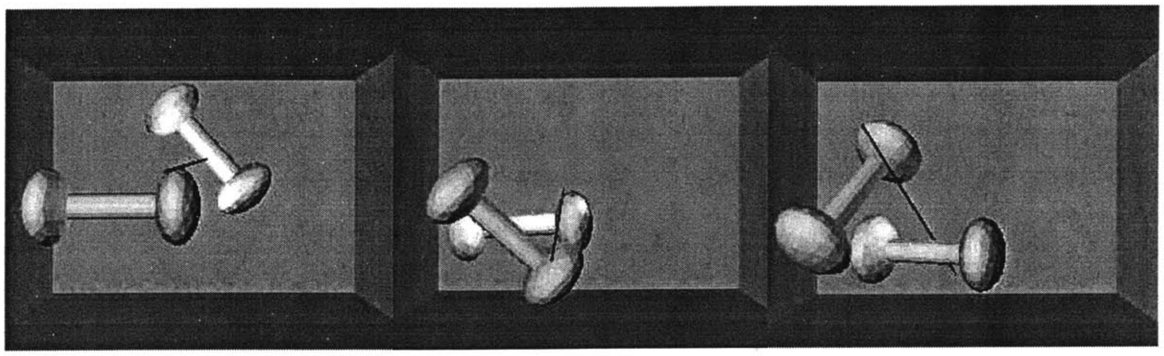


Figure 5-15. The floating dumbbells: The user can touch, explore, and manipulate two dumbbells at the same time with a single probe using the ray-based rendering technique.

5.5 An Experiment on the Haptic Perception of 3D Objects: Ray-based versus Point-based Interactions

In order to compare the effectiveness of the ray-based haptic rendering algorithm over the existing point-based rendering techniques in perceiving the shape of 3D objects, I designed and conducted a perceptual experiment. A total of 7 subjects participated in the experiment. Subjects were asked to identify the shape of 3D objects as quickly as possible using haptic cues only (i.e. no visual feedback was provided to the subjects). Four primitive objects (i.e. sphere, cone, cylinder, and cube), rendered either with the point-based or the ray-based technique, were randomly displayed to the subjects, one at a time, in various orientations. Each subject repeated the experiment for 304 times (i.e. each object was displayed for 76 times) under each of the two different conditions for haptic interactions (i.e. point-based versus ray-based). We measured the total time taken to identify an object for each subject and then compared the results for point-based versus ray-based rendering conditions. The results indicate that haptic perception of 3D objects with the ray-based technique is faster than the point-based for the cone and cube (see Figure 5-16). This is likely to be because (a) the sphere and cylinder can be efficiently identified solely by exploring them with the probe tip and (b) the cone and cube can be identified faster when side collision with the probe are rendered.

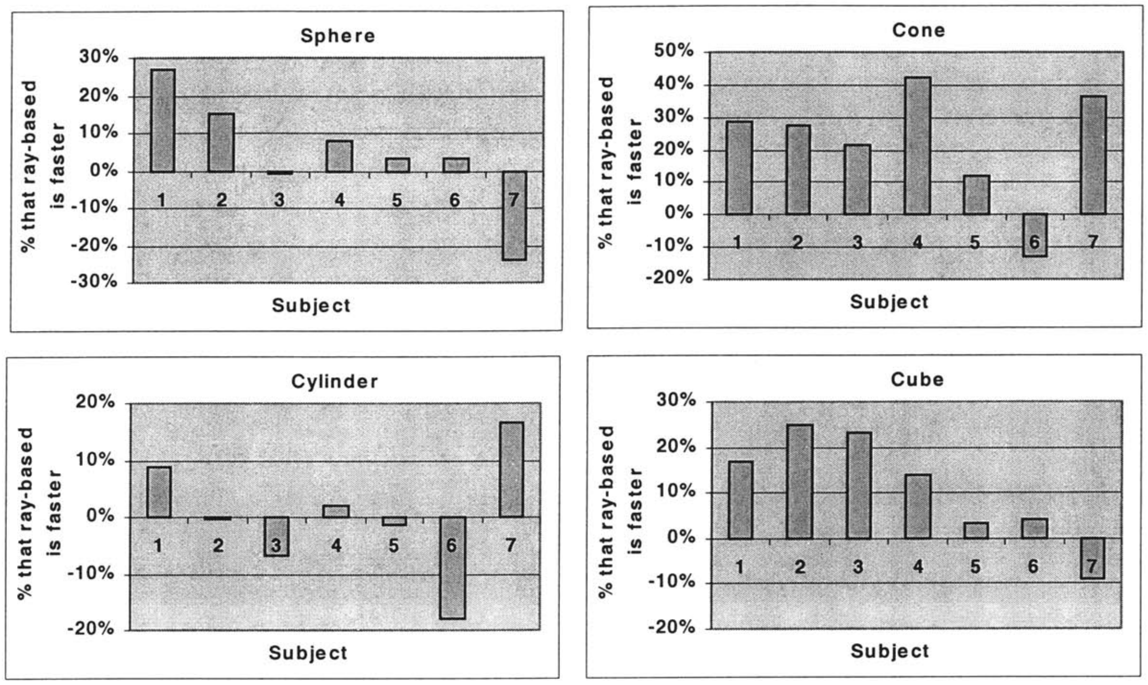


Figure 5-16. Comparison of the point- and ray-based rendering techniques: The use of the ray-based technique leads to a faster perception of cone and cube (on the average, 22% and 11% faster than point-based respectively). However, there were no significant differences for the sphere and cylinder.

5.6 Conclusions

In this chapter, I have presented a ray-based rendering technique for simulating the haptic interactions between the end effector (i.e. probe) of a haptic interface and 3D virtual objects. As compared to the point-based approaches that model the probe as a single point, the ray-based rendering technique models the probe as a line segment. Although the point-based approach simplifies the computations, it only enables the user to feel net forces. The ray-based rendering technique provides the user with torques in addition to the forces, both of which are essential in simulating tool-object interactions. Moreover, as the experimental study demonstrates, the haptic perception of some 3D objects using the ray-based rendering technique is better than the existing point-based techniques when there are no visual cues available. Although the ray-based rendering algorithm proposed in this chapter works with convex polyhedral objects only, it can also handle concave objects if they are represented as a combination of multiple convex objects. I have, for example, successfully rendered convexified 3D objects using the ray-based rendering technique (see Figures 5-15, 5-17, and 5-18).

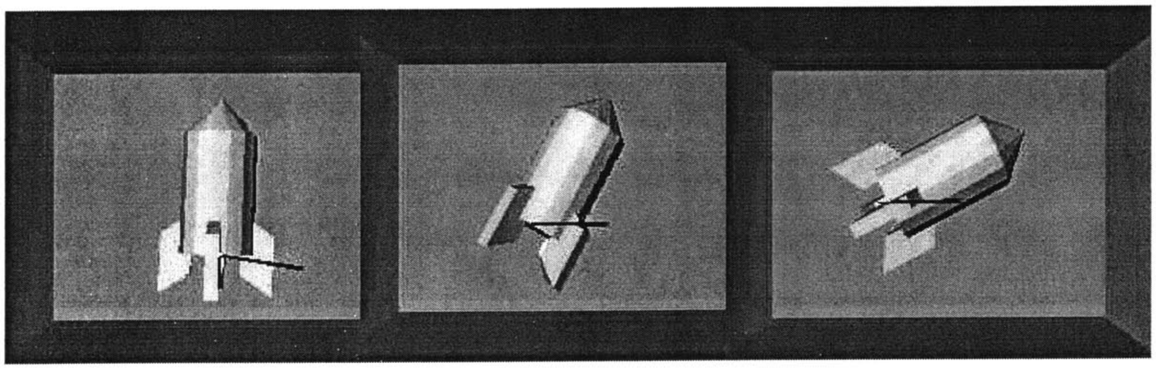


Figure 5-17. The flying concave rocket: The ray-based rendering technique has been used to manipulate a rocket with concave parts that have been sub-divided into multiple convex parts.

The ray-based rendering algorithm can be superior to the point-based techniques in many applications. If we need to explore an interior surface of an object where visual cues are limited or if the interaction torques are important in executing a task as in assembly planning and design, we can take advantage of the ray-based rendering. Moreover, the ray-based rendering can be considered as an intermediate stage in progress towards the full 6-dof haptic rendering. Since modeling of the haptic interactions between arbitrary shaped 3D objects is computationally too expensive (especially in interacting with dynamical or deformable objects with a 3D haptic probe), an intermediate step for rendering both forces and torques will be helpful. For example, the computational model of a 3D mechanical tool can be easily constructed from a few line segments to achieve faster haptic rendering rates (see Figure 5-18a). I have achieved real time update rates for rendering dynamical and deformable objects using the ray-based rendering which would not be possible if full 3D object-object interactions were considered (see Figure 5-18b).

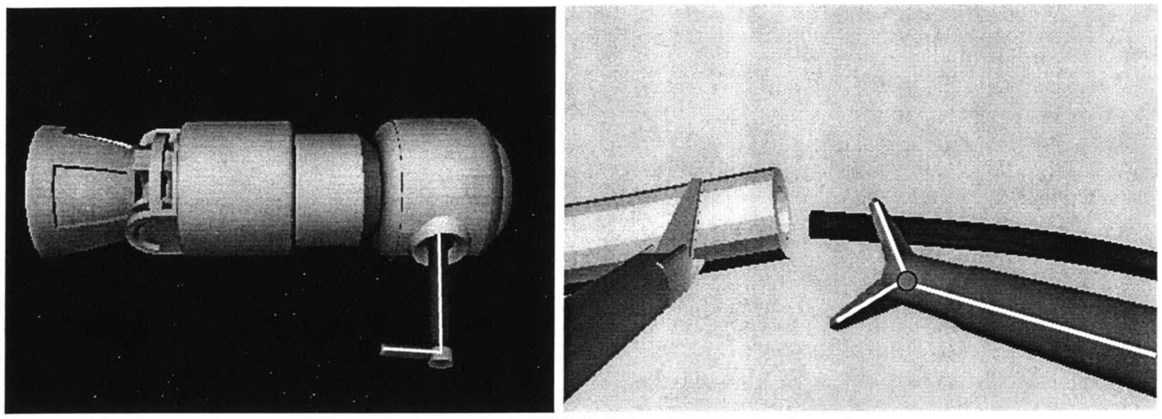


Figure 5-18. (a) Haptic Interactions between a mechanical tool and an engine block were simulated using the ray-based rendering technique. The engine block is made of convex objects and the mechanical tool is modeled as two line segments shown in white. (b) Haptic interactions between surgical instruments and flexible objects were simulated using the ray-based technique. The computational model of the surgical instrument seen in the figure consists of the three white line segments.

Chapter 6

Haptic Display of Object Properties

When we explore objects in the real world, we seldom contact smooth, frictionless, rigid surfaces. Therefore, in addition to the shape of objects, material property is another important factor that could increase naturalness of VEs. The material properties could be separated into three sub-areas: surface properties, dynamics, and compliance. In my simulations, surface properties are separated into two different categories: texture and friction. From a computational viewpoint, texture is the geometry that is too expensive to be represented as shape. Both friction and texture can be simulated by appropriate perturbations of the reflected forces. The major difference between the friction and the texture simulation via a haptic device is that the friction model creates only lateral forces and the texture model modifies both the lateral and normal forces.

The simulation of static and dynamic friction gives users the feeling as if they are stroking over the surface of sandpaper (Salisbury et al, 1995). By changing the mean value of friction coefficient and its variation, we can efficiently simulate various surfaces (Siira and Pai, 1996, Green and Salisbury, 1997). Textures can be simulated simply by mapping bumps and cavities onto the surface of objects. Minsky (1995) presented a method to simulate 2D haptic textures by perturbing the direction of reaction force. In contrast to the simulation of surface properties that add roughness to a smooth surface, *force shading* technique (Morgenbesser and Srinivasan, 1996, Basdogan, Ho, and Srinivasan, 1997, Ruspini, Kolarov, and Khatib, 1997) eliminates the surface roughness.

When the objects are represented as polyhedron, the surface normal and the force magnitude are usually discontinuous at the edges. By using force-shading technique, we can reduce the force discontinuities and make the edges of polyhedral object feel smoother.

Since an environment with moveable objects could always attract people's attention better than the one with only static objects, adding dynamic behavior to the VEs is another important factor in creating realistic VEs. Also, since many objects we encounter in our real life are compliant, the techniques that could render compliant virtual objects are also very important in improving the VEs. In this chapter, I first present the details of force-shading technique in section 6.1. The method to simulate frictions is presented in section 6.2. In section 6.3, I presented the techniques to add haptic textures to the virtual objects. The techniques to rendering dynamic and compliant objects are described in section 6.4 and 6.5, respectively. The conclusions are presented in section 6.6. It should be noted here that the techniques to render surface properties described in this chapter are for point contact only. They could be combined perfectly with the point-based algorithms. However, in the case with ray-based algorithm, they should be used only when the probe tip is contacting the objects. The techniques for rendering dynamic and compliant objects are for both point-based and ray-based algorithms.

6.1 Force Shading

During haptic rendering, we can compute the point of contact and retrieve information about the contacted primitive from the database using the haptic interaction paradigms mentioned earlier. The first step in calculating the shaded force is to find the shaded surface normal. To compute the shaded normal, we need to consider only the vertex normals of the original surface. If the contacted primitive is a vertex, the normal of the vertex is used directly as the normal of the collision point. If the contacted primitive is a line, the normal at the collision point is the average of the normal of the line's two neighboring vertices, weighted by the inverse of the distance from the collision point to the vertices. If the contacted primitive is a polygon, since our objects are made of triangles, the collision point will divide the collided triangle into three sub-triangles (see Figure 6-1). We then calculate the normal ($\bar{N}_s$) at the collision point by averaging the normals of the vertices ($\bar{N}_i$) of the contacted polygon, weighted by the areas A_i of the three sub-triangles (Equation 6-1).

$$\bar{N}_S = \frac{\sum_{i=1}^{3} A_i \cdot \bar{N}_i}{\sum_{i=1}^{3} A_i} \qquad (6\text{-}1)$$

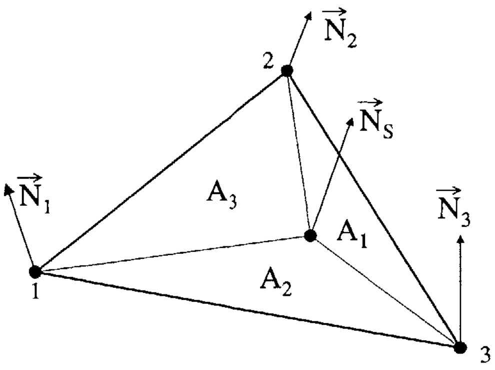


Figure 6-1. The collision point (IHIP) divides the collided triangle into three sub-triangles. The point in the center is the collision point. Points 1, 2, and 3 are the three vertices of the triangle. $\bar{N}_i$'s are the normals of the vertices. A_i's are the areas of sub-triangles. $\bar{N}_s$ is the normal at the collision point.

After the interpolation, the normal vector will become continuous across the whole surface. When the probe contacts the surface and strokes over the surface, the change of the surface normal along the stroking path will be continuous, too (see Figure 6-2). Once the shaded normal is available, we change the interacting force to be in the same direction as the shaded normal at the contact point (the magnitude of the force is still kept the same).

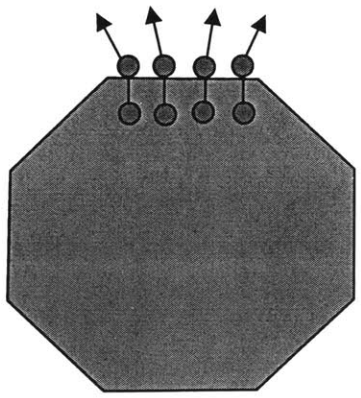


Figure 6-2. When the probe contacts the surface and strokes over the surface, the change of the surface normal along the stroking path will be continuous after the interpolation.

Although the interpolation of the vertex normals makes the normal vector $\vec{N}_s$ continuous at the edges, user can still feel the existence of the edges because of the discontinuities in the force magnitude (due to limited position tracking ability of the user, the depth of penetration is not held constant while the operator moves the probe of a haptic device from the surface of one polygon to another to explore the shape of a polyhedron). To minimize this problem, we project the HIP to the object surface along the direction of interpolated normal vector ($\vec{N}_S$). The distance between the projected HIP and the current HIP is then used to determine the magnitude of the force (see Figure 6-3). This method works well when the penetration of HIP is small compared to the size of the polygon. If the penetration depth is larger than the length of the edge of the contacted polygon, the users can still feel the existence of the edges. Force shading should be implemented appropriately depending on the geometry of the object that is being represented. It should be turned on to minimize the effects of artificial edges created by the polygonal representation. It should be turned off where the objects has natural edges (e.g. the force shading technique should be used to smooth the lateral surfaces of a circular cylinder made of polygons and not for the flat ends).

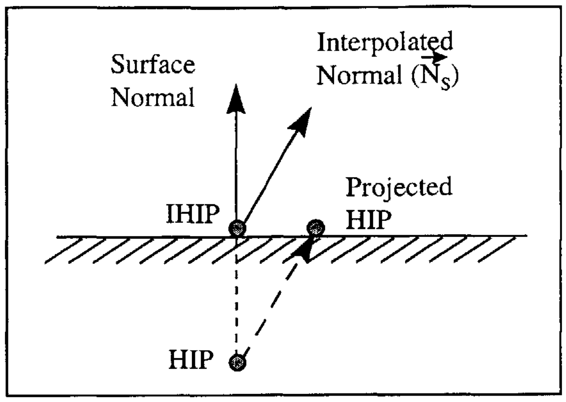


Figure 6-3. The direction and magnitude of force with force-shading technique. Before applying force shading, the vector from HIP to IHIP is used to determine the force. After applying the force shading technique, the vector from HIP to the projected HIP is used to compute the force.

6.2 Friction

As mentioned earlier, we can simulate friction by adding a lateral force vector to the normal force vector. We extended the friction model proposed by Salcudean and Vlaar (1994) and Salisbury et al. (1995) to simulate various types of bumpy frictional surfaces by changing the static and dynamic friction coefficients at different spatial locations and to improve the perception of object surfaces. We can simulate static and dynamic friction such that the user feels the stick-slip phenomenon when he/she strokes the stylus of a haptic device over the object surface.

In my model, there are two types of friction states: sticking and sliding. During the sticking and sliding states, the static and dynamic friction coefficients are used respectively. When the first collision is detected, the contacted point is stored as the sticking point and the friction state is set as the sticking state. When the users move the HIP, a tangential resistive force is applied to the user. The magnitude of the tangential force is decided by the linear elastic law ($F_s = k_s x$, where k_s is proportional to the static friction coefficient and x is the vector from the IHIP to the sticking point) (see Figure 6-4). If the tangential force is larger than the allowable static friction force (the normal force times the static friction coefficient), the friction state is changed to sliding state and the sticking point is updated and moved to a new location. The location of the new sticking point is on the line between the old sticking point and the IHIP. The distance from the IHIP to the new sticking point is calculated using the inverse of linear elastic law ($x = F_d / k_d$, where k_d is proportional to the dynamic friction coefficient, F_d is the friction force which is equal to the normal force times the dynamic friction coefficient). It should be noted that k_d is smaller than k_s, as is the case for real objects. Unlike the method described in Salisbury et al. (1995), we keep the state in the sliding state, instead of switching to the sticking state, following the movement of sticking point. The reason is that if we change back to sticking state right after the movement of sticking point, we observed that the feeling of dynamic friction is lost. In the next iteration, we calculate the tangential force ($F_d = k_d x$). If this force is larger than the dynamic friction force (the normal force times the dynamic friction coefficient), we know that the user intends to keep moving the HIP in the same direction and, therefore, we keep the state in the sliding state and use the method mentioned above to move the sticking point to a new position. However, if the force (F_d) is smaller than the dynamic friction force, meaning that the user has stopped moving in the same direction, we change the state to sticking state and do not update the position of the sticking point. The computation of frictional force is done continuously as long as the HIP is inside the object.

Uniform friction all over the object surface is created when the static and dynamic friction coefficients are constant and independent of the position. We can also create periodic frictional surfaces by varying the static and dynamic friction coefficients at different locations. More sophisticated surfaces can be simulated by changing the distribution of the friction coefficients. Green and Salisbury (1997) have shown that various grades of sandpaper can be simulated well by modifying the mean and variance of the friction coefficient.

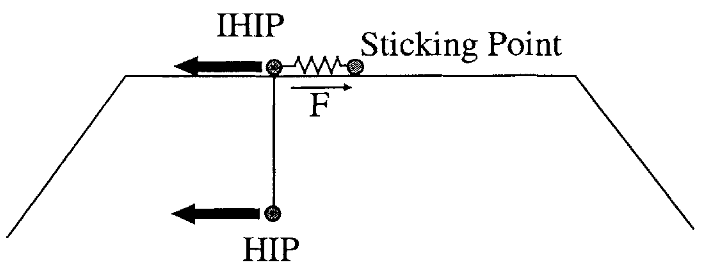


Figure 6-4. Description of the friction model. In addition to HIP and IHIP, a sticking point is created in this model. As shown in the figure, when the HIP and IHIP move towards left, the sticking point will cause a force to pull the IHIP to the right side.

6.3 Texture

Texturing techniques can be used to add roughness to a smooth surface. Similar to the role of textures in computer graphics, haptic textures can add complexity and realism to the existing geometry. Texturing techniques reduce the load on the geometry pipeline since they reduce the need for expressing texture geometry explicitly. In computer graphics, the final goal of the texturing computations is to decide the color in each pixel. Similarly, the final goal in haptic texturing is to decide the direction and magnitude of the force that will be reflected to the user. In order to simulate haptic textures, we need to know IHIP, HIP, and the height field that will be mapped to the surface. How to compute IHIP and HIP has already been described in Chapter 4, and how to map the height field over the object surfaces to simulate textures will be described in this section.

6.3.1 Magnitude of Force

When there is no texture on the surface, we can use the elastic law to decide the magnitude of the force based on the depth of penetration. When a height field is mapped over the surface to simulate haptic textures, the geometry of the surface will be changed, which, in turn, will change the depth of penetration. To correctly change the force, we simply need to add the texture height at IHIP to the depth of penetration. We choose the height value at IHIP instead of HIP because IHIP is the simulated contact point and is the point that stays on the surface of the object. If we use the height value at HIP, different penetration depths will have different height values, although the contact point is the same.

One other situation that has to be taken into account in texture display is the collision state. Adding a texture field over a surface may change the collision state from no collision to collision or vice versa when the probe is stroked over the textured surface (see Figure 6-5). There may be a collision between the HIP and the texture-mapped surface even when there exists no collision between the HIP and the original surface.

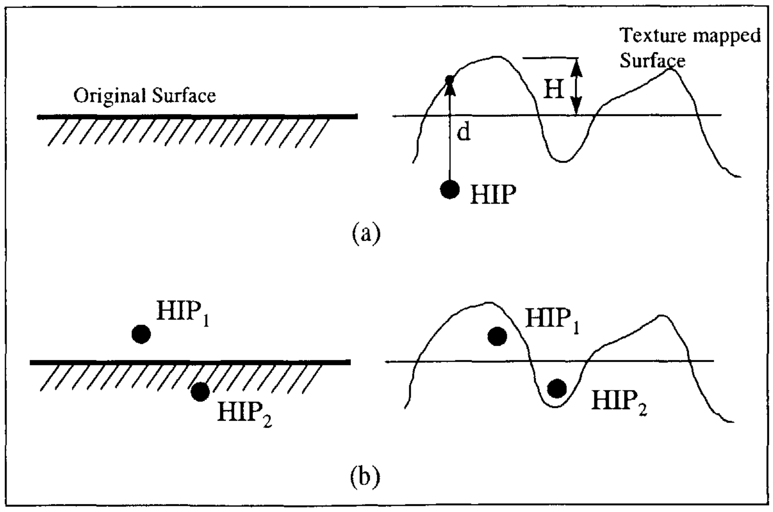


Figure 6-5. Collision situations may change after texture mapping. The original surface and the surface after texture mapping are shown in (a). In (b), there is no collision between HIP_1 and the original surface before the texture is mapped, but collision occurs after the texture mapping. The situation is the opposite for HIP_2.

To handle all such cases, we compute the "nearest distance" to HIP at each iteration using the following rules. If there is a collision with the original surface, the nearest point to the HIP is IHIP and the "nearest distance" is the distance between the IHIP and HIP. Since the HIP is below the contacted geometric primitive, we consider its value as negative. If there is no collision between the HIP and the primitive, the "nearest distance" is the distance between the HIP (the HIP is above the surface in this case) and the primitive and its value is positive. With the definition of "nearest distance" concept, we can easily check the collision status: the collision with textured surface actually happens only when the "nearest distance" is less than the height value at the nearest point on the original surface. Note that the height value could be positive or negative relative to the original object surface. We can define the depth of penetration to be the height value at the nearest point minus the "nearest distance". If this depth of penetration is negative, it is set to zero. The magnitude of the force is then calculated based on the depth of penetration and the mechanistic law that governs the interactions.

6.3.2 Direction of Force

To decide the direction of the force in the haptic display of texture, we modify the "bump mapping" technique of computer graphics. Bump mapping (Blinn, 1978) is a well known graphical technique for generating the appearance of a non-smooth surface by perturbing the surface normals. In haptics, if we perturb the direction of the force, we can also generate a similar effect that makes the users feel that there are bumps on the smooth surface. The first step in deciding the direction of force is to determine the direction of the surface normal. For graphics, Max and Becker (1994) improved the original bump mapping technique and suggested a direct method of mapping that does not require a transformation from global to parametric space. They developed a formulation that is purely based on the original surface normal and the local gradient of the height field that generates bumps on the surface of the object. Max and Becker utilized this technique to generate bumps to simulate clouds graphically. We used the same approach to calculate the perturbed surface normals ($\vec{M}$).

$$\vec{M} = \vec{N}_s - \nabla h + (\nabla h.\vec{N}_s)\vec{N}_s \qquad (6\text{-}2)$$

$$\nabla h = \frac{\partial h}{\partial x}\hat{i} + \frac{\partial h}{\partial y}\hat{j} + \frac{\partial h}{\partial z}\hat{k} \qquad (6\text{-}3)$$

where, $\vec{M}$ represents the normal of the surface after texture mapping, h(x,y,z) represents the height (texture) field function and ∇h is the local gradient vector, $\vec{N}_s$ represents the unperturbed surface normal at the collision point.

The most intuitive way of deciding the direction of the force is to use the perturbed surface normal ($\vec{M}$) as the force direction. Indeed, our experience shows that this can give the users very good feeling of texture for most of the cases. However, if the amplitude and spatial frequency of the texture are very high and the force applied by the user to explore the surface is very large, the haptic device will become unstable due to sudden changes in the force magnitude and direction when this algorithm is used. The reason for the instability is that the magnitude and direction of the force change abruptly with only a small change in position. To eliminate the force instability, we modify the approach slightly and use the normal of the original surface along with the perturbed surface normal to calculate the direction.

$$\vec{F} = (d - Kh)\vec{N}_s + Kh\vec{M} \qquad \text{if } d \geq \text{Kh} \qquad (6\text{-}4a)$$

$$\vec{F} = d\vec{M} \qquad \text{if } d < \text{Kh} \qquad (6\text{-}4b)$$

where, $\vec{F}$ represents the force that will be displayed to the users, d represents the magnitude of the penetration, $\vec{N}_s$ represents the normal of the original surface, $\vec{M}$

represents the perturbed surface normal, K is a scalar that depends on the properties of the texture, h is the height of the texture. From the equations (Equation 6-4a and 6-4b), we observe that the force will be in the same direction as the perturbed surface normal $\bar{M}$ when the penetration is small (i.e., the magnitude of the force is small) compared to the height of the texture. This is same as bump mapping approach. However, if the force is large, only a small amount (the Kh term in Equation 6-4a) of the force is perturbed and the remaining amount will still be along the original surface normal. This modification will make the interactions with fine textures more stable (In our experience with PHANToM force feedback device, the simulation gives the best results when the value of K is between 1 and 2. If the simulated texture is very smooth, K should be set to 2. If the changes in texture gradient are sharp, then K should be set to 1 or even smaller).

6.3.3 Height Field of Textures

In order to apply the proposed texturing techniques, textures must be C^0 and C^1 continuous (Foley et al., 1995). The simulation has the best effect if the height and the wavelength (i.e., the inverse of the spatial frequency) of the texture are of the same order.

Taking these constraints into account, we have ported several texturing techniques of computer graphics to simulate haptic textures. The haptic texturing techniques can be classified into two parts: (a) image-based and (b) procedural.

Image-based haptic texturing: This class of haptic texturing constructs a texture field from a 2D image data (see Figure 6-6). In computer graphics, the digital images are wrapped around 3D objects to make them look more realistic. The graphical texture map consists of texels with only 2D color or gray scale intensities, whereas the haptic texture map consists of texels with a height value (Basdogan, Ho, and Srinivasan, 1997).

The first step to create image-based haptic texture is to map the digitized image to the 3D polygonal object. We use the two-stage texture mapping techniques of computer graphics (Bier and Sloan, 1986; Watt and Watt, 1992) to map the 2D image to the surface of 3D objects. The first stage is to map the 2D image to a simple intermediate surface such as plane, cube, cylinder, or sphere. The second stage maps the texture from the intermediate surface to the object surface. After this two-stage mapping, we can obtain the height value for any point on the object surface (see Basdogan, Ho, and Srinivasan, 1997 for implementation details).

After the mapping, the only information we need to create the haptic texture is the gradient of the height field at the IHIP. The gradient of the height could be computed using the central difference approximation for partial derivatives:

$$\frac{\partial h}{\partial x} = \frac{(h_{x+\varepsilon} - h_{x-\varepsilon})}{2\varepsilon}$$

$$\frac{\partial h}{\partial y} = \frac{(h_{y+\varepsilon} - h_{y-\varepsilon})}{2\varepsilon} \qquad (6\text{-}5)$$

$$\frac{\partial h}{\partial z} = \frac{(h_{z+\varepsilon} - h_{z-\varepsilon})}{2\varepsilon}$$

$$\nabla h = \frac{\partial h}{\partial x}\hat{i} + \frac{\partial h}{\partial y}\hat{j} + \frac{\partial h}{\partial z}\hat{k} \qquad (6\text{-}6)$$

where (x, y, z) represents the coordinate of the collision point and ε is a small parameter. To compute the gradient at the IHIP, we estimate the height values that are ε distance away from the IHIP along the coordinate axes. Texture heights can be estimated at these points using the two-stage mapping technique (Basdogan, Ho, and Srinivasan, 1997). For example, $h_{x+\varepsilon}$ represents the estimated height value at the point (x+ε, y, z). Once the local gradient is known, it can be used to perturb the surface normal at the collision point for simulating image-based textures. Note that all the texture values indicated here are filtered values because of the need for C^0 and C^1 continuity.

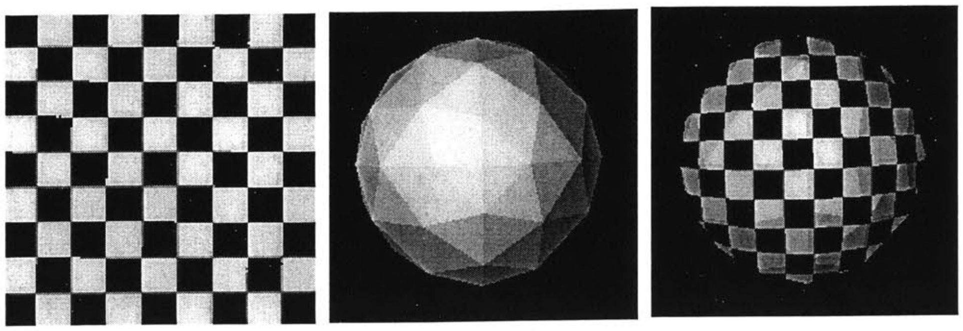


Figure 6-6. An example of image-based texture mapping. (a) The original image that is going to be mapped to the object. (b) The original object without texture mapping. (c) The object after the texture mapping.

(b) *Procedural haptic texturing*: The goal of procedural haptic texturing is to generate synthetic textures using mathematical functions(see Figure 6-7). Generally speaking, it is much more straightforward to obtain the height value and the gradient of height in this approach. The function usually takes the coordinate (x,y,z) as the input and returns the height value and its gradient as the outputs. For example, we have implemented the well-known noise texture (Perlin, 1985; Ebert et al. 1994) to generate stochastic haptic textures. Fractals are also suggested for modeling natural textures since many natural objects seem to exhibit self-similarity (Mandelbrot, 1982). We have used the fractal concept in combination with the texturing functions mentioned above (e.g. Fourier series, noise) with different frequency and amplitude scales (Ebert et al. 1994) to generate more sophisticated surface details. Other texturing techniques suggested in computer graphics can also be ported to generate haptic textures. For example, we have implemented haptic versions of reaction-diffusion textures (Turk, 1991, Witkin and Kass, 1991), the spot noise (Wijk, 1991), and cellular texture (Worley, 1996, Fleischer et al., 1995).

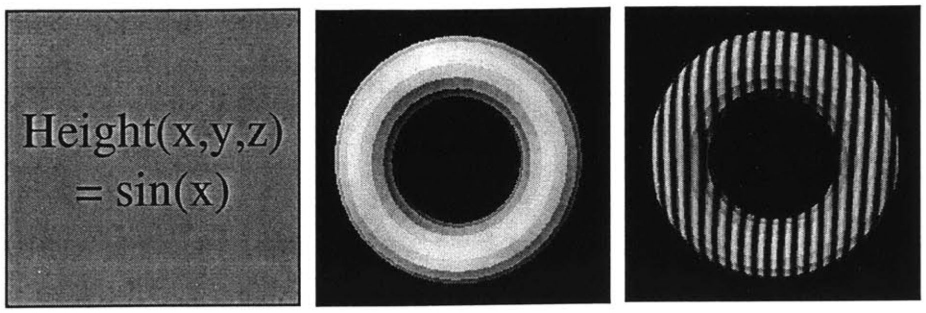


Figure 6-7. An example of procedure-based texture mapping. (a) The height field function of the texture. (b) The original object without texture mapping. (c) The object after the texture mapping.

6.4 Dynamics

Adding dynamic behavior to virtual objects could add dramatic fun and usefulness to the VEs. To simulate dynamic behavior, the first step is to attach a transformation matrix to each object. The transformation matrix contains the translation and rotation information of the object that it is attached to. When an object moves to a new location, we change only the transformation matrix, instead of changing the coordinates of all vertices of the object.

When checking collision between the object and the probe, we first use the inverse of the transformation matrix of the object to transform the probe from world coordinate to the object's local coordinate. We, then, check the collision in the local coordinate. Once the contact point and the interaction force are computed, we use the transformation matrix to transform them back to the world coordinate. In this way, we can save a lot of computational time in updating the object's coordinates.

The behavior of objects is governed by the equations of the motion. When an object is contacted by the probe or by other objects, we compute the contact point and the contact force in world coordinate. Once the contact point and the contact force are available, we can use them and object properties (such as the mass, inertia matrix, and mass center) to form the equations of the motion. The equations of the motion is then solved using the Euler integration technique and the location of the object is updated every iteration. In this way, the VEs could allow the user to manipulate the objects and change their position or orientation dynamically via the end-effector of the haptic device. Figure 6-8 is an example of simulation of dynamic object using point probe. Figure 5-15 and Figure 5-17 are examples of simulation of dynamic objects using ray probe.

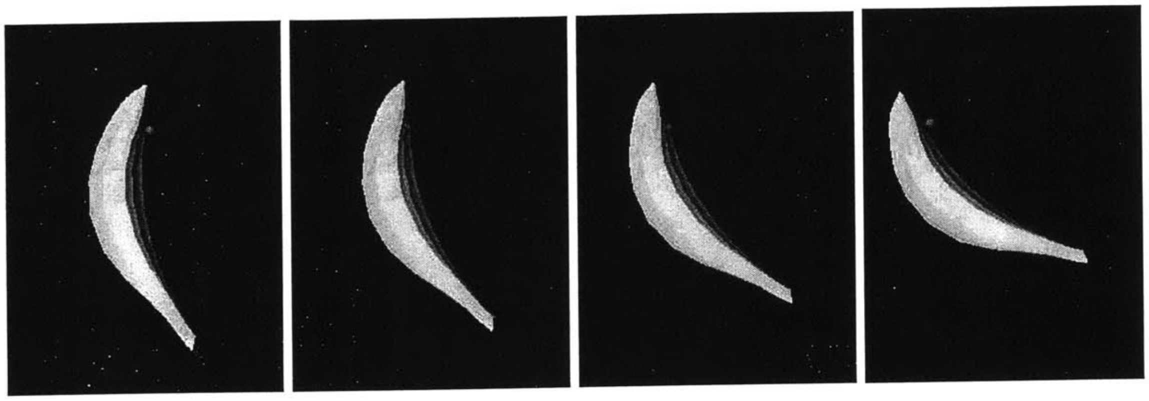


Figure 6-8. Haptic rendering of a dynamic object. The small dark dot represents the point probe. The user moves the probe from the right side to contact the banana. After the contact, the banana changes its position and orientation.

6.5 Compliance

It is very difficult to simulate physical-based deformable objects in haptic VEs due to the requirement of fast update rate. The finite element method (FEM) is a popular and available way to compute the physical-based behavior of a deformable object. However, it is computational too expensive for haptic rendering at this stage. The computation of the deformable behavior of an object is in the research area of material modeling. In this section, I am not trying to present another method of material modeling. Instead, I present the software structure that is good for simulating deformable objects in haptic VEs.

In section 3-2, I have presented software structures to simulate rigid objects for haptic VEs. Since a huge amount of data needs to be changed when simulating deformable objects, the two-threading technique is more appropriate than the two-processing technique when simulate deformable objects. The new software structure for simulating deformable object is shown in Figure 6-9. Compared to the one described in section 3-2, the modeling calculation is the new component in the software structure. It should be noted that the addition of the modeling computation would not affect the haptic rendering or the graphic rendering. The modeling computation is a separated loop. It could use different methods in calculating the new shape of the objects, as long as those methods are fast enough. If the objects are rigid, we could delete the modeling calculation loop and the software structure will change back to the one in section 3-2.

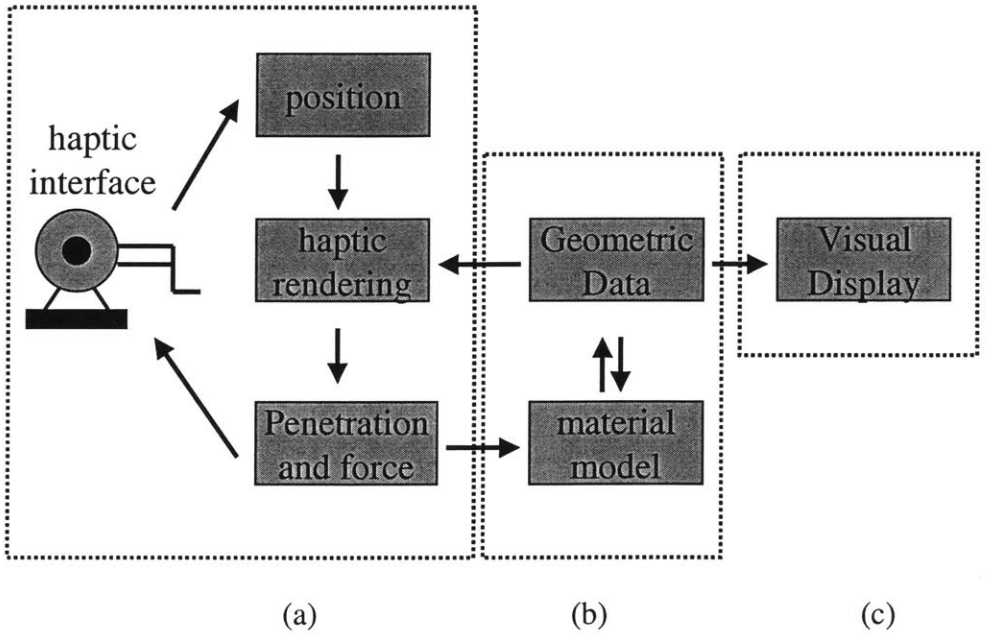


Figure 6-9. The software structure for simulating deformable objects in haptic VEs. (a) haptic rendering, (b) modeling computation, and (c) graphic rendering.

One of the easy ways to change the shape of an object is to deform it locally. Deforming an object locally is not a physical-based method. However, the advantage is that it is fast in computation. Also, since the human kinesthetic resolution is not very good, the users usually could not differentiate the difference between the real one and the fake one. Therefore, the local deformation is usually good enough and could convince the users that they are deforming an object.

To deform an object locally, the first step is to get the contact point and the penetration depth. This information is usually available from the results of haptic interaction paradigms (see chapter 4 and chapter 5). Once the contact point is available, the vertices of the contacted object that are in the close vicinity of the contact point are moved along the direction of penetration vector. A simple polynomial function could be used in deciding how much the vertex is going to be moved. The picture in Figure 6-10 shows an example of modeling computation using local deformation. In this example, I defined an affecting radius *R***.** The affecting radius R defines the size of the area that is going to be deformed. When the probe contacts the object, we can get the contact point *Pt* and the penetration vector $\vec{P}$ from the haptic interaction paradigms. Then, we compute the distance *Di* between each vertex *Vi* of the object to the contact point *Pt*. If the distance *Di* between the vertex *Vi* and the contact point *Pt* is less than *R*, the vertex *Vi* is going to be moved and the movement will be $(\frac{R - Di}{R})\vec{P}$. In this way, we can create a smooth deformation. To reduce the computation, we can create a neighbor list for each vertex during the pre-processing phase. The neighbor list of a vertex will contain all the vertices whose distance to the vertex is less than *R*. In this way, we can reduce the computational time in finding the vertices that is going to be deformed. More methods on material modeling are described in section 7.7.

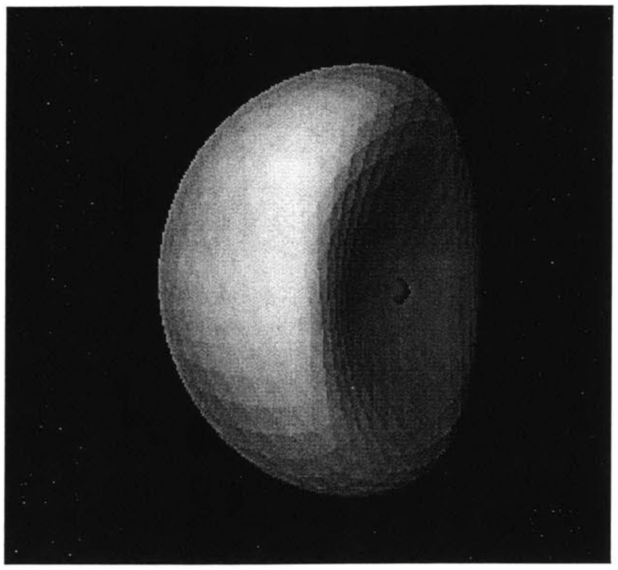


Figure 6-10. Haptic rendering of a deformable object. The small dark dot in the figure represents the probe. The original shape of the object is like a sphere. When the probe penetrates the surface of the object, the vertices of the object will be moved along the direction of penetration.

6.6 Conclusions

In this chapter, I have presented a couple of *object property display algorithms* to rendering material properties. These techniques all work with the *haptic interaction paradigms* described in chapter 4 and chapter 5. By combining the *haptic interaction paradigms* and these *object property display algorithms*, we can create versatile realistic 3D objects with various material properties. Different *object property display algorithms* could also be combined. For example, the friction, texture, and dynamic rendering algorithms could work together to create a moveable, frictional, and textured object. It should be noted that the force shading and the texturing algorithms should not work together since they are both used to change the shape of the objects. Any other combination of *object property display algorithms* will work well with each other.

Chapter 7

Application-Simulation of a Medical Process

7.1 Introduction

Human hand is a very versatile organ. People rely very much on their hands to explore and manipulate objects in the real world. Through haptic sensations, human beings are able to obtain information and then control the environment. In many situations, the interactions are very simple and it is easy to complete the tasks. However, there are many circumstances, for instance, navigating an airplane or dissecting human tissues during surgery, in which haptic sensations are critical for success. In such cases, where failure can be expensive or life-threatening, training is essential. The advantages of training in haptic virtual environments include: (1) it offers a consistent environment, (2) it offers unlimited opportunity to practice, and (3) it is cost effective. They are some of the reasons why the development of haptic VEs has become a popular research topic in recent years.

From chapter 3 through chapter 6, I have presented different techniques for haptic interaction and display of object properties in VEs. I have combined these techniques to build an initial demonstration of a surgical simulator, which when fully developed could help laparoscopic surgeons to practice specific medical procedures.

In the following section, an introduction to laparoscopic surgery is given. In section 7.3 the reasons why laparoscopic surgeons are in need of a trainer are discussed. In section

7.4, the specific procedure to be simulated is presented. The hardware setup of the simulator is presented in section 7.5. The software structure and how I implement the environment are described in section 7.6. The modeling methods for different deformable objects are presented in section7.7. The results are presented in section 7.8.

7.2 Laparoscopic Surgery

Laparoscopy has been used in a range of medical procedures since the early sixties. However, it was not applicable in surgery until the development of the clip appliers, which is required during a surgical procedure to clip off blood vessels. Major advantages of laparoscopic surgery to the patient are short hospital stay, short recovering time, less pain, and less scarring after the surgery. Moreover, it is cost effective. Because of the benefits associated with laparoscopic surgery, surgeons and their patients prefer it to the open surgery. Some of the most common laparoscopic procedures include cholecystectomies, appendectomies, and hernia repair. It is expected that more laparoscopic techniques will be developed in the near future as the instruments get smaller and more dexterous. It is a general trend that more surgical procedures will be performed laparoscopically. Researchers estimated that 60 to 80 percent of abdominal surgeries will be performed laparoscopically by the year 2000.

To perform a typical laparoscopic surgery, the patient is first placed under general anesthesia. The surgeon then uses narrow, tube like instruments called trocars to create a couple of holes on the abdomen. These trocars allow miniature surgical instruments at the end of long tools to access to the abdominal cavity. After that, A miniature camera is inserted through the trocar. The camera projects a clear magnified image of the patient's internal organs on a video monitor, which provides the surgeon a view of the insides of the abdomen as he proceeds with the surgery. Carbon dioxide gas is used to inflate the abdomen to help the surgeon see better and to expand the workspace. During the surgery, several miniature surgical instruments such as a scissors, clipping tool, and grasper are inserted through the trocars to perform the necessary haptic actions on the organs (see Figure 7-1 and Figure 7-2).

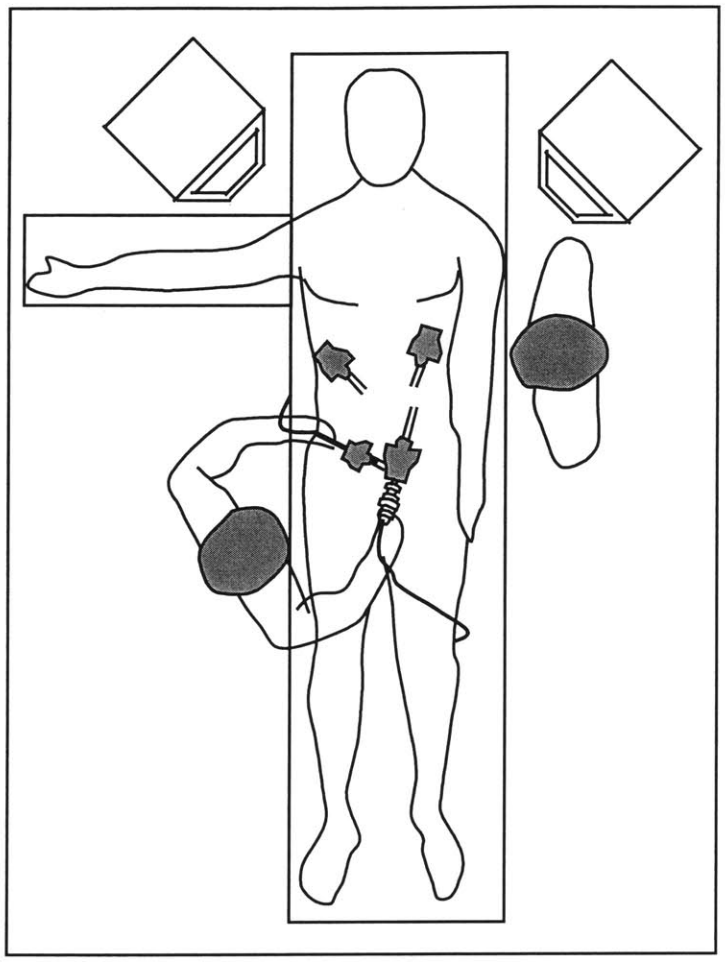


Figure 7-1. The setting of the operating room for a typical laparoscopic surgery. A couple of instruments are inserted through trocars to access to the internal organs. The screen of the monitor shows the view of the inside of the abdomen.

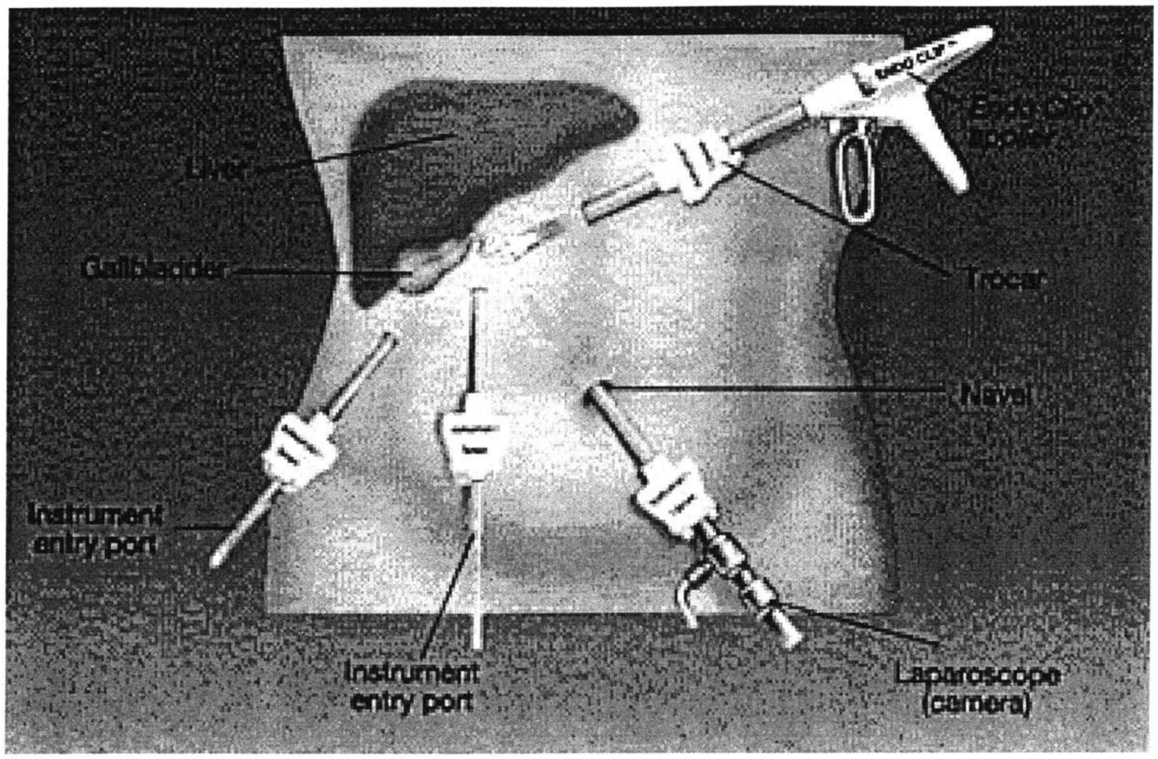


Figure 7-2. An example setup of the trocars and camera for a laparoscopic surgery.

Cholecystectomy is one of the most commonly performed laparoscopic procedures in the United States. The cholecystectomy is the surgical removal of the gallbladder. The surgery is done to get rid of problems associated with gallbladder diseases, such as gallstones and infections in the gallbladder. The gallbladder is a pear-shaped sac found on the liver (see Figure 7-3 and Figure 7-4). It is about 8cm long and its purpose is to store bile that is released by the liver. The bile helps in digesting fat. Gallstones are made out of cholesterol, bile salts, calcium and bilirubin. Gallstones usually stay in our gallbladder without causing problems. However, if they block the outlet of the gallbladder, it would cause pain in the abdomen. Occasionally, these stones can come out of the gallbladder, causing jaundice or inflammation of the pancreas. Sometimes very small stones grow in the bile ducts, which are the tubes that take bile to the intestines. During a cholecystectomy, the surgeon inspects the bile ducts and may x-ray them or examine them with a special instrument in search of gallstones. If stones are found in the common bile duct, they will be removed. In the past, patients were hospitalized for about seven days after having undergone such a procedure and usually spent another six weeks recovering at home. The laparoscopic technique makes it possible for most patients to go home the day after surgery and resume normal activities within a very short time. Gallstones and gallbladder disease affect approximately one out of ten people in the United States. It is estimated that more than 90% of gallbladder patients can be treated successfully by the laparoscopic method.

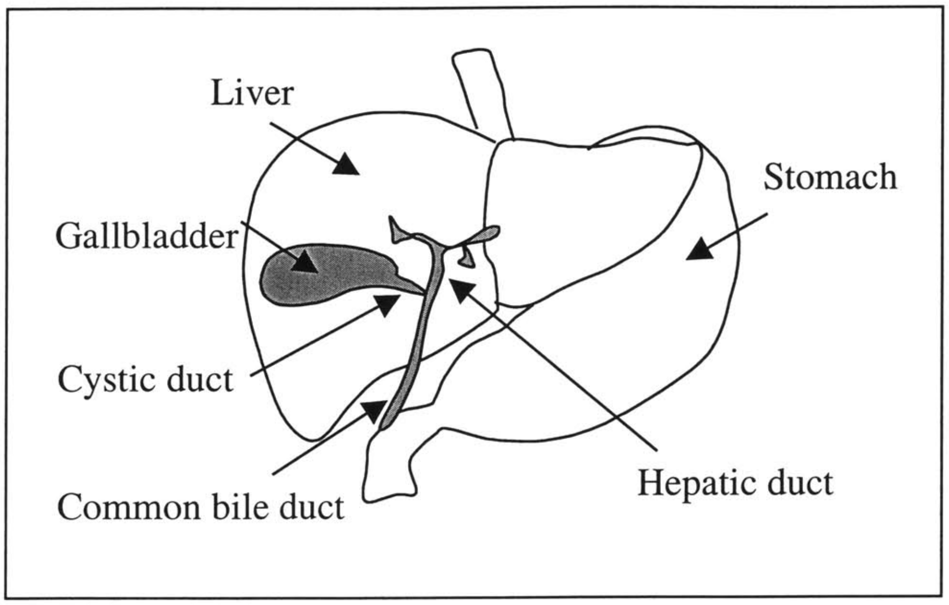


Figure 7-3. Anatomy of the gallbladder. The gallbladder is a pear-shaped sac found on the liver. It is about 8cm long and its purpose is to store bile that is released by the liver. The bile duct takes the bile to the intestines to help in digesting fat.

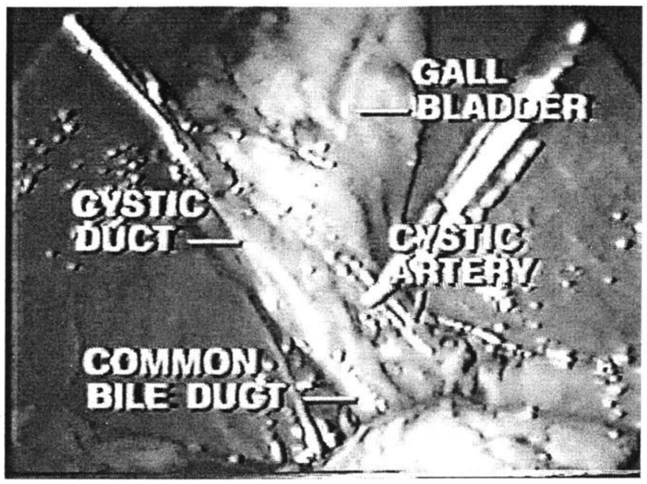


Figure 7-4. An image of a cholecystectomy. The surgery is done to get rid of problems associated with gallbladder disease, such as gallstones and infections in the gallbladder.

Removing the gallstones from the bile duct is an important procedure in cholecystectomy. Multiple steps are followed to clean up the small stones. First, the cystic duct is dissected and a clip is placed at the junction of the infundibulum of the gallbladder with the cystic duct. A needle is inserted directly through the abdominal wall such that the catheter will naturally point towards the operative site. A catheter attached to a syringe of saline is flushed to remove bubbles, and inserted through the needle. With an instrument behind the cystic duct to stabilize it, microscissors is used to incise the cystic duct. The catheter is then carefully inserted into the duct (see Figure 7-5). A special instrument can then be inserted through the catheter to go inside the bile duct to clean gallstones.

The major risks associated with the laparoscopic cholecystectomy during the operation are injury to abdominal organs from the laparoscopic insertion and injury to the bile duct that could require major reconstructive procedures. Other possible risks include bleeding, clipping impingement, leakage of bile from the bile ducts into the abdomen, and misidentification of common bile duct as the cystic duct.

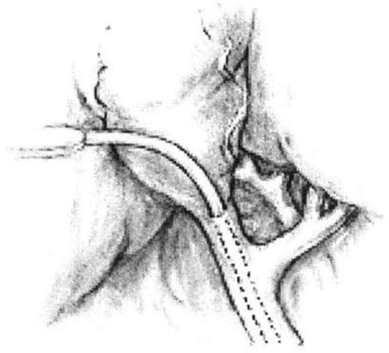


Figure 7-5. A catheter is inserted into the cystic duct for common bile duct exploration.

7.3 The Need for a Training System

As described above, in laparoscopic surgeries, surgeon uses a small video camera and a few customized instruments to perform surgery. The camera and instruments are inserted into the abdomen or chest through small skin incisions that enable the surgeon to explore the internal cavity without the need of making large openings. However, the surgeons are handicapped by limitations of the current laparoscopic technology in comparison to traditional open surgery. The laparoscopic surgeons face four main types of problems. First, the visualization of the internal organs can be achieved only by means of a camera places inside the body. Therefore, the vision is monoscopic and is limited by the field of view of laparoscope. Second, hand-eye coordination is a problem since the monitor reflects the mirror images of the actual hand movements and the anatomical landmarks. Third, the tactile sensing and force-feedback cues to the surgeons are substantially reduced since the instruments that interact with internal organs have to go through a long thin tube. Fourth, the movement of the instruments is constrained since they have to pass through the trocars, which can only rotate about a fixed point. The instrument cannot have a translation perpendicular to the long axis of the trocar.

While learning laparoscopic procedures, the trainees watch senior surgeons performing them first. Then, they start by performing simple tasks under the guidance of senior surgeons. It usually takes several months for the trainees to learn a procedure. Therefore it is necessary to find new training approaches or develop new devices to reduce the training time and reduce the risks related to the training process.

7.4 Selection of a Laparoscopic Procedure for Simulation

An important component of a cholecystectomy is the proper insertion of a flexible catheter (about 2-3 mm in diameter) into the cystic duct of (about 8-10 mm in diameter). Due to the limited visual and haptic cues and the small size of the catheter, this is not an easy task to perform for a junior level surgical resident. If a catheter is not inserted properly, it may cause the bile to come out or even damage the bile duct. To learn this procedure, the surgical residents usually practice on a simple training system first. The training system includes a laparoscopic training box, two laparoscopic tools, a camera, and several other objects such as a pen, needle, tube, and wire (see Figure 7-6). For example, they might practice inserting a wire into a tube guided by the view from the camera (see Figure 7-7).

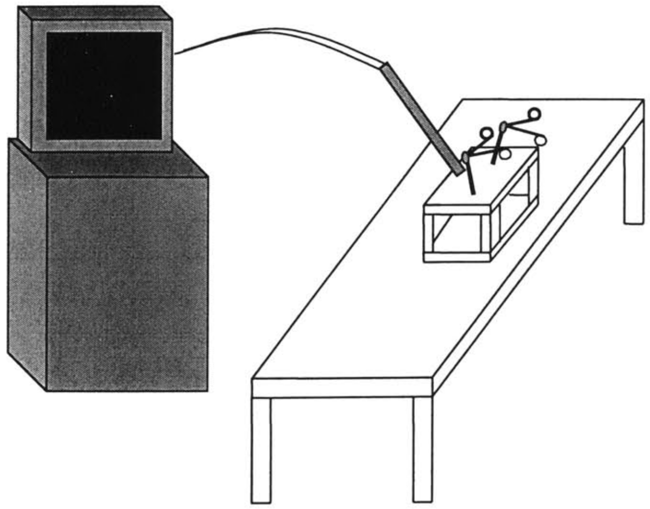


Figure 7-6. A system to practicing the skill for inserting a catheter into the bile duct. The training system includes a laparoscopic training box, two laparoscopic tools, and a camera. Inside the box, objects such as a pen, needle, tube, and wire are used for practicing manual and hand-eye coordination skills.

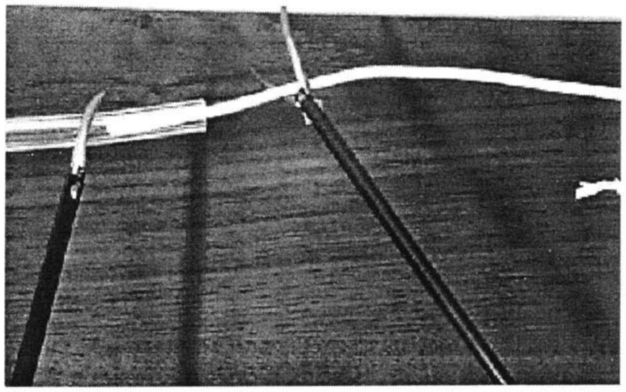


Figure 7-7. Using the training system shown in Figure 7-6, surgeons can practice inserting a wire into the tube guided by the view from the camera.

The purpose of our simulator is to improve this training stage by providing the user with more realistic environment. We selected the simulation of catheter insertion into the cystic duct based on a couple of reasons. First, the cholecystectomy is one of the most commonly performed laparoscopic surgeries in the United States and the insertion of catheter is a very important component of a cholecystectomy. Second, the current system in training catheter insertion is not realistic and can be improved by using haptic VEs. Third, it is possible to develop such a training system using haptic VEs with the techniques available today. The simulator could serve to verify the usefulness of the algorithms developed in this research. From the technical point of view, the development of such a part task simulator is also quite challenging since it needs to simulate the interaction between two deformable objects in VEs with both visual and haptic feedback.

Training the catheter insertion in haptic virtual environments has many advantages. First, the simulator could help in improving the hand-eye coordination. Second, we can have an objective measure of the performance. The computer could record all of the activities during the training process and the recorded data could be used to measure the performance. Third, we can easily change the material properties or change the objects in the VEs to simulate different tasks. We can also record and measure the forces applied by surgeons to help them understand whether the force they applied is too large or too small. The simulated tasks can be repeated many times and it is cost effective. And, the simulator could be continuously improved to create more realistic simulations.

7.5 Hardware Setup

In order to provide force-feedback interaction between the medical instruments and the virtual organs, we chose PHANToMs again as the haptic interfaces. Since two surgical tools are required in performing the catheter insertion, we used two PHANToMs in the simulator to provide force feedback for two instruments. One component task of the catheter insertion is to use gripper to grip the catheter and the cystic duct. In order to provide the force feedback for this gripping procedure, we attached an actuated laparoscopic tool (see Figure 7-8, developed by Ben-Ur, 1999) to the end of the PHANToM (see Figure 7-9 and Figure 7-10).

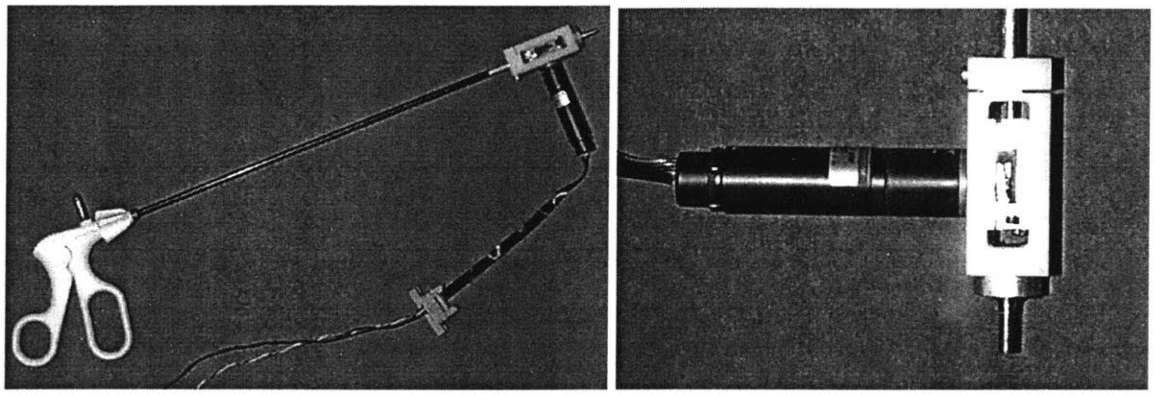


Figure 7-8. An actuated laparoscopic tool developed by Ela Ben-Ur, 1999. An encoder-actuator couple was attached to the distal end of a laparoscopic instrument.

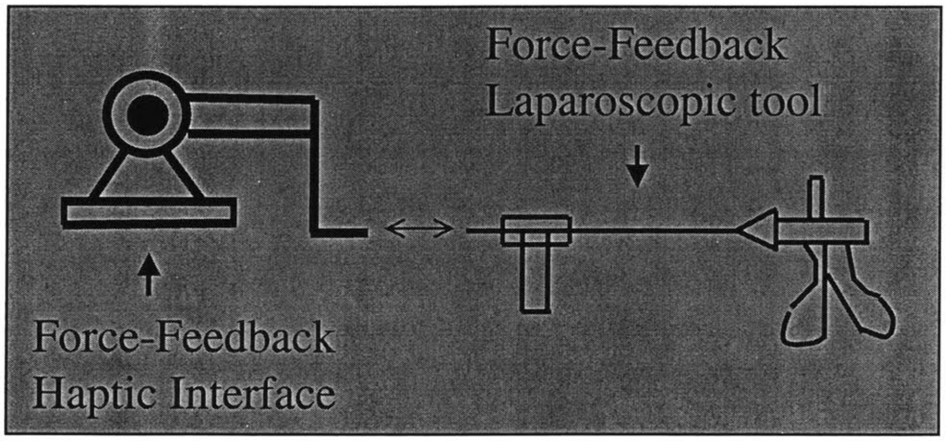


Figure 7-9. When connecting a force-feedback haptic interface and a force-feedback laparoscopic tool together, we can provide both gripping and interacting forces.

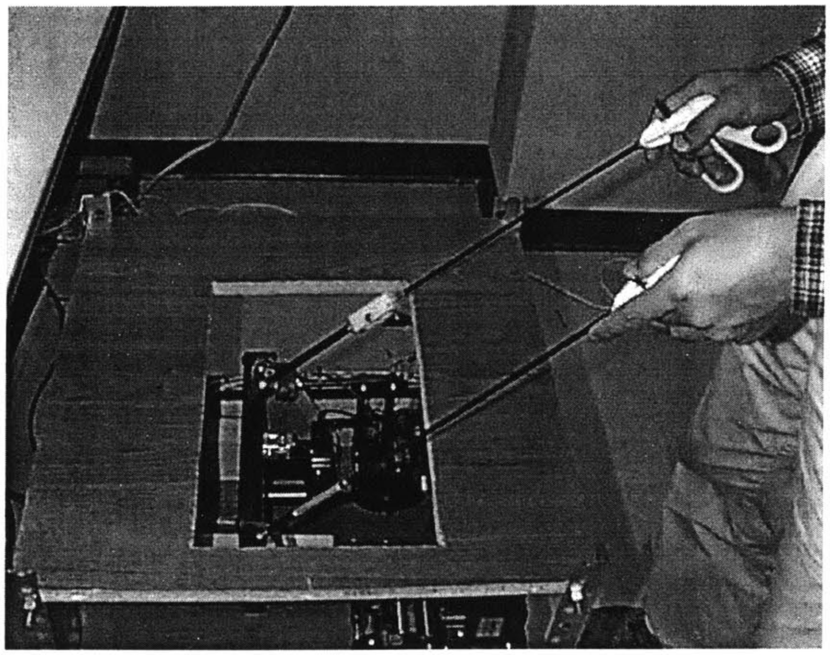


Figure 7-10. A force-feedback surgical simulator. Our simulator was designed to simulate laparoscopic procedures and to train medical residents and students. Two laparoscopic tools were attached to the end of two PHANToMs to provide both gripping and interacting forces.

In order to provide an environment similar to the one in real laparoscopic surgery, we put a torso (see Figure 7-11) on top of the simulator. The torso is produced by Simulab Corporation. By making a hole at the back of the torso and attaching it to the simulator, we can make the simulator very similar to what the surgeons will see in a real laparoscopic surgery (see Figure 7-12). With this training system, as the user manipulates the actual instruments and interacts with the virtual objects, the associated deformations of the organs are displayed on the computer monitor and the reaction forces are fed back to the user through the actuators.

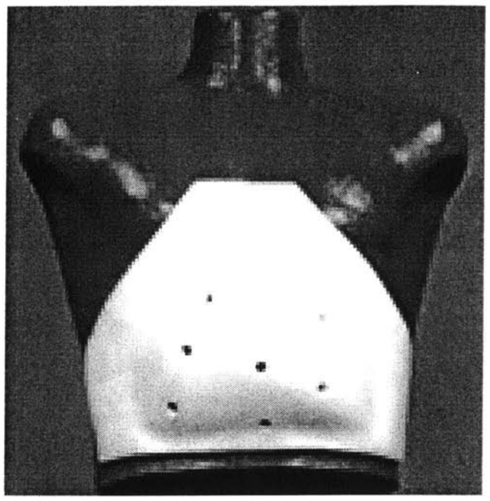


Figure 7-11. A torso by Simulab Corporation.

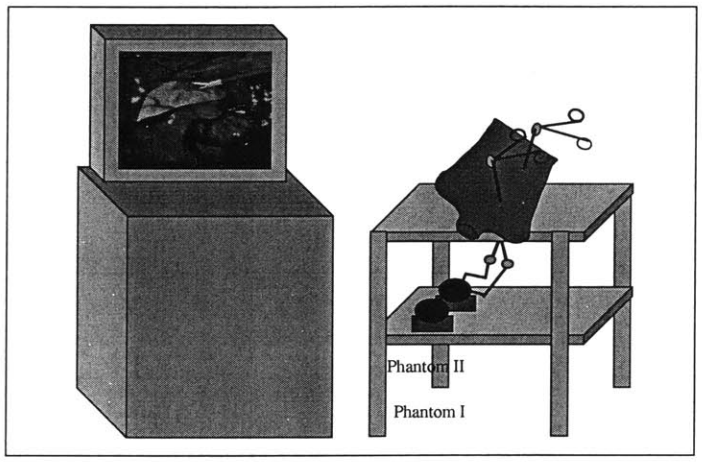


Figure 7-12. A surgical simulator for practicing laparoscopic procedures.

7.6 Software Setup

The software structure that was used to simulate the surgical procedure is similar to the one mentioned in section 6.5. Since there are two haptic interfaces and two force-feedback grippers, the servo loop for haptic rendering has to get position information from and send force commands to these four devices, instead of just one haptic interface shown in section 6.5. Since the computation of material modeling is very expensive, we separate it from haptic rendering. With the separation, there will be three different update rates during the simulation. The haptic rendering is updated fastest (around 500 Hz) and the modeling computation is updated slowest (a couple of Hertz). The graphic rendering rate is around 10 ~ 20 Hz.

7.6.1 Geometric Data on Human Organs

Visible Human Data set including graphical data in the forms of CT and MRI cross sectional images as well as slice photographs of actual body are available through the National Library of Medicine. Nowadays, 3D geometrical models of various body parts and organs generated from image slices are available in the market through many companies. To make the environment more natural, we add a couple of organ models (including liver, stomach, intestine, and fat) to our simulation. In order to use the rendering algorithms developed in this research, the organs are all represented in polyhedral format. A simple code was developed to generate and display the catheter. Texture maps were used to make those geometrical models appear more realistic. The image in Figure 7-13a shows the geometry and location of the organs we used in our simulation. The image in Figure 7-13b shows the wire frame representation of those organs.

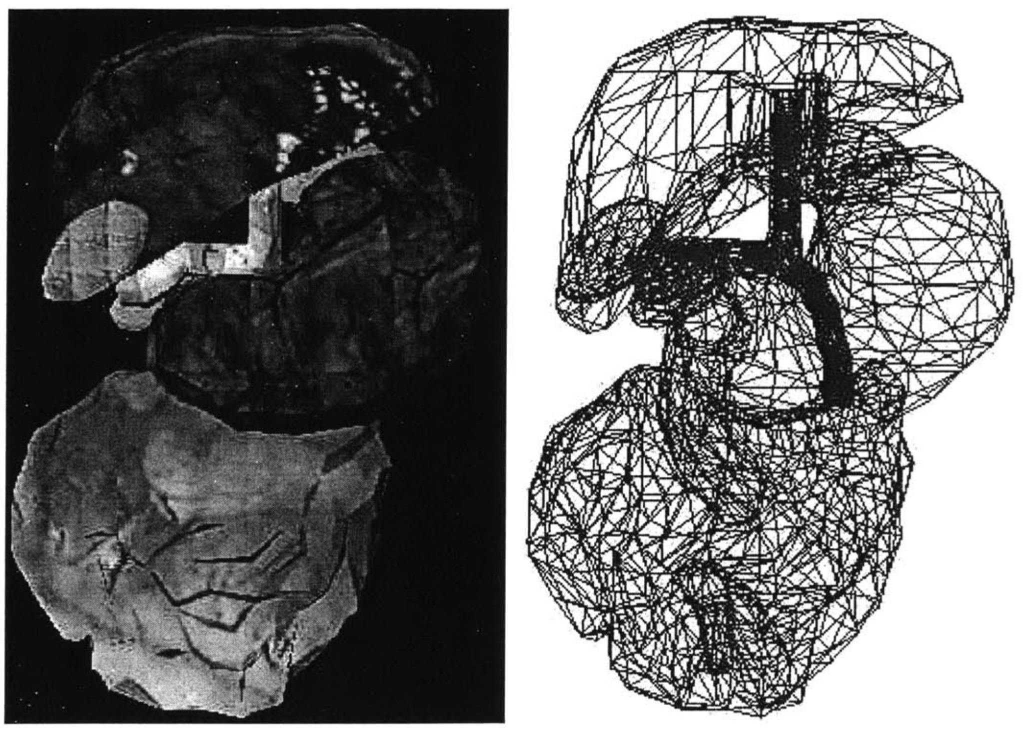
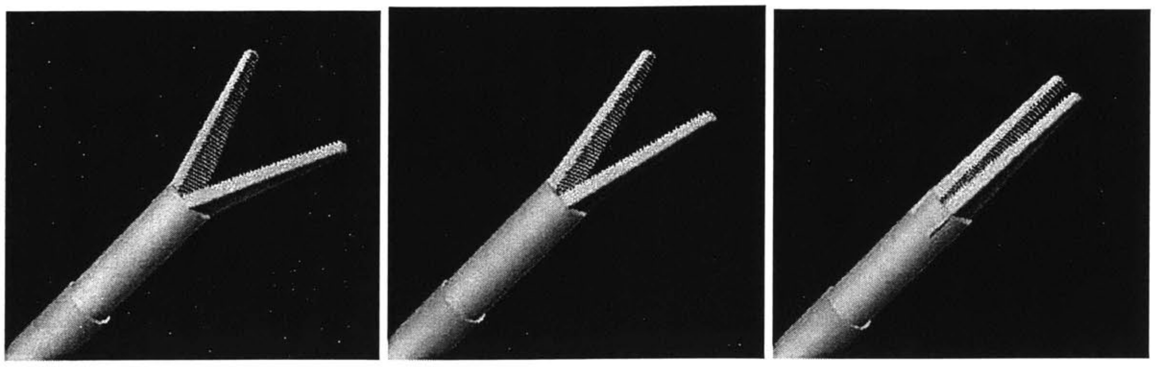


Figure 7-13. (a) The geometry and location of the organs we used in our simulation. The liver, stomach, gallbladder, and intestines were represented as the polyhedra. (b) The wire frame image of those organs.

7.6.2 Modeling of Surgical Instruments

3D computer models of the laparoscopic surgical instruments were by using computer-aided design techniques and packages. Since there are two force-feedback laparoscopic tools in our simulator, we need to show two laparoscopic tools on the screen. As catheter insertion is the process we are simulating, we need only the model of laparoscopic grippers as end-effectors. The pictures in Figure 7-14 show the model of the laparoscopic grippers we used to display on the screen. This laparoscopic gripper is the graphical representation of the tools manipulated by the users. When the users manipulate the force-feedback laparoscopic tools in the simulator, we change the positions of the laparoscopic grippers based on how the user manipulates the tools.



(a) (b) (c)

Figure 7-14. The model of the laparoscopic gripper used in our simulation. The laparoscopic gripper is composed of the tool-body and two grippers. The two grippers were modeled such that they could rotate about a hinge point. The images show that the gripper is (a) fully open, (b) half open, and (c) closed.

On the computer screen, the surgical instruments appear as 3D objects to the user. However, it is very difficult and computationally expensive to detect the collision between the 3D organs and the 3D surgical instruments. In order to reduce the number of computations and achieve fast collision detection rates, we simplify the model of the surgical tools when we perform haptic rendering. The laparoscopic gripper is modeled as combination of three line segments (see Figure 7-15) for haptic rendering. With this simplification, we can use the ray-based haptic rendering techniques described in Chapter 5 to perform real-time collision detection.

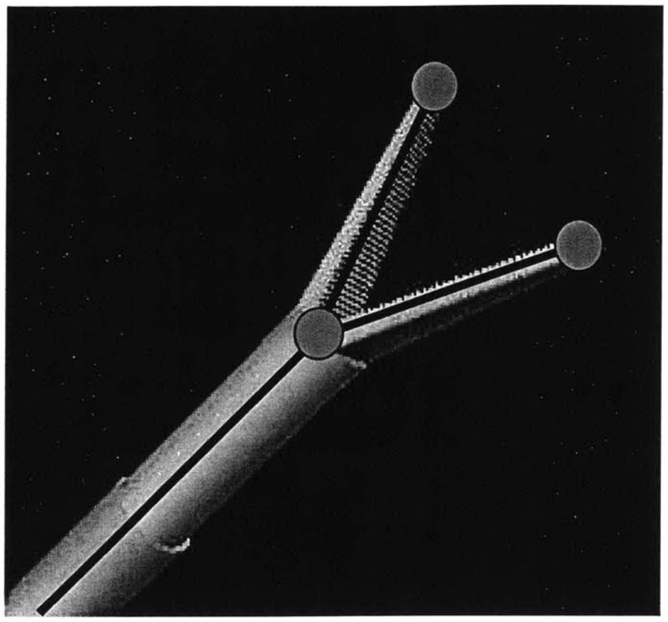


Figure 7-15. The model of the laparoscopic gripper used in haptic rendering. The laparoscopic gripper is shown as complex 3D objects on the screen. However, it is simplified to three line segments in haptic rendering to reduce the number of computations and achieve fast collision detection rates. In this model, one line segment is used to represent the tool-body and two line segments represent the two arms of the gripper.

7.6.3 Collision Detection

In virtual reality simulation, the detection of collision between objects is one of the most important components. A good collision detection algorithm can increase not only the realness of the VE, but also the stability of the interaction. As mentioned earlier in section 7.6.2, we have created a computer model to represent the real surgical tool in the VEs. We call these computer models as virtual tools since they are the representation of the real tools in virtual environments. In our simulator, as the user manipulates the force-feedback surgical tools, the encoders measure the new position and orientation of the surgical tools. This information is used to update the position and orientation of the virtual tools. We then perform collision detection between the virtual tools and virtual objects. If the virtual tool collides with an object, a reaction force will be calculated and sent to the users. In addition to the detection of collision between tools and objects, we also have to detect collision between objects. In the catheter insertion simulation, the catheter is a movable object. Therefore, we have to detect the collision between the catheter and organs. If the catheter collides with any of the organs, the reaction force needs to be calculated and sent to both the catheter and contacted organ. To reduce the computation, we set the deformable organs as static objects so that we do not need to check collisions among several organs present in the VE.

To check the collision between virtual tools and virtual objects, we used the ray-based rendering algorithms described in Chapter 5. The virtual tools shown on the screen are 3D complex objects. However, as described before, three line segments were used to represent each one of the 3D virtual tools when performing the collision detection. The first line segment is used to represent the body of the tool and the other two line segments represent the two arms of the gripper (see Figure 7-15). When the gripper arms are open, the two line segments will be open. When the arms are closed, the two line segments will be closed, too. For example, the images in Figure 7-16 show what the users will see on the screen when they manipulate the force-feedback surgical tools to manipulate the catheter. When performing collision detection, we check for collisions only between the three line segments and virtual objects (see Figure 7-17). If any one of the line segments contacts an object, the interacting force will be calculated and sent to the users. The interacting force will also be used to calculate the dynamics and deformations of the virtual objects in contact.

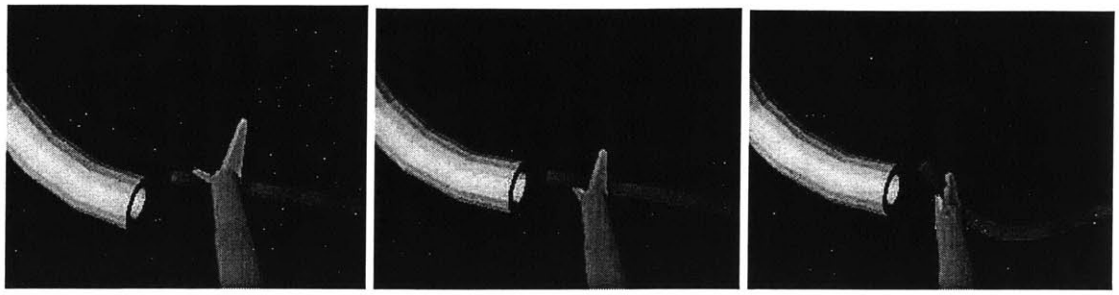


Figure 7-16. The images seen by the users on the screen when they manipulate the force-feedback surgical tools to move a virtual catheter.

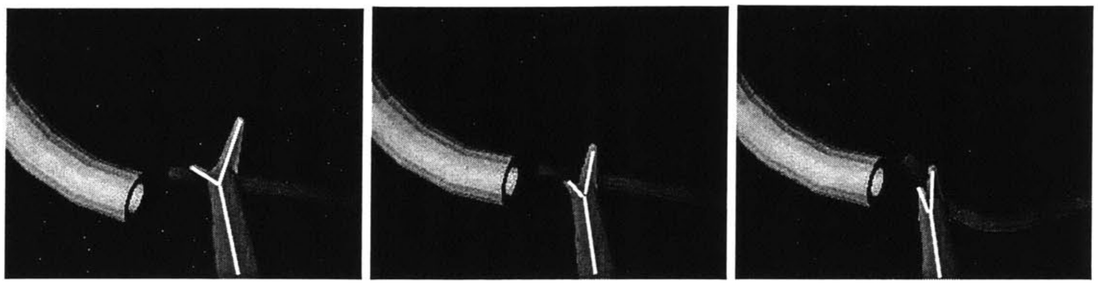


Figure 7-17. Three line segments are used to represent the tools in performing collision detection. Using this simplification, we can reduce the number of computations and achieve fast collision detection rates.

Simulating the interactions between the flexible catheter and deformable organs is challenging. Not only is the detection of object-object collisions difficult and computationally expensive, but also the modeling of "collision response" phase is complex. There are many groups focusing on the algorithms of collision detection and collision response (Cohen et al., 1995, Lin, 1993, Gottschalk et al., 1996, Smith et al., 1995, Hubbard, 1995, Moore and Wilhelms, 1988, Baraff, 1994, Mirtich, 1995 and 1996). However, they are all focused on rigid objects, not on deformable objects. Therefore, we need an alternative way to perform the collision detection. The way we chose was to use multiple points to represent catheter and then detect collision between these points and the organs. We located many points on the surface of the catheter and then used the point-based rendering algorithms described in Chapter 4 to perform collision detection between these points and the organs appropriately. If multiple points contacts an organ, the reaction force will be calculated and used to deform the catheter and the organ. If many of the points contact an organ at the same time, all of the interaction forces at these points have to be calculated. In this way, we will be able to perform collision detection between two 3D deformable objects. The image in Figure 7-18 shows an example of using this method to perform collision detection. The user can use two laparoscopic grippers to grip the vessel and catheter. When the catheter contacts the vessel, the interaction forces will be calculated and used to compute dynamic behaviors of the two compliant objects in contact.

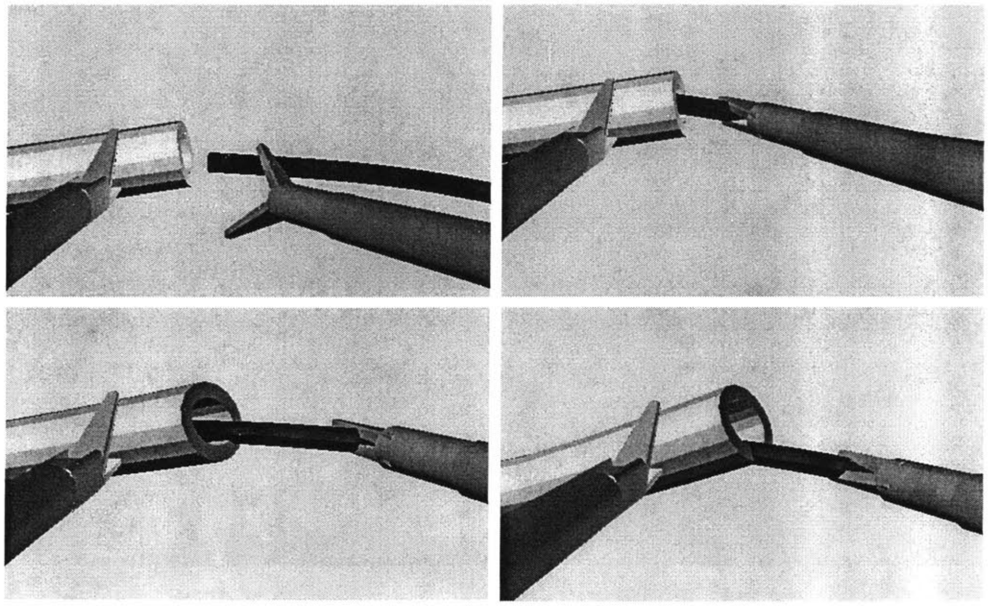


Figure 7-18. Simulation of the catheter insertion in virtual environments. In this simulation, the user could employ two laparoscopic grippers to grip the vessel and catheter and move them around. A group of points are placed on the surface of the catheter. Collision detection is performed between these points and the vessel. When one of the points contacts the vessel, the interaction forces will be calculated and used to compute the interaction dynamics and the associated deformations.

A further way to reduce the computational effort is to place the points representing the catheter along its central axis, instead of on the surface. The catheter used in cholecystectomy is a long thin flexible tube (around 2 mm in diameter). The position difference between the surface and the center is only about 1 mm. Since human kinesthetic resolution is quite poor, users usually are not able to notice the small difference. With this simplification, we can dramatically reduce the computation in collision detection. The images in Figure 7-19 show an example of collision detection with this simplification. In this example, the user is able to use a laparoscopic gripper to move the catheter. Several points are placed in the center of the catheter. The collision detection is performed between these points and the vessel.

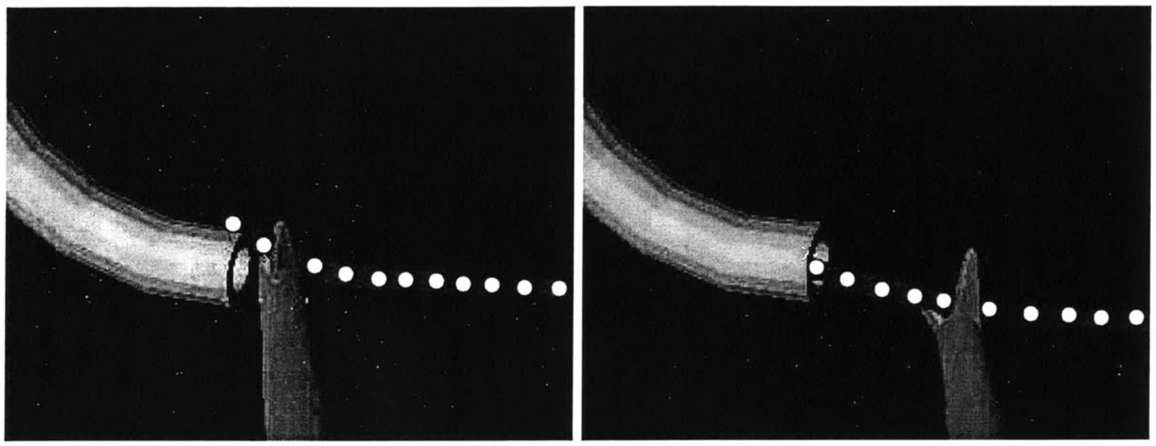


Figure 7-19. Simulation of catheter insertion. The user uses a laparoscopic gripper to move the catheter. Several points are placed in the center of the catheter. The collision detection is performed between these points and the vessel. When one of the points contacts the vessel, the interaction forces will be calculated and used to deform the catheter and the vessel.

7.7 Material Modeling

The techniques mentioned in section 7.6 allow us to perform the collision detection and compute the interaction force when a collision occurs. The next step would be to compute the deformation of the collided objects. Modeling and simulation of soft tissue behavior is always a challenging problem. Soft tissue mechanics is complicated not only because of non-linearity, rate and time dependence of stress-strain relationship, but also because the organs are layered and non-homogeneous. Some of the characteristics of living tissues that are difficult to simulate include (1) *nonlinear response*: displacement versus force profile is a nonlinear curve, resembling an exponential increase on force in relation to displacement, (2) *hysteresis*: force versus displacement profile is different for continuously increasing versus decreasing forces, (3) *non-homogeneous:* displacement versus force curve changes across the structure, (4) *anisotropic*: displacement versus force profile depends on the direction of the applied force, (5) *rate dependent*: the material is viscoelastic.

With the computational mechanics techniques available today, it is unlikely that we can accurately simulate the physical-based behavior of soft tissues in real time, especially on a personal computer. However, the good news is that we don't need highly accurate computation to convince the users that the simulated tissue behavior is realistic. If we can find a solution that makes the users believe what they are experiencing is colse to reality, it is good enough. Therefore, what we really want is to find a solution that is realistic enough for users to suspected disbelieve but efficient enough to be executable in real time. Since human haptic resolution while operating tools is quite poor, this goal is actually achievable.

In our catheter insertion simulation, the virtual objects could be divided into three categories: organs, ducts, and catheter. The main purpose of the organs is to increase the realness of the virtual environment. The users could use the laparoscopic tools to interact with these organs during the simulation. However, these organs play no role in the task of catheter insertion. But the duct and the catheter are important in this simulation. The duct is hollow and supported by a membrane. It can be deformed but not translated. The catheter is flexible. It can be both deformed and translated. Since the material modeling and haptic rendering are two separate loops in the program, we can use different modeling methods to model the different objects.

(1) Modeling of organs

As described earlier, the main purpose of the organs is to increase the realness of the environment. The user seldom interacts with the organs when performing the task. Therefore, the modeling of organs requires less accuracy compared to the modeling of other two virtual objects. To reduce the computation, I use the local deformation technique described in section 6.5 to deform the organs. The local deformation technique is quite fast in terms of computation but is not physically-based.

(2) Modeling of the duct

We modeled the duct using Finite Element Method (FEM) techniques. In FEM, the volume occupied by the object is divided into finite number of elements. Properties of each element is formulated and the elements are assembled together to study the deformation states for given loads. Modeling deformable objects using FEM could achieve very high accuracy. However, it is also very expensive in terms of computation. To apply it in a real time simulation, we have to reduce the accuracy in order to reduce the computation. The technique we used to model the duct was developed by Basdogan (1999). This technique is appropriate for a shell element only. Since the duct is a hollow tube, its wall can be treated as an assemblage of shell elements. In the pre-processing phase, the duct is divided into small shell elements first (see Figure 7-20). A modal analysis is then performed to find some of the most significant modes. The unwanted degrees of freedom will be eliminated at this stage. During the real time simulation, the computation focuses only on those significant modes. In this way, we could reduce the computation dramatically, yet achieve acceptable accuracy (more details could be found in Basdogan, 1999).

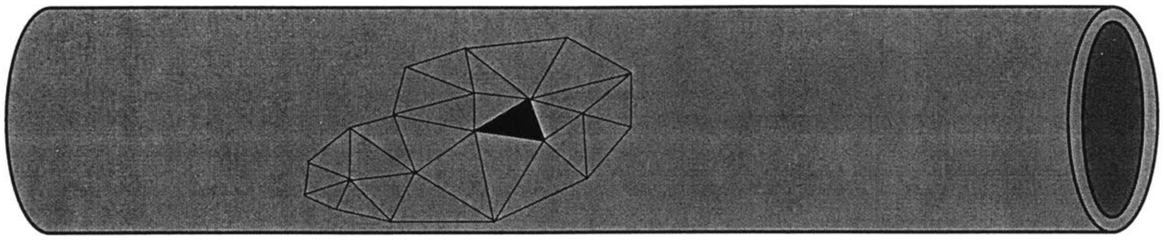


Figure 7-20. The duct is divided into small shell elements in the FEM technique. Properties of each element is formulated and the elements are assembled together to study the deformation states for a given loads.

(3) Modeling of the catheter

We model the catheter using particle-based modeling techniques (also known as lumped parameter or mass-spring-damper models) to reduce the computations and achieve faster haptic rendering rates. Particle systems have been extensively used in computer graphics to simulate the behavior ranging from clothes to fluid flow. This technique is simple to implement since the developer does not need to construct the equations of motion explicitly, but yet is able to simulate the physical-based behavior of deformable objects reasonably well. Particle systems consists of a set of point masses, connected to each other through a network of springs and dampers, moving under the influence of internal and external forces (see Figure 7-21). In this model, each particle is represented by its own mass, position, and velocity. The particle-based modeling is computationally less expensive and more flexible than FEM in implementing the boundary conditions for freely moving objects. In simulating catheter insertion, the catheter needs to be modeled as a freely moving visco-elastic tube that is going to be inserted into a flexible vessel. This makes the particle systems an ideal candidate for the simulation of catheter dynamics.

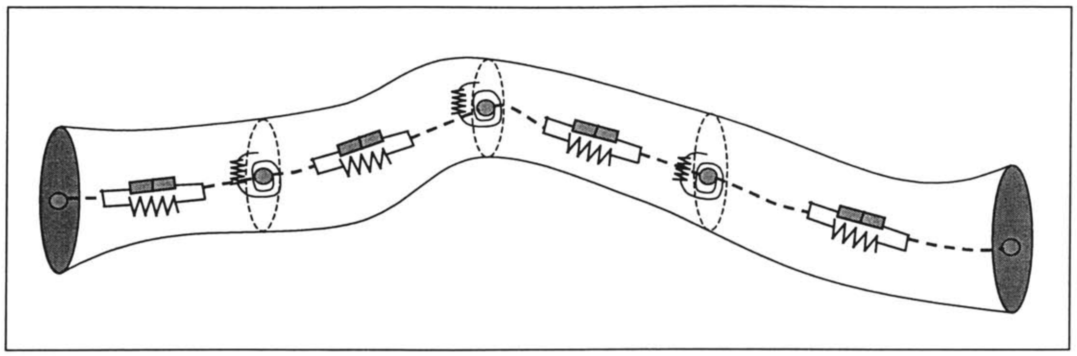


Figure 7-21. The catheter was modeled as a set of particles that lie along the central line of the catheter. Each particle has a mass and is connected to its neighbors with a damper and linear and torsional springs.

In our particle-based model, the particles are located along the central line of catheter. The forces applied to each of the particles could be external forces or internal forces. The external forces come from contact with the laparoscopic tools, the ducts, and the organs. The internal forces come from the linear spring, torsional spring, and damper (see Equation 7-1).

$$F_{internal} = \begin{Bmatrix} F_{linear\_spring} = \sum k_l (l - l_0) \\ F_{torsional\_spring} = \sum \dfrac{k_t(\Theta - \Theta_0)}{l} \\ F_{damper} = -bv \end{Bmatrix} \qquad (7\text{-}1)$$

where l and l_0 are the current and original distances between two particles, respectively, θ and θ_0 are the current and original relative angles between neighboring segments that connect the particle to its neighboring particles, respectively, v is the velocity of the particle, k_l and k_t are the spring constants, and b is the damping coefficient.

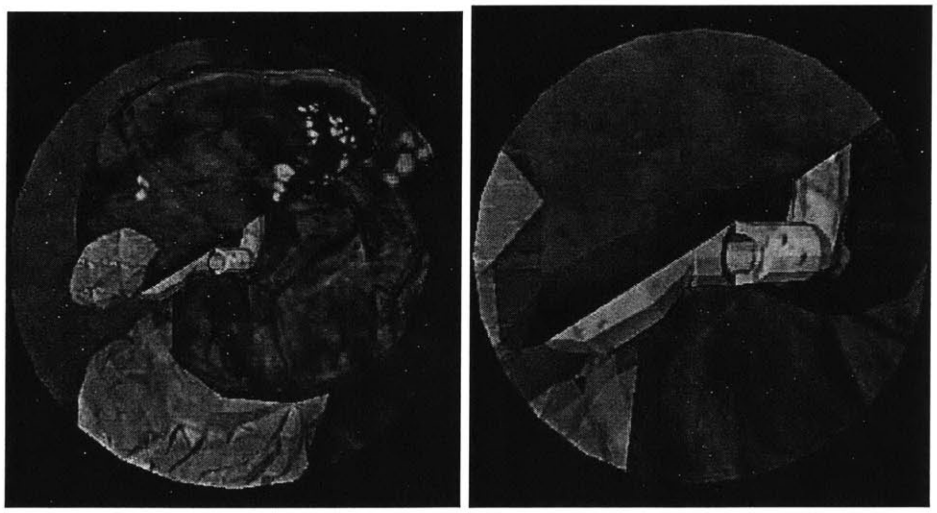
Then, the acceleration, velocity, and position of each particle can be updated in each servo loop using the Euler integration method (see Equation 7-2).

$$\begin{aligned} a_{t+\Delta t} &= F_{total} / m \\ v_{t+\Delta t} &= v_t + \Delta t \; a_{t+\Delta t} \qquad\qquad (7\text{-}2) \\ p_{t+\Delta t} &= p_t + \Delta t \; v_{t+\Delta t} \end{aligned}$$

where m is the mass of the particle, a is the acceleration, v is the velocity, p is the position of the particle.

7.8 Results

With the surgical simulator, the users can have unlimited opportunity to practice catheter insertion in a consistent and cost effective haptic VE. Figure 7-22 shows what users will see on the screen. I use the organ model shown on Figure 7-13 in the environment. I also added a camera to make the view more realistic. The location of the camera could be adjusted using keyboard. When the users want to spot an interesting area, they can zoom in for a bigger image (see Figure 7-22b for an example).



(a) (b)

Figure 7-22. This is the image that the users will see on the screen. (a) The users can see liver, stomach, intestines, and gallbladder. The cystic duct has already been cut, as indicated by a hole on it. (b) The users can change the camera location to view an area of interest from different view points and different distances.

As the user manipulates the force-feedback laparoscopic tools and interacts with the organs, the associated deformations of the organs are displayed on the computer monitor and the reaction forces are fed back to the user through haptic interfaces. If the users grip the catheter, they can feel the gripping force. The user also feels the forces that arise from pulling and pushing of flexible objects during the simulation. The images in Figure 7-23 are example images of the catheter insertion. In Figure 7-23a, the two tools are ready and the catheter is in its initial position. In Figure 7-23b, the user has already inserted a portion of the catheter into the cystic duct. It should be noted that the material properties used in the simulation are probably not realistic at this stage of simulator development. This is because that the material properties of live organs are currently unavailable. Although we have used three different ways to model the material properties (see Section 7.7), those material properties were chosen only based on what feels reasonably right. When the material properties become available in the future, we can easily incorporate them into the simulation and make the simulation more empirically based.

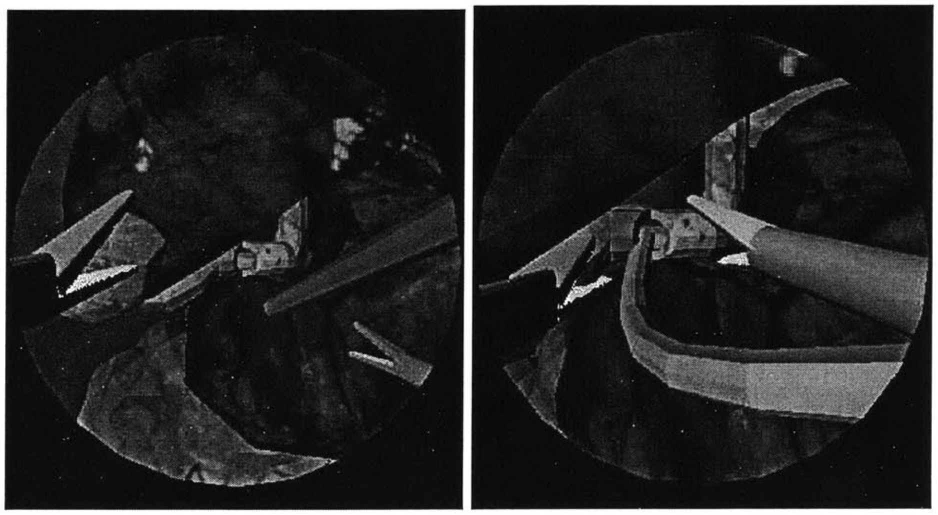


Figure 7-23. These are example images of the catheter insertion. (a) The two laparoscopic tools are ready and the catheter is in its initial position. (b) The user has grasped the catheter and has already inserted a portion of the catheter into the cystic duct.

To get an expert opinion, we have invited a surgeon to our laboratory to try the simulation. His response was very positive. He said that the setup was great and the interaction with the virtual objects was very realistic. He agreed that we could really use the simulator to practice the catheter insertion process. He had two suggestions on the improvement of the simulator. The first one is the texture mapping. The textures used at the current stage are real pictures of the organs. However, the resolution is not high enough. He said that we could have more realistic organs by using higher resolution images. The second suggestion is on the material properties. He believed that the simulation would be perfect if we use real material data.

Chapter 8

Suggestions for Future Work

The mission to create a better haptic VE and incorporate VE into our life is a never-ending task. It is like connected gears. Motion of any part will definitely change the state of whole mechanism. There are many facets to be continued on. First of all, in haptic interaction paradigms, research efforts can be made to improve rendering techniques in simulating interaction between arbitrarily shaped 3D probe and 3D objects. In addition, in object property display algorithms, new or better material modeling and surface property rendering algorithms in simulating different materials are needed. Secondly, in incorporating haptic VE into real-life applications, we need to establish a huge and reliable database, such as the material properties of live tissues, for more realistic material modeling. Finally, combining the techniques and data, we will be able to create better or different simulators. With the simulators, we will be able to simulate different applications or understand the effectiveness of using a haptic simulator.

In this research, I have proposed point-based and ray-based haptic interaction paradigms, which model the probe as a point and as a line segment, respectively. An extension in this area would be to develop algorithms to simulate interactions between 3D probe and 3D objects. If the probe could be modeled as a 3D object, it would give researchers more freedom when developing applications. A few groups have already made efforts in this direction (see section 2.2 for more details). However, there are many constrains to their algorithms. If a more efficient and powerful algorithm could be created to simulate 3D probe and 3D objects interactions, further benefits can be derived from haptic VEs since more diverse and realistic applications can be created.

In enhancing the displayed quality of virtual objects in haptic VEs, developers need to focus on different object property display algorithms. I have proposed a couple of rendering algorithms to simulate different object properties in this research such as friction, texture, dynamic behavior, and compliant behavior. However, there are a large number of different materials that exist in our environment, which cannot be simulated at this stage. For example, there is no algorithm to simulate the haptic behavior of bloody tissue, fur, or clothes. In many of these simulations, the proper property display is the key to a good haptic VE. In surgical simulation, for instance, a good bloody tissue display technique will definitely make the virtual surgery more realistic.

The other important aspect of a quality haptic VE calls for good material modeling, which provides fast and accurate displays of compliant objects. Since real time display is essential in a good haptic VE, a faster material modeling is critical. The existing techniques are either too slow or not sufficiently accurate for realistic haptic rendering. If a more efficient material modeling method is developed, researchers will be able to simulate more complex objects of interest. Aside from improving fundamental techniques, researchers might also head to build the fundamental data of different materials. A good material model requires empirically data on materials, which are usually nonlinear and unavailable at present.

In addition to focusing on the rendering techniques, other direction in continuing this research is to improve the surgical simulator developed in this thesis. As mentioned in section 7.8, the quality of the images and the material data of the organs need to be improved. The improvement of the image quality of the mapped textures is easy to achieve since we need only to take some higher resolution pictures of real organs. However, it is difficult to improve the material properties of the organs since it requires the material data of live organs, which are unavailable at present. One of the alternative ways in improving the material properties of the organs is to create several different material properties and ask experienced surgeons to identify the closest one. If an experienced surgeon feels that the material property is similar to the real one, we can safely assume that most people would be convinced that the material property is realistic. In this way, although we still cannot incorporate empirically based material property, we can implement a material property that is very close to the real one. In addition to improving the quality of the simulation, other direction to improve might be to extend the simulated process. Currently, the surgical trainer simulates only the process of catheter insertion. The process could be extended to simulate the whole cholecystectomy procedure or other laparascopic surgeries.

Another direction in continuing this research might be to use the simulator to do experiments to identify the usefulness of the simulator and understand the training effects of haptic VEs. For example, in flight simulation, extensive experiments have been done to understand the training effects. Researchers have found that ten hours of flight experience in the simulator is equal to one hour of real flight experience. Similarly, different psychophysical experiments have to be done to understand whether the training has any effect or which training method achieves the desired goal with the best efficiency. Only after we fully understand the effect of haptic VEs on human task performance, can we incorporate the haptic VE technology into real-life applications.

Appendix A

A.1. Projection of a point to a plane:

Given a point p, a plane with a unit normal $\vec{N}$ and another point p_0 in the plane, The projected point p' of point p on the plane is calculated as

$$p' = p + ((p_0 - p) \cdot \overrightarrow{N})\overrightarrow{N}.$$

A.2. Distance between a point and a plane:

Given a point p, a plane with a unit normal $\vec{N}$ and another point p_0 on the plane, the distance between the point p and the plane is calculated as d = $\|(P_0 - P) \bullet \vec{N}\|$.

A.3. Projection of a line segment to a plane:

Given a line segment (P_a, P_b), a plane with a unit normal $\vec{N}$ and another point p_0 on the plane, we want to calculate the projected line segment (P_a', P_b') of line segment (P_a, P_b) on the plane. Use the method mentioned in Appendix A.1 to project the point P_a to the plane and obtain the projected point P_a'. Same way, we can obtain the projected point P_b' of the point P_b. Then, the line segment (P_a', P_b') is the projection of the line segment (P_a, P_b) on to the plane.

A.4. Projection of a line segment to a plane parallel to the line segment and containing another line segment:

Two line segments (P_a, P_b) and (P_c, P_d) are defined in 3D space. First, we find a vector ($\vec{N}$) that is perpendicular to the two line segments (P_a, P_b) and (P_c, P_d) (i.e. cross product of two vectors (P_a, P_b) and (P_c, P_d)) and normalize it. The plane perpendicular to the line segment (P_a, P_b) and containing the line segment (P_c, P_d) will have the unit normal $\vec{N}$ and the point P_c. Use the method mentioned in Appendix A.3 to project the line segment (P_a, P_b) to this plane and obtain the projected line segment (P_a', P_b').

A.5. Intersection point between a line and a plane:

Given a line segment (P_a, P_b) and a plane with a unit normal $\vec{N}$ and another point P_0 on the plane, we want to find out the intersection point between this line segment and the plane. First, we check the signs of dot products (P_a-P_0) $\vec{N}$ and (P_b-P_0) $\vec{N}$. If any of them is equal to zero, then at least one of the points is on the plane. If both of them have the same sign, the two points P_a and P_b are on the same side of the plane and, therefore, there will be no intersection point. If the signs are different, we calculate the distances from the points P_a and P_b to the plane (let's say, d_1 and d_2, respectively). The intersection point will be (d_2-*P_a + d_1-* P_b)/(d_1 + d_2).

A.6. Nearest Distance between two line segments in 3D space:

Two line segments (P_a, P_b) and (P_c, P_d) are defined in 3D space. We want to calculate the nearest distance between the two line segments. First, we project the line segment (P_a, P_b) to a plane that is perpendicular to the line segment (P_a, P_b) and contains the line segment (P_c, P_d) using the method mentioned in Appendix A.4, which will be (P_a', P_b'). We also calculate the distance between the point P_a and the plane (Appendix A. 2), which is defined as d_1. We then find the nearest distance between the two line segments (P_a', P_b') and (P_c, P_d), which is defined as d_2. Then, the nearest distance between the two line segments will be the square root of (d_1 * d_1+ d_2 * d_2).

A.7. Movement of a point penetrates a triangular polygon:

A point is at point P_1 in time t_1 and at point P_2 in time t_2. A triangular polygon has vertices P_a, P_b, and P_c. We want to check if the movement of the point penetrates the triangular polygon. First of all, we find the normal $\vec{N}$ of the triangle which is equal to the cross product of the two vectors (P_a - P_c) and (P_b - P_c) and normalize it. We then check if there is an intersection between the line segment (P_1, P_2) and the plane containing the triangle (Appendix A. 5). If so, we check if the intersection point is inside the triangle or not. If the intersection point is inside the triangle, the movement of the point penetrates the triangle.

A.8. Collision between a moving line segment and a static line segment in 3D space:

At time t_0, the line segment l_{ab} is at $l_{ab}(t_0)$ and at time t_1, it moves to $l_{ab}(t_1)$. We want to know if the movement of l_{ab} from t_0 to t_1 passes the line segment l_{cd}. The analytical solution could be found in Schomer and Thiel (1995). Their method gives the exact solution for the collision time and the collision point. However, this solution is computational too expensive to be used in haptic rendering since the haptic loop needs to be updated in 1 kHz. Instead, we present a simplified method in here. Although this method does not calculate the exact collision time and the collision point, it reports whether the movement of one line crosses the other. For this method to be valid, the translation and rotation of the line segment should be very small. (This is absolutely the case in haptic rendering since the haptic loop is updated in 1 kHz and the movements of our hand are quite slow.) To detect the collision, we first calculate the vector $\overrightarrow{D_0}$ which represents the nearest distance from l_{cd} to $l_{ab}(t_0)$ (see Appendix A. 6). Let's call the nearest points on the two lines are P_{cd0} and P_{ab0}. We also calculate the nearest distance vector $\overrightarrow{D_1}$ from l_{cd} to $l_{ab}(t_1)$. Let's call the nearest points on the two lines are P_{cd1} and P_{ab1}.

If (1) the dot product of $\overrightarrow{D_0}$ and $\overrightarrow{D_1}$ is negative, and (2) neither P_{cd0} nor P_{cd1} are the end points of l_{cd}, (3) P_{ab0} is not the end point of $l_{ab}(t_0)$, (4) P_{ab1} is not the end point of $l_{ab}(t_1)$, we say the movement of l_{ab} from t_0 to t_1 crosses the line segment l_{cd}.

Reference

Adachi, Y., Kumano, T., Ogino, K. 1995, March 11-15, Intermediate Representation for Stiff Virtual Objects. Proc. IEEE Virtual Reality Annual Intl. Symposium, Research Triangle Park, N. Carolina, pp. 203-210.

Avila, R. S. and Sobierajski, L. M., 1996, A Haptic Interaction Method for Volume Visualization, IEEE Proceedings of Visualization, pp. 197-204.

Baraff, D., 1994, Fast Contact Force Computation for Nonpenetrating Rigid Bodies. *ACM, Proceedings of SIGGRAPH,* 28, pp. 23-34.

Basdogan, C., 1999, "Force-reflecting deformable objects for virtual environments", SIGGRAPH 99, Course Notes, No: 38.

Basdogan, C. 1999, "Integration of force feedback into virtual reality based training systems for simulating minimally invasive procedures", to appear as a chapter in "Human and Machine Haptics".

Basdogan, C., Ho, C., and Srinivasan, M.A., 1997, Nov 15-21, A Ray-Based Haptic Rendering Technique for Displaying Shape and Texture of 3D Objects in Virtual Environments, *ASME Winter Annual Meeting,* Dallas, TX, DSC-61, pp. 77-84.

Basdogan C., Ho, C., Srinivasan, M.A., Small, S., Dawson, S., 1998, January 19-22, Force interactions in laparoscopic simulations: haptic rendering of soft tissues, *Proceedings of the Medicine Meets Virtual Reality VI Conference*, San Diego, CA, pp. 385-391.

Bathe, K., (1996), "Finite Element Procedures", Prentice Hall, New Jersey.

Ben-Ur, E., "Development of a Force-Feedback Laparoscopic Surgery Simulator", M.S. Thesis, Mechanical Engineering Department, Massachusetts Institute of Technology, (1999).

Bier, E.A., Sloan K.R. (1986). Two-Part Texture Mapping. *IEEE Computer Graphics and Applications*, 40-53.

Blinn, J.F. (1978). Simulation of Wrinkled Surfaces. *ACM (Proceedings of SIGGRAPH),* 12(3), 286-292.

Burdea, G. (1996). *Force and Touch Feedback for Virtual Reality.* John Wiley and Sons, Inc., NY.

Buttolo, P., Hannaford, B., 1995, March, Pen-Based Force Display for Precision Manipulation in Virtual Environments, Proceedings IEEE Virtual Reality Annual International Symposium, North Carolina, March, pp. 217-224.

Chen, J., DiMattia, C., Falvo, M., Thiansathaporn P., Superfine, R., Taylor, R.M., 1997, Sticking to the Point: A Friction and Adhesion Model for Simulated Surfaces, Proceedings of the Sixth Annual Symposium on Haptic Interfaces and Virtual Environment and Teleoperator Systems, Dallas, TX, pp. 167-171.

Cohen, J., Lin, M., Manocha, D., Ponamgi, K., 1995, I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments, *Proceedings of ACM Interactive 3D Graphics Conference*, pp. 189-196.

Cotin, S., Delingette, H., Ayache, N., 1999, "Real-time Elastic Deformations of Soft Tissues for Surgery Simulation", IEEE Transactions on Visualization and Computer Graphics, Vol. 5, No. 1, pp. 62-73.

Ebert, D.S., Musgrave F.K., Peachey D., Perlin K., Worley S. (1994). *Texturing and Modeling.* AP Professional, Cambridge, MA.

Fleischer, K. W., Laidlaw, D. H., Currin, B. L., and Barr, A. H. (1995). Cellular Texture Generation. *ACM (Proceedings of SIGGRAPH),* August, 239-248.

Foley, J.D., van Dam, A., Feiner, S. K., Hughes, J.F. (1995). *Computer Graphics: Principles and Practice.* Addison-Wesley.

Fritz, J. and Barner K., 1996, Haptic Scientific Visualization, *Proceedings of the First PHANToM Users Group Workshop*, Eds: Salisbury J.K. and Srinivasan M.A. MIT-AI TR-1596 and RLE TR-612.

Gottschalk, S., Lin, M., and Manocha, D., 1996, August, OBB-Tree: A hierarchical Structure for Rapid Interference Detection, *ACM, Proceedings of SIGGRAPH.*

Green, D. F. and Salisbury, J. K., 1997, Oct. 19-22, Texture Sensing and Simulation Using the PHANToM: Towards Remote Sensing of Soil Properties, *Proceedings of the Second PHANToM Users Group Workshop.*

Green, Donald F., June 1998, Haptic Simulation of Naturally Occurring Textures and Soil Properties, MS thesis, MIT.

Gregroy, A., Lin, M., Gottschalk, S., and Taylor, R., 1999, H-Collide: A Framework for Fast and Accurate Collision Detection for Haptic Interaction, *Proceedings of IEEE Virtual Reality Conference,* 38-45.

Ho, C., Basdogan, C., Srinivasan M.A. 1997. Haptic Rendering: Point- and Ray-Based Interactions. *Proceedings of the Second PHANToM Users Group Workshop*, Dedham, MA, October 20-21.

Ho, C., Basdogan, C., Srinivasan M.A., 1999, "Efficient Point-Based Rendering Techniques for Haptic Display of Virtual Objects ", *Presence*, October, Volume 8, Issue 5, 477-491.

Ho, C., Basdogan, C., Srinivasan, M.A., 2000, "Ray-Based Haptic Rendering: Force and Torque Interactions Between a Line Probe and 3D Objects in Virtual Environments", submitted to International Journal of Robotics Research (IJRR).

Hubbard, P. (1995). Collision Detection for Interactive Graphics Applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3), 219-230.

Ikei, Y., Wakamatsu, K., Fukuda, S. (1997). Vibratory Tactile Display of Image-Based Textures. *IEEE Computer Graphics and Applications*, November, 53-61.

Iwata, H., 1993, Pen-Based Haptic Virtual Environment, Proc. VRAIS IEEE'93, Seattle, pp. 287-292.

Lin, M., 1993, Efficient Collision Detection for Animation and Robotics. Ph.D. thesis, University of California, Berkeley.

Mandelbrot, B. 1982, *The Fractal Geometry of Nature*. W.H. Freeman.

Mark, W., Randolph S., Finch, M., Van Verth, J., Taylor, R.M., 1996, August, Adding Force Feedback to Graphics Systems: Issues and Solutions, Computer Graphics: Proceedings of SIGGRAPH, pp. 447-452.

Massie, T. H. 1993, *Initial Haptic Explorations with the Phantom: Virtual Touch Through Point Interaction*. MS thesis, Massachusetts Institute of Technology.

Massie T.H., Salisbury J.K. 1994. The PHANToM Haptic Interface: A Device for Probing Virtual Objects. *Proceedings of the ASME Dynamic Systems and Control Division*, 55(1), 295-301.

Max, N.L., Becker, B.G. 1994. Bump Shading for Volume Textures. *IEEE Computer Graphics and App.*, 4, 18-20.

McNeely, W.A., Puterbaugh K.D., Troy, J.J., 1999, Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling, Proceedings of SIGGRAPH, pp. 401-408.

Millman, P., Colgate, J.E., 1991, Design of a four degree-freedom force reflection manipulandum with a specified force/torque workspace, IEEE International Conference on Robotics and Automation, Sacremento, CA, pp. 1488-1493.

Minsky, M. D. R. 1995, Computational Haptics: The Sandpaper System for Synthesizing Texture for a Force-Feedback Display. Ph.D. thesis, Massachusetts Institute of Technology.

Minsky, M., Ming, O., Steele, F., Brook, F.P., and Behensky, M. 1990, Feeling and seeing: issues in force display. *Proceedings of the symposium on 3D Real-Time Interactive Graphics,* 24, 235-243.

Mirtich, B., 1996, Impulse-based Dynamic Simulation of Rigid Body Systems. Ph.D. thesis, University of California, Berkeley.

Mirtich, B., Canny, J., 1995, April, Impulse-based Simulation of Rigid Bodies, Proceedings of Symposium on Interactive 3D Graphics.

Moore, M., Wilhelms, J., 1988, Collision Detection and Response for Computer Animation, *ACM, Proceedings of SIGGRAPH,* 22(4), pp. 289-298.

Morgenbesser, H.B., Srinivasan, M.A. 1996, Force Shading for Haptic Shape Perception. *Proceedings of the ASME Dynamic Systems and Control Division,* 58, 407-412.

Perlin, K. (1985). An Image Synthesizer. *ACM SIGGRAPH,* 19(3), 287-296.

Ruspini, D.C., Kolarov, K., Khatib O. 1996, Robust Haptic Display of Graphical Environments. Proceedings of the First PHANToM Users Group Workshop, Eds: Salisbury J.K. and Srinivasan M.A. MIT-AI TR-1596 and RLE TR-612.

Ruspini, D.C., Kolarov, K., Khatib O., 1997, July, The Haptic Display of Complex Graphical Environments, *ACM, Proceedings of SIGGRAPH,* pp. 345-352.

Salcudean, S.E. and Vlaar, T.D., 1994, On the Emulation of Stiff Walls and Static Friction with a Magnetically Levitated Input / Output Device, *ASME* DSC, 55(1), pp. 303-309.

Salisbury, J.K., and Tarr, C., 1997, Haptic Rendering of Surfaces Defined by Implicit Functions, *Proceedings of the ASME,* DSC-61, pp. 61-67.

Salisbury, J.K., Brock, D., Massie, T., Swarup, N., Zilles C., 1995, Haptic Rendering: Programming touch interaction with virtual objects, *Proceedings of the ACM Symposium on Interactive 3D Graphics,* Monterey, California.

Salisbury, J. K., Srinivasan, M. A. 1997, Phantom-Based Haptic Interaction with Virtual Objects. IEEE Computer Graphics and Applications, 17(5).

Schippers E., Schumpelick V., 1996, "Requirements and Possibilities of Computer-Assisted Endoscopic Surgery" in *Computer Assisted Surgery*, pp. 567-575, Ed: Taylor et al., MIT Press, Cambridge, MA.

Schomer, E. and Thiel, C., 1995, Efficient collision detection for moving polyhedra, ACM, 11th Computational Geometry, Vancouver, B.C. Canada, ACM 0-89791-724-3/95/0006.

Siira, J., Pai D. K., 1996, Haptic Texturing - A Stochastic Approach, *Proceedings of the IEEE International Conference on Robotics and Automation,* Minneapolis, Minnesota, pp. 557-562.

Smith, A., Kitamura, Y., Takemura, H., and Kishino, F. 1995. A Simple and Efficient Method for Accurate Collision Detection Among Deformable Polyhedral Objects in Arbitrary Motion. IEEE Virtual Reality Annual International Symposium, 136-145.

Srinivasan, M.A. 1995. Haptic Interfaces. In *Virtual Reality: Scientific and Technical Challenges*, Eds: N. Durlach and A. S. Mavor, National Academy Press, 161-187.

Srinivasan, M.A., and Basdogan, C., 1997, Haptics in Virtual Environments: Taxonomy, Research Status, and Challenges, Computers and Graphics, Vol. 21, No.4. 393 – 404.

Thompson, T.V., Johnson, D.E. and Cohen, E., 1997, April 27-30, Direct haptic rendering of sculptured models, Proc. Sym. Interactive 3D Graphics, Providence, RI, pp. 1-10.

Treat M.R., 1996, "Surgeon's Perspective on the Difficulties of Laparoscopic Surgery" in *Computer Assisted Surgery*", pp. 559-565, Ed: Taylor et al., MIT Press, Cambridge, MA.

Turk, G. (1991). Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion. *ACM (Proceedings of SIGGRAPH)*, 25(4), 289-298.

Watt, A., Watt, M. (1992). *Advanced Animation and Rendering Techniques*. Addison-Wesley, NY.

Wijk, J. J. V. (1991). Spot Noise. *ACM (Proceedings of SIGGRAPH)*, 25(4), 309-318.

Witkin, A., and Kass, M. (1991). Reaction-Diffusion Textures. *ACM (Proceedings of SIGGRAPH)*, 25(4), 299-308.

Worley, S. (1996). A Cellular Texture Basis Function. *ACM (Proceedings of SIGGRAPH)*, August, 291-294.

Zilles, C.B., and Salisbury, J.K., 1995, A Constraint-Based God-Object Method for Haptic Display, *IEEE International Conference on Intelligent Robots and System, Human Robot Interaction, and Co-operative Robots*, IROS, Vol 3, pp 146-151