2004

Networked haptic interaction in virtual reality for collaborative design

ChangEun Kim
Iowa State University

NOTE TO USERS

Networked haptic interaction in virtual reality for collaborative design

by

ChangEun Kim

A dissertation submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Program of Study Committee:
Judy M. Vance, Major Professor
Adrian Sannier
Greg R. Luecke
James H. Oliver
Shana S-F Smith

Iowa State University
Ames, Iowa
2004

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

ChangEun Kim

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. Judy Vance, for all of her help, knowledge and advice over the past three years. Her extraordinary patience and understanding make her an excellent advisor.

I would like to thank my committee members, Dr. Adrian Sannier, Dr. Greg Luecke, Dr. James Oliver and Dr. Shana Smith for their time and advice with my research and any other questions. I also wish to thank all of my friends and colleagues in the VRAC for their help and technical support.

Finally, I want to thank my wife and parent for their care and support they have given me over the years.

ABSTRACT

This research combines virtual reality with part interaction, force feedback, and network communication to facilitate collaborative design environment. By allowing collaborative tasks to occur with digital models though network communication as opposed to manipulation of real parts, industry can save time and money by trying many different options without gathering at one place.

The objective of this research is to investigate haptic feedback, collision detection, and object interaction within a networked collaborative virtual environment in order to provide realistic part interaction. Various collision detection and physical interaction packages were examined. The advantages and drawbacks of various network architectures were explored and the rationale for using the combination of client-server and peer-to-peer architectures was examined. Different multithreading structures were explored to maintain different update rates effectively.

The network inconsistency problem, which resulted from network delay and CPU latency, was investigated. The "Released-but-not-released" (RNR) synchronization method and timeout mechanism were developed and implemented in order to improve the synchronization of the object position/orientation over the network.

A networked haptic VR application was developed and tested for three network users with three force feedback devices, each with varying capabilities. The simulation speed and the update rates for each of the user's movements were calculated for the trial case when each user grabbed and collided objects with each other. Barrier synchronization was used to increase the speed of the simulation.

CHAPTER 1. INTRODUCTION

In the last ten years, virtual reality (VR) has emerged as an engineering design tool due to its ability to provide three-dimensional, interactive environments, which allow humans to interact with digital representations of products using natural human motions [1]. Increasing affordability of virtual environments has made VR applications possible in many fields, such as psychology [2], medicine [3], vehicle dynamics [4], architecture [5], education [6], and entertainment [7].

A key feature of VR is the ability of a user to be completely immersed in the computer-generated world. Stuart defines immersion as "the presentation of sensory cues that convey to users the sense of being surrounded by a computer-generated environment" [8]. Jayaram [1] defines the key elements of VR as: "a) immersion in a 3D environment through stereoscopic viewing, b) a sense of presence in the environment through tracking of the user and often representing the user in the environment, c) presentation of information of the senses other than vision, and d) realistic behavior of all objects in the virtual environment." The more senses are simulated, the more immersion can be provided. Most VR applications provide 3D visual and audio effects. However, the need to manipulate 3D CAD models within the VR environment requires the sense of touch.

Haptics refers to a category of technology that allows users to "touch" or "feel" virtual objects via mechanical simulation. In combination with a visual display, haptic technology can be used to train people for tasks requiring a significant amount of hand-eye coordination, such as performing surgery or maneuvering an aircraft. It can also be used in games in which a user feels as well as sees interactions with images. With the recent advent of fast network systems, real-time network communication technologies for distributed collaboration have begun to gather momentum towards allowing multiple users to share the same environment. Han defines a networked haptic environment (NHE) as "a software system through which people who are geographically dispersed over the world can interact with each other by sharing in terms of space, presence, and time" [9].

This research involves providing a network haptic environment to support collaborative assembly tasks.

1.1. Motivation

Virtual assembly, in the context of this research, is defined as the ability to assemble CAD models of parts using a three-dimensional, immersive user interface and natural human motion. Current CAD software allows sophisticated animation and planning of assembly and maintenance sequences. Digital humans can be positioned in order to identify ergonomic problems in the assembly or

maintenance of the product. These computer tools are used successfully to identify interference of parts and awkward assembly sequences. Assembling parts displayed on a computer screen, however, does not identify problems that occur when assembly workers, maintenance workers or users actually interact with the product. Awkward reach angles, insufficient clearance for assembly tools, additional part modifications to facilitate assembly or disassembly, and the need for additional fixturing are all assembly and maintenance issues in product design that cannot be identified through computer simulation using traditional computer interface tools such as a monitor and keyboard.

One of the features of VR that sets this technology apart from traditional computer simulations is the ability of the user to interact with the 3D objects using natural hand and head motion. This is accomplished through the use of motion trackers attached to the user's head and hand(s). Sitting at a traditional computer monitor, users interact with 3D computer models using the mouse and keyboard while viewing the results on the monitor. In VR, the user can move his/her head to change the view of the computer environment as if he/she were looking around in the natural world. Users can also reach out and "pick up" or "move" computer objects. The VR participant feels immersed in the computer-generated environment, and after a short time begins to interact with the objects as if they occupied positions in the real world.

VR presents the promise of "natural interaction" but the reality is that much research remains to achieve this vision. There are several levels of interaction that can be experienced in VR. The simplest type of interaction is accomplished using head tracking. The user's head position is detected and the computer adjusts the view displayed according to the viewing direction of the user. This allows the user to duck under an object or view the digital object from another viewpoint. The next level of interaction allows for navigation through the virtual space. Navigation is the process of moving a user's view through the virtual environment and can be accomplished using a position tracker on the user's head, hand or wand. This type of interaction is commonly used to explore virtual environments for architecture. A higher level of interaction occurs when the user is allowed to select and relocate objects in the environment. This can be accomplished by detecting when an object which represents the user's hand interacts with an object in the environment. After the collision has been detected, the selected object can move along with the hand movement. This action appears to the user as if he/she has picked up the selected object and is moving it around in space.

While the ability to look around, move around, select and release objects is generally well developed, simulating natural interaction between CAD models has not been achieved. Objects penetrate other solid objects or cease intersecting but remain at awkward positions. In addition, even if objects stop when a collision is detected, without force feedback, the user's hand continues to move

even as the display of the object is frozen. In order to provide realistic part interaction, research is needed in the area of integrating haptic feedback, collision detection and object interaction within a collaborative virtual environment. Until part interaction can be accurately modeled in the virtual environment, the promise of using VR to identify assembly and maintenance design issues as well as the ability to use VR for training will not be achieved.

Research in the area of projection screen VR environments with tracking devices, collision detection packages, physical constraints, haptic feedback devices, and real-time network communications has been active in the past ten years. Collision detection and physical constraints have been incorporated into a projection screen VR environment with tracking devices [10, 11]. A method to add haptic feedback to a projection screen VR environment has been investigated [12]. Collision detection packages and physical constraint methods have been developed and tested with haptic feedback [13, 14]. Collaborative virtual environments have offered new possibilities for remote collaborative working groups [15, 16], and real-time network communication has enabled multiple users to perform collaborative work together [17-21]. Combinations of available technologies in one application have already begun to create higher-quality immersive environments.

Developing a method to implement force feedback interactions in a networked haptic environment (NHE) is the goal of this work. Figure 1.1 shows the features needed to implement a networked haptic assembly application. The application will enable more than two users on the network to grab and move objects in a projection screen virtual environment with a haptic device. It will have real-time part interaction and force feedback when objects collide.

Combining projection screen VR, collision detection, physics constraints, force feedback, and network communication in a single application presents an optimization issue in regards to handling network delay, force feedback, collision detection, and physical interactions. The area of network haptic environment is relatively new and no known work has investigated the use of networking several haptic devices for interaction of the rigid-body simulation in a projection screen virtual environment. In order to integrate networked haptic force feedback with a projection screen virtual environment, and to develop a rigid-body interaction method for CAD models, several technical obstacles must be overcome. While an ideal solution would permit total freedom of a user's haptic interaction without any network delay for unlimited number of users, such efficiency is unattainable with the current state of NHE in VR. Various collision detection and physical interaction packages must be considered carefully in order to maintain the high update rate required to support haptics. The necessity of interacting with different geometry types while maintaining necessary haptics update rate (around 1000Hz) must be considered. Various network architectures and threading structures should

be investigated and suitable ones chosen after considering their advantages and disadvantages. Synchronization is also a major issue in NHE. Object synchronization methods for multiple users' interaction should be implemented.

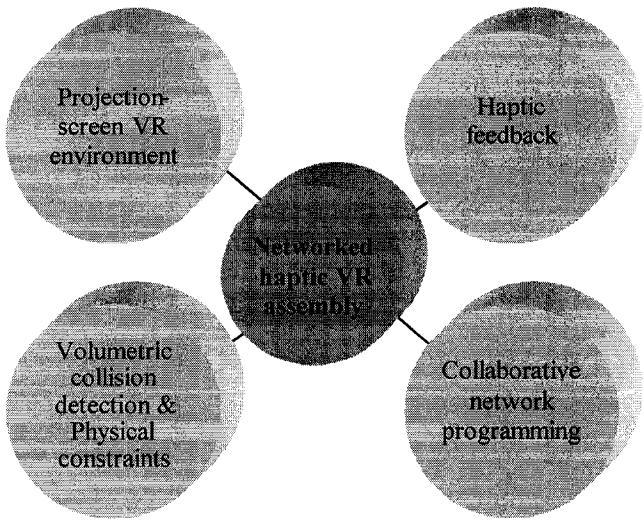


Figure 1.1: Components of a networked haptic environment in VR

1.2. Organization

Chapter 1 provides an introduction and outlines the motivation for the research. Chapter 2 presents a literature review of virtual assembly applications, collision detection and physical interaction packages, haptic devices, haptic collaborations, and network communication for distributed collaboration. The problems faced in this research follow, and a structure of the program is detailed in Chapter 3. In Chapter 4, performance results and discussion are presented. Finally, Chapter 5 contains the summary and conclusion of this research.

CHAPTER 2. LITERATURE REVIEW

2.1. Virtual Assembly Applications

Virtual reality provides a tool that allows users to interact with digital objects using natural human motions. In an immersive virtual environment the user interacts with objects just like in a real environment. For virtual assembly, VR can be used early in the design process to prototype assembly operations. Factory workers can be brought into the design process before the product design is finalized and asked to assemble products. Based on the findings of the virtual product assembly process, changes in product design can be recommended, which in turn could result in significant cost savings to the company.

Fraunhofer-Institute for Industrial Engineering (IAO) has developed an assembly planning system that makes it possible to interactively assemble and disassemble components and modules in a virtual surrounding [22]. It uses VirtualANTHROPOS – a virtual model of a human being – in order to carry out assembly operations. Since VirtualANTHROPOS is based on the anthropometrical module ANTHROPOS, VirtualANTHROPOS is able to simulate human kinematics in order to calculate assembly time and cost. This application uses collision detection to indicate part interaction, but does not implement part behaviors.

Jayaram et al. [11, 23-25] developed a virtual assembly application called VADE (Virtual Assembly Design Environment) at Washington State University. This application can input Pro/E CAD files to the virtual assembly program. Two-handed assembly can be performed using CyberGloves that detect finger bend angles for a realistic representation of the hand. Both a menu system and a voice recognition system can be used to manage the virtual environment. VADE also has the ability to detect collisions and model part behaviors. However, since VADE uses constraint-based part behavior modeling, reaction forces are not generated when objects collide with each other. Therefore force feedback cannot be implemented in VADE without an additional physics interaction method. VADE can display a virtual environment either through a head-mounted display or a single-pipe projection system but is not currently capable of being displayed in a multi-pipe stereo-projection environment.

Terrence Fernando et al. [26] developed a virtual assembly application called IPSEAM (Interactive Product Simulation Environment for Assessing Assembly and Maintainability) at the University of Salford that includes a limited ability to model part behavior. This application has been developed using the constraints-based geometric modeling approach. Modeling, however, is limited

to simulating part behavior of lower pair joints only (such as constraints between surfaces), leaving out constraints involving vertices and edges.

One virtual assembly application that has been tested using industrial examples is the Virtual Environment for General Assembly (VEGAS) [27] developed by Vance and Johnson at Iowa State University. It uses the geo file format for its graphics model input and Voxmap PointShell (VPS) for collision detection. VEGAS can be used in both single and multi-pipe display environments. VEGAS uses VPS as a collision detection package to indicate part interaction.

In order to develop a virtual assembly application that will provide adequate feedback to the user in his/her evaluation of the assembly process, several factors must be present in the application. Stereo viewing and position tracking of both the user's head and hands are required to provide the three-dimensional interface to the CAD data. Collision detection is needed between the user's body and the parts in the environment, and between the parts themselves, in order to indicate to the user that there are collisions occurring during the assembly process. This is different than the static interference detection available in most CAD packages. In order to simulate real assembly operations, physical constraints must be present in the environment to completely simulate part behavior. These physical constraints include interactions such as collars sliding on shafts and parts sliding on surfaces.

2.2. Collision Detection Packages and Physical Interactions

Many collision detection packages have been developed and tested with three-dimensional CAD data. I-collide [28], SWIFT [29], RAPID [30], V-collide [31], PQP [32] and SWIFT++ [33] have been designed by individuals by the University of North Carolina GAMMA (Geometric Packages for Modeling, Motion and Animation) research group. V-clip [34] was created by Brian Mirtich in 1998 at the Mitsubishi Electric Research Laboratories. These packages handle polygonal models which are prevalent in traditional computer graphic s applications.

Over the past few years, the interest in volumetric collision detection gets attention due to its fast calculation speed. Instead of complex mathematics needed to detect collision between polygonal surfaces, volumetric collision detection algorithms use a volume element, voxel, in order to calculate collisions between objects. Objects collide when two objects attempt to occupy the same voxel space. A major drawback of a volumetric collision detection algorithm is the contact accuracy. The accuracy of volumetric collision detection algorithm is inversely proportional to the voxel size.

Gibson proposed the use of a cubic voxel-based data representation for both visualization and part interactions of simple arbitrary model shapes [35]. She introduced the data structure and algorithms for two-dimensional modeling interacting, deformable voxel-based objects. Use of the cubic voxel-

based data representation with haptic devices was also developed by Avila and Sobierajski [36]. They focused on developing a haptic interaction method that is suitable for volume rendering. Much research has been focused on algorithms using sphere hierarchies similar to a cubic voxel-based data representation [37, 38]. Their algorithms represent a volumetric object using a collection of spheres. These spheres are computed based on the skeleton structure of the model. The main advantage of the sphere voxel-based data representation is that the spheres follow the shape of the object, so it can be used for rigid-body animation or elastic deformations without re-computing the voxel units. William McNeely et al. at Boeing developed Voxmap PointShell (VPS) in 1999 based on cubic volumetric based data representation [13]. Each voxel is allocated two bit of memory that indicates it as a free space, interior, or surface voxels. VPS enables the manipulation of moderately complex rigid objects with 6 degree-of-freedom (DOF) haptic rendering. By the nature of volumetric based data representation, voxmaps are insensitive to gaps or cracks that are smaller than the voxel width. The characteristics and comparisons of these packages will be addressed in Chapter 3.

In addition to collision detection, simulating physical part behaviors in the virtual environment is a key component of a realistic virtual assembly application. In general, there are two methods used to simulate physical behavior in VR: geometric constraint modeling and physically-based modeling. In geometric constraint modeling, certain geometric properties of the objects that would result in assembly constraints are identified. For each hole, for example, a sliding axis is identified. For each surface that could be used as a contact surface, a contact surface constraint is identified. Each part must go through a pre-processing step where all possible constraints must be identified [26]. Physically-based modeling, on the other hand, is a method that uses equations governing the motion of objects to model part interaction in the simulation. In this method, these equations of motion are solved at each time step based on forces and torques upon objects. Physically-based modeling can be divided into three categories: the penalty force method, the impulse method, and the analytical method. The penalty force method assumes there is a virtual spring damper system attached to the two colliding objects. The equation of motion is solved at every frame. This method is easy to implement and understand, but it is not easy to make robust because the dynamic differential equations can become stiff with a relatively large time step [39, 40]. The impulse method [41] uses the momentum conservation law to calculate an object's position and rotation. The impulse method is known as more stable and robust one than penalty force method. However this method does not apply to continuous contact such as resting or sliding. The analytical method [42] solves exact-surface boundary constraints through linear programming, emulating ideally rigid objects in quasi-steady equilibrium and applying conservation of momentum for non-equilibrium motion [41]. However, the

analytical method takes lot of computation time when many contacts occur at the same moment. Therefore, impulse method or analytical method may not be suitable for haptic displays.

2.3. Haptic Devices

Haptic perception is a rather complex subject that includes the very different mechanisms of tactile perception and kinesthetic/force perception [8]. Haptic device is the science of applying tactile sensation to human interaction with computers. A haptic device is one that involves physical contact between the computer and the user, usually through an input/output device that senses the body's movements. There are a wide variety of specialized research-oriented haptic devices available. Figure 2.1 shows some example devices.

(a) (b) (c) (d) (e)

Figure 2.1: Examples of haptic devices. (a) SensAble's Phantom Desktop, (b) MPB-technologies Freedom 6S, (c) Force dimension's 3-DOF Omega haptic device, (d) Immersion's CyberGrasp, (e) Immersion's Haptic Workstation

Salisbury et al. distinguished haptic devices in two ways: their ground locations and their mechanical behavior [43]. Ground-based device such as SensAble's Phantom device (Fig. 2.1 (a)) can reproduce object weight or inertia forces, while Immersion's CyberGrasp (Fig. 2.1 (e)), as an example of a non-ground device or body-based device, generates finger-specific force feedback, but not object weight or inertia forces. Immersion's Haptic Workstation (Fig. 2.1 (b)) combines a ground-based device with a body-based device in order to simulate ground force and finger forces at the same time. Another way to categorize haptic devices is by their mechanical behavior: impedance and admittance devices. Impedance devices read position and generate force, while admittance devices read force send position. Haptic devices shown in Figure 2.1 are all impedance-type devices. An example of the admittance-type device, Sarcos' Dextrous Arm Master [44], is a remote robot arm that is used as a human-size slave arm. It can be used in applications such as production assembly, undersea manipulation, and hazardous materials handling.

The most widely-used and commercially-available force feedback devices are SensAble Technology's PHANToM [45], Immersion Corporation's CyberForce [46], and MPB Technologies'

Freedom 6S [47]. The PHANToM (Personal HAptic iNTerface Mechanism) was designed as a relatively low-cost device to provide a sense of force interaction with virtual objects [48]. Its ease of use and commercial availability make it a popular haptic device interface. The CyberForce consists of the CyberGrasp and a 3 DOF force feedback robotic arm. The CyberGrasp is a linkage type device that applies forces to the fingers. Because it is body grounded, the weight and inertia of an object cannot be sensed. The CyberForce combines a CyberGrasp and a robot arm similar in appearance to the PHANToM. The robot arm is floor grounded and therefore the CyberForce can simulate object's weight and inertia. MPB Technologies' Freedom 6S has been developed by Prof. Vincent Hayward of McGill University to generate force feedback in 6 DOF. The Freedom 6S is also a linkage, but it has a smaller workspace (8.8 in by 9.6 in by 8.8 in) than PHANToM 1.5.

2.4. Haptic Collaborations

In addition to force feedback for a single user, researchers have recently focused on multi-user haptic environments. A main concern in this research is network latency [19]. Network latency does not only decrease the sense of immersion, but can also cause an application crash.

Ho et al. performed the first significant experimentation with a collaborative haptic environment [17]. They studied whether haptic communication through force feedback can facilitate a sense of togetherness between two people at two different locations who are interacting with the same virtual environment. In the experiment, subjects were asked to move a ring on a wire in collaboration (Fig. 2.2). The experiment showed that in the presence of haptic feedback, the feeling of togetherness significantly improved collaborative task performance. The simulation was performed using a single machine to display 3D graphics and to operate force feedback devices, in this case a pair of PHANToM controllers. The two users each handled a haptic device connected to the single computer so network latency was not a variable in this study. In addition, although physical interaction existed between the two users' haptic devices, they were not concerned with simulating actual part-to-part behaviors.

Sallnas et al. simulated a different maneuver where the objects were modeled to simulate simplified cubes with mass, damping, and friction [18] (Fig. 2.3). Simple object-object and object-wall interactions were implemented, but the objects lacked the ability to rotate. A setup utilizing a single host computer with two monitors eliminated network latency problems, but limited the physical distance between two users.

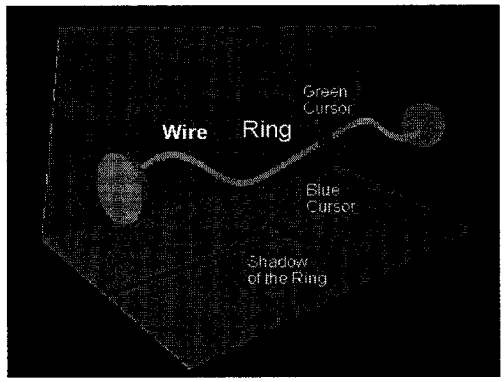


Figure 2.2: A distributed virtual environment used to play the "Ring on a Wire" game [17]

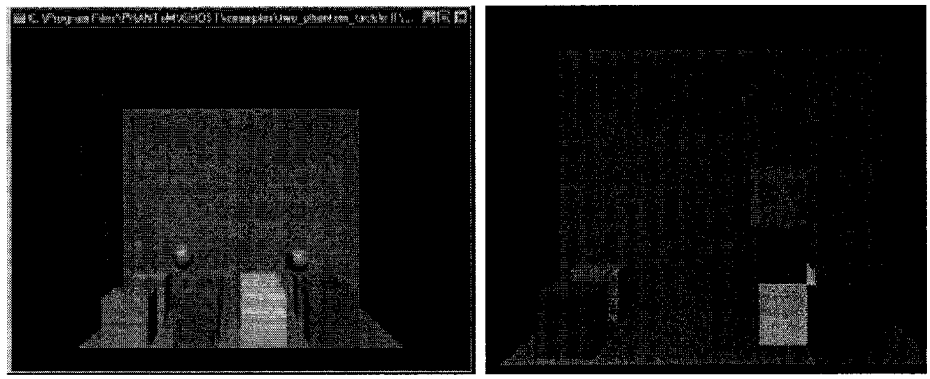


Figure 2.3: A collaborative virtual environment with eight dynamic cubes placed in a room representation [18]

In 2002, a haptic collaborative application called Virtual Handshake was developed over a network path that had significant physical distance [20]. This application consisted of a simulated room, a black box, and two tiny square pointers that showed the users' positions in the room. The users operated PHANToM desktop devices to collaboratively move the box. It was the first long-distance collaborative haptic application to utilize the Internet2 network. However, the cubes did not respond to torque, and only one simple pre-defined object, such as a box, could be displayed in a room representation. The users were also required to move very slowly in order to maintain synchronization between the networked computers.

The work of Gunn et al. is so far the closest to the work proposed in this thesis. They described a haptic collaborative virtual environment that spans global distance which allows collaborative manipulation of simulated human body organs and direct guidance of human hands between Australia and Sweden [49]. An instructor and student, communicating through a non-dedicated network line,

could view, touch, and interact with 3D objects. In addition, the instructor could take control of the student's haptic device and allow the student to feel the force of the instructor's guiding hand. The instructor and student could also collaboratively push, stretch, and pull on an object (Fig. 2.4).

Figure 2.4: Haptics collaboration for surgical training application [49]

Most NHE applications created so far can accommodate at most two users. When required to connect more than two users for haptic interaction, communication architecture is an important consideration in an NHE. Various communication architectures can be considered for use in an NHE network: client-server, peer-to-peer, multicast, or a combination of methods [50]. Matsumoto et al. summarized advantages and disadvantages of peer-to-peer and client-server architecture for the networked haptic environment [51]. Peer-to-peer architecture does not require a centralized server introducing further transmission delay between clients. Therefore, the network is alive as long as any peer is active. However, the connection for a peer-to-peer system becomes more complex as the number of clients increases, and additional delay handling processes are needed to guarantee consistency of objects in each client. In addition, each client requires significant computing facilities in order to run the interaction simulation independently. In client-server architecture, a complex interaction simulation runs in a server and only the results are distributed to the clients, which keeps the consistency of the collaboration. The computing power on the client side does not need to be high-end but faster connection speed is required in order to transfer results through the network. An additional concern is that propagation delay may cause excess reaction forces to be generated by a haptic device. A client-server NHE application was implemented by Matsumoto et al. [51] and a peer-to-peer NHE application was implemented by Jordan et al. [20]

The proposed method in this research enables simulation of repulsive forces and torques in a multi-user networked application with a mixture of peer-to-peer and client-server architecture,

without limitations on model complexity. This will make the application more useful for real haptic collaborative applications in industry.

2.5. Haptics for Projection Virtual Environment

While all current haptic collaboration research is developed for use on 2D monitor displays, a method to add force feedback for a single PHANToM user in a projection screen virtual environment has been developed (Fig. 2.5), with the ultimate goal of using it in a virtual assembly application [12]. Several software packages including VR Juggler, GHOST, and VPS were combined to explore the benefits haptic feedback can provide for various tasks. Two example applications—one that loads a NURBS surface that the PHANToM can interact with, and another that involves a simple virtual assembly task—have been developed and tested in a multi-projection screen virtual environment.

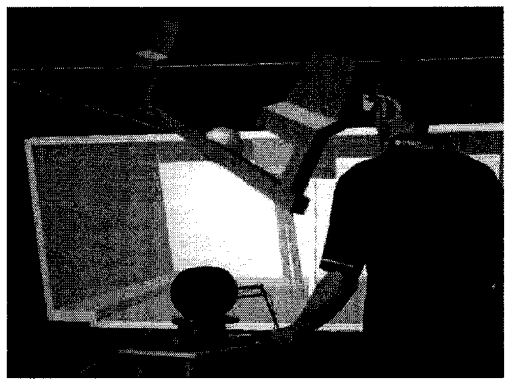


Figure 2.5: Single PHANToM user in a projection screen virtual environment [12]

The realization of multi-user haptic environments for virtual assembly applications in a projection screen virtual environment requires collision detection, physical interaction, haptic devices, and specified networking structures. Combining and optimizing these components is critical in development of NHE application.

CHAPTER 3. NETWORKED HAPTIC ENVIRONMENT IN VIRTUAL REALITY

Virtual assembly can be implemented using any of four levels of interaction. The first level includes the ability to grab, move, and release an object as well as the ability to examine part collisions. Physics interaction is added in the second level. This provides the ability to model object-to-object interactions to simulate the real world. There is no force feedback involved. The third level involves adding force feedback, also known as haptics. With a haptic device, a user can collide a part and feel the reaction force. The last level is networking, which enables multiple users to interact with each other.

Imagine that an engineer, a product designer, and an assembly worker are in virtual environments that are remotely located, each equipped with haptic device. They decide to collaborate on examining a part assembly in the early stage of the product design. Anyone can grab and move a virtual object and feel the reaction force when one user's part collides with another's part. This research is designed to provide a general framework for integrating currently available VR technologies, with the goal of using these technologies to explore collaborative haptic interaction.

3.1. Libraries

This application was developed using a variety of software toolkits. C++ was chosen as the programming language and the open source VR Juggler software toolkit was used for controlling the virtual environment (www.vrjuggler.org). Silicon Graphics (SGI) OpenGL Performer™ was used for rendering the virtual world and parts. The GHOST software from Sensable Technologies was used for driving the PHANToM and Voxmap PointShell (VPS) from Boeing was used for detecting collisions and calculating physical interactions. Figure 3.1 illustrates libraries needed for this research.

VR Juggler provides a platform for virtual reality application development and allows a user to run an application on different VR systems [52]. The VR Juggler Portable Runtime (VaPoR) library provides an operating system abstraction layer that simplifies the process of creating cross-platform software.

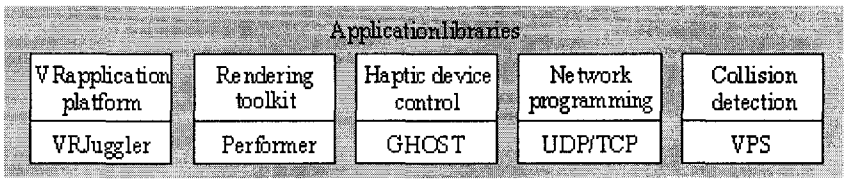


Figure 3.1: Application libraries

GHOST is a C++ object-oriented toolkit that represents the haptic environment as a hierarchical collection of geometric objects and spatial effects. It has a class that allows the developer to send force information to the haptic device.

The VPS software is a volumetric-based collision detection package developed as a fast collision detection method for complex models. In VPS, pointshell objects are represented by a set of surface point samples and their associated inward pointing surface normals, collectively called a pointshell. The environment of voxmap objects is represented by a single cubic occupancy map called a voxmap [13]. When the pointshell object's pointshell penetrates a voxmap object, the depth of penetration is calculated. Penalty forces are calculated using the depth of penetration and are summed to give a resultant force and torque. Besides the forces emerging from the virtual coupling, those resulting from a 6 DOF spring-damper system between the dynamic object and the virtual hand also act on the dynamic object. The dynamic rigid body behavior is calculated using Newton-Euler dynamics. The resulting position and velocity offsets between the virtual hand and the dynamic object change the force and torque in the coupling, which are then fed back to the simulation and displayed to the user.

3.2. Hardware

The virtual reality device used at Iowa State University is a multi-pipe stereo-projection environment. The Virtual Reality Applications Center (VRAC) has two large screen projector systems, the C4 and the C6 (Fig. 3.2). The C6 is a 3.048m by 3.048m by 3.048m. room equipped with 6 rear projection surfaces, which serve as the walls, ceiling and floor. The users wear stereo shutter glasses that are synchronized with the computer display to alternate the left and right eye views at a rate of 96 Hz in order to produce stereo images. A magnetic Ascension Motionstar tracking system tracks the user's head, hand, and arm positions. A 24-processor SGI Onyx2 Reality Monster supplies the computational power and six InfiniteReality2 graphic pipes, each with 256MB of texture memory, manage the graphics output. Each processor is a 400MHz MIPS R12000, and the computer contains 12Gb of RAM.

The C4 is a re-configurable projection system that has three walls and a floor projection surface. An Ascension Motionstar tracking system tracks the user's movement throughout the environment. The C4 system can be driven by a variety of computer systems. These systems include the following:

- An Onyx rack system with 16 MIPS R10000 CPU's, 1792Mb of RAM, and 3 InfiniteReality graphics pipes.
- An Onyx2 rack with 24 MIPS R12000 CPU's, 12Gb RAM, and 6 IR2 graphics pipes (same system used for C6).

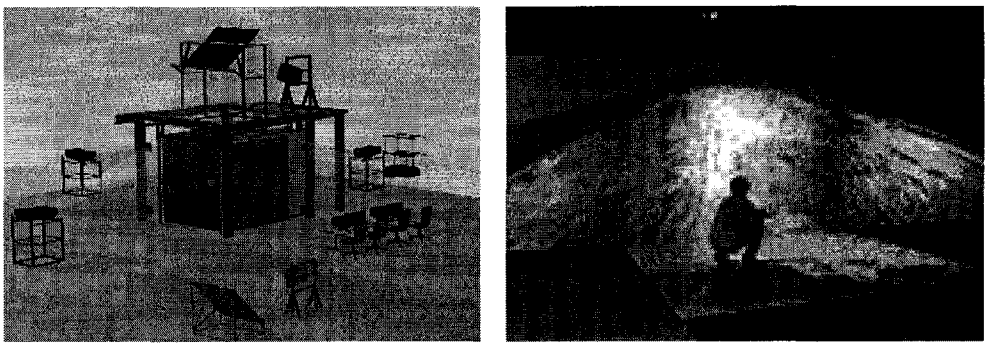


Figure 3.2: The C6 and the C4 at Iowa State University

The VRAC facilities also include several haptic devices, including different models of PHANToMs (Fig. 3.3). PHANToM Desktop provides a workspace of 15.24cm by 12.7cm by 12.7cm. PHANToM 1.5 provides a workspace of 19.05cm by 26.67cm by 38.1cm and PHANToM 3.0 has a workspace of 40.64cm by 58.42cm by 83.82cm.

Figure 3.3: PHANToM Desktop, PHANToM 1.5 and PHANToM 3.0, by SensAble Technologies (Images courtesy of Novint Technologies)

3.3. Collision Detection/Physical Interaction Packages

Since haptic rendering requires a high update rate, it is necessary to investigate collision detection and physical interaction packages carefully. There is considerable discussion in the geometric modeling community concerning the use of polygon-based vs. volume-based collision detection packages [13, 36, 53]. The decision to use VPS, which is a cubic volume-based package, is specifically tied to the need to perform virtual assembly. The rationale for the selection of VPS over other more common polygon-based packages is presented in this section.

The virtual assembly application takes complicated CAD model input and allows users to naturally pick up and assemble digital objects in the immersive virtual environment. The factors to be considered in selecting a collision detection package for this application include:

1. Ability to handle complicated part topology

2. Accuracy of collision detection
3. Performance
4. Preprocessing requirements for CAD input models
5. Ability to detect not only collisions, but to perform other types of part-to-part interaction queries

The collision detection packages investigated for this research include:

1. I-collide [28]
2. V-clip [34]
3. SWIFT [29]
4. RAPID [30]
5. V-collide [31]
6. PQP [32]
7. SWIFT++ [33]
8. SOLID [54]
9. Shape-based Volumetric Collision Detection [38]
10. VPS [13]

I-collide is an exact collision detection library developed in 1995 for large environments composed of convex polyhedra for multi-body collision detection. RAPID works with non-convex models but detects pair-wise collision only. V-collide, which is based on RAPID, includes the ability to detect multiple body collisions. PQP, which is also based on RAPID, is a pair-wise collision detection tool that supports non-convex modes. It also can perform distance computation and tolerance verification queries. SWIFT provides various queries such as intersection detection, tolerance verification, exact and approximate distance computation, and contact determination of convex models. SWIFT++ is based on SWIFT and supports non-convex 2-manifold objects. SOLID is especially suited for collision detection of objects and worlds described in VRML. All of these methods are polygon-based intersection packages. VPS, on the other hand, represents geometry using voxels, which are small cube elements [13]. VPS can detect collisions, verify tolerances, approximate distance, determine contact normals and center of mass, and also has the ability to implement physically-based modeling of part behavior.

In VPS, the geometric models of all parts are voxelized prior to start up of the virtual assembly simulation. Voxelization is the process of converting a geometry file, which is a set of triangular

polygons, to the VPS spatial representation called a "voxmap". The VPS method defines two objects in the environment: a pointshell object and a voxmap object. A voxmap object is represented by a single cubic occupancy map, called a voxmap. A pointshell object is represented by points placed at the center of each voxel. By definition, a dynamic object is an object moving in a virtual environment and a static object is any object that does not move in the environment. The voxmap object is conceptually static, in the sense that its voxels are not recomputed under motion, while the pointshell object is a collection of surface points that represents the moving object. When both the voxmap and pointshell objects are in motion, the relative transformation of the pointshell object is used to calculate collision information, while a voxmap object is considered as a static object.

When the pointshell penetrates the center of a voxel object or another pointshell of a pointshell object, a collision is detected. In addition, the penetration is used to determine the reaction forces. These forces can then be used to model object behavior.

The rest of this section will explore each of the five consideration factors used to distinguish between collision detection packages for virtual assembly.

3.3.1. Ability to handle complicated part topology

A virtual assembly application needs to be able to accommodate complicated part topology. Most collision detection packages can be divided into three categories according to the required topology of the input files: convex-based packages, non-convex manifold-based package, and polygon soup-based packages. Convex-based packages only work if the input geometry consists of convex polyhedra. These packages can give more information than the polygon soup-based packages, such as distances between objects, penetration depth, and tolerance check, with very fast speed. Non-convex manifold-based packages handle non-convex 2-manifold polygonal geometries, with fairly good speed. Polygon soup-based packages can deal with any collection of polygons but they do not have the ability to determine much beyond detecting a collision. Though convex-based packages and non-convex manifold-based packages are fast and can provide additional query information, they are not suited for virtual assembly applications where parts consist of arbitrary topology. I-Collide, V-clip, and SWIFT are convex-based packages, SWIFT++ is non-convex manifold-based package, and RAPID, V-collide, PQP, and VPS are polygon soup-based packages. Therefore, I-Collide, V-clip, SWIFT, and SWIFT++ are not suitable for virtual assembly applications.

3.3.2. Accuracy of collision detection

All polygon based collision detection packages can perform exact collision precision, while VPS is a surface approximation collision detection method. In this sense, absolute surface to surface accuracy is impractical to obtain and is not the goal when using VPS. In the applications, absolute accuracy is not required. VPS approximates the surface geometry using volume elements, or voxels, in order to calculate collisions, so the accuracy of VPS is inversely proportional to the voxel size. Therefore, voxmaps are insensitive to surface imperfections such as gaps or cracks that are smaller than the voxel width. Similar to the polygon-based methods, however, a trade-off exists between accuracy and performance. Smaller voxels require more computation time.

By definition, the voxmap is "a single spatial occupancy map" with a certain predefined size, and the pointshell is "the center point of the voxmap" [13]. The environment of static objects is represented by voxmaps and the dynamic object's motion is described as pointshells. When a pointshell interpenetrates a tangent plane that passes through the voxel's center point, a depth of penetration is calculated. Therefore the maximum distance offset between two parts in VPS is:

$$\text{MaxOffset} = \frac{\sqrt{3}}{2} \cdot (\text{voxel size of one part} + \text{voxel size of the other part}) \quad (1)$$

For example, if a 0.6096cm (0.02ft) voxel size is used as a global voxel size, then this results in the maximum offset of 1.0668cm. In other words, the voxel models are larger than the graphics models by, at most, the maximum offset of 1.0668cm. Therefore, tight-fit parts cannot be assembled because of this accuracy limitation. According to Equation (1), the maximum offset can be reduced if a smaller voxel size is used. However, the smaller the voxel size, the slower the object will move in the environment. A voxel size of 0.6096cm (0.02ft) is used in this project in order to guarantee an object speed of 0.6096m/s, a reasonable offset of about 1cm, and a 200-Hz update rate. More information about voxel size, object speed, and update rate is explained in the section, 'Part-to-part interaction limitations'.

3.3.3. Performance

Because of the need for real-time collision detection in virtual reality, performance of collision detection is a critical consideration. In general, polygon-based packages that deal only with convex topology (I-Collide, V-Clip, and SWIFT) are faster than those that deal with more general topology

(RAPID, V-Collide, PQP, SOLID, and SWIFT++). However, since the virtual assembly application must process general topology, it is limited to selecting from the non-convex packages.

In previous research, RAPID, V-Collide, and PQP proved to have faster execution times than SOLID [55]. However, SWIFT++ has recently proven to be the fastest method. Though SWIFT++ deals with non-convex 2-manifold objects, it maintains good performance because it uses SWIFT (a convex-based package) as its core [33]. The SWIFT++ package takes non-convex geometry and subdivides the geometry into a series of convex objects using its "decomposer" preprocessor. The convex-based package can then be applied to all of the sub-objects in the scene.

The collision detection scheme in a polygon-based package is conceptually different than in a voxel-based package. The first collision check level of a polygon-based package is to compute minimal bounding boxes for an object. Only a pair of objects whose bounding boxes overlap in three dimensions are passed to the pair-wise test for smaller bounding boxes till a bounding box contains one primitive. Then exact intersection tests between the triangles on the overlapping bounding boxes are performed [31].

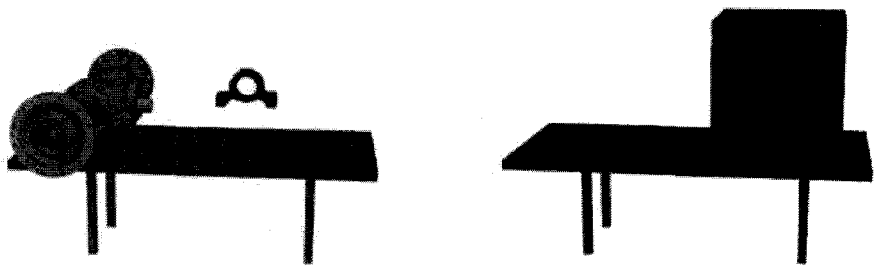
Collision detection for voxel-based objects is computed when one voxel intersects another voxel. In order to reduce collision checking calculation for each voxel/object/frame, most voxel-based algorithms use a collection of voxel units as a bounding box [13, 37, 38]. When a part is away from contact, only contact of those collections of voxels are tested. When a part is close to other parts, the smallest unit (voxel) is used to perform collision check. In VPS, the motion transformation of the pointshell object is applied to every pointshell during each time step, and collisions are detected as volumetric intersections between voxels and pointshells.

An experiment was performed in order to examine how the performance of PQP compares to VPS for models of different polygon and voxel counts. The collision query time was measured for both a complex model that needs many polygons to adequately describe the geometry and a simple model that can be described by only a few polygons. This query time comparison was made with various sampling densities that induce different surface offset. Sampling density in VPS is a voxel size, while it is pre-defined offset value in PQP. The voxel sizes were varied from 0.6096cm (0.02ft) to 2.4384cm (0.08ft) in this experiment and the maximum offset distance for PQP was calculated based on the corresponding voxel sizes (Eq. 1).

To examine the effect of model size and number of polygons on relative performance of VPS and PQP, several timed tests were performed. Figure 3.4 (a) shows the first performance test model set. The table and axle model are one object with 26,356 triangular polygons (model information needed for PQP) and 227,698 voxels (model information needed for VPS). The axle cap consists of 4,769

triangular polygons and 8,465 voxels. This represents a highly detailed haptic model. Collision query times for the minimum information to simulate part interaction (e.g., collision, minimum distance, and collision point) are shown in Table 1. As the voxel size and the pre-defined offset value in PQP were decreased, the collision query times in both packages were also decreased. However, VPS query time is more than 10 times faster than PQP in this case.

Figure 3.4 (b) shows the second performance test model set. The table is one object with 12 triangles and 92,986 voxels. The cube has 12 triangles and 94,932 voxels. This represents a more coarsely detailed model. Collision query times for the minimum information to simulate part interaction are shown in Table 2. As this test shows, VPS loses its performance advantage when the number of voxels used to represent the object is much greater than the number of polygons required.



(a) Model set #1 **(b) Model set #2**

Figure 3.4: Performance test model sets

Table 3.1: Collision query time of test model set 1 with various sampling densities

Voxel Size in VPS	VPS query time	PQP Offset value	PQP query time
0.0 (cm)	N/A (sec)	0.0 (cm)	0.013 (sec)
0.6096	0.00031	1.0546	0.0045
1.2192	0.00025	2.1123	0.0039
1.8288	0.00016	3.1339	0.0036
2.4384	0.00011	4.2245	0.0033

Table 3.2: Collision query time with test model set 2 with various sampling densities

Voxel Size in VPS	VPS query time	PQP Offset value	PQP query time
0.0 (cm)	N/A (sec)	0.0 (cm)	0.0014 (sec)
0.6096	0.0008	1.0546	0.0009
1.2192	0.00078	2.1123	0.00077
1.8288	0.0007	3.1339	0.0007
2.4384	0.0005	4.2245	0.0006

3.3.4. Preprocessing requirements for CAD input models

Collision detection packages require input files that contain model structure information, which is used to check various collision queries. The input file format of RAPID, V-collide and PQP consists of simple ASCII files that can be generated from a general CAD file format such as STL or ASE.

Creating input files for SWIFT++ is more difficult. SWIFT++ requires an extra file conversion step. Simple tessellated ASCII files that are generated from general CAD files are input into the "decomposer" software. Decomposer is a standalone executable library that takes basic model geometry and subdivides it into a series of convex objects for SWIFT++. The graphics model must be perfect for the decomposer process to work without an error. Most CAD packages create graphics models containing some geometrical errors. To fix these errors, an application program named IVECS (Interactive Virtual Environment for the Correction of STL files) [56] has been developed by Georges M. Fadel at Clemson University. IVECS displays the errors found in the STL file surface and allows the user to correct them manually. Once the STL file is consistent, decomposer can be used to prepare the file for processing by SWIFT++. IVECS is a powerful tool, but the process of fixing the errors in complicated .STL files is very time consuming and tedious.

VPS accepts standard ASCII files in the STL or ASE format. The conversion program called stl2vps in VPS converts STL files into binary VPS format files. This conversion creates the voxel representations needed for the collision detection. VPS has the ability to create the voxel model either within the VR application at run-time or through the use of the stl2vps conversion program.

3.3.5. Ability to detect not only collisions, but to perform other types of part-to-part interaction queries

If a physical constraint interaction model is needed during the assembly process operation, the collision detection package needs to query tolerance verification, exact and approximate distances,

nearest features, center of mass, and contact normal vectors in addition to intersection status. Types of queries for four different collision detection packages are shown in Table 3.1.

Coutee and Bras [53] compared collision detection methods for disassembly applications according to the following five features: closest point, collision features, depth of penetration, programmatic geometry construction, and n-body detection. The collision features comparison in this research expanded to include examination of intersection, tolerance verification, distance, contact normal, and center of mass capabilities of each collision detection package.

VPS does not provide exact distance, nearest features, or nearest point calculations, but provides part-to-part interactions using a physically-based modeling approach. Within VPS are functions that calculate the interaction forces between colliding objects. These forces are used to model the part-to-part interactions. Another viable collision detection option, SWIFT++, provides most of the queries but does not have part-to-part interaction packages. These would then have to be created separately by the developer.

Table 3.3: Types of queries in four collision detection packages (o = present, x = not present)

	V-collide	PQP	SWIFT++	VPS
Intersection	o	o	o	o
Tolerance verification	x	x	o	o
Exact distance	x	o	o	x
Approximate distance	x	x	o	o
Nearest features	x	x	o	x
Nearest points	x	x	o	x
Contact normal	x	x	o	o
Center of mass	x	x	o	o

3.3.6. Physical constraint interaction

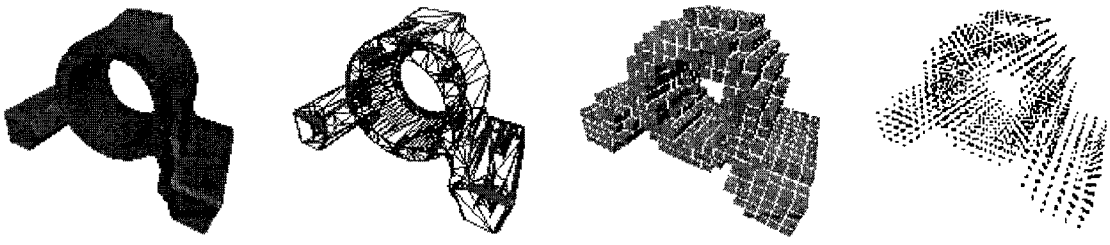
Along with collision detection, physical constraints in the virtual environment are implemented to make users feel immersed. VPS has a built-in physical constraint interaction capability called PBM (Physically-Based Modeling). It generates a collision response and calculates the subsequent motion. Details about implementation issues concerning the physical interaction capabilities of VPS are further detailed later. Without physical constraint modeling, the user must pre-define geometry

constraints between objects before the virtual reality application starts. The use of physically-based modeling, therefore, is a more general approach to modeling interaction constraints.

VPS does not have any restriction on the input model shape and has been shown to be compatible with the graphic interface, SGI OpenGL Performer™. It provides sufficient query results at the expense of contact accuracy. It uses easy-to-make input files, and has a built-in interaction generation library including swept volume generation, and a PHANToM haptic device driver. A dynamic part is defined as an object moving in a virtual environment and a static object consists of all other objects that do not move in the environment. Since VPS is a voxel-based data representation, the performance depends on the physical size of a model, while the performance of the polygonal-based package depends on the number of polygons. Therefore a reasonable number of objects, polygons, and size for an assembly application will be defined and used for the test.

3.4. Voxmap PointShell (VPS)

The VPS (Voxmap PointShell) software is a based on voxelization of the geometry model. Voxelization is the process of converting a geometry file, a set of triangular polygons, to the VPS spatial representation called a "voxmap". This voxelization is an outside-in process. First a bounding box is created that encloses the entire part. Then the package propagates exterior-marked voxels inward until it meets the object surface. All voxels touched by any vertex, edge or face of a triangle is marked as a surface voxel. Each part has only one voxel size, but different parts can have different voxel sizes. Solid and wire-frame representations are illustrated in the Figures 3.5 (a) and (b). The voxmap object is represented by a collection of voxels (Fig. 3.5 (c)). Pointshell objects are represented by a set of center point samples of a voxmap and their associated inward pointing normals, collectively called a pointshell (Fig. 3.5 (d)) [13].



(a) Solid object **(b) Wire-frame object** **(c) Voxmap object** **(d) Pointshell object**

Figure 3.5: Solid, wire-frame, voxel, and pointshell representation

When the pointshell object's pointshell penetrates the tangent plane of a voxmap object's center point, a depth of penetration is calculated (Fig. 3.6). The penalty forces are calculated using the depth of penetration and are summed to give a resultant force and torque. By the nature of volumetric representation, voxmaps are insensitive to surface imperfections such as gaps or cracks that are smaller than the voxel size (McNeely, 1999 #106).

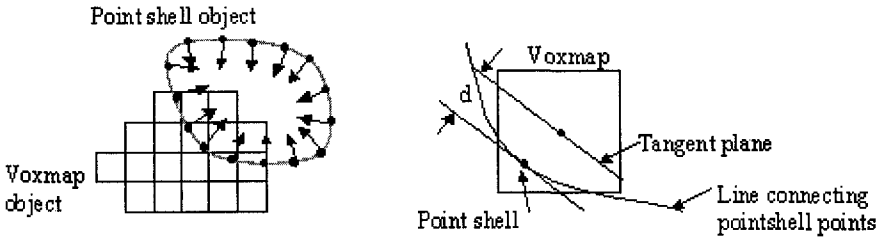


Figure 3.6: Point shell colliding with a voxmap

3.4.1. Physically-based modeling in VPS

Besides the forces emerging from the virtual coupling, those resulting from a 6 DOF spring-damper system between the dynamic object and the virtual hand also act on the dynamic object. VPS contains a part interaction library called PBM (Physically-Based Modeling). This library models part interaction using rigid body dynamics principles. The basic equation of motion used to describe the model reaction is the Newton-Euler equations of motion:

$$\begin{aligned} M\ddot{\vec{x}} &= \vec{F}(t) \\ I\dot{\vec{\omega}} + \vec{\omega} \times I \cdot \vec{\omega} &= \vec{N}(t) \end{aligned} \tag{2}$$

where x is the linear displacement, $F(t)$ is an linear force which is a function of time (t), M is an object's mass, ω is the angular velocity, I is the moment of inertia, and $N(t)$ is the angular force. VPS PBM solves the Newton-Euler dynamic equation numerically by using finite difference approximations to describe rigid body dynamics. The resulting position and velocity offsets between the virtual hand and the dynamic object change the force and torque in the coupling, which are then fed back to the simulation and displayed to the user [57]. The appropriate time marching step, linear/angular spring constants, and linear/angular damping coefficients need to be defined for stability. The computational cost of the physically-based modeling method is relatively more expensive than that of the constraint-based geometric modeling, and numerical instability can also be

an issue. However, physically-based modeling techniques enable the realistic dynamic manipulation of a complex rigid object. The theory of physically-based modeling has been studied extensively [42, 58].

VPS PBM can be used in real-time simulation for a time-critical application. It differs from collision and proximity detection methods, however, in that those methods detect exact and proximity collisions, while PBM generates a collision response and calculates subsequent motion. PBM does not provide highly detailed collision information because the time required for such calculations would be too great for the use of a force feedback device.

3.4.2. Object speed

When the user grabs and moves a part in the application, the part's speed should be fast enough to follow the natural motion of a human hand. If the application cannot keep up with the user, the user may see the object lagging behind the hand movement. More significantly, this object-dragging effect generates strong spring forces and torques in the haptic device, which further prevents natural human motions in the virtual environment.

In VPS, two factors affect a part's speed: voxel size and VPS PBM update rate. VPS allows the user to define a maximum time period, called *maxTime* for the part interaction calculation. The CPU attempts to perform a part interaction calculation until it either solves the calculation within *maxTime* or decreases the maximum distance that an object can move in a frame in order to keep calculation time less than *maxTime*.

Initially, the maximum distance an object can move while still maintaining part-to-part interaction is defined in VPS as *maxTravel*, which can be as high as approximately 32 times the voxel size. VPS has a chunk and hyperchunk tree system similar to a bounding box tree [30]. A chunk is a cubical collection of voxels, and is a mid-level node in the voxel tree. A hyperchunk is a cubical group of chunks, and is a top-level node in the voxel tree. When a part is away from any contact, *maxTravel* is equal to ½ x hyperchunk. One hyperchunk is a 64 x 64 x 64 collection of voxels under default settings in VPS. Collision detections at the chunk level are performed when the hyperchunk overlaps with another object's hyperchunks. When the part is close enough to other objects, the smallest unit (voxel) is used in order to detect more accurate collisions. In this case *maxTravel* is ½ x VoxelSize. This method prevents a part from penetrating other objects in the environment. However, *maxTravel* can be smaller than ½ x VoxelSize when a low-capacity CPU or a large number of voxels is being used. If the CPU is incapable of executing the required number of part interaction

calculations within *maxTime* because there are too many voxels, the application reduces *maxTravel* until the calculation can be done within *maxTime*. Therefore, *maxTravel* is expressed as:

$$maxTravel \leq 32 \; x \; VoxelSize \tag{3}$$

Since *maxTravel* is the maximum distance that a part moves per frame, the maximum speed with which an object can be moved is defined as *maxSpeed*, which is expressed as:

$$maxSpeed = maxTravel \; x \; PUR \tag{4}$$

where PUR is the physics interaction loop update rate.

3.4.3. Merging verses pair-wise collision detection

VPS PBM has two main queries: 'VpsPbmCollide' and 'VpsPbmEvolve'. 'VpsPbmCollide' calculates reaction forces between two objects and 'VpsPbmEvolve' generates a new position for the dynamic object based on reaction information generated from 'VpsPbmCollide'. The physics calculation flowchart is illustrated in Figure 3.7.

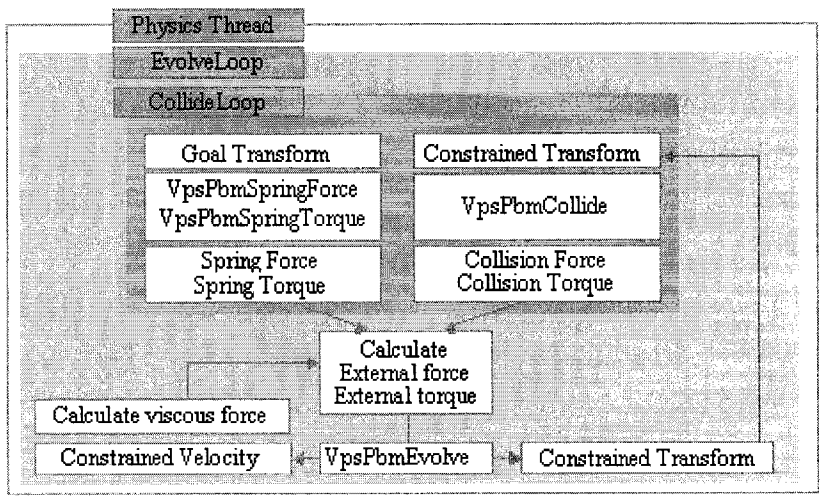
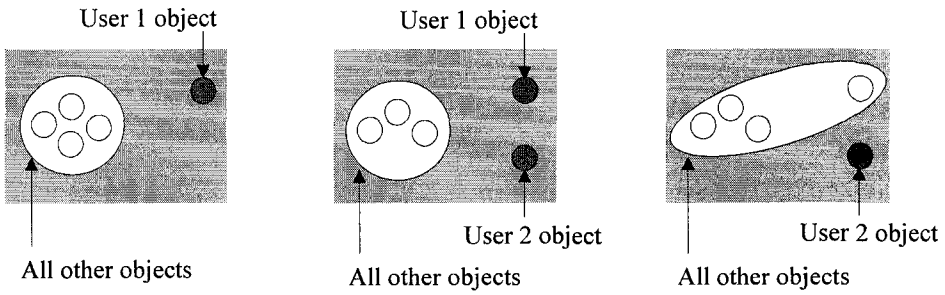


Figure 3.7: Physics calculation flowchart

Since VPS is a pair-wise collision detection package, 'VpsPbmCollide' is called to test for collisions between the dynamic object and all other objects in the environment. 'VpsPbmEvolve' is also called for each dynamic object. Pair-wise collision detection means that collision checks on the order of $O(N^2)$ are needed for N objects. Although a single 'VpsPbmCollide' calculation requires time on the order of milliseconds, total collision checking time will grow dramatically as the number of parts increases.

VPS uses a merging process in order to decrease the number of collision checks. Merging is the process of combining the voxels from groups of parts to form one object. It reduces the amount of pair-wise testing and improves the performance by considering the merged parts as a single object. One drawback to using the merging method is that the merging task can require up to several seconds, depending on model sizes (i.e., number of voxels). Furthermore, the merging process should be performed whenever any user grabs an object in a scene, and occasionally when one is released. Figure 3.8 illustrates when to merge parts for two users. Red dots are the dynamic parts that are grabbed by users, and the white dots are the static parts. The first merging process is needed when the first user grabs an object (Fig. 3.8 (a)). The scene has now two objects. The second merging process is performed when the second user grabs an object (Fig. 3.8 (b)). The scene has three objects until the third merging process is performed when one of the users releases his/her object (Fig. 3.8(c)).



(a) First user grabs **(b) Second user grabs** **(c) First user releases**

Figure 3.8: Merging times

In order to reduce the delay that occurs whenever one user grabs or releases an object, a performance check was performed. A pair-wise PBM check was executed, and the change in calculation time examined as the number of objects was increased. The total number of PBM checks, *N*, is expressed as:

$$N = (T - 1) \times D \tag{5}$$

where T is the total number of parts and D is the number of dynamic parts.

In order to examine the extent to which changing the number of dynamic objects affects the performance time, a performance test was conducted using from 1 to 20 dynamic parts. Figure 3.9 shows the test model setting. When the application is started, all dynamic parts fall down, bounce slightly on the table box, and collide with other dynamic parts.

In order to maintain natural human motion as users assemble parts in the virtual environment, a minimum speed of hand movement was chosen as 0.6096m/s. The voxel size for the assembly parts in the test was set to 0.6096cm (0.02ft). The voxel size is a variable that is selected based on the complexity of the geometry and the accuracy desired. In this simulation, collisions within 1.0668cm of the surface are detected. According to equation (4), PUR should be over 200Hz to meet the 0.6096m/s *maxSpeed*. The 200 Hz is called the target frame rate.

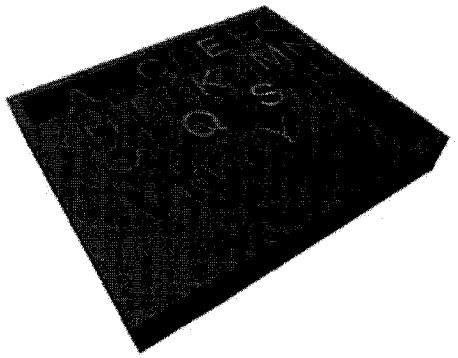


Figure 3.9: Performance test model setting

When the number of dynamic parts exceeds 16, the time consumed for physics calculations increases exponentially. Figure 3.10 is a graph of physics interaction loop update times versus number of dynamic parts, for various quantities of voxels. The graph shows that the number of dynamic objects has a greater effect on application performance than does the total number of voxels.

According to the performance test, VPS cannot maintain the minimum speed (0.6096m/s) with a maximum 1.0668cm offset if a scene has more than 15 dynamic objects. If it is assumed that one user grabs one part each, VPS can have 15 multiple users at the same time without degrading the performance (0.6096m/s) for a 0.6096cm (0.02ft) voxel size.

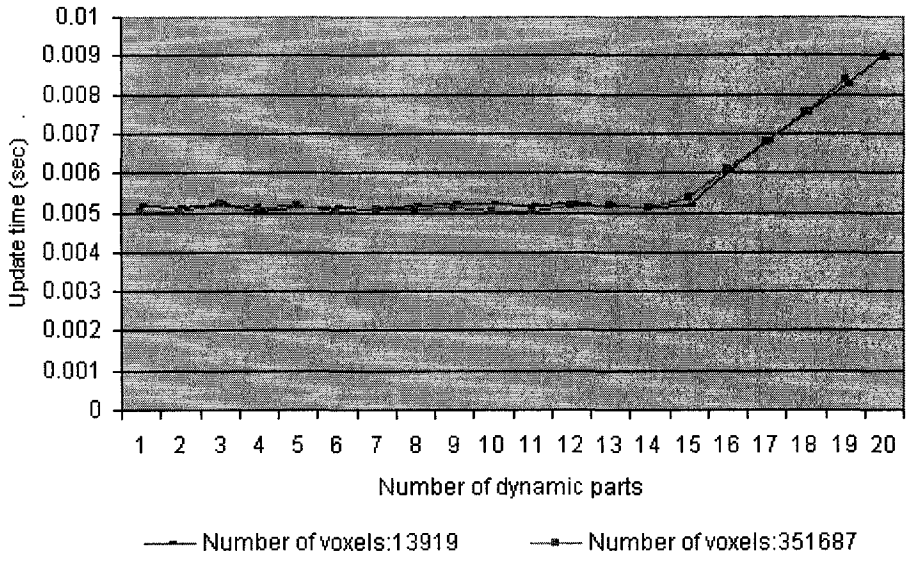


Figure 3.10: Change in update time with increasing number of dynamic parts and with different numbers of total voxels

3.5. Network Architecture

A main feature of NHE distinguishing it from single-user applications is the use of networking to exchange packets of information among the multiple users. These users communicate with each other via a network in order to share the dynamic states of other users who are participating in the application. Dynamic states include information such as a user's position, orientation, and action keys. These data are generated in each host and relayed to the other hosts. In an ideal situation, the dynamic states are mirrored in the other hosts instantly.

3.5.1. Consistency-throughput tradeoff

However, in every network communication some tradeoff exists between having real-time updating and keeping absolute consistency. It is a fundamental rule concerning network communication.

> "It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state."

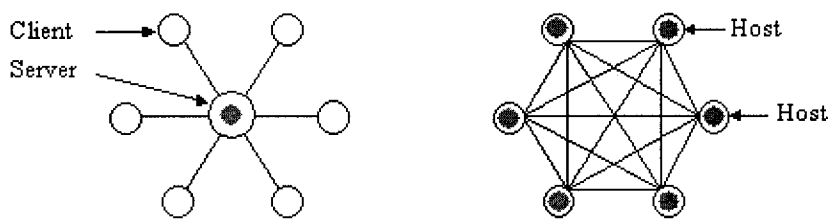
This rule is called the 'Consistency-Throughput Tradeoff' [50]. The network latency, or network delay, is the amount of time required to transfer a bit of data from one point to another. Network delay is mostly composed of propagation, transmission, and processing delay. Network latency is due primarily to propagation delay as the request message travels from one station to the end host. This delay is fairly constant if there are no route changes. Besides propagation delays, latency may also involve transmission delays, which are the time actually transmit the packet into the link. It depends on properties of the physical medium, such as modem or 1Gb/s connection. Transmission delays are typically on the order of microseconds or less in practice. Processing delays is the time required to examine the packet's header and determine where to direct the packet. Processing delays are typically on the order of microseconds or less. In order to have absolute consistency under the network delay situation, all users have to wait for the longest delay time and perform simulations at the same time in all hosts. Because of this waiting time, dynamic shared states cannot be updated frequently. On the contrary, when each host updates and simulates at its top speed, each will then work independently. While this situation results in near real-time simulation, it lacks some consistency of collaboration. Consequently, network communication can be either a dynamic world or a consistent world, but not both. Various network architectures – for example, client-server, pure peer-to-peer, combination of client-server and peer-to-peer, multicast, and broadcast – have different features and manage dynamic states differently. In order to decide which architecture is appropriate for this research, it is necessary to discuss how each method manages data flow and communication between remote users.

3.5.2. Client-server architecture and peer-to-peer architecture

Client-server architecture is a network architecture in which each computer or process on the network is either a client or a server. Servers are powerful computers or processors dedicated to managing heavy calculations and network traffic. Clients are PCs or workstations on which users run applications. They rely on servers for resources, and send packets to other clients through a server. Packet management is easy in client-server architecture. A server can sort all packets and decide which packet is to be sent to which client. Packets also can be compressed on a server side and sent to clients less frequently. This server-side packet management can reduce the network loads and improve performance. In addition, administrative tasks can be performed, such as creating a secure login process, accounting for secure information, or metering for time spent [50]. Simulations can be performed on the server side and have only the results distributed to the clients. Figure 3.11(a) illustrates client-server architecture, where the dot inside the server represents where a simulation or heavy computation is performed.

Peer-to-peer architecture is a type of network in which each workstation has equivalent capabilities and responsibilities. Peer-to-peer architecture has no intermediate server introducing further transmission delay between hosts. A sender host sends packets to the other hosts directly. Multicast and broadcast can be categorized as subsets of the peer-to-peer system in the sense that there is no server involved. Broadcast sends packets to everyone on the network. Although technically it is more effective than a one-to-one data communication method, broadcast schemes cannot be used because most routers block broadcast packets to prevent broadcast storm. Multicast sends packets to a selective group. A simple example of multicast is sending an email to a mailing list. Multicast, however, is still being developed and uses the UDP (User Datagram Protocol) protocol, which can cause packet loss. There is no retransmission process in multicast protocol when a packet is lost. Multicast-capable routers are not universally deployed yet and multicast packets cannot go through a firewall. Therefore, both broadcast and multicast are not an option for a network environment of this research.

Pure peer-to-peer architecture is defined as an architecture in which one sending host sends a packet to one receiving host. Figure 3.11 (b) illustrates pure peer-to-peer architecture. In this architecture, simulation or management of various states are processed in each host machine. The fact that peer-to-peer architecture does not depend on a server connection means that the network connection lasts as long as at least one host is alive. In this case, all clients handle environment information independently and experience less network delay than in client-server architecture. However, as the number of hosts increases, the connection structure becomes more complex. In addition, there is no main source of control to maintain administrative tasks.



(a) Client-server architecture **(b) Peer-to-peer architecture**

Figure 3.11: Various network architectures

3.5.3. Network structure for virtual assembly

The NHE developed as a result of this research has two network communication modules: local network and global network (Fig 3.12). The local network connects a PC machine to the local virtual

environment system, which controls graphics updates and transfers force feedback information. Each PC has a haptic device connected to it. The global network connects multiple virtual environment systems for collaboration tasks. Global network communication consists of a mixture of client-server and pure peer-to-peer architecture, while local network communication is pure peer-to-peer architecture. The global network is built with TCP (Transmission Control Protocol) and the local network is formed with UDP. TCP establishes a connection between two hosts and streams data packets. It guarantees packet delivery, flow control, and error correction. UDP is a connectionless protocol. Like TCP, it runs on top of an IP (Internet Protocol) network. UDP provides faster communication than the TCP protocol, but has less error correction and no flow control.

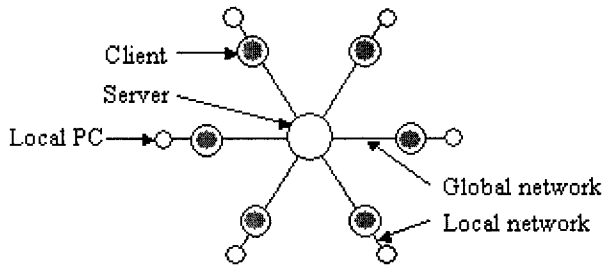


Figure 3.12: NHE network architecture

A combination of peer-to-peer and client-server architecture has been implemented here for the global network in order to maintain stability and simple network structure. The server generates a new thread when a new client accesses the server for the first time. The server then sends a user list back to the client in order to assign a user number to the client. This is called a user list handshake. After the handshake, clients send their position and orientation information to the server and receive other users' information via the server. When a client disconnects, its thread is automatically removed (Fig 3.13).

Physics calculations run independently on each client. Only PHANToM position and orientation information is transmitted to the server. The server updates user list information and distributes it with the packet to other users. Since the connection speeds are different for each client, inconsistency problems still exist. In addition, each client requires a high CPU power to calculate physics simulation.

Haptic rendering is performed on a dedicated processor in order to guarantee the haptic update rate requirement of 1kHz. The local network is used to connect the dedicated processor (PC) and the virtual environment (SGI UNIX).

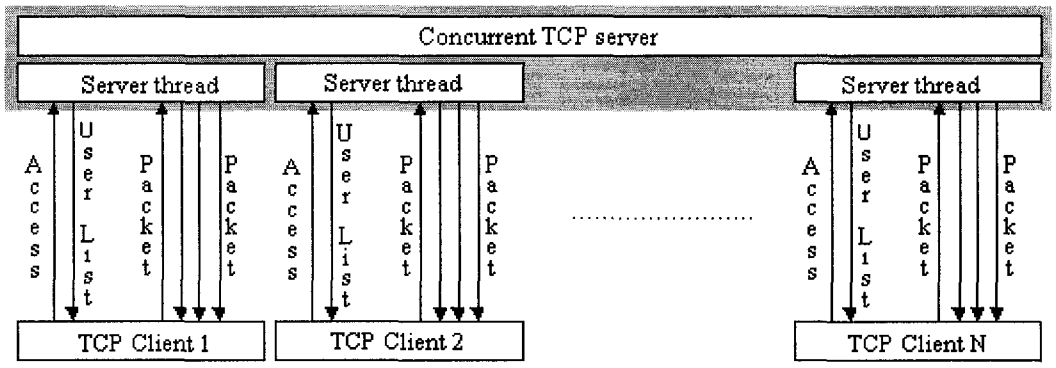


Figure 3.13: Global network TCP client-server architecture

3.6. Multithreading Structure

Threads are often defined as lightweight processes. Threads are very small compared with processes since processes require their own resource bundle, and threads share resources. All threads associated with a given task share the task's resources while a process contains both an executing program and a bundle of resources. Thus a thread is essentially a program counter, a stack, and a set of registers. Threads give programmers the ability to write concurrent applications that run on both uniprocessor and multiprocessor machines. If a machine has more than one processor, a second processor picks up execution of the background thread while running at the same time that the first processor continues executing the application. If a machine has only one main processor, multithreading will give one thread a small amount of CPU time, then switch to another thread fast enough to appear as if it multitasks.

3.6.1. Examples of the multithreading structures for NHE

The NHE has several loops running separately at different update rates. These include the graphics, physics interaction, haptic rendering, local network, and global network loops. The graphics loop update rate is set to a maximum of 48Hz. If graphics data are very intense, then the graphics update rate will be lower than 48Hz. The haptic rendering loop update rate must be around 1kHz to achieve stability and high disturbance rejection in the haptic device. The physics interaction, local network, and global network loop update rates vary based on physics and network conditions. Physics conditions are determined by the number of objects, part sizes, and voxel size. Network conditions are dependent upon network delay and number of users.

In order to maintain different update rates effectively, two separate threads must run on the PC: the haptic thread and the local network thread. The haptic thread updates the haptic loop and the local network thread communicates with the local network loop in the virtual environment. Three loops running on the virtual environment side are the physics interaction, local network, and global network loops. The graphics loop updates simulation graphics on the projectors. The physics interaction loop runs collision detection packages and physically-based modeling simulation. When a part collides into other objects, the physics interaction loop calculates the penetration depth and reaction force and torque information. This reaction force and torque information is used to calculate the next position and orientation of the part. Because of this situation, when the physics interaction loop becomes slow, the whole simulation slows down. While the physics interaction loop updates the objects' positions, the network loops refresh users' hand positions. The local network updates the local user's hand position and orientation and the global network updates the position and orientation of the other users on the network. Any slowness of the local network loop can lead to discontinuous force feedback and instability of the haptic device. The ideal update rate of the local and global network loops would be that of the physics interaction loop. If the local network or global network loops are faster than the physics interaction loop, they exchange the same data packets from the physics interaction loop multiple times over the network. Besides being a waste of bandwidth and CPU, the fast update of the network loops may slow down the update speed of the physics interaction loop because the network loops and physics interaction loop share information such as hand position and orientation, reaction force and torque, and action status.

Four thread structures for use in the virtual environment side were designed and examined (Fig. 3.14). In Figure 3.14 (a), the local and global network loops are in one thread separate from the physics and graphics thread. Because this architecture maintains the update rates of the users' hand position and orientation independently of the physics and graphics update rate, the physics interaction loop can perform optimally whether the local or global network thread is relatively fast or slow. The drawbacks of this architecture are the local network slowdown due to global network delay and the waste of bandwidth and CPU. Since the local network loop is in the same thread as the global network loop, the network delay in the global network slows down the local network, and vice versa. If the local network becomes slow, the user feels discontinuous force even though the physics thread calculates smooth force curves with a fast update rate. On the contrary, if the network thread is much faster than the graphics and physics threads, it is, above all else, a waste of bandwidth and CPU capacity because the extra bandwidth can do nothing to speed up the simulation. Another aspect of the application that further reduces the physics update rate is the use of a mutex, which is a program

object that allows multiple program threads to share the same resource (such as file access), although not simultaneously. When a program is started, a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource. In the situation described above, the mutex for the data sharing between the physics and network threads will decrease the speed of the physics thread when the update rate of the network thread is too high.

The easiest way to synchronize the physics and network loops is to put them into the same thread as in Figure 3.14 (b). With this architecture, the system optimizes the tasks by transmitting the data through the network only when they are updated in the physics interaction loop. However, if there is a delay in either the local or global network, it slows down the thread speed and results in physics simulation slowdown. In order to solve this problem, the global network and the local network are each built in a separate thread, as shown in Figure 3.14 (c). This architecture separates the physics, local network, and global network loops from each other in order to prevent delay in one loop from affecting the other threads. A drawback of this structure is that the more complicated thread structure requires more mutex objects, which increases the overhead in each thread.

In order to simplify the thread structure and optimize performance, the local network loop is combined into the physics interaction loop and the global network loop is built in a separate thread. This structure is shown in Figure 3.14 (d). In this structure, the local network always feeds the newest data from the physics interaction loop to the haptic rendering loop on the dedicated PC. In this way, the physics update speed is independent of the global network update speed. It is possible, however, for the local network to slow down the physics interaction loop when the local network has any network delay. Therefore, this architecture relies on a negligible local network delay. Update times for the architectures illustrated in Figures 3.14 (b) and 3.14 (d) are investigated in Chapter 4.

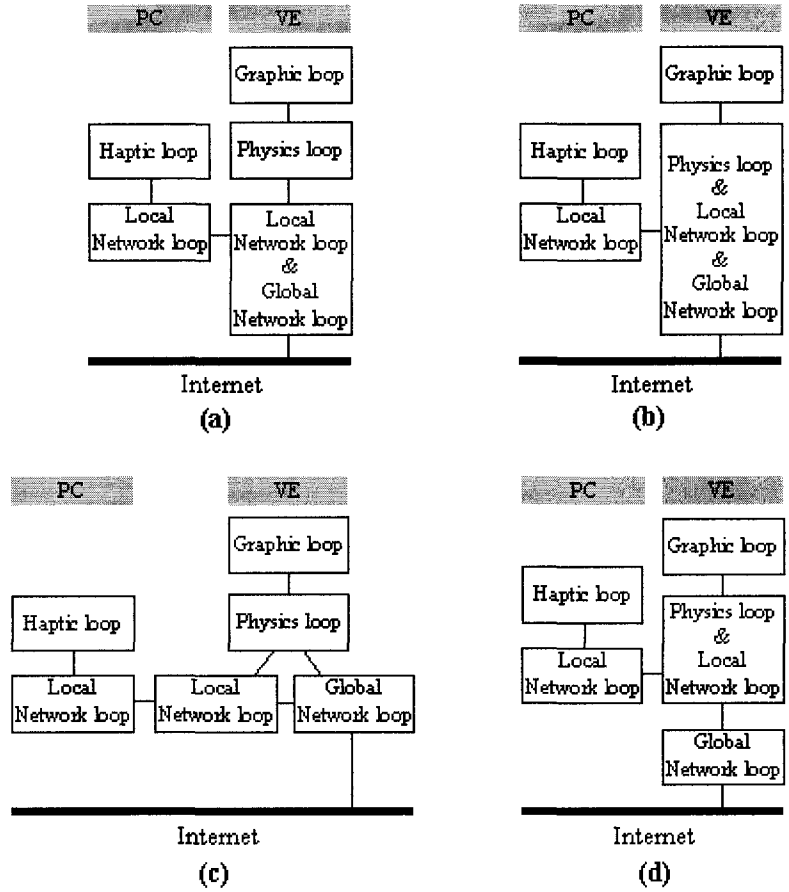


Figure 3.14: Various threads structures in virtual environment

3.6.2. Threading speed optimization

Since the physics/local network loop and the global network loop share some resources, one thread's update rate affects the other thread's update rate. The physics/local network loop should be maintained at as high a speed as possible, since the higher the physics update rate is, the smoother and faster the simulation will be. The global network transports the data generated from the physics/local network loop to users in other NHE's. If the global network update rate is higher than that of the physics/local network update rate, then it may slow down the physics/local network loop, in addition to being a waste of bandwidth and CPU. The global network speed can be slowed down by using a technique called barrier synchronization. A barrier synchronization is a logical point in the control flow of an package at which all the members of the subset of the processes must arrive before any of the processes in the subset are allowed to proceed further [59]. Barrier synchronization is applied

inside of the physics/local and global network loops when the global loop runs at a faster speed. This increases the physics interaction loop speed by decreasing mutex conflicts and preventing waste of bandwidth and CPU. This technology should not, however, be applied when the physics/local network loop is running faster than the global network loop. This situation would slow the physics/local network loop and result in a sluggish simulation. Update rates with and without barrier synchronization are measured and analyzed in Chapter 4.

3.6.3. Multithreading structure

The NHE uses the threads architecture illustrated in Figure 3.14 (d) in order to simplify the threads structure and isolate the physics/local network loop from the global network loop. The local network delay was assumed to be negligible, since the dedicated PC for the haptic device is usually physically close to the virtual environment machine. The user's position and orientation variables are transferred from the haptic device to the network thread. When the virtual environment machine gets those variables in its physics and local network thread, it calculates reaction forces and a new position and orientation. The new data from this physics and local network thread are passed to the graphics thread to update the display, and to the global network thread in order to communicate with other users on the network. The reaction force and torque variables generated from the physics interaction loop go to the haptic thread through the local network to operate the haptic device. A more detailed diagram of this thread structure can be found in Figure 3.15.

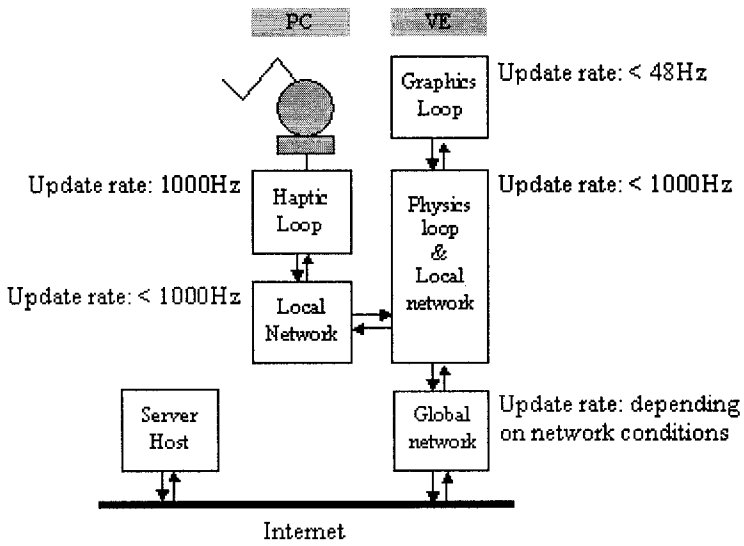


Figure 3.15: NHE threads structure

3.7. Object Synchronization

A fundamental goal of the NHE is to provide users with the illusion that they are sharing the same space and interacting with objects in real-time. When a user picks up an object and moves it, the other users need to see the object's current position and movement in their own virtual environments. If this object synchronization is broken, the users are no longer sharing the same time and space. This inconsistency stems from two main issues: CPU latency and network latency.

3.7.1. CPU latency

Different CPU speeds in network hosts produces CPU latency in peer-to-peer network architecture. Hosts in peer-to-peer architecture receive dynamic states – hand position and orientation – of other users through the network and run simulations independently based on the information. Because CPU power varies in each host, one host may end the simulation while another is still running the simulation. This is known as CPU latency.

In the global network, peer-to-peer architecture is used for physics simulations and client-server architecture for transferring dynamic states. The main advantage of peer-to-peer physics simulation in NHE is unconditional stability for a haptic device, while the main drawback is the simulation discordance problem, which is caused by CPU latency. This situation becomes worse when the simulation is computationally intense. The NHE physics simulation consists of three main parts: collision check, penetration depth calculation, and a 6 by 6 inverse matrix calculation for the Euler numerical difference method. For example, if the PUR is 500Hz with 4 dynamic parts out of a total of 10 objects, 18000 instances of collision and penetration depth checks and 2000 instances of 6 by 6 inverse matrix calculations per second would be required. This is a very intensive computation. Since PUR varies with the CPU power, a shared object in a distributed computing system may have different speeds or positions on different clients.

3.7.2. Discordance due to CPU latency

In VPS, the maximum distance that the object can move per frame is defined as *maxTravel* (see Chapter 3.3). *maxTravel* serves two purposes. For one, it has an absolute upper bound of ½ x VoxelSize (Eq 3) in order to prevent points from skipping over the penalty-force region and penetrating into the object's interior when two parts are close each other. The other purpose of *maxTravel* is to limit *maxSpeed* (Eq 4), so that available processing power can be used to keep predicting future collisions. Using a point's maximum speed, VPS can predict how many frames will elapse before a contact will possibly occur. This can help avoid unnecessary collision tests for

pointshells that are not near contact. In general, *maxTravel* is adjusted on a frame-by-frame basis. It is dependent on the CPU power since it varies with the number of points that the CPU can test per frame and the number of points that is mandatory in each frame. *maxSpeed* is a function of both *maxTravel* and the PUR. For example, assume there is an object moving at its maximum speed in the NHE, which consists of a fast host machine (Host 1) and a relatively slower host machine (Host 2). In this case, an object in Host 1 moves faster than the same object in Host 2. Figure 3.16 illustrates this discordance issue. The virtual spring-damper system is removed between the user's hand and the object when the object is released, so the object will be in different positions and orientations in Host 1 and Host 2 – the result of inconsistency of dynamic states.

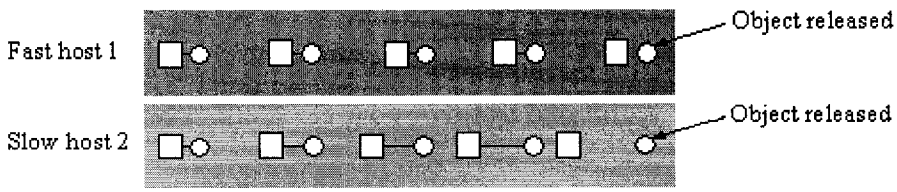


Figure 3.16: Discordance due to CPU latency in a fast host

3.7.3. Network latency in the global network

The network latency in the global network represents a bigger challenge in client-server architecture than in peer-to-peer architecture because latency may cause not only time lag, but also instability of the haptic device, in the form of excessive vibrations and reaction forces. Without the presence of delay, reaction forces are calculated and applied to the haptic display instantly. However, when there is some delay, a time lag will occur and allow deeper penetration depth that the haptic display (on the client side) cannot handle. In this case, the excessive vibrations will stop the haptic simulation.

In NHE client-server architecture, the physics are calculated on a server, and there is a time lag until the reaction force is transmitted through the network. In order to avoid this problem, a combination of client-server architecture and peer-to-peer architecture can be used. Physics interaction calculations are performed on each client (peer-to-peer architecture) while the user's hand position information is transferred through client-server architecture. This combination provides unconditional stability for a haptic simulation. However, the different update frequencies, which are caused by network latency on each host machine, results in the discordance issue.

Assume that there are two hosts connecting and communicating via a server where Host 1 has 1ms network delay while Host 2 has 5ms network delay. The user moves his/her hand in Host 1. The longer delay time means a less frequent update. Because Host 2 picks up the user's hand motion only once while Host 1 updates his/her motion 5 times, data in Host 2 will not be as smooth as in Host 1. This will also cause Host 2 to skip some motions that would be shown in Host 1. Figure 3.17 illustrates the user's hand motion trajectories in the two hosts.

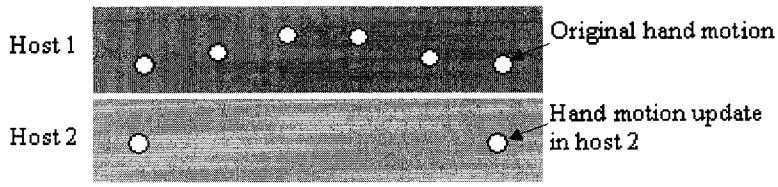


Figure 3.17: Hand motion in two hosts

3.7.4. Discordance and global network latency

Synchronization can be broken in multiple hosts because of the frequency difference caused by global network latency. When the user releases the grabbed object in Host 1 before Host 2 updates the user's hand motion, the object carried by the user will be located in different positions on the two hosts. Figure 3.18 illustrates this object position discordance issue.

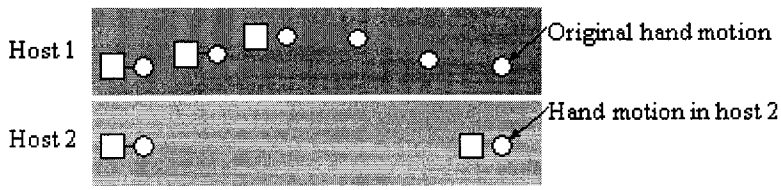


Figure 3.18: Discordance due to global network latency

3.7.5. RNR method

The simplest way to solve the network latency problem is to apply the longest network delay time to all hosts at every frame ahead of the actual data exchange. This method synchronizes all network speeds with the longest delay time. However, it cannot be used to fix the discordance due to CPU latency because the CPU latency time is not consistent in each host. The CPU latency time varies

according to the number of dynamic parts, the number of users, model complexity, voxel size, and so on.

In order to cope with this problem, Jordan et al. suggested that each user should complete tasks very slowly to maintain synchronization in peer-to-peer architecture for their particular haptic collaboration environment [20]. Although this may resolve the discordance problem to some extent, it restricts the user's freedom in a virtual environment.

In order to avoid this speed restriction of users' movement while still maintaining synchronization, an additional process, called the "Released-but-not-released" method, is proposed for this research. The proposed solution is to build the spring-damper system between the object's last position and the current object position until the object moves to the position and orientation where the original object is stopped. Using this method, the objects will be located at the same position in all network machines after a certain time period. Figure 3.19 shows how the RNR method synchronizes an object position in different CPU latency conditions. The upper figures in Figure 3.19 show the object movement on a fast CPU host, while the lower figures show the same object movement on a slow CPU host. When the object is released, the spring-damper system is removed. A new spring-damper system between the original object position (on Host 1) and the object's last position (on Host 2) is generated on Host 2, which drags the object to the original object's position and orientation. Then the object will be at the same position and orientation on both hosts.

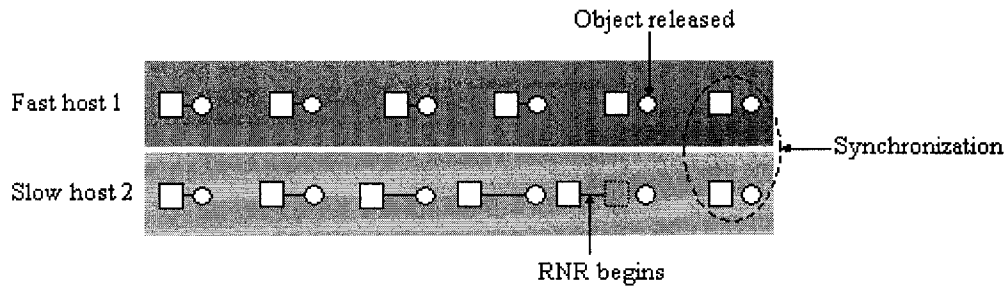


Figure 3.19: The RNR method synchronizing dynamic state between a fast CPU and a slow CPU

Figure 3.20 illustrates the RNR method running under different network latency conditions. When an object is released in different positions on multiple hosts because of different network latency conditions, Host 1 sends the last position and orientation information to Host 2. The rest of the procedure is the same as in the case involving different CPU latencies, as described above.

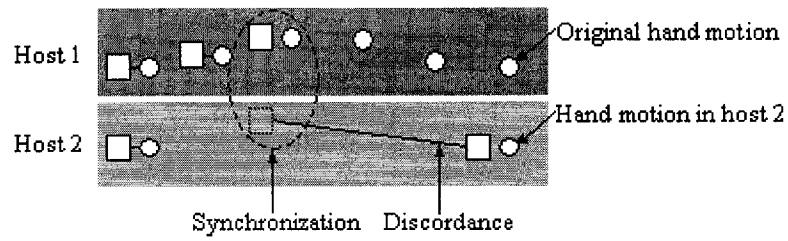


Figure 3.20: The RNR method synchronizing dynamic state in between a small network delay host and a large network delay host

3.7.6. Timeout mechanism

Although the RNR method was developed to synchronize object position and orientation in a distributed computing environment, synchronization can be broken in some cases. Figure 3.21 illustrates when the synchronization is broken in a CPU latency condition. The upper figures in Figure 3.21 show the simulation on the fast CPU and the lower figures in Figure 3.21 show it on the slow CPU. The user grabs the cube object, moves and releases it, while another object is interfering with its path. Because the RNR method assumes that there is no interference while an object converges to the sync position, the object position and orientation on the slow CPU cannot converge to the original position as it is on the fast CPU when another object blocks its way.

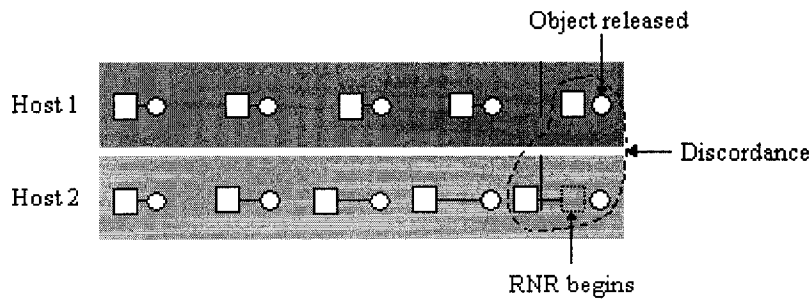


Figure 3.21: Broken synchronization with RNR method in CPU latency condition

Figure 3.22 shows when the RNR method cannot synchronize dynamic states under the network latency condition. If there is an obstacle in the object's path, the host with a small network delay can go around but the host with a large network delay may not be able to pass the obstacle. In this case, the RNR method will not be able to synchronize the object's position and orientation because the object is blocked by the obstacle on Host 2.

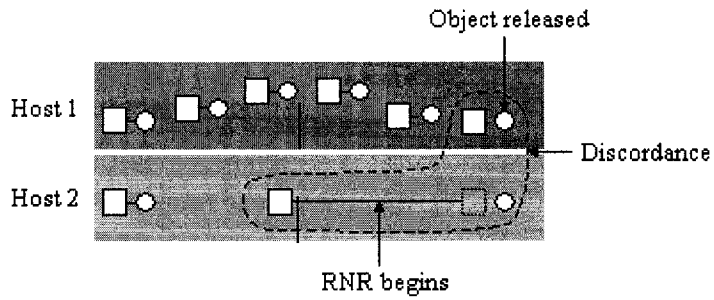


Figure 3.22: Broken synchronization with RNR method in network latency condition

A timeout mechanism is implemented to solve this problem. If the RNR mode is running for longer than a pre-defined amount of time (i.e. 10 seconds) and the object is still not in sync, the physics interaction is turned off until the synchronization is met. Thus the application can keep consistency in any situation.

CHAPTER 4. APPLICATION SETUP, RESULTS AND DISCUSSION

The NHE application was implemented and tested for three users with three PHANToM devices. The performance results were reported in order to measure the network efficiency and the physics interaction simulation speed, which is the maximum speed of a grasped object when it is moved and collided into other parts in the virtual environment.

4.1. Application Setup

A virtual reality application has been developed to allow multiple users on the network to collaboratively assemble 3D parts and feel the reaction force when an object collides into other parts. Three haptic devices (PHANToM Desktop, PHANToM 1.5, and PHANToM 3.0) were set up on their own LINUX PC machines, and each PC was connected to the virtual environment run by SGI machines. Figure 4.1 shows the device and environment setup. The PHANToM Desktop was set up on a PC connected to an SGI machine with a two-dimensional monitor display. PHANToM 1.5 was set up in one of the multi-pipe stereo projection environments, the C4, at the Virtual Reality Applications Center. PHANToM 3.0 was set up in the other multi-pipe stereo projection environment, the C6, also at Virtual Reality Applications Center.

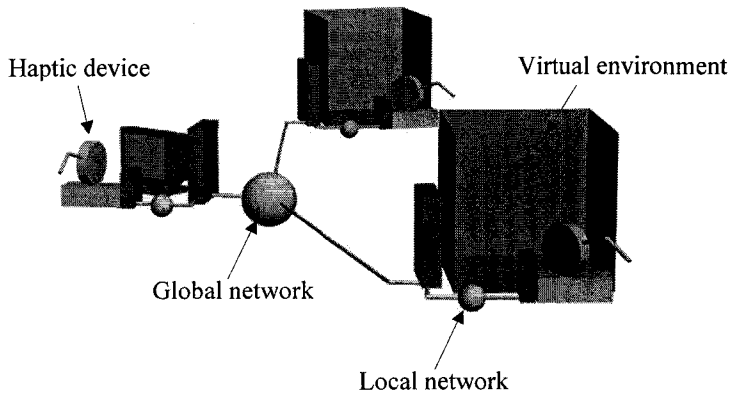


Figure 4.1: Device setup for the application test

4.1.1. Program use

Before running the NHE application, the user must create a configuration file in order to set up the input device, input model names, and voxel sizes. An example text configuration file is shown in Figure 4.2.

```
Haptic_device: 0 (1=yes/0=no)
Number_of_models: 8
axle_part01.3ds     -0.434  0.0     3.6   1.0   0.02
axle_part02.3ds     0.0     0.0     3.0   1.0   0.02
axle_part03.3ds     0.7     0.0     3.0   1.0   0.02
axle_part04.3ds     2.034   -0.2    3.2   1.0   0.02
axle_part05.3ds     2.413   1.437   3.4   1.0   0.02
axle_cab01.3ds      1.3  1.5     3.2    1.0    0.02
axle_cab02.3ds      0.1     1.5     3.2   1.0   0.02
axle_base.3ds       0       1.0     2.5   1.0   0.02
Barrier_synchronization: 1 (1=yes/0=no)
```

Figure 4.2: Example configuration file for networked haptic environment application

The first line sets up the input device. If it is set to 1, the NHE application expects the user to use a PHANToM force feedback device as the input hardware. If it is set to 0, the NHE application expects a mouse as the application input device. The second line of the configuration file is the total number of objects. The following lines contain the model file names and their initial positions and voxel sizes. Each model has its own voxel size. The last line of the configuration file is the barrier synchronization flag to toggle barrier synchronization mode in the physics interaction and global network threads. As addressed in Chapter 3.6, enabling barrier synchronization may decrease application performance in the presence of certain physics interaction conditions and global network delay. Physics interaction conditions include number of users, number of objects, sizes of object, and voxel sizes. It is recommended to turn the barrier synchronization flag on only when the global network loop update rate is higher than the physics interaction loop update rate.

Before running the NHE application the user must create two input files for each model: a graphics input file and a collision input file. Since the NHE application uses SGI OpenGL Performer™ as its graphics rendering toolkit, it can load any of the file formats listed in Appendix A. The VPS collision input file is a simple ASCII file format, ITRI, which includes the xyz components of all three vertices of the triangular polygons that define the face polygons of the solid model. An example of ITRI file is shown in Appendix B. The first line of the ITRI file is the number of triangles. The following lines contain vertex positions. A standard STL model file in ASCII format is converted into the ITRI format by using a simple conversion program. Using the ITRI file, VPS creates the voxel model in memory within the VR application at run-time.

The first user must run the network server application that manages and distributes packets to the clients before he/she runs the NHE application. The server application can be run either on the same

machine as one of the NHE applications or on a separate machine. While the network server application is running, the NHE application is executed with the server hostname, the global network port number, the local PC hostname, and the VR Juggler configuration files.

When the NHE application is executed, it first loads 3D model files for graphics and physics interaction. For this research a vehicle axle model from the Deere & Company Product Engineering Center (PEC) was used. Snapshots and specifications for each model are shown in Appendix C. After loading model files, the NHE initiates the global network thread, the physics interaction/local network thread, and the graphics thread in a sequence. The global network thread first connects with the server and performs the user-list handshake in order to determine how many users are on the network. Once clients receive the user list from the server, they begin transferring data packets of dynamic status to the server. If there is only one user logged into the network, the packets are not transmitted via a network server until another user logs in. The local network loop, on the other hand, is activated only when the user selects the haptic device option for the input device type in the configuration file. If the user selects the mouse option, the local network loop is not activated. Instead, the user moves the virtual hand model with mouse control.

When the users enter the virtual environment, they first see virtual parts on the virtual table. Each user moves their hand model by manipulating the haptic handle attached to the haptic feedback device. All users can see all virtual hands. The user can move the virtual hand model into the objects on the virtual table and grab an object by holding the stylus button on the haptic handle. When the user intersects his/her virtual hand model with an object, the object turns its color to green to signal to the user that this part is ready to be grabbed and moved. The users can move, release a part, and grab another one by pressing and holding a stylus button on the haptic handle. When two or more parts collide in the virtual environment, the objects turn their color to red and the reaction force is generated. The user feels those forces through the PHANToM devices. Figure 4.2 shows a simulation snapshot when three users are participating in the application.

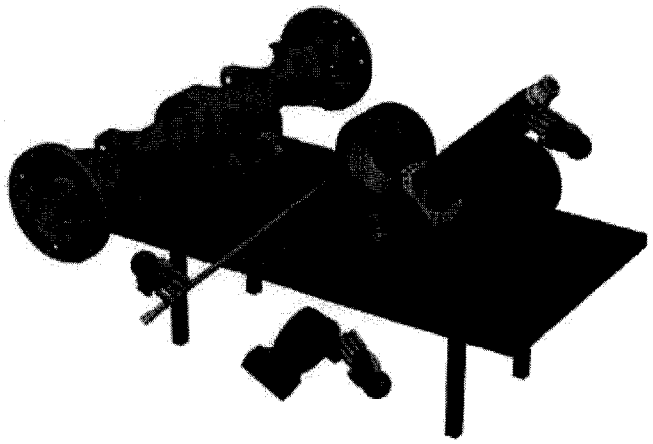
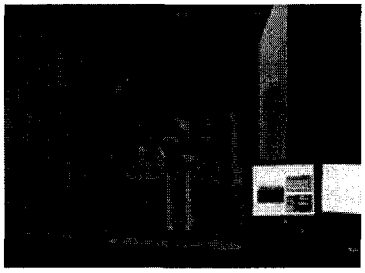


Figure 4.3: Simulation snapshot

The three users and their respective hardware setups are shown in Figure 4.3. Specifications of the machines and haptic devices used in each virtual environment are shown in Table 4.1.

(a) User 1



(b) User 2

(c) User 3

Figure 4.4: Hardware setups

Table 4.1: Hardware specifications

	VE #1	**VE #2**	**VE #3**
CPU quantity/MHz	2/195	16/194	24/400
Memory (MB)	384	1792	12288
Graphics board	MXI	Infinite-Reality	Infinite-Reality2E
Display device	Monitor	4 stereo projectors	6 stereo projectors
Haptic device	PHANToM Desktop	PHANToM 1.5	PHANToM 3.0

4.2. Performance Measurement for Example Model Set

Performance in this chapter is measured by the loop update times for each thread of the NHE application. The time periods were measured at the end of the loops. Performance of the physics interaction/local network and global network loops in the NHE were measured for the three virtual environments when all three users in the network grabbed objects and collided them into others at the same time.

The first objective of this section is to calculate the maximum and minimum speed of the simulation while three users connect and grab different object at the same time. The simulation speed is *maxTravel* multiplied by PUR (Eq. 4). The physics/local network update time and *maxTravel* were measured for both a small part (ring) and a large part (driveline housing) in order to measure the slowest and fastest speed of the simulation. Figure 4.5 shows the physics loop update time and *maxTravel* when the driveline housing is grabbed and moved from empty space and collided into the virtual table (axle base) in virtual environment VE#1, which is controlled by the slowest machine. Movement of the largest part on the slowest machine generates the slowest simulation speed. The slow simulation speed generates strong reaction forces and torques that make controlling the haptic device difficult. The fastest simulation speed, on the other hand, would therefore be given by measuring the physics loop update time and *maxTravel* in the fastest machine (VE #3) when the smallest part (ring) is grabbed and collided into the "axle base" model. Figure 4.6 shows the physics loop update time and *maxTravel* in VE#3. The numbers of voxels for each part are shown in Appendix C.

In all VE's, the part was moved close to the virtual table at about the 50^{th} loop and actually collided into the virtual table between the 110^{th} and 120^{th} loops. In Figure 4.5 and 4.6 *maxTravel* drops and the physics loop update time runs slightly higher around the 50^{th} and 120^{th} loops. These results show how the hyperchunk and chunk affect the object speed according to the distance between parts. In both VE#1 and VE#3, the hyperchunk and chunk determine the *maxTravel*, while *maxTravel* in VE#3 is higher than that in VE#1. Secondly the maximum and minimum speeds of the parts were calculated based on *maxTravel* and the physics loop update time which are shown in Table 4.2.

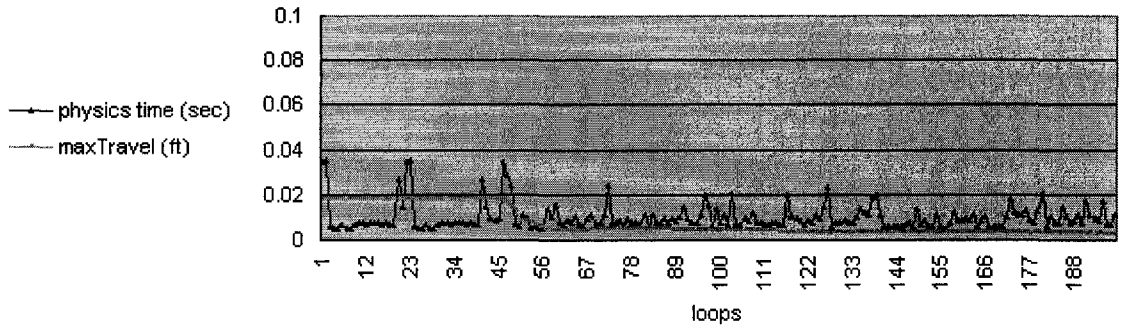


Figure 4.5: Physics update time and maxTravel of "housing" part when collided into "axle base" in VE #1

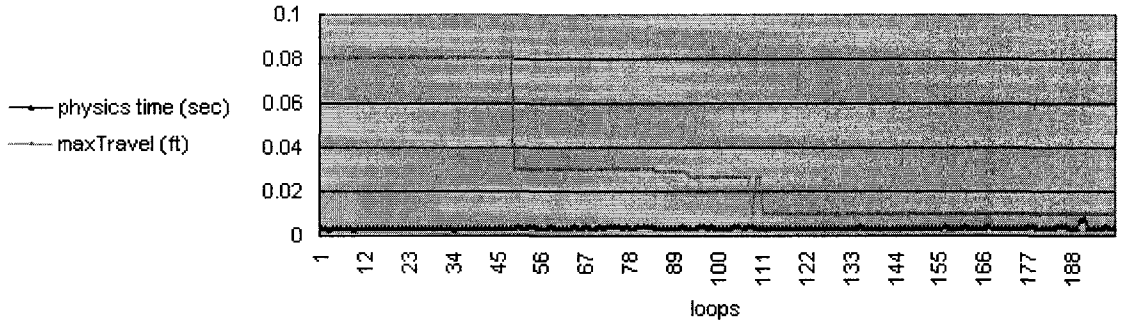


Figure 4.6: Physics update time and maxTravel of "ring" part when collided into "axle base" in VE #3

Table 4.2: Physics loop update time and *maxTravel* in each VE

	Object	Dynamic Status	Physics loop update time (sec)	*maxTravel* (cm)	*maxSpeed* (m/sec)
VE #1	Ring	Not colliding	0.005078	2.4384	4.8019
	Ring	Colliding	0.01416	0.2652	0.1873
	Housing holder	Not Colliding	0.004969	0.5563	1.1195
	Housing holder	Colliding	0.02316	0.1074	**0.0464**
VE #2	Ring	Not colliding	0.005247	2.4384	4.6472
	Ring	Colliding	0.006728	0.3048	0.4530
	Housing holder	Not Colliding	0.004415	0.9144	2.0711
	Housing holder	Colliding	0.008993	0.0904	0.1005
VE #3	Ring	Not colliding	0.003043	2.4384	**8.0131**
	Ring	Colliding	0.004164	0.3048	0.7320
	Housing holder	Not Colliding	0.003846	0.9144	2.3775
	Housing holder	Colliding	0.005119	0.1908	0.3727

The second objective of the section was to measure the effect of thread synchronization. In each VE, the same performance tests were performed with and without barrier synchronization in order to

determine how barrier synchronization affects performance in different hardware setups. The test action consisted of the "driveline", "ring", and "housing holder" parts being grabbed by each user and collided with each other. The average times for one update in both the physics/local network loop and the global network loop in each of the three virtual environments, without barrier synchronization, are shown in Table 4.3. The results with barrier synchronization are shown in Table 4.4.

The actual update times for 100 loops in VE #1 with and without barrier synchronization are shown in Figures 4.7 and 4.8. The update times in virtual environment #2 for the same conditions are shown in Figures 4.9 and 4.10, while those for virtual environment #3 are shown in Figures 4.11 and 4.12.

Table 4.3: Average time for one update in both the physics/local network loop and the global network loop in three virtual environments without barrier synchronization

	Average time for one physics/local network loop (sec)	Average time for one global network loop (sec)	Physics/local network update rate (Hz)	The other users' position/orientation update rate (Hz)
VE #1	0.0052	0.0107	192.31	93
VE #2	0.0043	0.0055	232.56	181.81
VE #3	0.0030	0.0017	333.33	588.24

Table 4.4: Average time for one update in both the physics/local network loop and the global network loop in three virtual environments with barrier synchronization

	Average time for one physics/local network loop (sec)	Average time for one global network loop (sec)	Physics/local network update rate (Hz)	The other users' position/orientation update rate (Hz)
VE #1	0.0118	0.0116	84.76	86.21
VE #2	0.0047	0.0048	212.77	208.33
VE #3	0.0024	0.0024	416.67	416.67

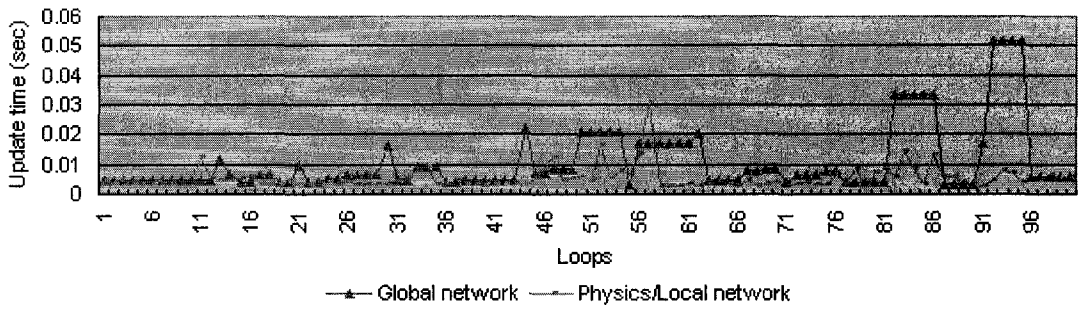


Figure 4.7: Global network & physics/local network update time in VE #1 without barrier synchronization

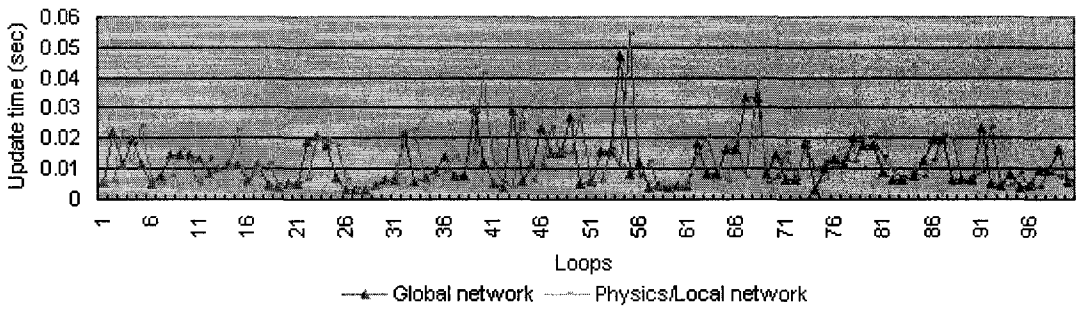


Figure 4.8: Global network & physics/local network update time in VE #1 with barrier synchronization

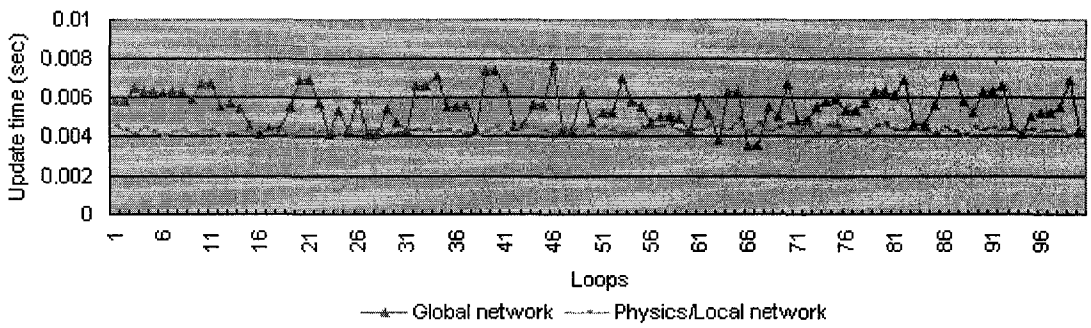


Figure 4.9: Global network & physics/local network update time in VE #2 without barrier synchronization

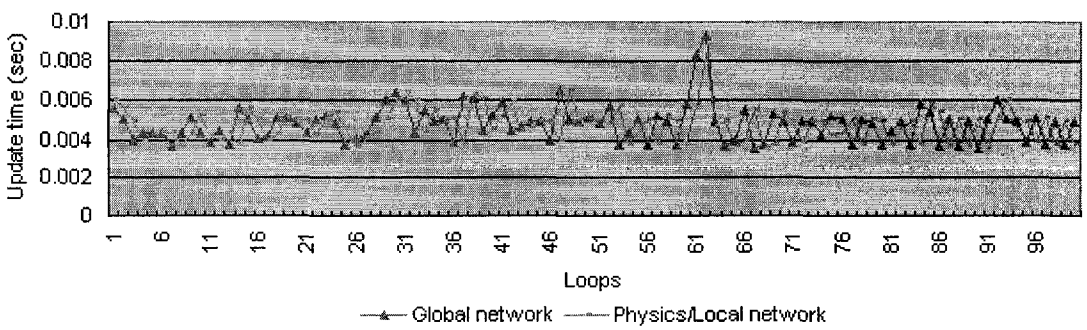


Figure 4.10: Global network & physics/local network update time in VE #2 with barrier synchronization

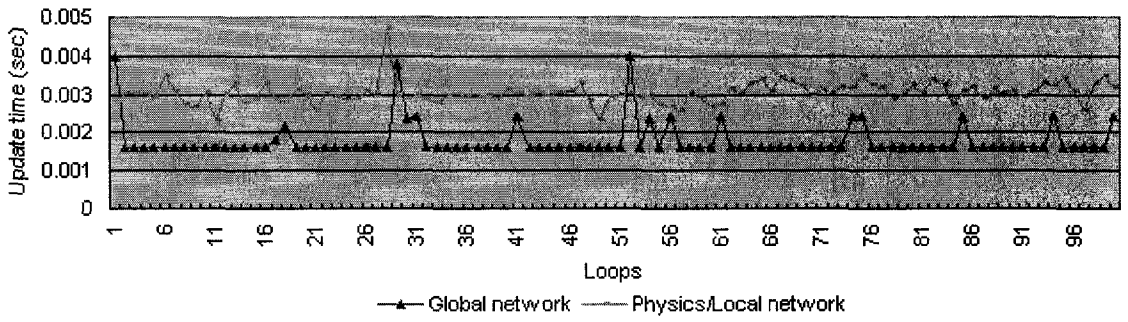


Figure 4.11: Global network & physics/local network update time in VE #3 without barrier synchronization

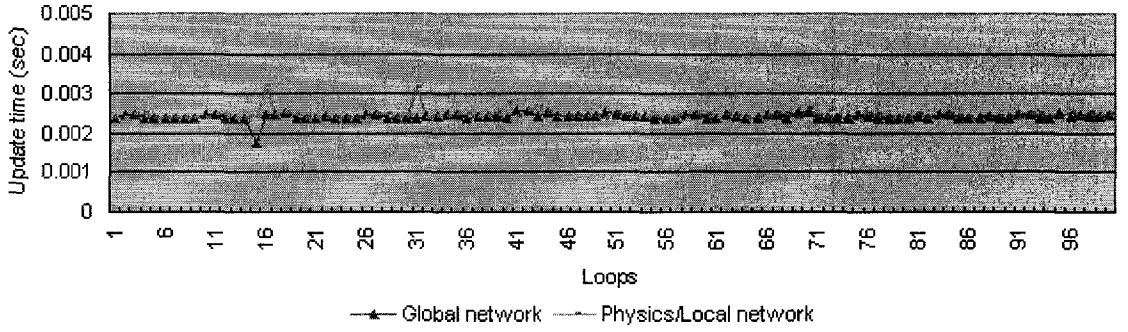


Figure 4.12: Global network & physics/local network update time in VE #3 with barrier synchronization

Figures 4.7 to Figure 4.12 show that the barrier synchronization method improves the simulation performance only in virtual environment #3, where the global network update speed is significantly faster than the physics interaction/local network update speed. That is because the VE#3 host machine has enough CPU power to take advantage of the multithreading architecture. In addition, running the server application in virtual environment #3 reduces the network delay and speeds up the global network loop update rate in virtual environment #3. Therefore, barrier synchronization is applied only to virtual environment #3.

The multithreading structure illustrated in Figure 3.14(b) was next implemented and the performance compared to the multithreading structure shown in Figure 3.14(d) in order to verify the effect of the network delay. The global network and the physics/local network loop are running in one thread. Figure 4.10 is a graph comparing the structure of Figure 3.14(b), in which the global network, physics interaction, and local network loops are in one thread, and the structure of Figure 3.14(d), which is the multithreading structure of the current NHE application. In this situation, there is very small (<1ms) network delay. Figure 4.13 is a comparison of the same two multithreading structures

when 10ms of network delay is artificially made on the server side. Figure 4.14 shows that the thread structures do not exhibit great differences in performance when network delay is small. Figure 4.13, however, shows that the NHE multithreading structure can maintain a relatively high physics/local network update rate in the presence of large global network delay.

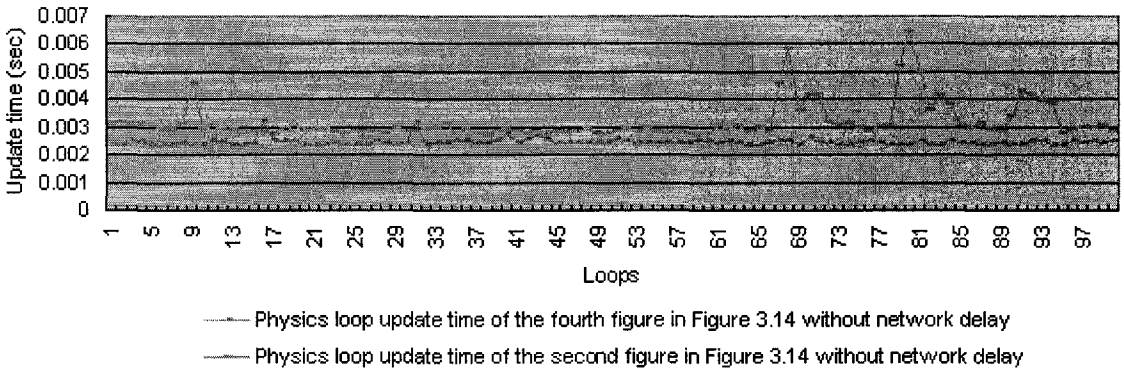


Figure 4.13: Physics loop update times in different multithreading structures

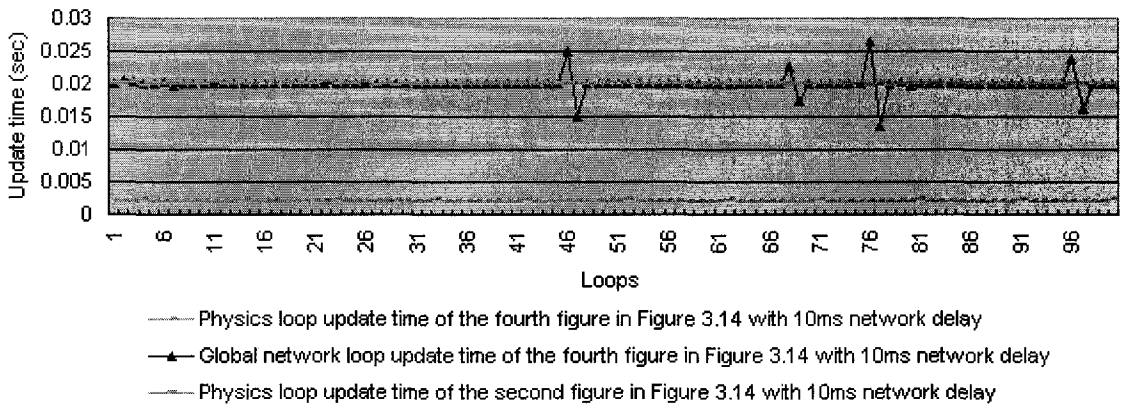


Figure 4.14: Physics and global network loop update times in different multithreading structures

CHAPTER 5. CONCLUSION AND FUTURE WORK

The purpose of the research is to develop a method to implement force feedback interactions in a networked virtual environment. The intent is also to develop this method to allow the users on the network to grab and move objects in a projection screen virtual environment with a haptic device to enable remote users to communicate and to transmit their simulated motions and forces to each other through non-dedicated networks.

5.1. Summary

In order to provide realistic part interaction, haptic feedback, collision detection, and object interaction within a collaborative virtual environment were investigated. This research seeks to provide combinations of available technologies in one application to create higher-quality immersive environments that require the physics interaction computations to be performed at high frame-rate speeds and the network structure to handle unlimited number of users and to guarantee the high frame-rate (1000Hz) of the haptic devices.

Various collision detection and physical interaction packages were examined. From this comparison, VPS software was chosen for the collision detection and physical interaction modeling. The advantages and drawbacks of client-server and peer-to-peer network architectures were explored and the rationale for using the combination of these two architectures was examined. Four different multithreading structures were explored. A mixture of client-server and peer-to-peer architecture was used for the final application's network structure. The advantage of peer-to-peer physics simulation is unconditional stability for a haptic device, while the drawback is a simulation discordance problem due to CPU latency. The discordance problem was defined due to this network and CPU delay. The RNR synchronization method and timeout mechanism were developed and implemented. These methods improved the synchronization of the object position/orientation over the network.

The NHE application was developed and tested for three users with three PHANToM devices, each with varying capabilities. The time periods of the physics/local network and the global network loop updates in the NHE were measured in three virtual environments when three users in the network each grab an object and collide them at the same time. Based on the results, the speeds of the simulation and the update rates of the network user's movements were calculated.

The accomplishments of this research include:

- No restriction on the input model shape and type.

- Creation of network architecture without limitation on the number of users for haptic simulation.
- Development of the position synchronization method.
- Investigation of multithreading structures for networked haptic environments.
- Simulation of 3 DOF rigid-body force feedback with 6 DOF reaction force/torque.

5.2. Conclusion

An important aspect to collaborative product design is to understand how the users will interact with each other as well as with other objects. The focus in this research is on collaboration over a non-dedicated channel where users experience real-time rigid-body force feedback with synchronized communication. Because of the three dimensional interaction with force feedback and human centered viewing in a shared virtual environment, engineers and product designers in different places can interact with their models to perform collaborative tasks as they do in the physical world. By allowing collaborative tasks to occur with digital models though network communication as opposed to manipulation of real parts, industry can save time and money by trying many different options without gathering at one place.

Increasing realism, accuracy, and performance are still issues to overcome in order to achieve more realistic NHE application.

5.3. Future Research

In order to increase realism, the next goal of this work is to develop a method to accommodate the assembly of large and small parts in the same environment. The current voxmap-pointshell interaction used to generate a reaction force and torque constrains the user to assembly of objects with uniformly-sized voxels per model in the virtual environment. It is necessary to develop a method that can perform collision detection using models consisting of multiple voxel sizes within the same part in order to increase the simulation speed and collision accuracy.

Another challenge is implementing a new architecture for the global network. Current client-server architecture requires an independent server program running in the network to enable clients to communicate with each other. When the server is disconnected, the whole network loses communication because all packets are sorted and distributed through the server. In order to avoid this problem, an embedded server system needs to be implemented. An embedded server system is an extension to the traditional client-server architecture, where the first client runs both a server program and a client program at the same time. The server architecture included in every client is

automatically activated in the first host and acts as a server while it also runs a client program. The other users connect to the first client and consider it as a server. If the first client, a server, is disconnected, the second client inherits the server property from the first client and acts as a server at this time. In this case, all other users automatically reconnect to the new server to maintain the network communication. The fact that any client can be a server means that the network connection lasts as long as at least one host is alive.

APPENDIX A. SGI OPENGL PERFORMER™ INPUT FILE FORMATS

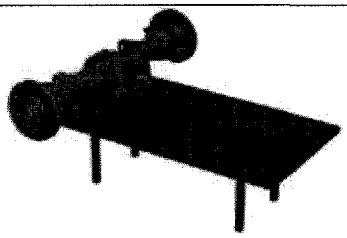
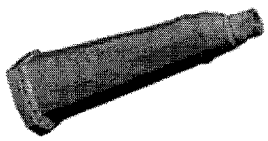
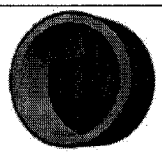
Name	Description
3ds	AutoDesk 3DStudio binary data
bin	SGI format used by powerflip
bpoly	Side Effects Software PRISMS binary data
byu	Brigham Young University CAD/FEA data
csb	OpenGL Optimizer Format
ct	Cliptexture config file loader - auto-generates viewing geometry
dwb	Coryphaeus Software Designer's Workbench data
dxf	AutoDesk AutoCAD ASCII format
flt11	MultiGen public domain Flight v11 format
flt	MultiGen OpenFlight format provided by MultiGen
gds	McDonnell-Douglas GDS things data
gfo	Old SGI radiosity data format
im	Simple OpenGL Performer data format
irtp	AAI/Graphicon Interactive Real-Time PHIGS
iv	SGI Open Inventor format (VRML 1.0 superset)
lsa	Lightscape Technologies ASCII radiosity data
lsb	Lightscape Technologies binary radiosity data
medit	Medit Productions medit modeling data
nff	Eric Haines' ray tracing test data
pfb	OpenGL Performer fast binary format
obj	Wavefront Technologies data format
pegg	Radiosity research data format
phd	SGI polyhedron data format
poly	Side Effects Software PRISMS ASCII data
ptu	Simple OpenGL Performer terrain data format
rpc	ArchVision rich photorealistic content
sgf	US Naval Academy standard graphics format
sgo	Paul Haeberli's graphics data format
spf	US Naval Academy simple polygon format

Name	Description
sponge	Sierpinski sponge 3D fractal generator
star	Astronomical data from Yale University star chart
stla	3D Structures ASCII stereolithography data
stlb	3D Structures binary stereolithography data
stm	Michael Garland's terrain data format
sv	John Kichury's i3dm modeler format
tri	University of Minnesota Geometry Center data
unc	University of North Carolina walkthrough data
wrl	OpenWorlds VMRL 2.0 provided by DRaW Computing

APPENDIX B. EXAMPLE OF ITRI FILE FORMATS

```
9
0.00741279    1.7957    2.98301
-0.212391     1.5445    2.4178
0.227216      1.5445    2.4178
0.00741279    1.7957    2.98301
0.227216      1.5445    2.4178
0.227216      2.0469    2.4178
0.00741279    1.7957    2.98301
0.227216      2.0469    2.4178
-0.212391     2.0469    2.4178
0.00741279    1.7957    2.98301
-0.212391     2.0469    2.4178
-0.212391     1.5445    2.4178
-0.212391     1.5445    2.4178
0.00741279    1.7957    2.4178
0.227216      1.5445    2.4178
0.227216      1.5445    2.4178
0.00741279    1.7957    2.4178
0.227216      2.0469    2.4178
0.227216      2.0469    2.4178
0.00741279    1.7957    2.4178
0.227216      2.0469    2.4178
0.227216      2.0469    2.4178
0.00741279    1.7957    2.4178
-0.212391     2.0469    2.4178
-0.212391     2.0469    2.4178
0.00741279    1.7957    2.4178
-0.212391     1.5445    2.4178
```

APPENDIX C. SPECIFICATIONS OF INPUT MODEL FILES

Graphic image	Model name	Number of triangles	Number of voxels (voxel size 0.6096cm)
	Axle base	26365	227968
	Driveline	1418	1736
	Ring cap	6836	1483
	Ring	2168	1458
	Housing holder	4350	8396
	Driveline housing	4796	23211
	Right cap	6304	20998
	Left cap	6325	19875
	Total	58562	305098

REFERENCES

[1] Jayaram, S., Vance, M.J., Gadh, R., Jayaram, U., and Srinivasan, H., March, 2001, "Assessment of VR Technology and its Applications to Engineering Problems", ASME Journal of Computing and Information Science in Engineering. **1**(1), pp. 72-83.

[2] Hodges, L.F., Anderson, G., Burdea, G.C., Hoffman, H.G., and Rothbaum, B.O., November/December, 2001, "Treating Psychological and Physical Disorders with VR", IEEE Computer Graphics and Applications. **21**(6), pp. 25-33.

[3] Stansfield, S., Shawver, D., Sobel, A., Prasad, M., and Tapia, L., December, 2000, "Design and Implementation of a Virtual Reality System and Its Application to Training Medical First Responders", Presence: Teleoperators and virtual envi-ronments. MIT Press. **9**(6).

[4] Balling, O., Knight, M., Walter, B., and Sannier, A., 2002, "Collaborative Driving Simulation", in SAE 2002 World Congress & Exhibition. Detroit, MI, SP-1656.

[5] Good, M. and Tan, L., November/December, 1994, "VR in Architecture: Today's Use and Tomorrow's Promise", Virtual Reality World. **2**(6), pp. 58-64.

[6] Winn, W., May, 1997, "The Impact of Three-Dimensional Immersive Virtual Environments on Modern Pedagogy", HITL Technical Report R-97-15. Discussion paper for NSF Workshop.

[7] Pausch, R., Snoddy, J., Taylor, R., Watson, S., and Haseltine, E., November 3-4, 1996, "Disney's Aladdin: First Steps Toward Storytelling in Virtual Reality", Computer Graphics Proceedings, Annual Conference Series.

[8] Stuart, R.,2001, The Design of Virtual Environments: Barricade Books Inc. 66.

[9] Han, S. and Lim, M., 2003, "ATLAS-II: Scalable and Self-tunable Network Framework for Networked Virtual Environments", in The Second Young Investigator's Forum on Virtual Reality (YVR 2003). Bonn, Germany.

[10] Kim, C. and Vance, J., September 2-6, 2003, "Using VPS (Voxmap PointShell) as the basis for interaction in a virtual assembly environment", in Proceedings of ASME 2003 Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Chicago, Illinois, DETC2003/CIE-48297.

[11] Jayaram, S., Jayaram, U., Wang, Y., Tirumali, H., Lyons, K., and Hart, P., November/December, 1999, "VADE: A Virtual Assembly Design Environment", Computer Graphics and Applications. **19**(6), pp. 44-50.

[12] Fischer, A.G. and Vance, J.M., 2003, "PHANToM Haptic Device Implemented in a Projection Screen Virtual Environment", in Immersive Projection Technology and Virtual Environments 2003 (Eurographics/Fraunhofer IAO Workshop Proceedings). Zurich, Switzerland, pp. 225-230.

[13] McNeely, W.A., Puterbaugh, K.D., and Troy, J.J., 1999, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", in SIGGRAPH 99 Conference Proceedings, Annual Conference Series, ACM Press, pp. 401-408.

[14] Kim, Y.J., Otaduy, M.A., Lin, M.C., and Manocha, D., March 24-25, 2002, "Six-Degree-of Freedom Haptic Display Using Localized Contact Computations", in The Tenth Symposium on Haptic Interfaces For Virtual Environment and Teleoperator Systems. Los Alamitos, California, pp. 209.

[15] Codella, C.F., Jalili, R., Koved, L., and Lewis, J.B., 1993, "A Toolkit for Developing Multi-User, Distributed Virtual Environments", in IEEE Virtual Reality Annual International Symposium, pp. 401-407.

[16] Broll, W., 1995, "Interacting in Distributed Collaborative Virtual Environments", in Proceedings of the IEEE Virtual Reality Annual International Symposium - VRAIS'95. Los Alamitos, California, pp. 148-155.

[17] Ho, C., Basdogan, C., Slater, M., Durlach, N., and Srinivasan, M.A., June 10-11, 1998, "An Experiment on the Influence of Haptic Communication on the Sense of Being Together", BT Presence Workshop BT Labs.

[18] Sallnas, E.-L., Rassmus-Grohn, K., and Sjostrom, C., 2000, "Supporting Presence in Collaborative Environments by Haptic Force Feedback", ACM Transactions on Computer-Human Interaction (To CHI). 7(4), pp. 461-476.

[19] Hespanha, J.P., McLaughlin, M., Sukhatme, G.S., Akbarian, M., Garg, R., and Zhu, W.,2002, Haptic Collaboration over the Internet: Touch in Virtual Environments: Haptics and the Design of Interactive Systems. Prentice-Hall.

[20] Jordan, J., Mortensen, M., Oliveira, M., Slater, M., Tay, B.K., Kim, J., and Srinivasan, M., 24 October, 2002, "Collaboration in a Mediated Haptic Environment", in 2nd EQUATOR Conference. Careys Manor, Brockenhurst, United Kingdom.

[21] McLaughlin, M., Sukhatme, G., Peng, W., Zhu, W., and Parks, J., 2003, "Performance and co-presence in heterogeneous haptic collaboration", in IEEE VR 2003, the Haptics Symposium. Los Angeles, California.

[22] Bullinger, H.J., Richer, M., and Seidel, K.-A., May, 2000, "Virtual Assembly Planning", Human Factors and Ergonomics in Manufacturing. **10**(3), pp. 331-341.

[23] Jayaram, U., Tirumali, H., and Jayaram, S., September 10-13, 2000, "A Tool/Part/Human Interaction Model for Assembly in Virtual Environments", in The ASME Design Engineering Technical Conferences 2000. Baltimore, Maryland, DETC 2000/CIE-14584.

[24] Taylor, F., Jayaram, S., and Jayaram, U., September 10-13, 2000, "Functionality to Facilitate Assembly of Heavy Machines in a Virtual Environment", in The ASME Design Engineering Technical Conferences. Baltimore.

[25] Jayaram, S., Jayaram, U., Wang, Y., and Lyons, K., September 10-13, 2000, "COBRA-based Collaboration in a Virtual Assembly Design Environment", in The ASME Design Engineering Technical Conferences 2000. Baltimore, Maryland, DETC 2000/CIE-14585.

[26] Fernando, T., Marcelino, L., Wimalaratne, P., and Tan, K., Febrary 16-18, 2000, "Interactive Assembly Modeling within a CAVE Environment", Portuguese Chapter of Eurographics, pp. 43-49.

[27] Johnson, T.C. and Vance, M.J., September 9-12, 2001, "The use of the voxmap pointshell method of collision detection in virtual assembly methods planning", in The ASME Design Engineering Technical Conferences 2001. Pittsburgh, Pennsylvania, DETC2001/DAC-21137.

[28] Cohen, J.D., Lin, M.C., Manocha, D., and Ponamgi, M.K., 1995, "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments", in The 1995 ACM International 3D Graphics Conference. Los Angeles, California, pp. 189-196.

[29] Ehmann, S.A. and Lin, M.C., "SWIFT: Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching", in Technical report, Computer Science Department, University of North Carolina at Chapel Hill. 2000.

[30] Gottschalk, S., Lin, M.C., and Manocha, D., 04-09 August 1996, 1996, "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection", ACM SIGGRAPH'96, pp. 171-180.

[31] Hudson, T., Lin, M.C., Cohen, J., Gottschalk, S., and Manocha, D., 1997, "V-COLLIDE: Accelerated Collision Detection for VRML", in Proceedings of the second symposium on Virtual Reality Modeling Language. New York City, NY, ACM Press, pp. 119-125.

[32] Larsen, E., Gottschalk, S., Lin, M.C., and Manocha, D., 1999, "Fast Proximity Queries with Swept Sphere Volumes", Technical Report TR99-018, Department of Computer Science, University of North Carolina.

[33] Ehmann, S.A. and Lin, M.C., 2001, "Accurate and Fast Proximity Queries between Polyhedra Using Surface Decomposition", Eurographics. Computer Graphics Forum. **20**(3), pp. 500-510.

[34] Mirtich, B., 1998, "V-Clip: fast and robust polyhedral collision detection", ACM Transactions on Graphics. **17(3)**, pp. 177-208.

[35] Gibson, S.F., "Beyond Volume Rendering: Visualization, Haptic Exploration, and Physical Modeling of Voxel-based Objects", in Mitsubishi Electric Research Laboratories Cambridge Research Center Technical Report 95-04. 1995.

[36] Avila, R.S. and Sobierajski, L.M., 1996, "A haptic interaction method for volume visualization", in Proceedings of the Conference on Visualization '96. San Francisco, California, pp. 197.

[37] Hubbard, P.M., July, 1995, "Real-Time Collision Detection and Time-Critical Computing", in Workshop on Simulation and Interaction in Virtual Environments. U. of Iowa, pp. 92-96.

[38] Gagvani, N. and Silver, D., October 09-10, 2000, "Shape-based Volumetric Collision Detection", in Proceeding of the 2000 IEEE symposium on Volume visualization. Salt Lake City, Utah, pp. 57-61.

[39] Moore, M. and Wilhelms, J., 1998, "Collision detection and response for computer animation", Computer Graphics (proc. SIGGRAPH). **22**, pp. 289-298.

[40] Platt, J. and Barr, A., 1988, "Constraint methods for flexible models", Computer Graphics (proc. SIGGRAPH). **22**, pp. 279-288.

[41] Mirtich, B. and Canny, J., 1995, "Impulse-based Simulation of Rigid Bodies", in Symposium on Interactive 3D Graphics. New York, ACM Press.

[42] Baraff, D. and Witkin, A., 1997, "Physically Based Modeling: Principles and Practice (Online Siggraph '97 Course Note), http://www-2.cs.cmu.edu/~baraff/sigcourse". (Date Accessed: April 16, 2004)

[43] Salisbury, K., Conti, F., and Barbagli, F., September-October, 2004, "Haptic Rendering: Introductory Concepts", IEEE Computer Graphics and Applications(March/April), pp. 24-32.

[44] Sarcos, 2002, "Utah/MIT Dextrous Hand Master, http://www.sarcos.com/telespec.dexarm.html". (Date Accessed: April 16, 2004)

[45] SensAble, 2001, "Product Brochure, http://www.sensable.com/products/datafiles/phantom_ghost/PHANTOM_GHOST%20Brochure.pdf". (Date Accessed: April 16, 2004)

[46] Immersion, 2002, "CyberForce Users Guide, http://www.immersion.com/manuals/cyberforceuserguide_v1.3.pdf". (Date Accessed: April 16, 2004)

[47] MPBTechnologies, 2003, "Freedom 6S, http://www.mpb-technologies.ca/space/freedom6_2000/f6s/freedom6s.html". (Date Accessed: April 16, 2004)

[48] Salisbury, J.K. and Srinivasan, M.A., September-October, 1997, "Phantom-Based Haptic Interaction with Virtual Objects", IEEE Computer Graphics and Applications. **17**(5), pp. 6-10.

[49] Gunn, C., Hutchins, M., Adcock, M., and Hawkins, R., 2003, "Trans-World Haptic Collaboration", in SIGGRAPH 2003 Sketches and Applications. San Diego, California.

[50] Singhal, S. and Zyda, M.,1999, Networked Virtual Environments: Design and Implementation, New York: addison-wesley.

[51] Matsumoto, S., Fukuda, I., Morino, H., Hikichi, K., Sezaki, K., and Yasuda, Y., 2000, "The influences of Network Issues on Haptic Collaboration in Shared Virtual Environments", in 5th Phantom Users Group Proceedings. Aspen, Colorado.

[52] Just, C., Bierbaum, A., Baker, A., and Cruz-Neira, C., May 1998, 1998, "VR Juggler: A Framework for Virtual Reality Development", in 2nd Immersive Projection Technology Workshop (IPT98) CD-ROM. Ames, Iowa.

[53] Coutee, A.S. and Bras, B., 2002, "Collision Detection for Virtual Objects in a Haptic Assembly and Disassembly Simulation Environment", in 2002 ASME Design Engineering Technical Conference/Computers in Information Engineering. Montreal, Canada, DETC2002/CIE-34385.

[54] Bergen, G., 1999, "User's Guide to the SOLID Interference Detection Library, http://www.win.tue.nl/~gino/solid/solid2_toc.html". (Date Accessed: April 16, 2004)

[55] Caselli, S., Reggiani, M., and Mazzoli, M., May, 2002, "Exploiting Advanced Collision Detection Libraries in a Probabilistic Motion Planner", in International Conference in Central Europe on Computer Graphics and Visualization. Plzen, Czech Republic.

[56] Morvan, S.M. and Fadel, G.M., August 21-22, 1996, "IVECS: An Interactive Virtual Environment for the Correction of .STL files", in The ASME Design Engineering Technical Conferences 1996. Irvine, California, DETC 1996/DFM-1305.

[57] Preusche, C., Hirzinger, G., and Rettig, A., June, 2002, "Assembly Verification in Digital Mock-Ups Using Force Feedback", in Proceedings of 12th International Symposium on Measurement and Control in Robotics Towards Advanced Robot Systems and Virtual Reality 2002. Bourges, France.

[58] Mirtich, B., 1998, "Rigid Body Contact: Collision Detection to Force Computation", MERL Technical Report, TR-98-01, pp. 8-10.

[59] Xu, H., McKinley, P.K., and Ni, L.M., 1992, "Efficient Implementation of Barrier Synchronization in Wormhole-Routed Hypercube Multicomputers", Journal of Parallel and Distributed Computing. **16**, pp. 172-184.