DEVELOPMENT OF REAL-TIME HAPTIC APPLICATION

by

FARIDA SHIRAJ VAHORA, B.S.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

May, 2000

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge my sincere gratitude to Dr. Bharti Temkin for accepting me as her graduate student and serving as an advisor for my master's thesis in computer science. She has gone beyond the call of duty in making time available for guiding me in all aspects of my thesis work. I have come to admire her infinite patience and continued encouragement during many roadblocks that I have encountered.

I would also like to acknowledge my gratitude and thanks to my committee member, Dr. Bill Marcy. Dr. Marcy, in spite of his very busy schedule, has been available to me for feedback, support and problem solving.

Thanks are also due to my husband for his encouragement. In addition, I am grateful to my children for their support in taking care of their own needs and making time available for me to work on my thesis.

TABLE OF CONTENTS

ABSTRACT

This on going research focuses on developing a PC/NT based real-time, virtual reality haptic application. Our main goal is to resolve the critical real-time issues required to develop a stable haptic application. Critical real-time issues addressed include: (1) interprocess communication, (2) synchronization of haptic and graphics interface and (3) finding application size limits for achieving high haptic rendering rates with stable haptic interactions. We are using a breast biopsy simulator prototype as an example to show the viability of developing such a system. This system monitors and indirectly guides the surgeon's movements by providing high fidelity visual and force feedback cues as the area of surgical interest is approached.

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Computer haptics, although in its infancy, is generating a considerable amount of research. Newer techniques are being developed. When full capability is achieved, its implication will be profound and widespread. Applications of this technology include educational training, entertainment, healthcare (surgical simulation), scientific visualization, telecommunication, and design manufacturing and marketing.

One of the broad application areas of haptic technology is training people to perform real world tasks using simulation models. Simulations involved with touch and feel (e.g., medical surgery) require realistic real-time feedback, since the skills being learned include feeling and manipulating the environment.

Virtual reality simulation holds the potential to improve surgical training in a manner similar to simulated pilot training. The advantages of virtual reality for surgical training includes a reduction in operating room time required for training, quantification of trainee's performances and an

ability to provide experiences independent of a hospital's limited patient population. All this results in accelerating the learning curve.

Surgery is a human activity that requires both visual and manual tasks. In order to optimize an effective virtual surgical environment, real time interactivity and realistic visualization are essential. The requirements for a computer system to simulate complex interactions are many. The system must be able to generate realistic visual and haptic cues that provide temporal and spatial coherence with data manipulation. It must provide high fidelity haptic and visual cues to "convince" and thereby engage the user. And finally, it must provide interaction and behavior that are similar enough to the real world in order to maximize transfer to actual practice. It is these requirements that make the haptic applications very critical and need the adjustment of parameters for the best possible haptic interaction and stability. The focus of present work is an analysis of critical real-time issues needed for development of a real time haptic application.

CHAPTER II

BACKGROUND

History of Haptic Interface

Specialized devices to provide virtual haptic feedback are the result of decades of active research in both industry and university laboratories. In the 1950s and 1960s, when virtual reality did not exist, research was aimed at developing and improving telerobotic systems. In 1954, Goertz at Argonne National Laboratory developed an electrical servomechanism [1]. In this system, the operator controls a "master" arm that transmits his commands to a remote slave. The presence of electrical servoactuators in the master arm receive feedback signals from slave sensors and apply forces to the user's hand grasping the master which makes the user feel like as if he is manipulating the remote environment directly.

In 1964, researchers at General Electric Company developed a power amplifying exoskeleton called "Hardiman" which was used to amplify the power of the user [2]. The Hardiman consisted of 30 hydraulically powered servo joints. There were several drawbacks with this system: (1) unstable control at normal operating speed because control technology was not

developed well at the time. (2) The possibility of a leak in hydraulics was potentially hazardous to the user enclosed in Hardiman.

A much simpler and safer device was build in 1966 by Jones and Thousand called the first dextrous master manipulator using pneumatic bladders [3]. A dextrous master measured user hand commands (usually through sensing gloves) and provided feedback to independent fingers. Here the user receives a very natural sensation as if he was grasping the object directly.

By 1971, GROPE-I demonstrated a two-dimensional continuous force field simulation [4]. In 1976, the research team used a surplus Argone arm to simulate three-dimensional collision forces [5].

In 1981, Zarudiansky developed a more complex dextrous master with feedback to each finger phalange. His design used actuators connected to rings attached to the user's inner glove. These actuators provided force feedback to the fingers as well as the palm and the wrist [3].

In 1989, NASA and JPL [3] created a teleoperation testbed using the Salisbury/JPL arm, a six-degree-of-freedom generalized master with force feedback. The advance brought by the Salisbury/JPL arm was the introduction of computer-based Cartesian control, allowing the master to be

more compact and to be able to teleoperate slave with different kinematic configuration.

All the above masters were developed originally for telerobotics application and not to serve as I/O devices for virtual reality, which appeared in the late 1970s. Researchers then started to develop special-purpose tactile/force feedback hardware.

One of the first prototypes to provide tactile feedback from a graphics simulation was the "Sandpaper" system developed at the Massachusetts Institute of Technology Media Laboratory [6]. This prototype consisted of a two-degree-of-freedom joystick with large electrical actuators in an enclosure placed by the computer. The high bandwidth of the feedback loop (500 to 1000Hz) allowed for both force and tactile feedback in a single simple and easy to use device. Thus, it was possible to move a cursor over various samples of virtual sandpaper and feel their surface texture.

The drawback of using feedback joystick is the reduction in the user's freedom of motion, since the hand has to stay on the joystick. In order to allow natural freedom of motion of the user hand, and still provide tactile and force feedback, masters have to be portable and light. One such device was demonstrated in 1992 at the Rutgers University CAIP Center [7]. This

device uses four pneumatic microactuators placed in the palm to give users the feel of the hardness of virtual objects being manipulated. It weighs only 100 grams; therefore, it does not tire users during prolonged simulations.

The first commercial system designed for virtual I/O became available at the end of 1993 through the introduction of the Touch Master and SAFIRE Master [8]. These were followed by the recent introduction of lower-cost masters called PHANToM [9] and Impulse Engine [10]. With these devices, developers have the tools to complement the visual and sound feedback created by earlier I/O devices. As the price of commercial tactile and force feedback systems drop, it will allow widespread acceptance and integration with most of today's virtual simulations.

Haptic System: Basic Concepts

Human Haptic system

We use our hands to both change and measure our environment [11]. To manipulate an object, we apply forces to move it, while simultaneously sensing the result of these actions. Thus in almost all hand activities, we use both the sensory and motor parts of our haptic system [12] to get information from or to alter the environment.

Human haptic sensory information [13] consists of tactile and kinesthetic data (force feedback). Tactile and force feedback differ in several respects such as physiology, control requirements and functionality. When interacting with a surface, tactile feedback is sensed by the high bandwidth receptors placed close to the skin. Force feedback is sensed by low bandwidth receptors placed in the skin around the joints, tendons and muscles. These receptors provide information on the total contact force and weight.

In the real world, whenever we touch an object, it imposes forces on our skin. These net forces, plus the posture and motion of hands and arms are transmitted to the brain as kinesthetic information. This is how we feel the properties of an object. In contrast, fine texture, small shapes and softness are felt through our tactile sensors [12].

Computer haptic system

Human haptics is the sense of touch. Computer haptics is simulating human haptics via a device. In a virtual world tactile feedback is a sensation applied to the skin, typically in response to contact or other actions. It is simply a sensation that indicates some condition. On the other hand, force

feedback is a sensation of weight or resistance in a virtual world. Force feedback produces a force equivalent (or scaled) to that of a real object [12].

Computer haptic process

To provide haptic interaction a user interacts with computer through hardware I/O device that tracks a position and in return exerts a force to the user through this same device. The basic model for haptic interface is described in Figure 1. The physical interface to the user is called PHANToM haptic interface from SensAble Technologies Inc. The fundamental algorithm for haptic rendering (haptic servo loop) is presented in Figure 2. In this process, position and orientation of the PHANToM device are retrieved by the computer model; the computer detects the collision between the device and the geometry of the virtual object and calculates a reaction force vector based on the spring compression model. In the spring compression model, the force feedback depends on the stiffness of an object (K) and the distance of the compression (d) determined by how far the haptic device's virtual position penetrated the object in the virtual world (see Figure 3). This process (haptic servo loop) repeats at a regular interval. Varying the time interval between the iteration can effect the stability of the haptic simulation.

Since the human hand is very sensitive to vibration, we are able to feel these vibrations up to 1kHz. Due to this, haptic displays must be updated at 1kHz.

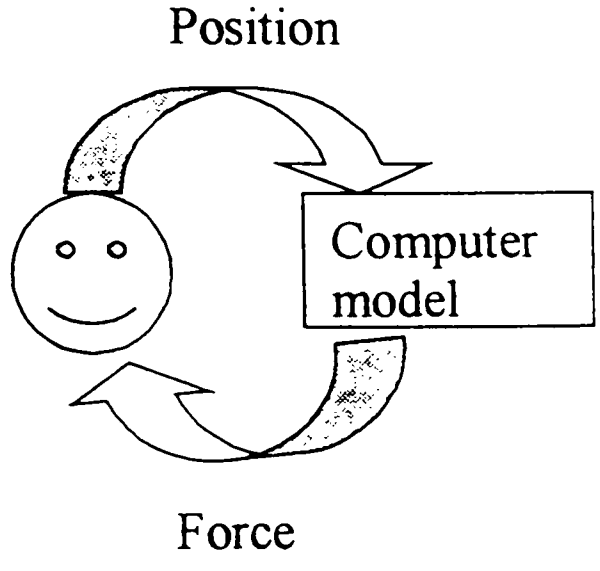

Figure 1. Basic haptic Interface Model

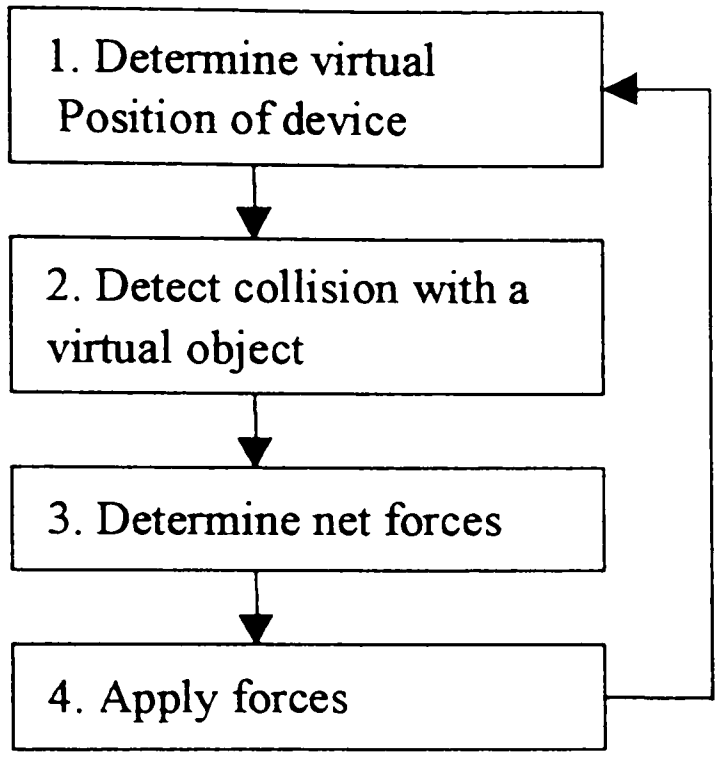

Figure 2. Haptic rendering loop

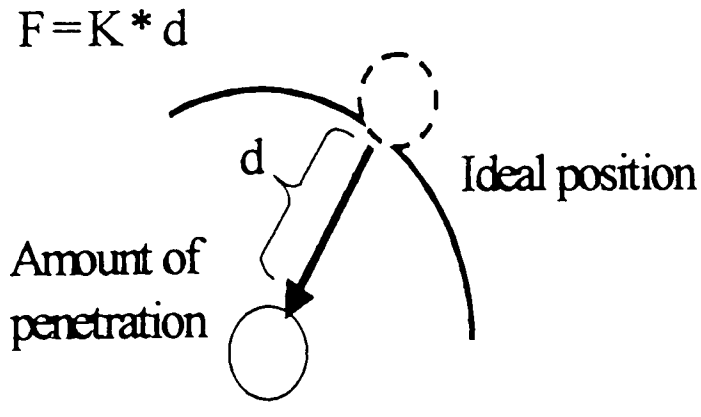

Figure 3. Spring force model

Haptic Rendering Techniques

In general, two calculations must be performed for all geometries to be rendered haptically. First, the software must check for collision between the user's fingertip and the virtual object. If the user is not in contact with the virtual object, then further calculation for that particular object is not required. However, if a collision has occurred, a reaction force must be calculated. To compute the reaction force, the surface normal must be determined at the location touched by the user.

Initial haptic rendering methods focused on providing force feedback for simple, rigid and frictionless objects [14]. Massie and Salisbury [14] developed the Phantom haptic interface device and proposed a point-based method. In this method, computation for determining the force vector involves dividing the object in to sub-spaces associated with particular portion of object surface. If a haptic interface point penetrates into a region shared by multiple sub-spaces, superposition of surface normal is used to calculate the direction of resultant force vector (see Figure 4). However, there are problems with this haptic rendering technique: (1) it is not easy to divide an object into sub-spaces or to construct virtual environments from

primitive objects and (2) the superposition of force vectors may breakdown for thin or complex shaped objects.

Zilles and Salisbury [15] developed a more sophisticated point-based method to hapticaly render polygonal surfaces. They defined a new point called the god object to represent the location of the surface point. The location of god-object point is calculated each time the user manipulates the generic probe of the haptic interface by using constrained optimization technique. In constrained optimization technique, the distance between the god-object and the haptic interface point is minimized, and the god-object always remains on the surface of the object even though the haptic interface point can penetrate it. This method is the most robust and flexible to haptic rendering achieved to date, and has been used to model the feel of complex polyhedral objects.

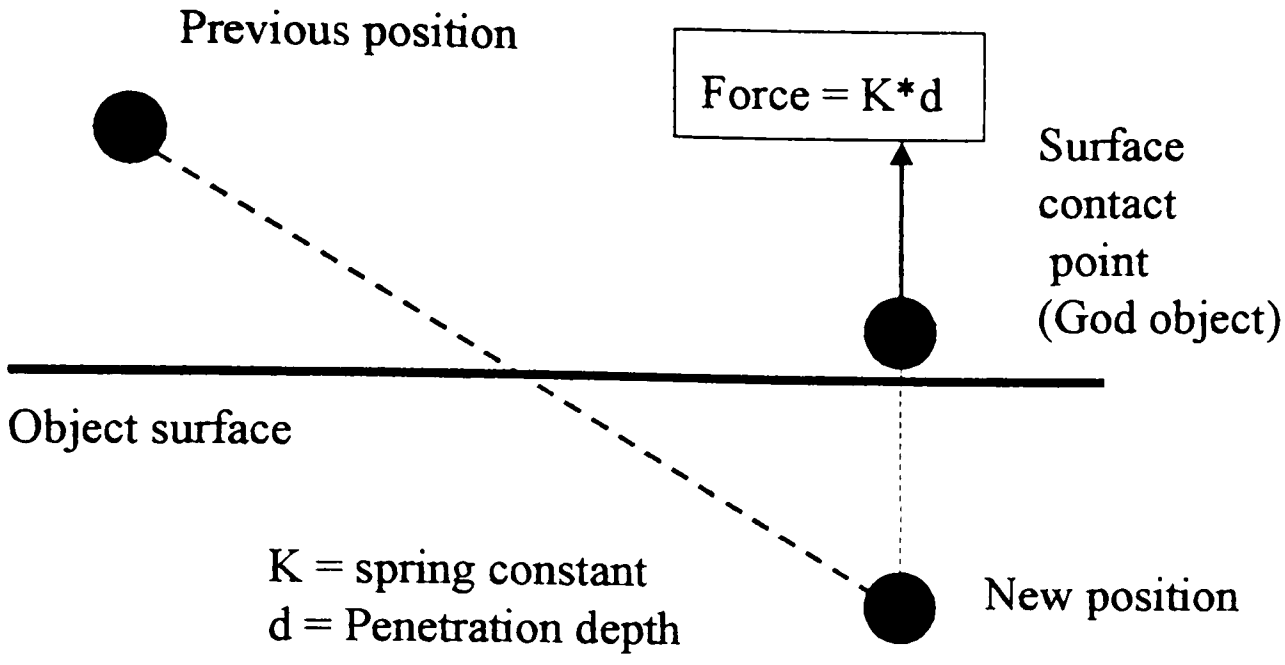

Figure 4. Point-base haptic interactions.

Review of Surgical Simulation Systems

Computer simulated biopsy procedure holds promise for training and further refining of minimally invasive surgical procedures. Although minimally invasive surgical procedures have many advantages over standard surgical procedures, they are not risk free. There are several surgical simulation systems that have been developed in research laboratories. For example, Georgetown University Medical Center [16] developed a spine biopsy simulator for training purposes. This training simulator is designed to mimic existing procedure as closely as possible. This system uses force-

feedback and is able to track the procedure on the computer screen. When the procedure is complete, the user is evaluated on how close the final position was to the target point.

The surgical simulation system developed by Boston dynamic Inc. in collaboration with Hershey Medical Center of Pennsylvania State University [17] simulates anastomosis of blood vessels using high quality graphics, phantom based haptic display and force feedback. This surgical trainer compares the performance of surgeons and medical students on a simulated surgical task. The results obtained from this experiment suggest that the surgical trainer is useful in quantifying surgical skills.

The present research uses additional force-feedback to modulate the surgeon's movement and also provides visual cues that help in preventing or aborting potential adverse consequences.

CHAPTER III

OBJECTIVE

This research focuses on developing a PC/NT based real-time, virtual reality hatpic application. Our main goal is to resolve the critical real-time issues required to develop a stable haptic application. Some of the real-time issues necessary for stable haptic environment are: (1) interprocess communication, (2) synchronization of haptic and graphics interface and (3) finding application size limits for achieving high haptic rendering rates with stable haptic interactions. We are using a minimally invasive biopsy prototype as an example to show the viability of developing such a system.

A surgical simulation system requires the use of a multi-layered structure to represent the anatomical details of the human body. Use of a multi-layered structure, however, requires haptic interaction with the inner layers. In this situation, surface based model would have limitations. This research has also focused on overcoming this limitation to penetrate beyond the surface.

Penetration Beyond Surface

The surgical simulator that we use as an example simulates a breast biopsy system. To simulate a surface geometry such as simulating a tumor, the structure of the object needs to be multi layered, the structure having a surface and a sub-surface. Implementing this simulator with a surface based model (PHANToM) has limitations. The surface based model does not allow penetration beyond the surface. What this means is that geometric objects simulated with GHOST SDK are currently limited to geometries consisting of rigid surfaces. Thus, when the position of haptic device (gstPhantom node: representing the physical position of PHANToM haptic interface end-point) passes through the surface of an object, the shape of the object does not deform. Instead, a surface contact point (SCP) is maintained for each gstPhantom node; the SCP is forced to a point on the surface of any object intersecting with the gstPhantom node. Therefore, in surface based model the SCP never penetrates the surface of an object even if the position of the associated gstPhantom node does penetrate the logical boundaries of the surface. For example, consider the geometric model which has an outer surface and a subsurface representing a tumor. When the haptic device passes through these surfaces they are considered being in contact with the

haptic device. However, the force feedback is calculated relative to the outer surface. What we want for this simulator is to be able to touch the tumor independent of the outer surface and get force feedback depending on the tissue density of the tumor. To overcome this limitation, we have two options: use a layer-based model or implement this simulator with volumetric object representation.

Layer-Based Model

In the layer-based model, the structure is separated into multiple layers modeling the upper surface and sub-surface geometry. For example, muscle tissue on top of a bone could be modeled as the muscle surface and a separate bone surface with high stiffness. In this model, it is assumed that the layers are separated by certain distance. The system maintains a flag for each layer in the structure. When the haptic device is in contact with the outer layer, the system calculates the basic haptic force feedback depending on the surface based model (see Figure 3). The system also calculates the additional force feedback depending on the differential force feedback (DFFB) model (explained in later section). This differential force feedback is sent to the haptic device and the layer currently in contact with the haptic

device is haptically removed. At this point the system sets the associated flag for that layer to provide additional FFB until the haptic device comes in contact with the next layer. This is necessary for two reasons (1) once we remove the outer surface and if the haptic device is not in contact with any object there will be no force feedback. (2) The structure of object is represented in multi-layer surface with distance between each layer. This creates a gap between the two consecutive layers. To keep the additional force active for this gap until the haptic device is in contact with the next layer there has to be some mechanism for the haptic device to provide force feedback. By setting the flag, the additional force feedback is kept active. As soon as the haptic device makes contact with the next layer, the flag for the previous layer gets reset and the flag for the current layer gets set. This process is repeated for each layer until the target is reached, thus achieving penetration beyond surface. This model has the ability to remove the outer surface or layer and calculate the precise FFB for the next layer depending on the DFFB model. One of the major constraints with this model is that it requires separation of haptics and graphics processes.

Separation of graphics and haptics

A haptic interface is usually implemented with a visual display. When using pictures and force feedback devices, the synchronization of stimuli is important. In using a layer-based model, when we remove the surface haptically, it is necessary to remove the same surface graphically for the inner layers to be visible when interacting with haptic device. Otherwise we will have a poor visual/haptic registration which could result in inconsistent situations where the user feels reaction forces from the haptic device even though the object or surface is not visible (hidden under the outer surface). If haptics and graphics are not separated in using a layer-based model, haptically we can remove the outer surface but graphically it will look unrealistic. Separating haptics and graphics gives more flexibility in manipulating the visual representation depending on the haptic interaction.

So, for a realistic visualization, the system architecture is divided into two different processes: the haptic process and the graphic process. The haptic process is a typical Ghost application that uses Ghost libraries for the haptic scene and display of force feedback. The graphic process uses OpenGL for the graphic representation.

Need for Volumetric Object Representation

As compared to surface model, volumetric object representation can represent both surface and interior of the object, providing a better anatomical model. Burgin et al. [18,19] at Texas Tech university/computer science department have developed a computer model to implement (on 266 MHz PC/NT) haptic rendering of 3D volumetric objects (see Figure 5) using occupancy-map algorithm (OMA) for collision detection and a chainmail algorithm (CMA) for generation of the real-time force feedback while allowing deformation of the soft-bodied object to occur. Using these techniques, they have established a work that will allow for haptic representation of materials with non-homogenous internal behavior. The increased resolution of tissue representation improves the entire process, from better collision detection to more accurate differential force-feedback modeling to refine tissue deformation. The hardware requirements for this implementation using volumetric object representation make the technique memory intensive but allows for haptic presentation of materials with non-homogenous internal behavior. Although this would indicate a need for substantial memory requirement, the continuing decline in price of memory chips would offset the cost.

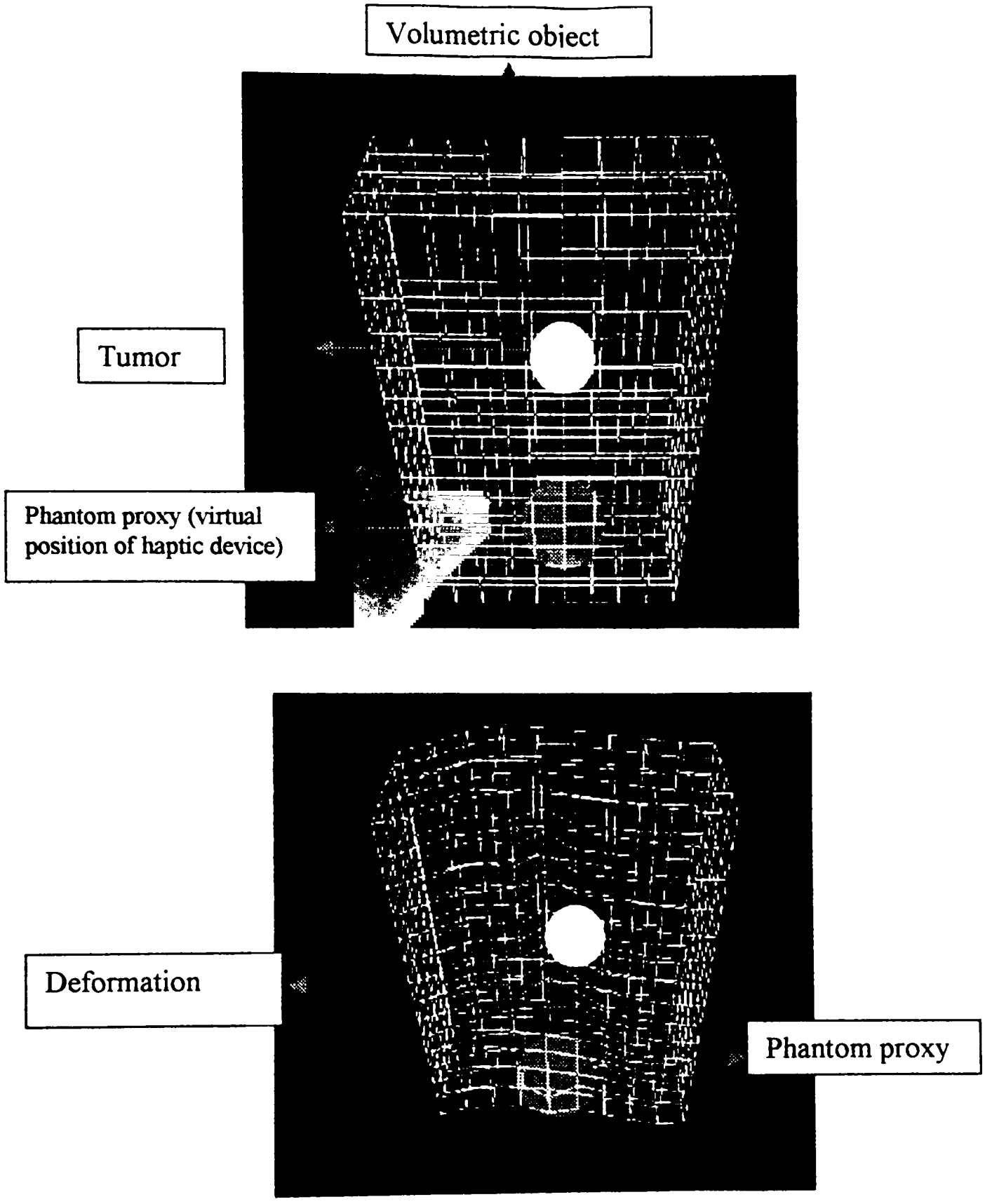

Figure 5. 3D-chainmail - deformation

CHAPTER IV

VIRTUAL SURGICAL TRAINER

WITH DIFFERENTIONAL FORCE FEEDBACK

Key Features

Any haptic application provides position information (PI) of the device in 3D space at any moment with 1kHz resolution. This knowledge of where we are touching makes it possible to develop the three important features for this surgical simulation: (1) monitor surgeon's movement, (2) haptically guide surgeon's movement by applying different forces for different scenarios, and (3) provide appropriate helpful visual cues.

The haptic process uses (1) tissue type providing stiffness of object and (2) the surgeon's position to do the basic haptic force feedback (BFFB) calculation (see Figure 3).

Monitoring task of the simulator keeps track of the surgeon's position (the position of the virtual instrument) and the surface in contact with the device (see Figure 6). The system then takes these parameters along with the forward-looking data such as proximity to a vital area (nerve, blood vessel, etc.), closeness to a tumor and tissue type (skin, bone, muscle and tumor)

and calculates an additional force feed back (AFFB) which is used for preventive structural damages.

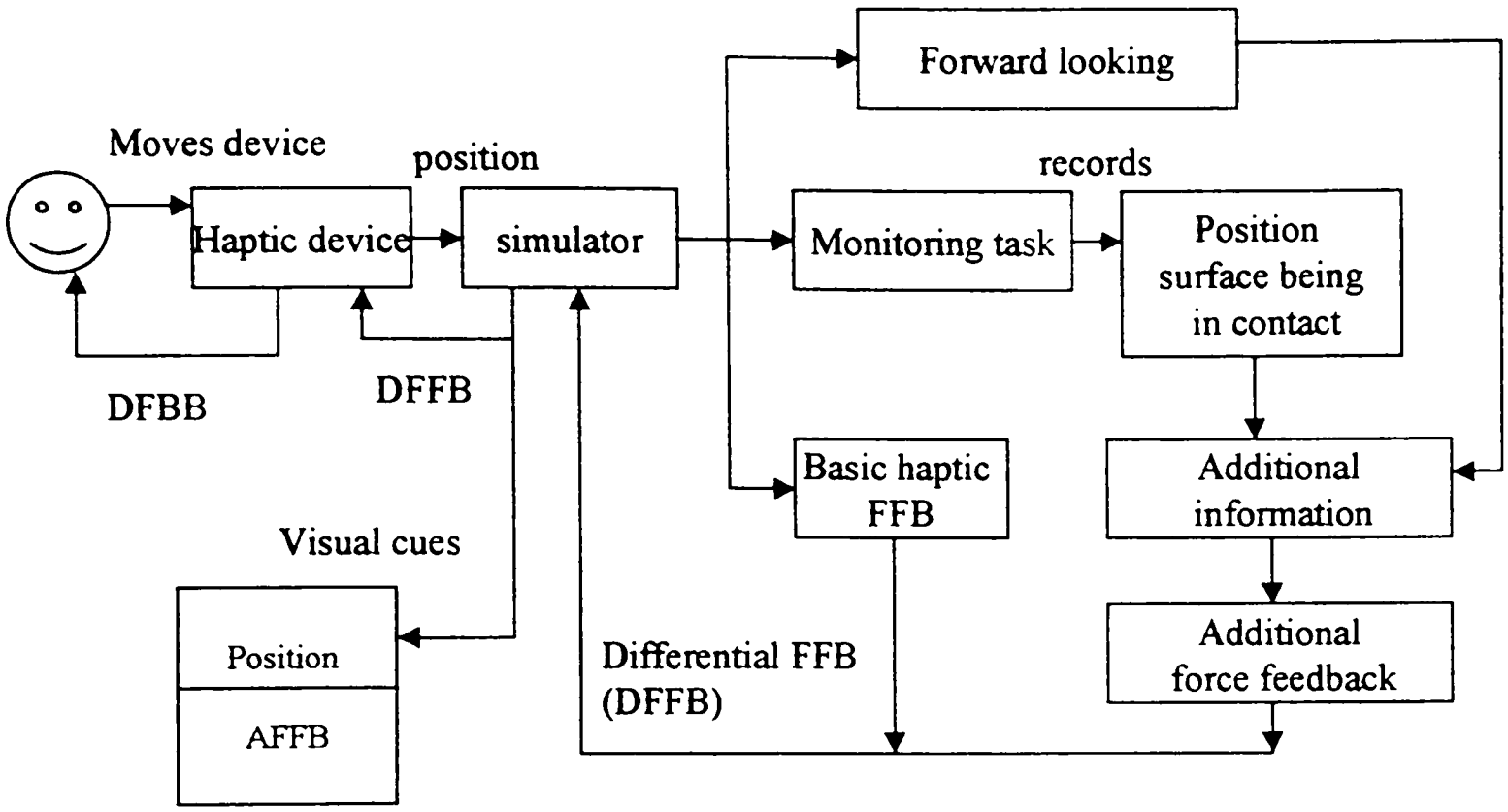

Figure 6. Monitoring and guiding process

The system guides the surgeon's movement by generating and activating the differential force feedback (DFFB) and by displaying appropriate visual cues. In this simulator, when the surgeon is touching an object, a surgeon feels a differential force feedback. This differential force feedback (see Figure 7) is a combination of a basic haptic force (object-

surface collision force, see Figure 3) and an "additional" force calculated by the simulator. The haptic force feedback depends on the tissue density and the depth of penetration of the virtual instrument. The additional force feedback generated by the simulator is based upon parameters provided by the monitoring task and the forward-looking data. Hence by generating and activating the AFFB, the system forewarns the surgeon, making it possible to abort those maneuvers that may lead to adverse results.

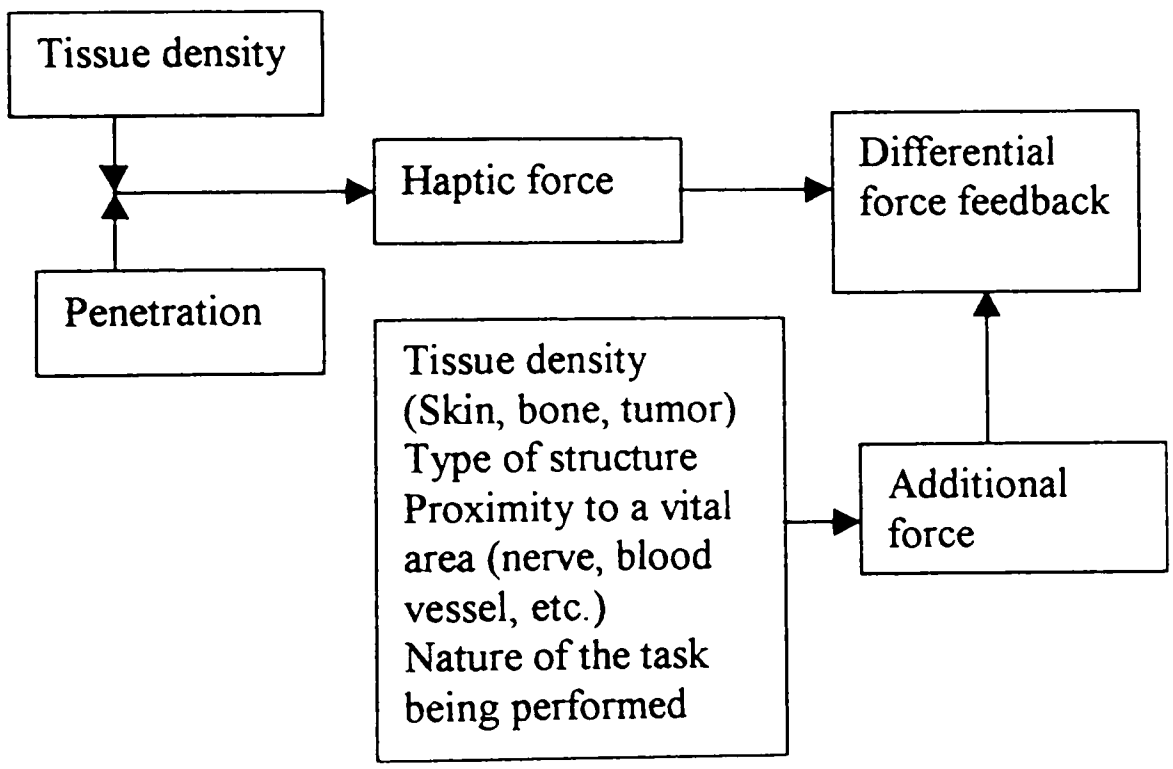

Figure 7. Simulators force feedback

Graphic display provides information related to the position of the virtual instrument, closeness to a vital area, contact/damage to a vital object and the current amount of the force feedback being felt. By providing visualization of the surgeon's movement, the system allows a surgeon to correct any potential contact/damage before it occurs. This corrective measure is reinforced by a haptic visual cue showing the amount of AFFB that is increased.

Thus, this differential force feedback mechanism of the haptic device and visual cues help in improving the accuracy of the surgical procedure by minimizing tissue damage, avoiding unnecessary exploration and wastage of time in reaching the target and aborting of potential adverse consequences.

Differential Force Feedback Model

As mentioned earlier the differential force feedback is a combination of basic haptic force feedback and additional force feedback. The following equation [21] describes the calculation of differential force feedback such that.

$$0 < F_{DFFB} = F_{BFFB} + F_{AFFB} <= 8.5N \quad (1)$$

Where F_{DFFB} is a differential force feedback, F_{BFFB} is a basic haptic force feedback and the F_{AFFB} is an additional force feedback.

From Figure 3, we know that

$$F_{BFFB} = K^* d \qquad \text{(spring compression model)} \qquad (2)$$

Where K is the stiffness matrix and d is the penetration vector.

The F_{AFFB} is a force depending on number of parameters and is described as follows:

$$F_{AFFB} = F_{\text{type of surgery}} + F_{\text{type of structure}} + F_{\text{type of tissue}} + F_{\text{proximity to a vital area}} + F_{\text{nature of task being performed}} \qquad (3)$$

Where user defines $F_{\text{type of surgery}}$, $F_{\text{type of structure}}$, $F_{\text{type of tissue}}$ and $F_{\text{type of task being performed}}$.

$$F_{\text{proximity to a vital area}} = (F_{\text{type of structure}} + F_{\text{type of tissue}}) / \text{distance to a vital area} \qquad (4)$$

distance to a vital area = the distance between the haptic device and the Nearest_vital_structure

Nearest_vital_structure is assigned a value from the lookup table. The lookup table (provided by the user) provides the sequential arrangement of the vital structures in the haptic scene. At the commencement of the program, the nearest_vital_structure is assigned the first value from lookup table. When contact is established with the first vital structure, the nearest_vital_structure gets assigned to the next value from lookup table. This process continues until the target is reached. After calculating $F_{proximity\ to\ a\ vital\ area}$, the value is used in equation 3.

Implementation of Differential Force Feedback

In this program, the additional force feedback is calculated using Ghost SDK gstEffect – gstInertiaEffect subclass. This class simulates inertia effect at the endpoint of a haptic device as if a mass were attached there using a spring/damper model. Additional force feedback based on spring/damper model is shown in equation below.

$$F_{AFFB} = M_{AFFB} \frac{d^2 (d_{AFFB})}{d t^2} = -K_{AFFB} * d_{AFFB} - C_{AFFB} \frac{d (d_{AFFB})}{d t} \quad (5)$$

Where M_{AFFB} = mass, d_{AFFB} = displacement, K_{AFFB} = spring stiffness, C_{AFFB} = damping coefficient and t = time.

By substituting the values of mass, spring stiffness and damping in equation 5 and solving for differential equation, additional force feedback is calculated. Hence, by specifying the force value in terms of mass, spring stiffness and damping coefficient for variables $F_{type\ of\ surgery}$, $F_{type\ of\ structure}$, $F_{type\ of\ tissue}$, $F_{proximity\ to\ a\ vital\ area}$, and $F_{nature\ of\ task\ being\ performed}$, the value of M_{AFFB}, K_{AFFB} and C_{AFFB} will be

$$M_{AFFB} = M_{type\ of\ surgery} + M_{type\ of\ structure} + M_{type\ of\ tissue} + M_{proximity\ to\ a\ vital\ area} + M_{nature\ of\ task\ being\ performed} \quad (6)$$

$$K_{AFFB} = K_{type\ of\ surgery} + K_{type\ of\ structure} + K_{type\ of\ tissue} + K_{proximity\ to\ a\ vital\ area} + K_{nature\ of\ task\ being\ performed} \quad (7)$$

$$C_{AFFB} = C_{type\ of\ surgery} + C_{type\ of\ structure} + C_{type\ of\ tissue} + C_{proximity\ to\ a\ vital\ area} + C_{nature\ of\ task\ being\ performed} \quad (8)$$

Substituting for M_{AFFB}, K_{AFFB} and C_{AFFB} in equation 5, F_{AFFB} is calculated at any instant. There is not any restriction put by Ghost on specifying values

for mass, damping and spring stiffness. However, we must remember that limitations (force can not be greater than 8.5N) are built into the hardware with respect to the maximum force which GHOST will permit. For example, if we specify excessive value for spring stiffness within gstInertiaEffect, the required forces would make the Ghost to exit with an error value –14 (GST_PHANTOM_FORCE_ERROR). The following sample code explains how the additional force feedback is activated.

inertiaEffect->setMass(M_{AFFB}); // set the mass value for additional FFB

inertiaEffect->setDamping(C_{AFFB}); // set the damping value for additional FFB

inertiaEffect->setSpringStiffness(K_{AFFB}); // set spring constant for additional FFB

PHANToM->setEffect(inertiaEffect); // set inertia effect

PHANToM ->startEffect(); // start the effect of additional FFB

CHAPTER V

IMPLEMENTATION

This section provides the hardware and software environment, explains the task analysis and the software architecture. It also discusses the real-time issues concerning the stability of the haptic system.

Environment

The hardware platform is a 300MHz Intel Pentium PC with 128 Mbytes of SDRAM. The operating system used for this work is Windows NT version 4.0 from Microsoft Corporation. The graphics are implemented using Silicon Graphics OpenGL API. The haptic interaction was provided by a PHANToM haptic interface manufactured by SensAble Technologies, Inc. The GHOST (General Haptics Open Software Toolkit) API was used to provide the IO to and from the PHANToM haptic interface.

Task Analysis

The software library (GHOST) associated with the haptic device uses a surface based model. As mentioned earlier, the problem with surface based

model is that it does not allow haptic interaction with inner structures. Thus, in order to interact with inner structures, we use a layer-based model that requires separation of haptic and graphic processes.

Task description

The haptic process has 9 tasks: H1 through H9. The graphic process has 3 tasks: G1 through G3. See Figure 8 for the task flow diagrams and Table 2 for task description.

Table 1: Task description

Tasks	Task description
H1	Application-specific task in haptic process
H2	Get device position
H3	Detect collision
H4	Calculate force feedback
H5	Send forces to haptic device
H6	Monitoring/forward looking task
H7	Calculate additional force feedback (AFFB)
H8	Display warning message
H9	Display haptic scene graphically
G1	Application-specific task in graphic process
G2	Read surface being in contact and position from haptic process
G3	Display graphics in graphic process window

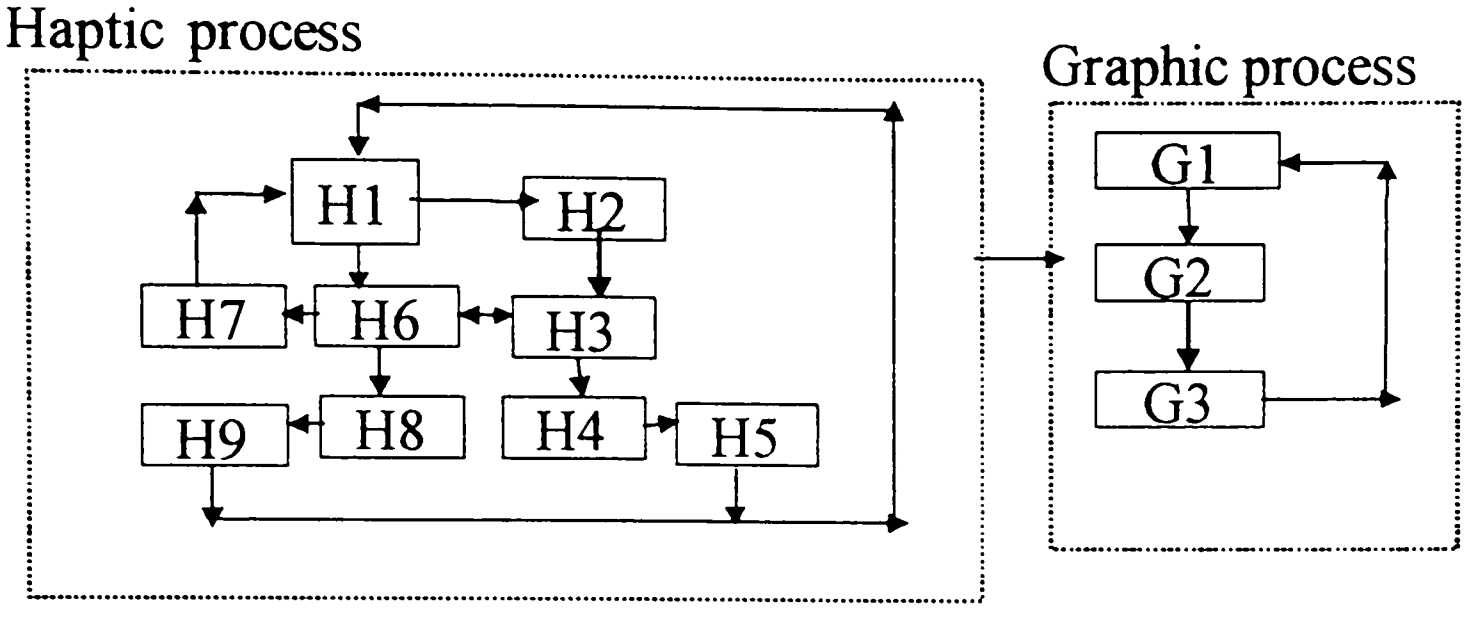

Figure 8. Task flow diagram

Task H1 performs the application specific functions that include setting up the communication process, starting monitoring/forward looking task, initializing the graphic and the haptic state, and terminating the application. Initializing the graphic state involves creating the graphic display window for the haptic process and allowing for later OpenGL calls to be made to render the 3D environment. The haptic initialization creates the haptic scene and initializes the haptic hardware.

Haptic servo loop task is made up of tasks H2, H3, H4 and H5. Task H2 gets its input from the haptic device and sends it to task H3. H3 provides

the status of collision detection. Depending on the stiffness of an object and the penetration vector, H4 calculates appropriate forces and transmits it to H5. Task H5 forwards the forces to the haptic device. The haptic servo loop task handles all communications with the haptic interface. This task is owned by the GHOST library and is repeated at 1kHz update rate. This real-time task has a highest priority over all other tasks. This ensures that the haptic interaction is not compromised.

The monitoring part of task H6 continuously checks for the contact between the virtual instrument and the virtual surface and gets its input from task H3. The forward-looking part of task H6 continuously checks for the proximity to next virtual surface and/or vital area. The task H6 provides input for H7 and H8. Task H7 calculates AFFB. Similarly task H8 displays the amount of AFFB. For the system to run in real-time, task H7 and H8 must finish its AFFB calculation and display a warning before the next servo loop starts. Depending on the position of the virtual instrument, task H6 sends the message to be displayed to task H8. The display graphic task (H9) is simply in charge of rendering the virtual environment (visual scene containing the surface and the tool) for the user to see. This task runs in a loop redrawing the virtual environment with each iteration. The graphic

display task waits for a signal at the beginning of each iteration before redrawing the screen. The haptic servo loop running at 1kHz signals the display graphic task every 33 iterations to help the graphic task maintain a steady 30Hz update rate. This task gets its input from H1, H3 and H8. This task updates the graphic display and sends information (the position of haptic device and the SCP) to graphic process. This task must update at 30Hz.

Task G1 performs the application specific graphic functions that include setting up the process to connect to haptic process, initializing graphic state and terminating graphic process. Initializing the graphic state involves creating the graphic display window for the graphic process and allowing for later OpenGL calls to be made to render the 3D environment.

Task G2 gets the phantom's position information and the surface contact point from the haptic process and passes it to G3. Task G2 must receive the phantom's position before it gets updated graphically in the haptic process. Task G3 displays the phantom's position graphically. At this stage, graphic displays in both processes must be synchronized for real-time operation.

Real Time Requirements

For this system to work in real time, the following requirements must be satisfied.

1. The haptic servo loop task must update at 1kHz.
2. Forward-looking, monitoring and differential force feedback tasks must finish their calculation before the next servo loop starts (before 1 millisecond is finished).
3. The graphic must update at 30Hz in the haptic process.
4. There has to be some mechanism, which allows the haptic interaction with inner layers since the haptic device is a surface-based model and does not allow penetration beyond the surface.
5. The communication between processes should be in real time.
6. The graphic display must update simultaneously in both haptic and graphic processes.
7. The graphic process must receive position (virtual information) information before it graphically gets updated in the haptic process.

Task Timing Model

As this research focuses on resolving critical real-time issues required to develop a stable haptic application, we need to study the processes that make the haptic application work in real-time. A haptic interface is usually implemented with visual display. When using pictures and force feedback, synchronization of stimuli is important. For a haptic application to be stable, haptic must update at 1kHz and graphic must update at 30Hz. In this application haptic stability depends on 3 tasks: (1) haptic servo loop task (H2,H3,H4,H5), (2) monitoring /forward looking task (H6) and (3) additional force feedback task (H7). The visual representation depends on graphic display task (H9). In this section we will study the process time requirement for each task for the given number of objects. A task-timing model will be created from this analysis.

To measure the process time for each task that influences the haptic stability, "Hload" program supplied by Ghost SDK is used. This program monitors the load placed by haptic processes on the computer system. The haptic servo loop runs at 1kHz, which means all tasks involved with haptic communication must finish their work within 1ms. By monitoring the bar

graph of hload program, we can find out what percent of the 1 millisecond is used by each task for a given number of objects.

Haptic servo loop task

The haptic servo loop performs the following tasks:

1. Setting the device position (H2).
2. Detecting collision (H3).
3. Calculating force feedback (H4).
4. Sending force feedback to the haptic device (H5).

To measure the process time for each task in the servo loop the haptic application is run for different number of objects and hload reading is recorded (see Table 2) for the events described in Table 3. Figures 9-12 depict the plot of number of objects vs. the hload reading for events A1, A2, A3 and A4 as mentioned in Table 2. As we can see from the graphs that process time taken by the haptic servo loop depends linearly on the number of objects. From the best fit of the data, we determine the slope and the y-intercept for each event using "Origin" program. These values are included in Table 4.

Table 2: Experimental data for tasks

G-on : Graphics on
G-off : Graphics off
A5 = 0.9 (0.1ms), A6 = 0.8 (0.1ms)

No. of objects	Touching				Not touching			
	G-on		G-off		G-on		G-off	
	A1 0.1ms	A7 0.1ms	A2 0.1ms	A8 0.1ms	A3 0.1ms	A9 0.1ms	A4 0.1ms	A10 0.1ms
25	1.6	2.1	1.5	2.0	1.2	1.4	1.05	1.3
50	1.9	2.3	1.8	2.2	1.4	1.6	1.3	1.5
75	2.125	2.6	2.025	2.5	1.7	1.9	1.6	1.8
100	2.4	2.9	2.3	2.8	1.9	2.1	1.8	2.0
150	2.9	3.4	2.8	3.3	2.4	2.6	2.3	2.5
200	3.35	3.9	3.25	3.8	2.85	3.2	2.8	3.1
250	3.8	4.4	3.7	4.3	3.3	3.6	3.2	3.5
300	4.25	4.8	4	4.7	3.8	4.1	3.7	3.9
350	4.8	5.2	4.6	5.1	4.3	4.6	4.2	4.5
400	5.35	5.8	5.15	5.7	4.75	5.1	4.7	5.0
450	5.9	6.2	5.65	6.1	5.4	5.6	5.2	5.4
500	6.55	6.9	6.3	6.6	6	6.0	5.75	5.9
550	6.9	7.3	6.675	7.1	6.45	6.5	6.25	6.3
600	7.4	7.7	7.2		6.9	6.9	6.7	6.8

Table 3: Events

Events (H=Haptic, G= graphics, T= touching, NT = not touching, AF=AFFB)	Haptic/graphic active tasks (*=active, - = not active)						
	H2	H3	H4	H5	H8,H9	H6	H7
A1: H-on-G-on-T	*	*	*	*	*	-	-
A2: H-on-G-off-T	*	*	*	*	-	-	-
A3: H-on-G-on-NT	*	*	-	-	*	-	-
A4: H-on-G-off-NT	*	*	-	-	-	-	-
A5: H-off-G-on	*	-	-	-	*	-	-
A6: H-off-G-off	*	-	-	-	-	-	-
A7: H-on-G-on-T-AF-on	*	*	*	*	*	*	*
A8: H-on-G-off-T-AF-on	*	*	*	*	-	*	*
A9: H-on-G-on-NT-AF-on	*	*	-	-	*	-	*
A10: H-on-G-off-NT-AF-on	*	*	-	-	-	-	*
A11: H-off-G-on-AF-on	*	-	-	-	*	-	*
A12: H-off-G-off-AF-on	*	-	-	-	-	-	*
A13: H-off-G-on-AF-off	*	-	-	-	*	-	-
A14: H-off-G-off-AF-off	*	-	-	-	-	-	-

Table 4: Linear fit data

Events	Slope = m		y-intercept = c	
	Value (* 0.1ms)	Error (* 0.1ms)	Value (* 0.1ms)	Error (* 0.1ms)
A1	0.01009	1.00431E-4	1.34746	0.03428
A2	0.0098	1.20632E-4	1.2676	0.04118
A3	0.00999	1.07684E-4	0.88569	0.03676
A4	0.0098	5.48663E-5	0.81092	0.01873
A7	0.00978	8.96409E-5	1.86607	0.03174
A8	0.0096	8.03373E-5	1.78949	0.0244
A9	0.00972	4.64103E-5	1.15241	0.01409
A10	0.00959	6.47935E-5	1.06219	0.01968

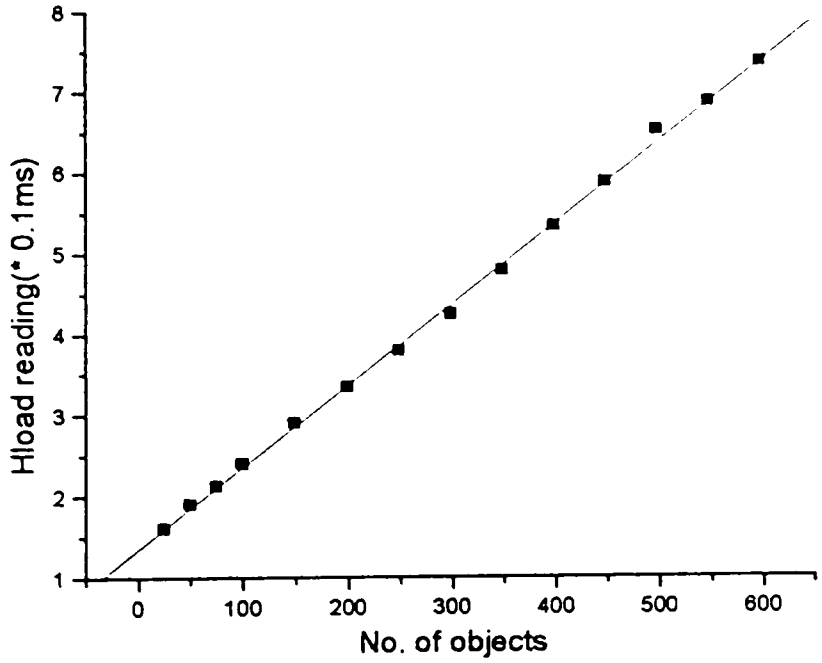

Figure 9. Graph for event A1

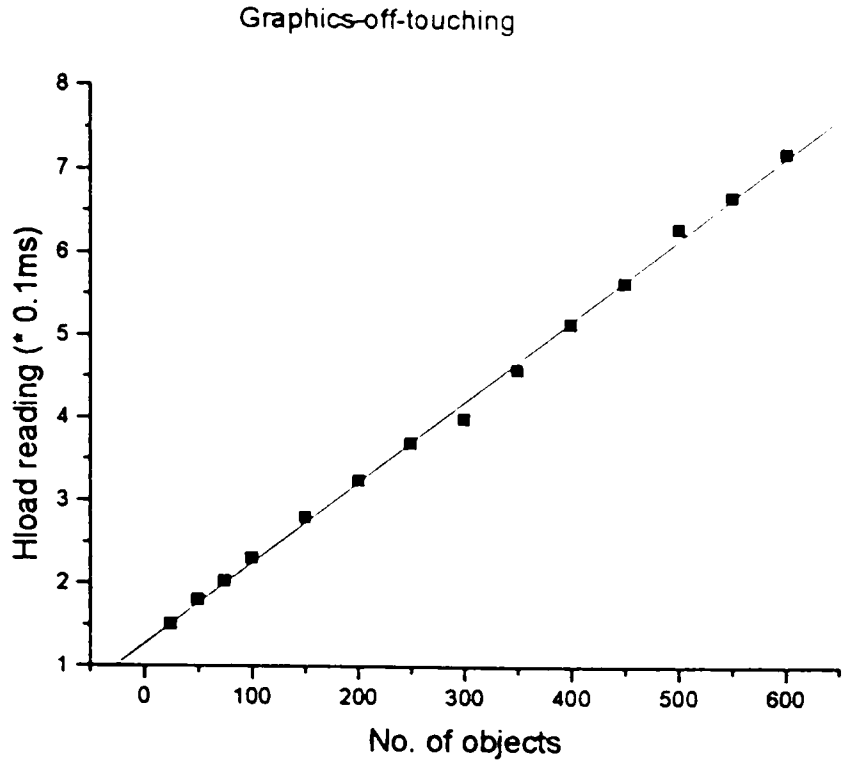

Figure 10. Graph for event A2

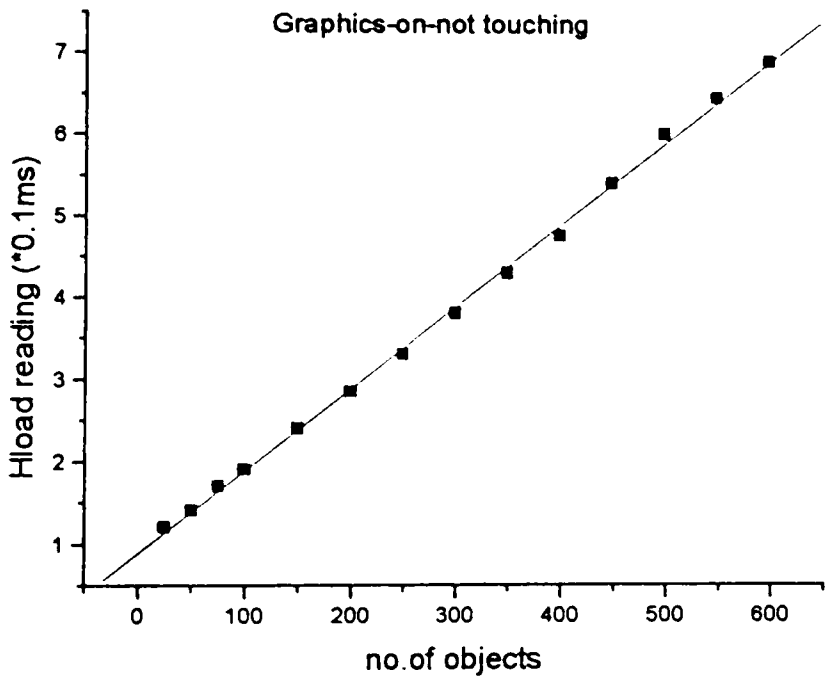

Figure 11. Graph for event A3

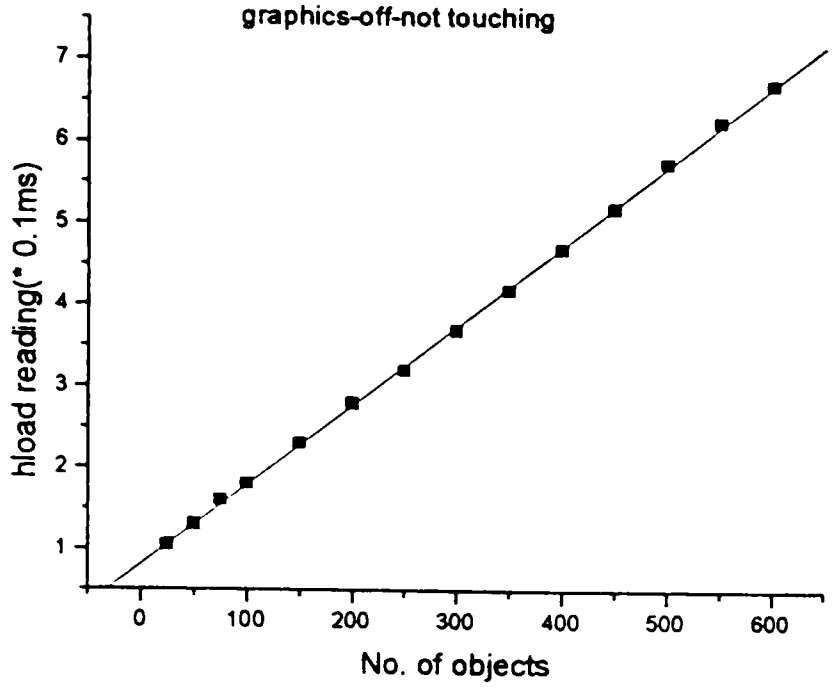

Figure 12. Graph for event A4

Process time for graphics (Y_G)

Since process time for graphic task will be needed for the calculation of collision detection process time, we will present this analysis first. As we can see from Tables 2 and 3, the process time for graphic task for x number of objects can be obtained by subtracting hload reading for A2 from hload reading for A1. The equation representing the hload reading for events A1 and A2, respectively, are

$$y_1 = m_1 x + c_1 \quad (9)$$

$$y_2 = m_2 x + c_2 \quad (10)$$

Subtracting equation (10) from equation (9) we obtain

$$y_1 - y_2 = (m_1 - m_2)\, x + (c_1 - c_2)$$

or, $$Y_G = m_G\, x + c_G \qquad (11)$$

Where $Y_G = y_1 - y_2$, $m_G = m_1 - m_2$ and $c_G = c_1 - c_2$

Figure 13 shows the plot of number of objects versus the Y_G calculated from equation (11). As the number of object increases, the process time for graphic task increases. We have also obtained the graphic process time experimentally by subtracting the hload reading for A2 from hload reading for A1 and hload reading for A4 from hload reading for A3 for a number of objects. This is presented in Table 5. The graphic process time Y_G calculated using equation (11) is also presented in Table 5. The rate at which the graphic process time changes with increase in number of objects is clearly seen when it is calculated using equation (11). The trend is not clear for experimental data because the change is so small that it can not be measured precisely.

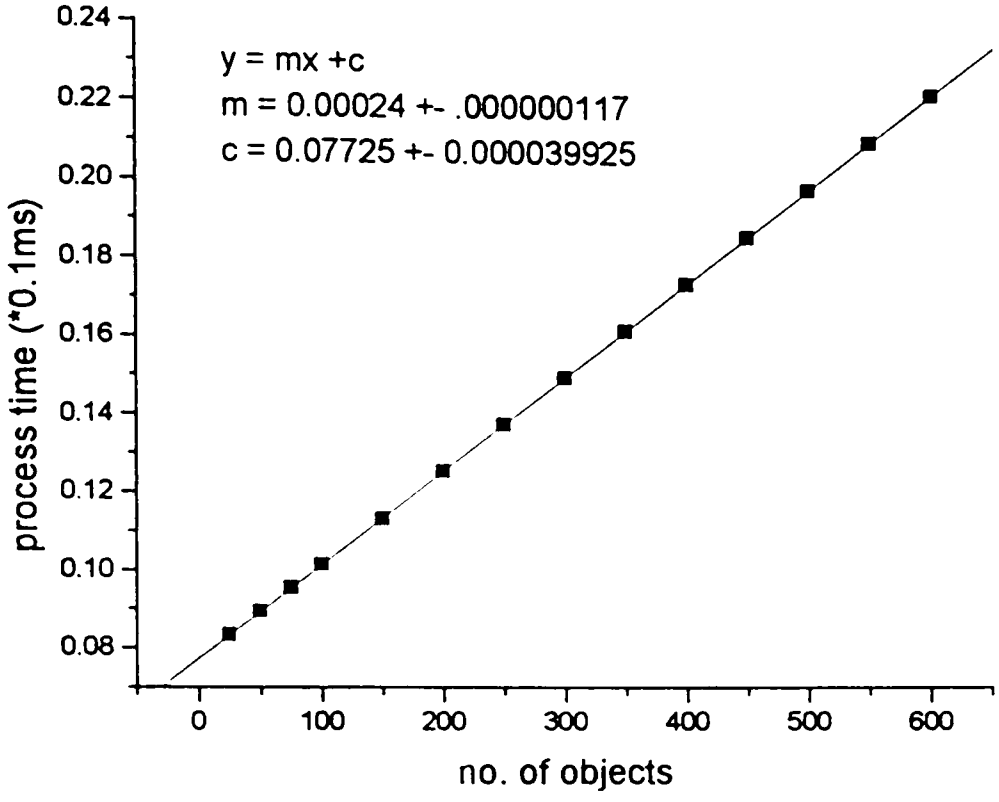

Figure 13. Graphic process time

Table 5: Graphic process time

No. of objects	A1-A2 (*0.1ms)	A3-A4 (*0.1ms)	Calculated Y_G (*0.1ms)
25	0.1	0.15	0.083
50	0.1	0.1	0.089
75	0.1	0.1	0.095
100	0.1	0.1	0.101
150	0.1	0.1	0.113
200	0.1	0.05	0.125
250	0.1	0.1	0.137
300	0.25	0.1	0.149
350	0.2	0.1	0.161
400	0.2	0.05	0.173
450	0.25	0.2	0.185
500	0.25	0.25	0.197
550	0.225	0.2	0.209
600	0.2	0.2	0.221

Process time for setting position task (Y_P)

As shown in Table 3, only setting position task is active for event A6. Hence hload reading for A6 in table 2 is the process time for setting position task. Figure 14 shows the plot of number object versus the hload reading for A6. The graph shows that the process time for setting the position task does not depend on the number of objects ($Y_P = 0.08$ ms).

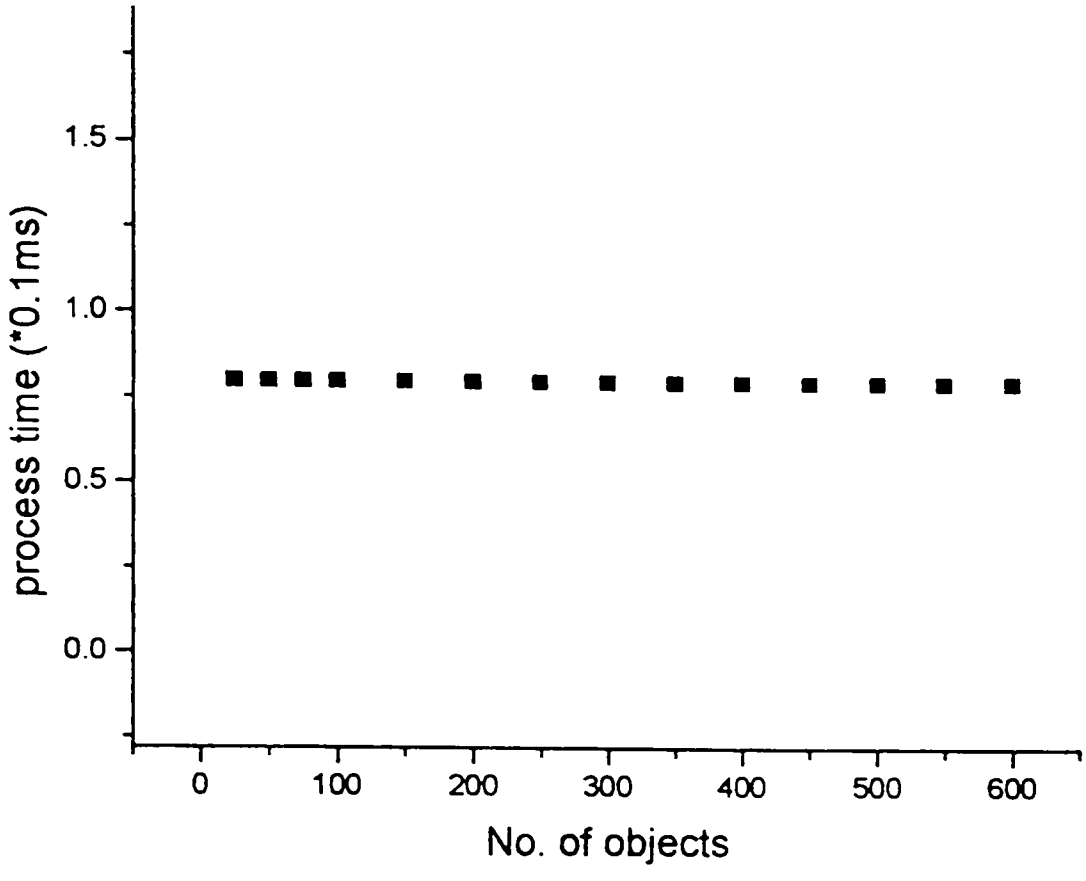

Figure 14. Process time for setting the device position

Process time for collision detection (Y_{CD})

The process time for collision detection for x number of objects can be obtained using following equations

$$Y_{CD} = A4 - A6 \quad (12)$$

$$Y_{CD} = A3 - A5 \quad (13)$$

$$Y_{CD} = A3 - Y_P - Y_G \quad (14)$$

Table 6 represents the experimental value of process time for collision detection calculated by using equations 12,13 and 14. Figure 15 shows the plot of number of objects vs. the process time for collision detection calculated by using equations 12,13 and 14. The graph shows that the process time increases as the number of object increases.

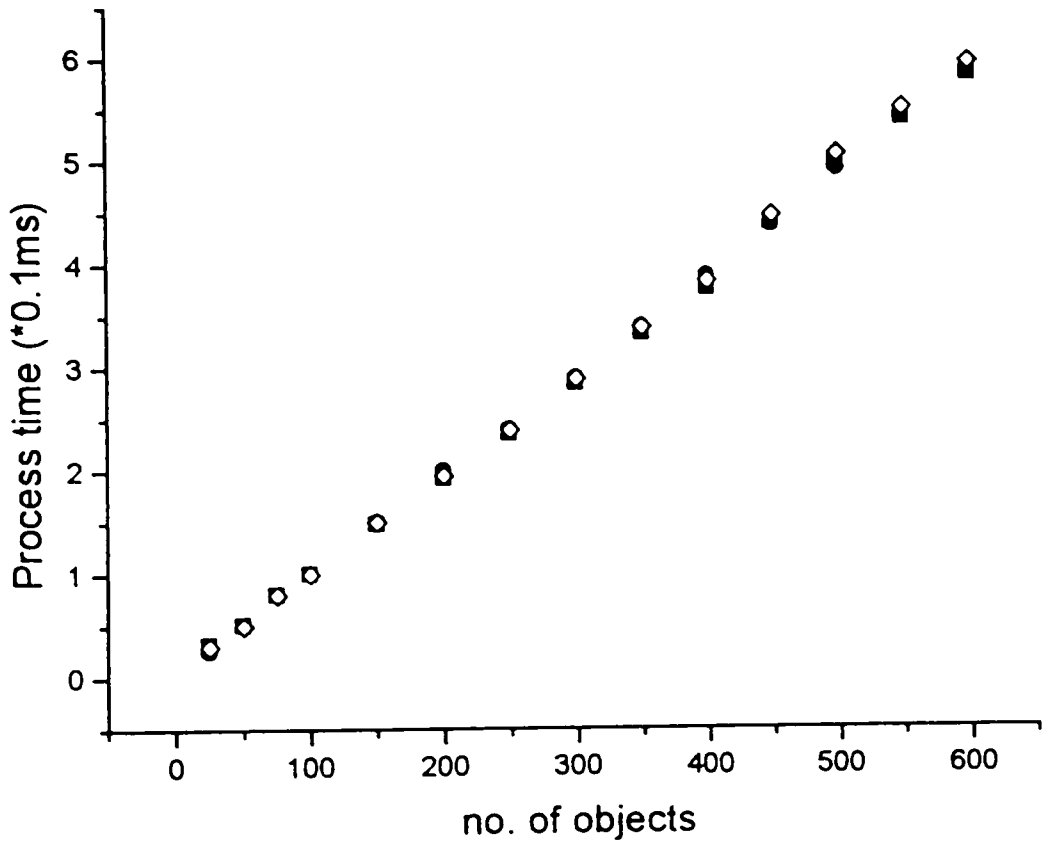

Figure 15. Process time for collision detection

Table 6: Collision detection process time

No. of objects	A4-A6 (*0.1ms)	A3-A5 (*0.1ms)	Calculated Y_{CD} (*0.1ms)
25	0.25	0.3	0.317
50	0.5	0.5	0.511
75	0.8	0.8	0.805
100	1	1	0.999
150	1.5	1.5	1.487
200	2	1.95	1.925
250	2.4	2.4	2.363
300	2.9	2.9	2.851
350	3.4	3.4	3.339
400	3.9	3.8	3.777
450	4.4	4.5	4.415
500	4.95	5.1	5.003
550	5.45	5.55	5.441
600	5.9	6	5.879

Process time for force feedback (H4 + H5) (Y_F)

The process time for H4 and H5 can be calculated by subtracting hload reading for A4 from hload reading for A2. The equation representing hload reading for A2 and A4 events are

$$y_2 = m_2 x + c_2 \qquad (15)$$

$$y_4 = m_4 x + c_4 \qquad (16)$$

Subtracting equation (16) from equation (15) we obtain

$$y_2 - y_4 = (m_2 - m_4)\,x + (c_2 - c_4)$$

or, $$Y_F = m_F\,x + c_F \qquad (17)$$

Where $Y_F = y_2 - y_4$, $m_F = m_2 - m_4$ and $c_F = c_2 - c_4$

Figure 16 shows the plot of number of object vs. the Y_F calculated using equation (17). As the number of object increases, the total process time for H4 and H5 remains constant ($Y_F = 0.0459$ ms).

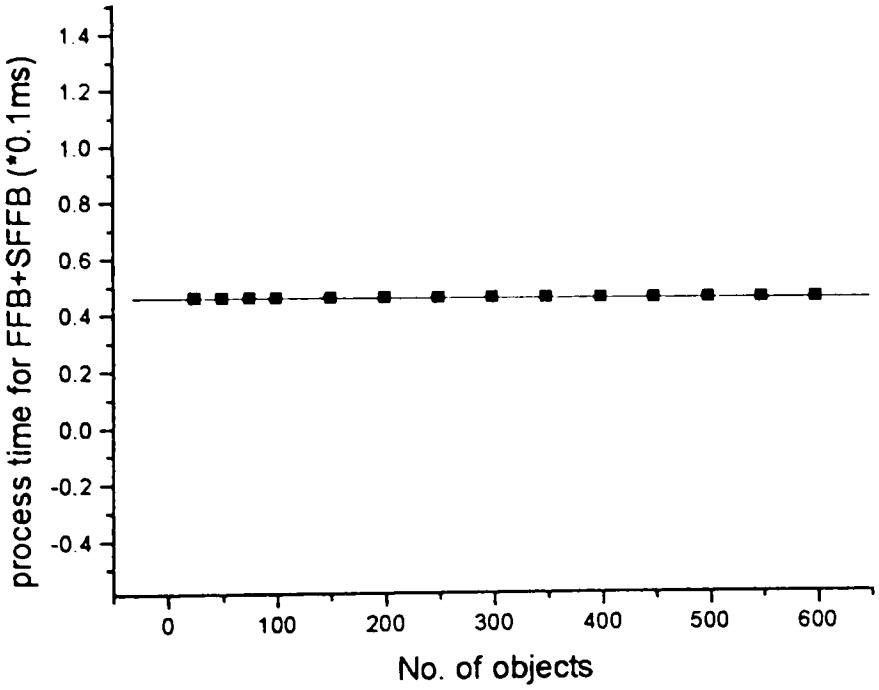

Figure 16. Process time for force feedback

Discussion

In this section we will compare time requirement for each task. Figure 17 shows the plot for each servo loop tasks (H2, H3, H4 and H5) and graphic process time (H9) as a function of the number of objects. As shown in the graph, the process time requirements for task H2, H4, H5 and H9 is smaller compared to the process time for collision detection (H3). The total process time for servo loop tasks can be presented by the following equation:

$$Y_{SV} = Y_P + Y_{CD} + Y_F$$

Or, $Y_{SV} = 0.8 + (0.00975)\,x + 0.459$

$$Y_{SV} = 1.259 + 0.00975\,x. \qquad (18)$$

The total process time including graphics can be obtained by adding Y_G to equation (18)

$$Y_{SV+G} = Y_{SV} + Y_G$$

Or, $Y_{SV+G} = 1.259 + 0.00975x + 0.00024x + 0.077$

$$Y_{SV+G} = (0.00999x + 1.336) * (0.1\text{ms}). \qquad (19)$$

Figure 17 also shows the plot of total (calculated Y_{SV+G} and experimental: A1) process time as a function of number of objects. The graphs indicate the

good overlap between the two readings and verify the validity of the task-timing model.

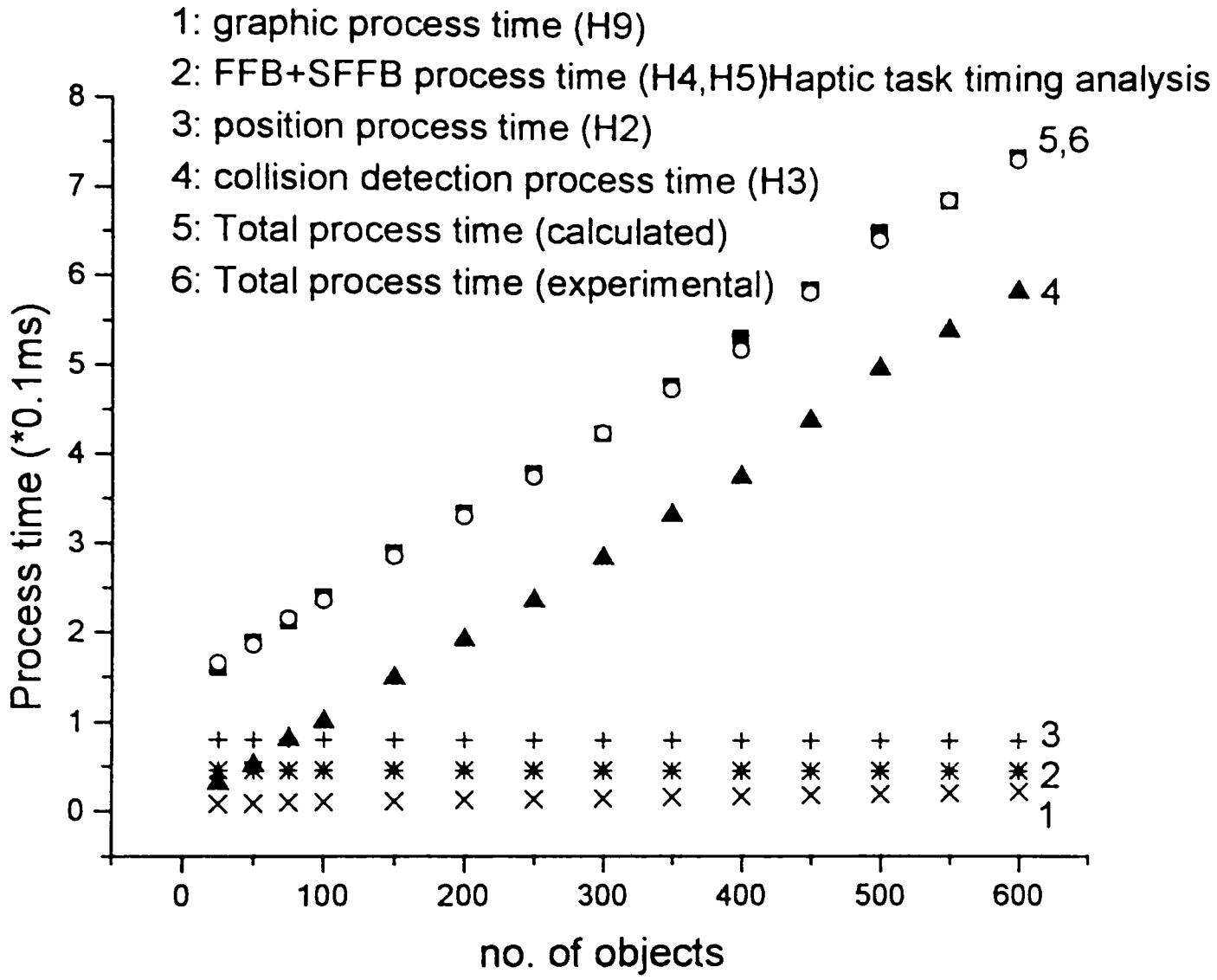

Figure 17. Haptic task timing analysis

Monitoring/Forward Looking Task (Y_{MT})

The process time for monitoring task can be measured by using following equations

$$y_7 = m_7 x + c_7 \qquad (20)$$

$$y_9 = m_9 x + c_9. \quad (21)$$

Subtracting equation (21) from equation (20), we obtain

$$y_7 - y_9 = (m_7 - m_9) x + (c_7 - c_9) \quad (22)$$

or, $$Y_{MT+F} = m_{MT+F} x + c_{MT+F} \quad (23)$$

Where $Y_{MT+F} = y_1 - y_7$, $m_{MT+F} = m_1 - m_7$ and $c_{MT+F} = c_1 - c_7$.

Equation (21) gives the process time for monitoring task and force feedback task (Y_F). Since we already calculated Y_F in previous section (see equation (17)), subtracting its value from equation (20) will give the process time for monitoring task.

Subtracting Y_F from equation (21), we get

$$Y_{MT} = Y_{MT+F} - Y_F. \quad (24)$$

In this project, we are using layer-based model where haptic device interacts with one layer at a time. This gives the flexibility of working with fewer numbers of objects in the haptic scene at any instant. In this project the number of objects are constrained to 4 at any instant. The process time for monitoring task in this application is 0.026ms. Generally, the process time for monitoring task will depend on the nature of the task and the number of objects constraint in the haptic scene at any instants.

Process time for AFFB task (Y_{AF})

The process time for AFFB is calculated by subtracting hload reading for A1 and Y_{MT} from hload reading for A7. The hload reading for event A1 and A7, respectively, are

$$y_1 = m_1 x + c_1 \qquad (25)$$

$$y_7 = m_7 x + c_7 \qquad (26)$$

Subtracting equation 26 from equation 25 we obtain

$$y_1 - y_7 = (m_1 - m_7) x + (c_1 - c_7)$$

or, $$Y_{MT+AF} = m_{MT+AF} x + c_{MT+AF} \qquad (27)$$

Where $Y_{MT+AF} = y_1 - y_7$, $m_{MT+AF} = m_1 - m_7$ and $c_{MT+AF} = c_1 - c_7$

Subtracting Y_{MT} from equation 27 we get

$$Y_{AF} = Y_{MT+AF} - Y_{MT} \qquad (28)$$

Y_{AF} is calculated using equation (28). We found that as the number of object increases, the process time for additional force feedback remains constant ($Y_{AF} = 0.026$ms).

Final Model

The following model can determine the final equation for the total process time, which includes haptic/graphic critical tasks

$$Y_T = Y_{SV+G} + Y_{MT} + Y_{AF}$$

$$Y_T = (0.00999x + 1.336) + Y_{MT} + 0.26$$

$$Y_T = 0.00999x + 1.336 + m_{MT}x + 0.26$$

$$Y_T = 0.00999x + m_{MT}x + 1.596 \qquad (29)$$

Where Y_T depends on the number of objects and the nature of the monitoring task. In this research, the value of m_{MT} is equal to (0.065 * 0.1ms).

Options for Implementation

Depending on the haptic system (real-time) requirements the interprocess communication can be achieved using either file mapping or via pipes. In fact, because windows NT (the operating system for this project) does not give processes direct access to each other's data, pipes and file mapping are the only data-sharing mechanism that are available to windows NT processes [20].

File mapping

File mapping [20] allows multiple processes to map virtual pages to the same physical memory so that the processes can share data. The shared memory is backed up either by a user-specified file or by a system swap file a special file maintained by the system to ensure file coherence during file-mapping operations.

The most common use for memory-mapped file is to access data stored on disk. It can also be used for loading and executing applications and DLLs and it can also be used for sharing blocks of data in memory between multiple processes. Memory-mapped files should not be used to share writeable files over a network, because Windows NT cannot guarantee coherent views of data. File mapping also does not guarantee the orders of read and write operation.

Pipes

Pipes [20] are another mechanism that Windows NT processes can use to share data. Pipes are easy to use because they work much like files. When an application has set up a pipe, processes send information through the pipe by writing to it in the same way they would write to a file.

Similarly, processes retrieve information from the pipe by reading from it in the same way they would read from a file. The sending and receiving information from processes can run on the same machine or on different computer systems.

Pipes can be divided in to two categories: one-way pipes and two-way pipes. A one-way pipe has one end that is read-only and one end that is write-only. In contrast, both ends of a two-way pipe can be either read from or written to. Pipes can also be categorized as either anonymous pipes or named pipes.

Anonymous pipes

To access an anonymous pipe, a process needs to use a process handle. Because of complexities in managing handles, anonymous pipes are not as useful for transferring data between unrelated processes, and they cannot be used for communication over a network [20].

Named pipes

To access a named pipe, a process must provide the name of the pipe to the process that created the pipe. Once the named pipe is created, it can be

accessed by any process running on the same computer or by a process running on any computer that is connected over a network. Only named pipes support asynchronous or overlapped operations. By overlapping pipe transmission, an application can speed processing through named pipes by allowing background processing during transmission of data. An overlapped structure is also useful with named pipes whenever an application expects to perform simultaneous read and writes [20].

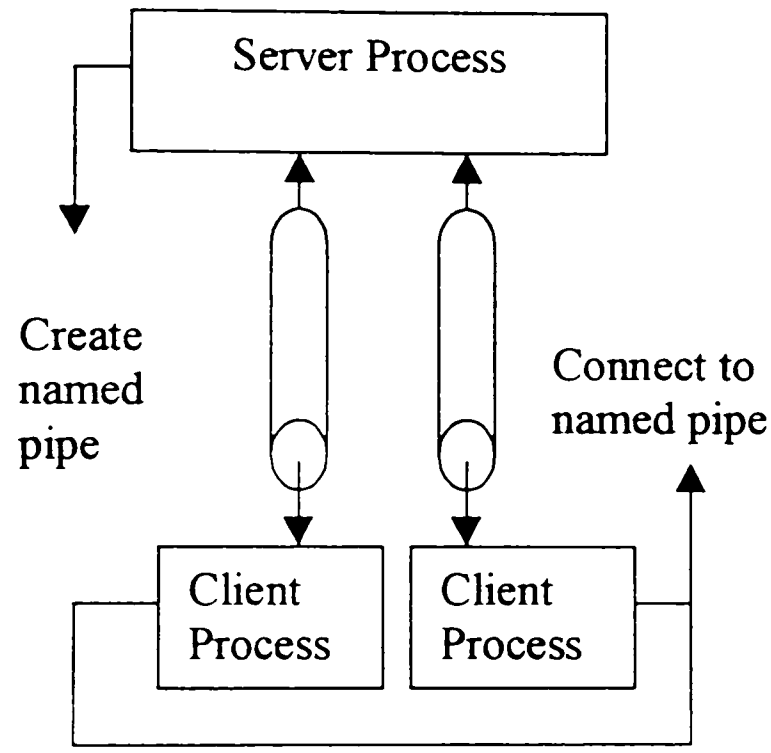

Figure 18. Local communication through named pipes

Choice of Options

In this project, the haptic servo loop must update at 1kHz and graphics must update at 30Hz. Because of the update rate and the simultaneous

reading and writing information between processes, named pipes are used for interprocess communication. The asynchronous I/O operation of named pipe helps with the update rate. When functions run asynchronously--using overlapped I/O-- each function that is called can return immediately, even before its operations are finished. This means that time-consuming operations can be executed in the background while the calling thread is free to perform other tasks. Overlapped operations make it possible for a single process to read and write data to and from other process simultaneously.

System Architecture

Implementing this simulator with a surface based model (PHANToM) has limitations. As mentioned earlier, the surface based model does not allow penetration beyond the surface. In order to interact with inner structure and view region beyond the outer surface, this simulator system is developed using two different techniques: (1) for haptic interaction with the inner layers we have developed a layer based model where we divided the surface in to separate layers. This model has the ability to remove the outer surface and calculate the precise FFB for the next layer depending on the tissue density for that layer. (2) For a realistic display of graphical representation

of the simulation system, the system architecture is divided into two different processes: the haptic process and the graphic process (see Figure 19). The haptic process is a typical Ghost application that uses Ghost libraries for the haptic scene and displays force feedback. The graphic process uses OpenGL for the graphic representation. It receives the position of phantom stylus from haptic process and displays the proxy (graphical representation of phantom stylus) in its own window creating an illusion of phantom interacting with multiple scenes in two separate windows. Graphic process also generates the warning cues about the amount of force generated. Many of the Ghost library Callback functions are used in finding the position of phantom and in generating different force feedback. The interprocess communication between haptic process and graphic process is achieved using named pipes.

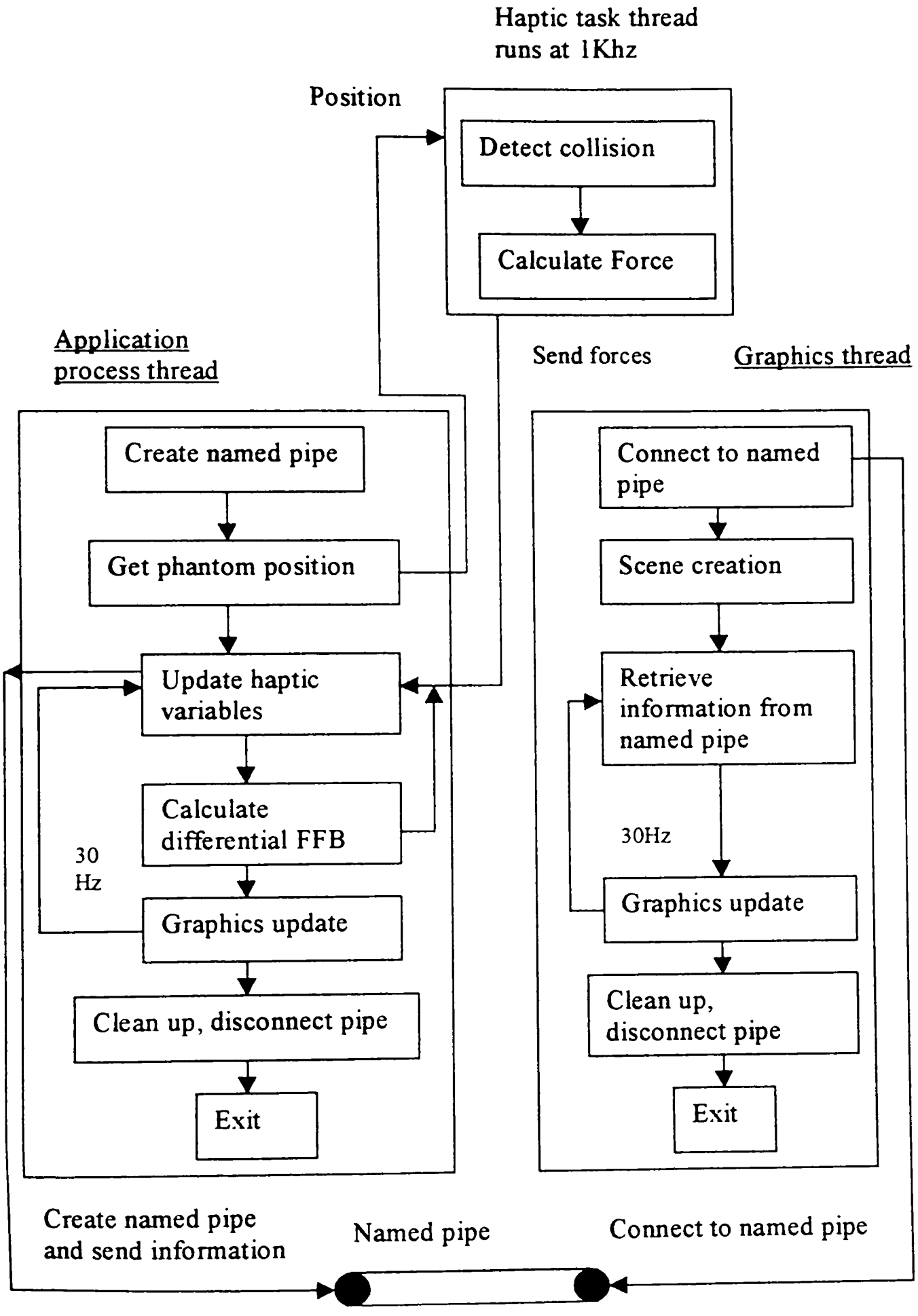

Figure 19. System architecture

CHAPTER VI

REAL-TIME ISSUES

The major challenges in real-time performance are:

1. The communication mechanism to be used between the tasks and processes.
2. Specifying process time required to make the haptic and graphic updates synchronized and stable (creating a stable haptic servo loop).
3. Finding application size limits for achieving high haptic rendering rates with stable haptic interactions.

Communication Mechanism

As mentioned earlier, our system records surgeon's position. This information is sent from haptic process to graphic process to be displayed. If the system does not work in real-time, there will be discontinuity between the surgeon's movement and the visual representation. Since our system runs on a Pentium II-300MHz under Windows NT 4.0, named pipe is used for interprocess communication (see under choice of option). The support of asynchronous I/O operation using overlapped I/O allows the time consuming

operations such as reading and writing information between processes to be executed in the background while the calling thread is free to perform other tasks such as updating graphics. This indeed helps maintain the graphic update. Furthermore, the information is transferred in real-time between processes, which we can observe in experiment 1 in testing section. The following Figures 11 and 12 explain how using these parameters help in optimizing the system. In Figure 11, which uses the optimization parameters, we can see that the graphic is updated within the time limit of 30ms to keep up with the 30Hz of graphic update rate. Compare this with Figure 12, which does not use optimization parameters. Here the graphic update takes longer than the 30ms, which may create visual glitches making the haptic environment, look unrealistic.

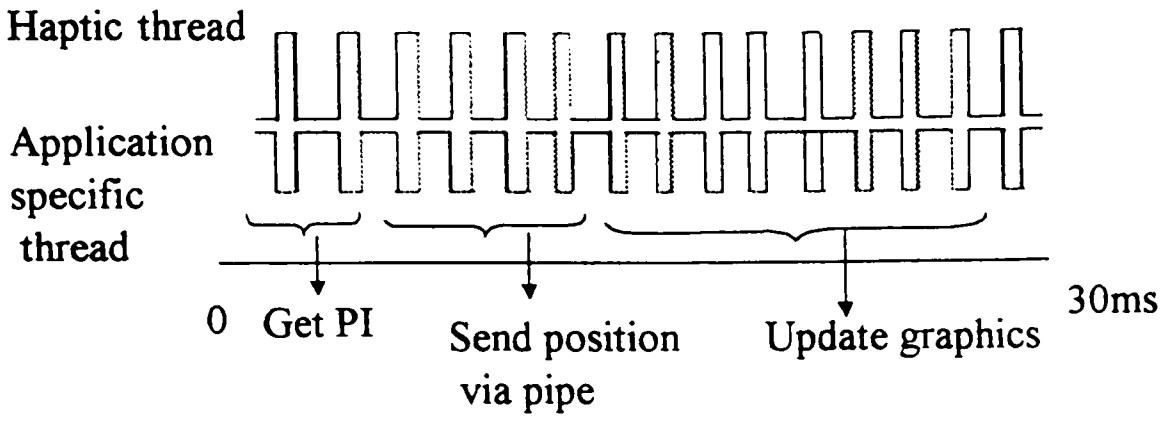

Figure 20. Graphic update using optimization

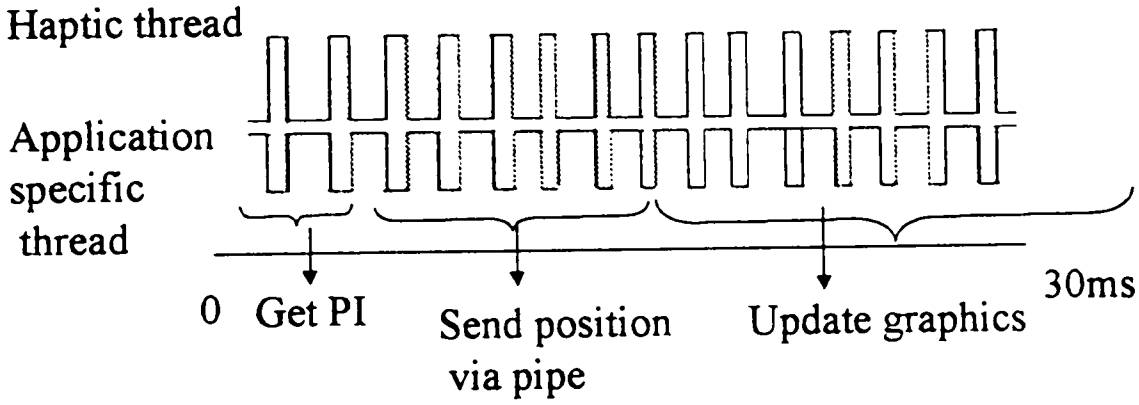

Figure 21. Graphic update without using optimization parameters

Synchronization of Tasks

For the application to work in real-time, the graphic display must update simultaneously in both haptic and graphic processes. In addition the haptic and graphic updates must be synchronized in the haptic process.

In the haptic process, the display graphic task (H9) renders the visual scene containing the surface and the graphical representation of the haptic device. The haptic servo loop running at 1kHz signals the display graphic tasks every 33 iteration to update. This produces a 30Hz graphic update rate and is maintained (assuming haptic environment is stable). Graphic updates in both processes (haptic and graphic) must be synchronized. Since graphic update in haptic process updates at 30Hz, it implies that graphic update in graphic process must update at a minimum rate of 30Hz.

In order to create a virtual environment that satisfies the haptic update rate of 1kHz and graphic update rate of 30Hz, we have to impose a limit on the time taken by tasks such as haptic rendering, forward looking and AFFB calculation. To synchronize these tasks and create a stable haptic environment, we need to understand some limitations on the size of an application, i.e., scene complexity. This refers to the maximum number of

non-overlapping objects allowed for all real-time tasks. This information allows imposition on process time limits on all other tasks.

Scene Complexity

Determining the complexity of a haptic environment is a challenging problem. In order to solve this problem, we must first identify the factors that effect the complexity of a scene. These factors vary from the size of the object, object arrangement in x,y,z direction, spacing between objects to the number of objects. To study the influence of these parameters on haptic applications in real-time, a number of experiments were performed [22], indicating that if non-overlapping spheres are used, haptic interactions remain stable for up to about 750 spheres using GHOST SDK version 1.2 libraries (see Figure 22). In comparison to GHOST 1.2 libraries, if Basic IO is used then the maximum number of objects that can be defined for a stable haptic environment are much higher in the range of 2600-2700. Since Basic IO provides a low-level interface to the PHANToM hardware and has fewer overheads, it takes less system resources. These features are very useful when dealing with volumetric data.

In a surgical simulation system, allowing for a complex scenario of 100 layers, scene complexity should not be an issue. This is seen in Figure 22. The graph shows the relation between the number of objects defined in the haptic scene and the Hload reading. For a load of 100 objects, the Hload reading is only 30% of 1 millisecond (the update rate) leaving the other 70% of 1 millisecond for the other tasks such as forward looking, calculating differential FFB, operating system related tasks etc to finish their work. Moreover, since in the layer-based model the haptic device will be interacting with one layer at a time, it allows ample time for other tasks to be accomplished. From our empirical results [22], we are assuming that as long as haptic rendering is completed within about 0.08ms (leaving the rest of the time for system tasks) the haptic stability will be maintained.

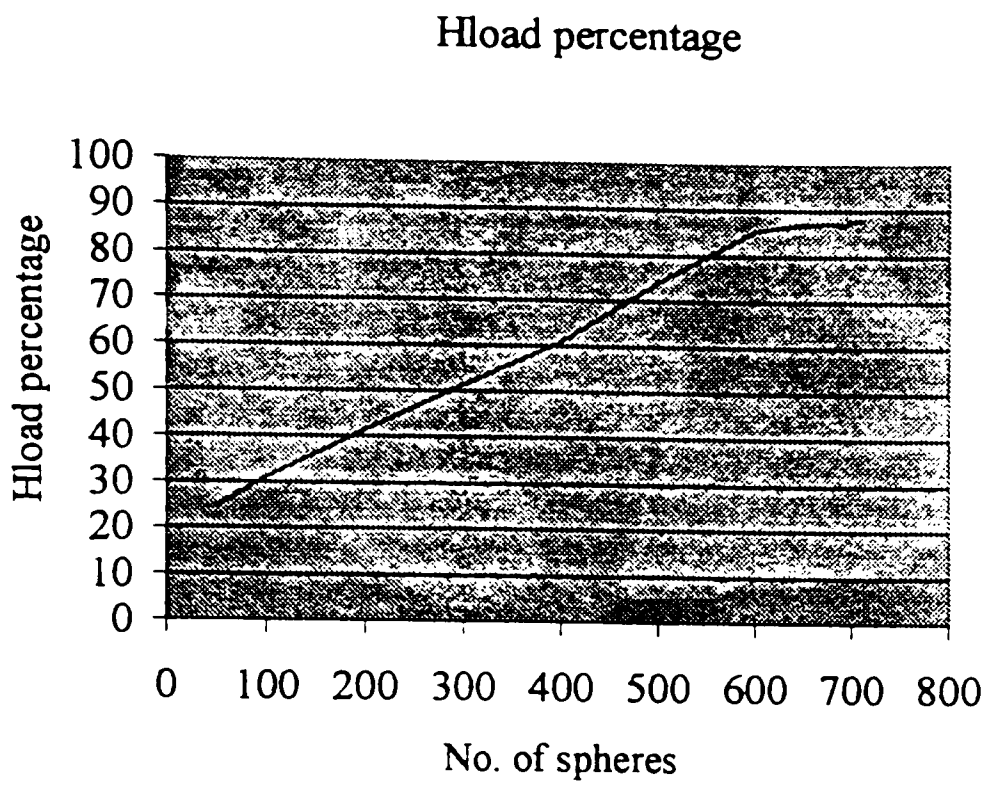

Figure 22. Graph showing relation between Hload and the Number ofobjects (spheres)

Performance

This project has been tested for it to work in real time by performing following experiments.

Experiment 1: test for real-time interprocess communication

In this experiment the time is recorded in the haptic process before the information is send via pipes and in graphic process after the information is retrieved from pipe. The difference in time is calculated for each transfer of information to check the amount of time taken to transfer information via pipe.

Table 7: Test 1 for experiment 1

Time (h:m:s.ms) before Sending position coordinate in haptic process = HS	Time (h:m:s.ms) after receiving position coordinate in graphic process = GR	Difference in time in millisecond = GR-HS
16:14:30.836	16:14:31.286	450
16:14:30.836	16:14:31.296	460
16:14:30.846	16:14:31.306	460
16:14:30.856	16:14:31.326	470
16:14:30.856	16:14:31.336	480
16:14:30.866	16:14:31.346	480
16:14:30.866	16:14:31.356	490
16:14:30.876	16:14:31.366	490
16:14:30.876	16:14:31.376	500
16:14:30.886	16:14:31.386	500
16:14:30.886	16:14:31.406	520
16:14:30.896	16:14:31.416	520
16:14:30.906	16:14:31.426	520
16:14:30.906	16:14:31.436	530
16:14:31.457	16:14:31.457	<1ms
16:14:31.477	16:14:31.477	<1ms
16:14:31.487	16:14:31.487	<1ms
16:14:31.507	16:14:31.507	<1ms
16:14:31.527	16:14:31.527	<1ms
16:14:31.547	16:14:31.547	<1ms
16:14:31.567	16:14:31.567	<1ms
16:14:31.587	16:14:31.587	<1ms
16:14:31.597	16:14:31.597	<1ms
16:14:31.617	16:14:31.617	<1ms

Table 8: Test 2 for experiment 1

Time (h:m:s.ms) before sending position coordinate in haptic process = HS	Time (h:m:s.ms) after receiving position coordinate in graphic process = GR	Difference in time in millisecond = GR-HS
16:59:06.753	16:59:06.934	181
16:59:06.763	16:59:06.944	181
16:59:06.783	16:59:06.954	171
16:59:06.793	16:59:06.964	171
16:59:06.803	16:59:06.974	171
16:59:06.834	16:59:06.994	160
16:59:06.844	16:59:07.4	160
16:59:06.844	16:59:07.14	170
16:59:06.854	16:59:07.24	170
16:59:06.864	16:59:07.34	170
16:59:06.874	16:59:07.44	170
16:59:06.874	16:59:07.64	190
16:59:06.884	16:59:07.74	190
16:59:06.894	16:59:07.84	190
16:59:07.104	16:59:07.104	<1ms
16:59:07.124	16:59:07.124	<1ms
16:59:07.134	16:59:07.134	<1ms
16:59:07.154	16:59:07.154	<1ms
16:59:07.174	16:59:07.174	<1ms
16:59:07.194	16:59:07.194	<1ms
16:59:07.204	16:59:07.204	<1ms
16:59:07.224	16:59:07.224	<1ms
16:59:07.244	16:59:07.244	<1ms
16:59:07.264	16:59:07.264	<1ms

Table 9: Test 3 for experiment 1

Time (h:m:s.ms) before sending position coordinate in haptic process = HS	Time (h:m:s.ms) after receiving position coordinate in graphic process = GR	Difference in time in millisecond = GR-HS
17:14:13.858	17:14:13.938	80
17:14:13.868	17:14:13.958	90
17:14:13.878	17:14:13.968	90
17:14:13.888	17:14:13.978	90
17:14:13.978	17:14:13.998	20
17:14:14.8	17:14:14.8	<1ms
17:14:14.28	17:14:14.28	<1ms
17:14:14.48	17:14:14.48	<1ms
17:14:14.58	17:14:14.58	<1ms
17:14:14.78	17:14:14.78	<1ms
17:14:14.108	17:14:14.108	<1ms
17:14:14.118	17:14:14.118	<1ms
17:14:14.138	17:14:14.138	<1ms
17:14:14.158	17:14:14.158	<1ms
17:14:14.178	17:14:14.178	<1ms
17:14:14.208	17:14:14.208	<1ms
17:14:14.228	17:14:14.228	<1ms
17:14:14.248	17:14:14.248	<1ms
17:14:14.268	17:14:14.268	<1ms
17:14:14.288	17:14:14.288	<1ms
17:14:14.298	17:14:14.298	<1ms
17:14:14.318	17:14:14.318	<1ms
17:14:14.338	17:14:14.338	<1ms
17:14:14.358	17:14:14.358	<1ms
17:14:14.379	17:14:14.379	<1ms

Table 10: Test 4 for experiment 1

Time (h:m:s.ms) before Sending position coordinate in haptic process = HS	Time (h:m:s.ms) after receiving position coordinate in graphic process = GR	Difference in time in millisecond = GR-HS
17:22:18.755	17:22:18.835	80
17:22:18.765	17:22:18.845	80
17:22:18.775	17:22:18.855	80
17:22:18.785	17:22:18.875	90
17:22:18.865	17:22:18.885	20
17:22:18.895	17:22:18.895	<1ms
17:22:18.915	17:22:18.915	<1ms
17:22:18.895	17:22:18.895	<1ms
17:22:18.935	17:22:18.935	<1ms
17:22:18.975	17:22:18.975	<1ms
17:22:18.995	17:22:18.995	<1ms
17:22:19.15	17:22:19.15	<1ms
17:22:19.35	17:22:19.35	<1ms
17:22:19.45	17:22:19.45	<1ms
17:22:19.65	17:22:19.65	<1ms
17:22:19.85	17:22:19.85	<1ms
17:22:19.106	17:22:19.106	<1ms
17:22:19.116	17:22:19.116	<1ms
17:22:19.136	17:22:19.136	<1ms
17:22:19.156	17:22:19.156	<1ms
17:22:19.176	17:22:19.176	<1ms
17:22:19.196	17:22:19.196	<1ms
17:22:19.216	17:22:19.216	<1ms
17:22:19.236	17:22:19.236	<1ms

Results

Four sets of data are collected for experiment 1. From the data given above in table 7-10 we can see that in each set first few transfer of information takes longer than the others. This is due to the set up time for interprocess communication which system needs at the commencement of the program. Once the system is set up for interprocess communication the time taken in transferring information between processes decreased significantly.

Experiment 2. Test for synchronized graphic updates

This experiment is performed to check if the graphic updates in both the processes (haptic and graphic) are synchronized. The time is recorded before the update graphic is called in haptic process and in graphic process. The difference in time is calculated to check the time interval between the graphic updates in both processes.

Table 11: Test 1 for experiment 2

Time (h:m:s:ms) before calling update graphic in Haptic process = HUP	Time(h:m:s:ms) before calling update graphic in graphic process = GUP	Difference in time in millisecond = GUP – HUP
17:26:22.282	17:26:22.372	90
17:26:22.302	17:26:22.382	80
17:26:22.312	17:26:22.402	90
17:26:22.322	17:26:22.412	90
17:26:22.382	17:26:22.422	40
17:26:22.432	17:26:22.443	11
17:26:22.453	17:26:22.453	0
17:26:22.473	17:26:22.473	0
17:26:22.493	17:26:22.483	-10
17:26:22.513	17:26:22.503	-10
17:26:22.523	17:26:22.523	0
17:26:22.543	17:26:22.543	0
17:26:22.563	17:26:22.553	-10
17:26:22.583	17:26:22.573	-10
17:26:22.593	17:26:22.593	0

Table 12: Test 2 for experiment 2

Time (h:m:s:ms) before calling update graphic in haptic process= HUP	Time(h:m:s:ms) before calling update graphic in graphic process = GUP	Difference in time in millisecond =GUP-HUP
17:30:53.833	17:30:53.923	90
17:30:53.853	17:30:53.943	90
17:30:53.863	17:30:53.953	90
17:30:53.883	17:30:53.973	90
17:30:53.963	17:30:53.983	20
17:30:53.993	17:30:53.993	0
17:30:54.13	17:30:54.13	0
17:30:54.33	17:30:54.33	0
17:30:54.53	17:30:54.43	-10
17:30:54.73	17:30:54.63	-10
17:30:54.83	17:30:54.83	0
17:30:54.103	17:30:54.103	0
17:30:54.123	17:30:54.113	-10
17:30:54.143	17:30:54.133	-10
17:30:54.163	17:30:54.153	-10

Results

Two sets of data are collected for experiment 2. From the data given above in Tables 11-12, we can see that in each set the difference in time when update graphics is called in both processes is longer than the others. Since the information is transferred and retrieved in update graphics in both the processes, the time delay is due to the set up time for interprocess

when update graphics is called in both processes is longer than the others. Since the information is transferred and retrieved in update graphics in both the processes, the time delay is due to the set up time for interprocess communication which system needs at the commencement of the program. Once the system is set up for interprocess communication the graphic update in both the processes run simultaneously (as indicated by 0 value in Tables 11 and 12). In fact the negative values in difference in time column in Tables 11and 12 indicate that the graphic update in graphic process is ready even before the graphic update in haptic process sends information, to retrieve it from the pipe and update it graphically.

CHAPTER VII

FUTURE ENHANCEMENTS

The new gstForceField class available with Ghost version 2.0 provides a complete flexibility for force in terms of both magnitude and direction. Differential force feedback model can further be extended using this class to provide further approximation of real world simulation.

The use of volumetric soft-bodied objects to represent non-homogenous tissues will provide a better anatomical model and give required efficiency needed for a real-time system simulation.

Using higher computational power of the computer platform or Pentium PCs with dual processor can further improve the system. The dual processor Pentium PC will allow the critical haptics process to run with real-time priority on one processor while the graphics and other processes can compete for the remaining resources.

CHAPTER VIII

CONCLUSION

In summary, we have developed a prototype of a PC/NT based surgical simulation using force feedback to show feasibility of creating a preventive forward looking surgical simulation. We have also resolved some of the real-time issues that will be encountered when such a simulation is developed using patient specific data. Transferring information works in real-time by synchronizing parameters such as overlapped I/O structure and WaitForSingleObject (semaphore). For development of real-time application, all haptic rendering has to be completed within about 0.08ms (using the remaining time for system tasks) and corresponding graphics rendering must be synchronized within 30Hz.

REFERENCES

1. Goertz, R. and R. Thompson, 1954, "Electrically controled manipulator," Nucleonics, pp. 46-47.

2. Makinson, B., 1971, "Research and development prototype for machine augmentation of human strength and endurance: Hardiman 1project," Technical Reports S-71-1056, General Electric Co., Schencectady, NY, May.

3. Burdea, G. C., "Force and touch feedback for virtual reality," working paper, Electrical and Computer Engineering Department, Rutgers—The State University of New Jersey.

4. Batter, J. and F. Brooks Jr., 1971, "GROPE-I: A computer display to the sense of feel," Proceedings of ITIP Congress, pp. 759-763

5. Kilpatrics, P., 1976, "The Use of Kinesthetic Supplement in an Interactive System," Ph.D Dissertation, Computer Science Department, University of North Carolina at Chapel Hill.

6. Ouh-young, M. M., O. Steele, F. Brooks Jr. and M. Behensky, 1990, "Feeling and Seeing: Issues in Force Display," Computer Graphics, ACM Press, Vol.24, No. 2, pp. 235-243, March.

7. Burdea, G., J. Zhuang, E. Roskos, D, Silver and N. Langrana, 1992a, "A Portable Dextrous Master with Force Feedback," Presence-Teleoperators and Virtual Environments, Vol. 1. No. 1, MIT Press, Cambridge, MA, pp. 18-27, March.

8. EXOS Co., 1995, "Sensing and Force Reflecting Exoskeleton (SAFIRE) Specifications," Woburn, MA, Company Brochure, 1p.

9. Massie, T. and K. Salisbury, 1994, "The PHANToM Haptic Interface: A Device for Probing Virtual Objects," ASME Winter Annual Meeting, DSC-Vol. 55-1, ASME, New York, pp.295-300.

10. Jackson, B. and L. Rosenburg, 1995, "Force Feedback and Medical simulation," in K. Morgan, R. Satava, H. Sieburg, R. Mattheus and j. Christensen Eds., Interactive Technology and the New Paradigm for Healthcare, IOS Press, Amsterdam, Chapter 24, pp.147-151, January.

11. Massie, T. H., "Initial haptic exploration with Phantom virtual touch through point interaction," Massachusetts Institute of technology, Cambridge, MA, (1993).

12. Lawrence, J., Rosenblum, M., and R. Macedonia, " Projects in VR," IEEE (September/October 1997).

13. Burdea, G. C., Force and touch feedback for virtual reality, Wiley-Interscience Publication, John Wiley and Sons Inc., New York.

14. Mandayam, A. S., and C. Basdogan, " Haptics in virtual Environment: Taxomony, Research Status, and Challenges," Computers and Graphics, Vol. 21, No.4. (1997).

15. Zilles, C. B. and J.K. Salisbury, "A Constraint-based God-Object Method for Haptic Display," IEEE International Conference on Intelligent Robots and System, (1995).

16. Cleary, K., C. Lathan, R. C. Platenberg, K. Gary, F. Wade, and L. Traynor, "Developing a PC-based spine biopsy simulator," The Second PHANToM Users' Group Workshop, MIT Press. Cambridge, MA, (1997).

17. O'Toole, R., R. Playter, W. Blank, N. Cornelius, W. Robert, and M. Raibert, "A novel Virtual Reality Surgical Trainer with Force Feedback: Surgeons vs. Medical student performance," The Second PHANToM User's Group Workshop, MIT Press. Cambridge, MA, (1997).

18. Burgin, J., S. Bryan, F. Vahora, B. Temkin, W. Marcy, P. Gorman, and T. Krummel, "Haptic Rendering of Volumetric Soft Bodied Objects" by, the Third PHANToM User's Group Workshop, October 3-6, 1998

19. Burgin, J. "Haptic rendering of soft-bodied objects with non-homogenous internal structures," Ph.D. Dissertation, Computer Science Department, Texas Tech University, (1998).

20. Andrews, M. C++ Windows NT programming, second edition, M&T Books, New York, New York, (1996).

21. Sensable Technologies, "Ghost software developer's toolkit--- Programmer's guide version 1.2," Sensable Technologies, Inc. Cambridge, MA, (1997).

22. "Haptic Scene complexity," To be published, contact temkin@coe.ttu.edu for more information.

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University or Texas Tech University Health Sciences Center, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Agree (Permission is granted.)

Student's Signature _____ Date _____

Disagree (Permission is not granted.)

Student's Signature _____ Date _____