HAPTIC RENDERING OF TRIMMED NURBS MODELS WITHIN AN ACTIVE PROTOTYPING ENVIRONMENT

by

Thomas V Thompson II

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

School of Computing

The University of Utah

December 2006

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Thomas V Thompson II

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

	Chair: Elaine Cohen
	Rich Riesenfeld
	John Hollerbach

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the dissertation of Thomas V Thompson II in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Elaine Cohen
Chair: Supervisory Committee

Approved for the Major Department

Martin Berzins
Chair/Director

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

This dissertation presents techniques for direct haptic rendering of complex trimmed NURBS models. More specifically, the research concentrates on developing a complete collection of algorithms for haptic interaction with Computer Aided Design (CAD) models without requiring any intermediate representations or simplifications. The complex underlying structure of trimmed NURBS further compounds the difficulty of haptically rendering them. However, for many reasons, trimmed NURBS are a highly popular CAD representation. This approach allows users to interact haptically with the massive collection of real-world NURBS models.

The design and implementation of a target application, *APE* (Active Prototyping Environment), as a test-bed system that integrates haptic rendering into the *Alpha_1* modeling package is described. Among the algorithms required for such interaction are model proximity; global and local closest point tracking; surface evaluation of contact point, normal and tangent vectors; and rapid and robust transitioning across trimmed edges. Since current computer systems are not powerful enough to perform all of the necessary tasks within the alloted time using a single processing thread, distributed system techniques, including multiplatform, multiprocessing and multithreading, are also presented. In addition, an approach to verify the quality of a haptic algorithm is presented and applied to direct haptic rendering.

To my grandmothers Margaret and Naomi.

Your faith and love got me here.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to sincerely thank the members of my committee, Elaine Cohen, Rich Riesenfeld, and John Hollerbach, for their time and advice. Each of you has helped improve and shape this research. Special thanks to the chair of my committee, Elaine Cohen, for creative ideas, interesting diversions, historical perspective, and grounding my theories in reality. Thanks to the staff of the *GDC* research group, Dave Johnson and Jim de St Germain, for their support and helping me flesh out ideas. Thanks to the members of the Biorobotics group, Don Nelson, Rod Freier, and Milan Ikits, for helping operate and setup the robotic devices, their various control systems, and the networking software. Further thanks goes to the graduate community and in particular David Weinstein, Helen Hu, and Steve Parker.

I thank my parents and grandparents for giving me the drive to continue my education. Without their encouragement I could not have succeeded. Thanks to Don Smith and Joe Fuller for setting me down this path by inspiring me at West Virgina Tech. And finally, great thanks to my wife Tamaryn. Without her love, support, insistence, and willingness to pick up my slack during the long hours of writing, I am certain this document would never have been completed.

This work has been supported in part by DARPA grant F33615-96-C-5621, NSF Grant MIP-9420352, and by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). All opinion, findings, conclusions, or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

CHAPTER 1

INTRODUCTION

Current modeling systems offer limited feedback to a model designer. The passive graphical display of complex models can convey only limited visual information. Even with the array of presentation options supplied to a user by modeling packages, such as iso-line drawings, shaded images, and animations, the designer is often left without information that could easily be gathered if the model could be interrogated by touch [1].

One promising approach for increasing the information available to a designer is haptic rendering. Haptic rendering supplies touch feedback to the user by simulating the forces generated by contact with, and surface tracing of, a virtual model. In conjunction with visual feedback, haptic rendering increases the level of interaction. This facilitates a greater understanding of complex models and adds to the sense of realism in virtual environments [23, 24, 63].

While the graphical display of a model is almost exclusively accomplished by first converting it to a collection of polygons, the model itself often starts with a different geometric representation. In fact, a master CAD model is almost always described by NURBS throughout its life [44]. Being a parametric representation, NURBS surfaces have the advantage of compactness, higher order continuity, and exact computation of surface tangents and normals. All of these properties are useful in complex, realistic virtual environments [57].

The ability to trim away arbitrary portions of a NURBS surface and define adjacencies under boolean set operations is a powerful extension to many modeling packages. These constructive solid geometry results form a large class of models which can be expressed much more readily and succinctly using trimmed rather than nontrimmed NURBS. Furthermore, the use of trimmed NURBS is fairly widespread,

making it important for a haptic rendering algorithm to handle them. This increase in expressiveness does not come without a cost. Since the trimming curve (or curves) representing the boundary of a surface is frequently expressed as a piecewise linear curve with a high number of segments, determining when haptic contact is to transition from one surface to another can be a complex task. However, once such a transition is detected, topological adjacency information can allow for efficient computation of the exact transition point.

The goal of the research reported here is to enable the user to trace directly on the actual CAD model instead of an intermediate or alternate representation; achieving the most accurate haptic rendering results. To this end, this research introduces direct haptic rendering of complex models constructed from trimmed NURBS surfaces. This approach is realized through development of algorithms for model proximity testing; fast update of global and local closest point approximations; computationally efficient surface evaluation and normal evaluation techniques; and smooth efficient transitioning across trimmed surface boundaries. Furthermore, these results are extended to include model manipulation, dimensioned probes, and multiple probe contact. A complete system, *APE*, is presented as a target application in which the algorithmic results are tested. This distributed system integrates the haptic rendering results, through various haptic devices, to *Alpha_1* [53, 54], a research CAD package.

1.1 The Problem

CAD systems are not built with real-time interaction as a primary goal. The focus of CAD systems, secondary to the actual design process, tends to be directed toward data management, representation, and complexity issues. While this is appropriate for the overall task of product design, it is not an optimal space for haptic rendering. Further, even when computation is shifted to concentrate on haptic rendering of the CAD model, the complexity of trimmed NURBS models results in their not lending themselves to haptic rate computation.

Another difficulty in the development of a haptic design environment is that tight update rate constraints on the visual and haptic displays must be met. It may be acceptable for the visual display to update at twenty frames per second [9], but the haptic display must maintain rates of several hundred *Hz* [40] for the virtual surfaces to feel solid. Both the visual and haptic display individually tax a system; therefore, it is necessary to distribute computation onto multiple processors or even multiple machines [66, 67].

1.2 Trimmed NURBS Models

Non-Uniform Rational B-Spline (NURBS) surfaces are highly compact and yet very expressive as a representation for modeling. A NURBS surface is a bivariate vector-valued piecewise rational function of the form

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} w_{i,j} B_{j,k_v}(v) N_{i,k_u}(u)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{i,j} B_{j,k_v}(v) N_{i,k_u}(u)}, \tag{1.1}$$

where the $\{P_{i,j}\}$ form the control mesh, the $\{w_{i,j}\}$ are the weights, and the $\{N_{i,k_u}\}$ and $\{B_{j,k_v}\}$ are the basis functions defined on the knot vectors $\{u\}$ and $\{v\}$ for a surface of order k_u in the u direction and k_v in the v direction.

The various properties of a NURBS surface (presented in greater detail in Appendix A), including a local convex hull property, and the ability to evaluate surface points, normals and tangents, along with its intuitive control characteristics make it a good representation for modeling and design. These properties have led to NURBS becoming the de facto industry standard for the representation and data exchange of geometric models [44].

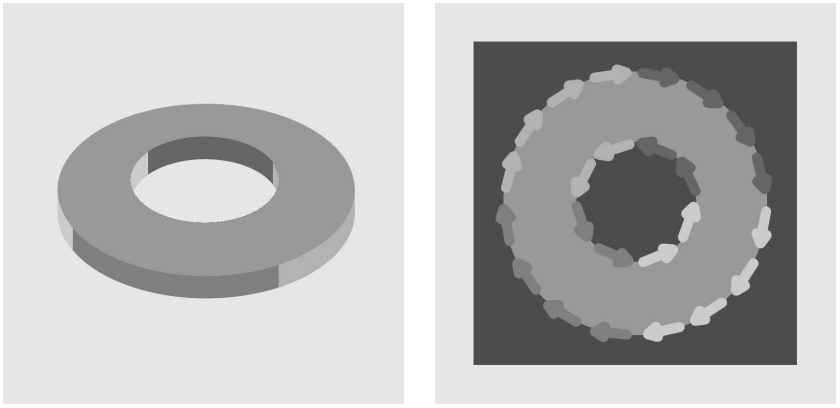
Trimmed NURBS models are constructed by cutting away portions of a NURBS surface using trimming curves in parametric space. In *APE*, trimming information is represented in parametric space as directed closed polygons called *trimming loops*. Each individual linear portion of the loop is called a *segment*. An ordered set of connected segments that represents a shared boundary between two surfaces is referred to as an *edge*. The trimming loop being directed enables the classification of the surface domain. Portions of the surface domain to the "left" of a loop are

considered cut-away while pieces to the "right" are deemed part of the model. Note that each surface that is part of a model contains at least one trimming loop. If there is no portion of the surface being cut away then this loop simply surrounds the domain of the surface.

Consider the model in Figure 1.1a of a simple disc with a hole cut through it. The top surface will have two trimming loops within its parametric domain as shown in Figure 1.1b. Notice that the direction of the two loops indicate that the dark regions are to be cut away. The outer clockwise loop cuts away the outermost region while the inner counterclockwise loop cuts away the center region representing the hole. The edges in Figure 1.1b are illustrated in alternating brightness to indicate the patch in Figure 1.1a that is adjacent to each edge.

1.3 Haptic Rendering

The goal of a haptic rendering system is to reproduce a sense of physical presence for a virtual model. One approach to achieving this result is to accurately generate forces and apply them to a probe attached to a user's hand or arm. A haptic device renders the calculated forces and applies them to the probe. These forces, called restoring forces, resist penetration into the virtual model and are calculated using



(a) (b)

Figure 1.1. A trimmed NURBS model (a) and its parametric domain containing the trimming loop information (b).

a response model called a wall model. The more accurate these forces are the more realistic the simulation of reality.

1.3.1 Haptic Devices

Currently, there are not a large number of commercially available haptic devices. Those that are available present interesting trade-offs in fidelity, workspace, and degrees-of-freedom (dof). The device handling feature of *APE* is object oriented and as such is extensible to further device development. At the University of Utah we currently employ two highly disparate haptic devices: the Sensable Phantom [37] and the Sarcos Dextrous Arm Master [26]. The *APE* system supports both of these devices as well as other nonhaptic devices and trackers.

The Phantom (Figure 1.2a) is a small desktop force feedback device. It is a low inertia device with 6-dof for input and 3-dof for force reflection. This device is controlled by a Linux Dual $2.4GHz$ Intel Pentium4 processor based workstation.

The Dextrous Arm (Figure 1.2b) is an advanced hydraulic force-reflecting exoskeleton. It is a high inertia device with 10-dof and a complex dynamic structure. Among the 10-dof are two for the thumb and one for the finger, allowing force reflection to two points on the hand instead of one. This device is controlled by a hybrid PowerPC 604 and Motorola 68040 VME system.

One example of a nonhaptic device supported by *APE* is the Immersion CyberTouch [12] tactile feedback glove (Figure 1.2c). The glove contains 18 sensors to monitor the motions of the hand and fingers. Further, it provides small vibrotactile stimulators for each finger and the palm. When used in conjunction with the Ascension Bird [5], the hands position can be tracked with 6-dof.

1.3.2 Response Models

There are two basic types of response models: compliance and stiffness. The compliance model [69] takes force measurements and uses a control strategy to render acceleration or another form of motion to the probe and the virtual object. The stiffness model uses measured position to compute and display force. Wall models

(a) (b) (c)

Figure 1.2. *APE* supported devices: Sensable Phantom (a), Sarcos Dextrous Arm Master (b), and Immersion CyberTouch (c).

based on the stiffness response model often have a restoring force proportional to the depth the probe penetrates into the model [11] and in the direction of the surface normal at the contact point. The stiffness model is the prevalent approach used for haptic rendering and the one adopted for *APE*.

The intrinsic differences in the two haptic devices supported by *APE* require different wall model definitions for each. The combination of low inertia and low friction in the Phantom allows a high fidelity force display to be produced with a very simple wall model. The wall model, recommended by the manufacturer, is defined as

$$F = k_p x, \tag{1.2}$$

where x is the penetration depth and k_p is the spring coefficient. For the Phantom we set the surface stiffness to $1500 N/m$.

The Dextrous Arm requires a more complex wall model that can compensate for the inertia of the device. For the Dextrous Arm we use a nonlinear damping model developed by Marhefka and Orin [35]. The wall model is defined as

$$F = k_p x^n + k_v x^n x', \tag{1.3}$$

where x' is the velocity of the probe, and k_v is the damping coefficient. Notice that the penetration depth is present in the second term, which requires that the force starts from zero during initial contact regardless of velocity. For the Dextrous Arm

we use a surface stiffness of $6000N/m$ and through experimentation have found $n = {}^1\!/_2$ to be a sharp and stable exponent value.

1.3.3 Direct Haptic Rendering

At best, a haptic algorithm can accurately reproduce the model upon which it acts. If this differs from the original even the most sophisticated algorithms cannot faithfully reproduce the original model. Therefore, the most accurate haptic rendition of a virtual model is produced when the haptic algorithm acts directly on the actual model and not an intermediate or alternate representation.

In the modeling and design communities the de facto standard model representation is NURBS [44]. Many systems also provide trimming information that permits constructive solid geometry to be accurately represented. Further, some more powerful systems provide adjacency information within the models representation as well. *APE* and *Alpha_1* support both trimming and topological adjacency information. Obviously, the addition of trimming information to the model representation complicates the haptic algorithm. The algorithm must detect not only when a haptic trace exits the surface's domain, but also any intersection with trimming boundaries. However, the combination of trimming and topological adjacency information can be used to both simplify and improve the efficiency of the haptic algorithm. For example, if a haptic trace crosses a surface boundary, indicated by an intersection with a trimming boundary, the adjacency information can be used to efficiently and accurately calculate the contact point on the newly contacted surface. This allows a collection of trimmed surfaces with adjacency information to be treated as a solid model, both conceptually and within the haptic algorithm.

1.4 Active Prototyping Environment

As both a test-bed application and proof of concept, we present the Active Prototyping Environment (*APE*). While haptically tracing a virtual model can provide information to a designer that visual rendering alone cannot, it is the

combination of these two effects that most completely produces the experience of tracing a physical model. This convergence of rendering techniques, however, is neither straightforward nor an obvious combinatoric result.

Using currently available resources, it is not feasible to both visually and haptically portray a virtual model using a single computational thread. Decoupling the computation of these two displays into simulation and haptic processes can alleviate this computational bottleneck, but it introduces the problem of synchronizing the presentation. For example, when the user feels contact with a virtual model through the haptic device the visual display should, simultaneously, display the graphical representation of the haptic device contacting the virtual model. Each feedback channel re-enforces the other, producing a more powerful, and therefore realistic, experience.

For this reason, *APE* is a highly distributed system that attempts to maximize the use of available resources. Further, concentrated effort was placed on making all data accessible, without mutex locking, as soon as it becomes available, the combination of which results in a highly interactive environment with synchronized visual and haptic feedback.

1.5 Document Overview

The dissertation continues in Chapter 2 with a review of previous work in haptic rendering and environments that integrate haptic feedback. Chapter 3 focuses on the ideas that have been developed to solve the Direct Haptic Rendering problem for trimmed NURBS models. In particular, the problem is decomposed into several phases and subproblems with a separate section discussing each in turn. Results are reported for each individual section. Following this, Chapter 4 presents the *APE* system by giving both a system overview and by addressing specific implementation issues. Next, Chapter 5 brings the dissertation to a close with some conclusions and final remarks. Finally, a collection of appendixes are provided to help explain, in greater detail, some information related to research topics and data structures discussed in the earlier chapters.

CHAPTER 2

PREVIOUS WORK

Many researchers have worked toward bringing the full design process into an integrated computer design system. Each stage of the design process has been targeted for research. This section provides an overview of previous work relevant to the presented research. Primarily, work is included that is aimed at enabling a user to use 3D input to perform meaningful work within a virtual environment.

Several groups have investigated the problem of bringing the conceptual stage of design into a 3D environment. Not withstanding Hatvany's [20] claims that it is nearly impossible to sketch CAD designs, work in this area has progressed and the results are becoming more usable.

Others have decided that the ability to store and track early sketches is of more importance than having the sketch be actually in computer form. This approach is actually rather sound since design ideas come to designers at random inspirational moments throughout the day. Therefore, the initial sketches often occur on scraps of paper found at that creative moment. These systems allow the sketches to be scanned into image form and kept with the design throughout the design process.

The modeling stage of design is currently heavily supported with solid CAD systems. Over time, as the power of the computer systems upon which these design environments run has increased, many modeling systems have added support for a variety of input and output devices. However, these are generally for the purposes of visualization, such as the use of a head mounted display, or for passive manipulation, such as 3D trackers for manipulating a collection of models.

The prototyping stage is important to the design process as the results may signify the completion of the design. Problems that are not apparent in the design can often be brought out quickly in the prototype. However, the construction of

a physical prototype can be very expensive both in cost and time. The necessity of both cutting costs and bringing products to their fruition more quickly has given rise to the desire for virtual prototypes. There are three main problems to making virtual prototypes a reality. These are the construction of an appropriate virtual environment extension to the design space, development of haptic rendering algorithms that work directly on the designer's model, and the integration of these two components into a single system via distributed computing methods.

2.1 Virtual Environment

An immersive environment permits the designer to enter the design space. This is a powerful addition that allows the designer to view, manipulate, and explore the design using natural and intuitive 3D body motions. The transfer of real world skills into the virtual world in order to make the designer more efficient and reduce training times is a formidable research challenge.

Current CAD systems provide rather basic visual display capabilities but researchers in other disciplines have been working toward creating realistic virtual worlds [32]. The first look into an overlaid virtual world was provided by Sutherland in 1968 with his invention of the first computer graphics driven head-mounted display [65]. The commercial success of such completely immersive systems has been limited to entertainment and simulator purposes, but new applications have been shown to be effective as well.

Architectural walk-throughs [6, 17] have been a success nearly from the beginning. These environments are easy for the user to become immersed within since they represent something familiar to the user. The combination of relatively simple and mostly static geometry with algorithms that prune away geometry that is not visible allows for high display rates.

The Virtual Wind Tunnel was developed by Bryson and Levit in 1991 to allow testing of aircraft aerodynamics [7]. This was an important step in the virtual reality community as it proved to be an effective application even though the computation rate for the simulation was not able to keep up with the display rate.

They have since improved this system to allow it to be extensible for new devices and new visualization tools [8].

Medical applications that allow the surgeon or doctor to see a virtual version of the operating space have also been developed. Augmented displays have been used to show ultrasound images directly on a patient during ultrasound-guided needle biopsies [62]. Some researchers have worked on constructing virtual environments out of helical CT scan data to allow a doctor to fly through a patients colon in presurgical exams and surgery planning procedures [25].

Much of the virtual environment research has dealt with user interaction methods for manipulating items within the environment. There are two basic components to this problem: selection and manipulation. The different techniques developed make various trade-offs to meet these goals. Laser beam techniques [39] provide superior selection but suffer from poor manipulation capabilities. Arm-extension techniques such as Go-Go arms [51] provide intuitive hand-centered manipulation but imprecise selection. The worlds in miniature approach [48, 64] provides both easy selection and manipulation but the usability of this method may degrade as the environment size and number of objects increases. Image plane techniques [49] allow hand centered manipulation and simple pointing selection, however arm fatigue, eye dominance, and the hand obscuring objects all limit this approach. Only recently has the sense of touch been considered feasible as an additional channel of input for virtual environments.

2.2 Haptic Rendering

The goal of a haptic rendering system is to produce a sense of contact with a virtual model. This is accomplished by generating forces that can be applied to the user's hand or arm via a haptic device. These forces, called restoring forces, prevent penetration into the virtual model and are calculated using a wall model. There are two basic types of response models, compliance and stiffness, with the stiffness model being most prevalent in haptic rendering systems. Wall models based on the stiffness model often have a restoring force proportional to the penetration depth

[11] and in the direction of the closest point's surface normal. In order to maintain the stiffness of the virtual surface, the force servo loop must run at several hundred Hz [40]. This high update rate limits the complexity of the algorithms that can be used to find the closest point and has also restricted the types of models that can be rendered.

Zilles and Salisbury have traced polygonal models using a constraint-based system that tracks a point on the polyhedrons surface [70]. They calculate the penetration depth and surface normal from the tracked surface point. In order to portray sculptured models, they recommend interpolating the surface normals (much like Phong shading in graphics). Systems of this type are often limited to relatively simple models since too much processing time is required for complex models with a high polygon count. Ruspini et al. have extended this work to handle larger polygon counts as well as permit more general graphics primitives, such as points and lines, to be traced by a dimensioned probe [56].

Adachi et al. [2] and Mark et al. [36] advocate the use of intermediate representations to simplify haptic rendering of sculptured models. Stewart [63] also demonstrated this approach by applying a globally convergent numerical method to the system of equations describing the orthogonal projection onto a spline surface. These systems haptically render the model by using relatively slowly changing planar approximations to the virtual model. This method allows more complex models to be rendered but is limited when trying to approximate surfaces with high curvature. Further, since the planar approximations are sampled in time and not by position, the surface felt by the user is not necessarily repeatable during multiple tracings.

Free-form surfaces have been traced by Adachi using distribution functions [1] and by Salisbury et al. using implicit surfaces [58]. Both approaches permit quality tracing of smooth surfaces. However, parametric surfaces, such as NURBS, have become the surface representation of choice in CAD. As such, to use these methods requires a conversion from the original model into one of these other

representations. This conversion is difficult and often results in models defined by complex numerically unstable high order functions.

Thompson et al. have demonstrated direct haptic rendering of sculptured models constructed from NURBS patches [66]. Parametric surfaces such as NURBS have the advantage of a compact representation, higher order continuity, and exact computation of surface normals which are all useful in complex, realistic virtual environments [61]. This method has been extended to support more complex trimmed NURBS models [68] and to permit the models to be manipulated [67]. Using this method, designers can touch, trace and manipulate a CAD model at interactive rates without the use of an intermediate representation.

Subsequent to the work described here, Johnson and Cohen followed up upon the results of Thompson et al. by extending direct parametric tracing to include second order surface information [28]. Nelson et al. demonstrated surface-to-surface haptic interaction of sculpted models [42]. Patoglu and Gillespie also presented a surface-to-surface algorithm [45, 46]. Their method is based on control theory and maintains the extremal distance even with imprecise seed values. The pair has also presented a novel closest point tracking algorithm for parametric models constructed from convex tiling [47]. The approach is shown to be both patch invariant and globally uniformly asymptotically stable.

Dachille et al. simulated sculpting of surfaces through a physics based approach [13, 14]. This approach allowed the haptic force to act upon a discreatized representation constrained to the NURBS surface representation.

Research has continued using more recently available 6-dof devices. The additional degrees-of-freedom allow the resulting forces to include torques along with translational forces.

Kim et al. have created incremental methods for computing the penetration depth for collections of convex polygonal bodies [33]. The convex decomposition approach was extended by Otaduy and Lin to include perceptual level of detailing [43]. This can accelerate haptic rendering of very large models.

Johnson et al. use spatialized normal cones combined with local descent to facilitate polygonal model-model haptic rendering [30, 31]. This approach prevents penetration by deriving repulsive forces and torques and is therefore suitable for accessibility analysis.

Duan et al. propose a PDE-based surface flow approach [15]. Their work supports both implicit, distance-field based shape modeling, and dynamic, force-based shape design. The work was embedded in an immersive stereo environment by Hua et al. and extended to support locallized model modification [21].

Rather than use a method based on penetration depth, McNeely et al. at Boeing have created a voxel-based approach [38]. Their system allows for interaction with the voxelized scene by way of a point-sampled model. By creating the voxel boundary they insure valid virtual prototyping.

Perhaps the most glaring absence in this body of work is the ability to readily modify the underlying geometry of the model. Many of the above techniques require significant preprocessing to setup hierarchical bounding structures in order to speed contact detection. Others require a conversion from the design into an alternate form before haptic rendering can begin. Those that use a distributed model approach complicate geometry modification by introducing synchronization issues. This problem must be solved if haptic rendering is to be pervasive to the design process.

2.3 Distributed Computation

In order for a virtual environment to present a realistic and immersive experience to the user the update rate for the visual display must be kept above $20Hz$ [9]. Similarly, a haptic display must have an update rate maintained at hundreds of Hz [40]. Neither display can be allowed to slow the other's update rate and therefore each must be run in a separate process. These processes must maintain a consistent view of the model if the visual and haptic presentations are to produce a realistic, synchronized, portrayal of the tracing experience. This forces a distributed design approach that can, if approached properly, drastically improve the quality of both

processes within one system. Previous work approached the distributed design of virtual environments from a slightly different angle. In these prior works the system needed to be segmented since the components were the visual display and an environment simulation. The simulation would often run at much slower rates than the visual display and therefore needed to be placed in its own process. This would permit the visual display to be kept at a high enough update rate.

The Cognitive Coprocessor Architecture was developed at Xerox as a tool for building virtual reality user interfaces [55]. This architecture was designed to support smooth animation and multiple asynchronous interactive agents. This work was based on the Three Agent Model for supervisory control and interactive systems developed by Sheridan [60].

Distribution over multiple workstations to support the interactive rates of virtual reality interfaces is a main goal in the IBM VUE system [4]. This system assigns a workstation to each device including one for each graphics renderer. This distribution of a single process for each device has become more common since 3D devices are currently noisy and therefore require filtering. The more data used in the filter the better it performs. Therefore, a separate process is used to gather the data from the device at device rates and then supply the filtered results to the application upon request.

The Decoupled Simulation Model is included within the MR toolkit and provides low level support for the design of virtual reality environments [59]. This system provides a unified view of a tracker so that the system need not be recompiled in order to support new equipment or a new organization of equipment. A separate process is created for each tracker, the simulation component and for a geometric model component. This distribution is slightly different from the previous systems as it allows for dynamic model geometry adjustment. The geometric model component supplies different versions of the model to the viewer depending on the current view update rate.

Adachi et al. [2] and Mark et al. [36] have both presented distributed systems for haptic environments. The distribution consists of a graphical viewer and a haptic

control process. The haptic device supplies its position to the viewer so that a graphical representation of the devices end-effector can be presented for the user. A simple local representation of the geometry is provided to the haptic controller by the graphical process so that it can render an appropriate force for the user.

CHAPTER 3

DIRECT HAPTIC RENDERING

The goal of direct haptic rendering is to enable the user to feel the model actually designed instead of some secondary representation. In addition to an enhanced tracing experience, using the actual model allows the designer to modify the model without having to wait for the haptic system to convert the model in a time-consuming preprocessing step. We break direct haptic rendering of trimmed NURBS models into several phases.

- The system checks each model for proximity to the probe and activates those models deemed proximal.
- The tracking algorithm tracks the probe's global closest point on the model until contact is made.
- While in contact, the tracing algorithm maps the movement of the probe to movement of a local closest point along the surface of the model.
- The transitioning algorithm computes the local closest point on an adjacent surface when the probe's movement intersects a trimming loop boundary.

As with other forms of haptic rendering, this direct approach must find an appropriate closest point and surface normal for the force response model. These computations must cycle at haptic rates of around $500Hz$. More specifically, each instance of these computations must complete in under 1/500sec. Therefore, we can establish a budget to which the sum of all computations must adhere. The difficulty in this approach is that the computation of the requisite data requires surface evaluation, a time consuming operation when dealing with NURBS models.

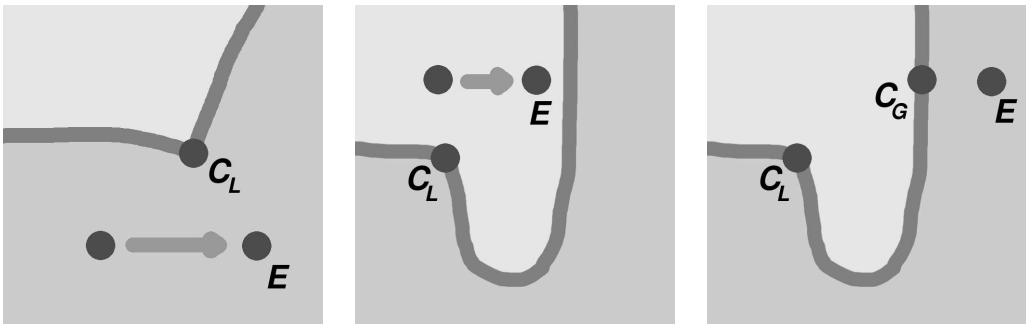
Minimizing the number of surface evaluations was therefore one of the primary goals in the development of this approach.

3.1 Local vs Global Closest Point

The *APE* system uses two types of closest points: *local* and *global*. When the probe is in contact with a surface, the tracing algorithm uses a local closest point to determine the virtual point of contact with the model's surface. However, when the probe is not in contact, a global closest point is used to track the next possible contact point. By using a global closest point, our system ensures proper contact detection by permitting the closest point to jump across concavities and to climb convex regions. Figure 3.1 illustrates the different closest point requirements.

During tracing, the probe (or end-effector in robotics terms), E, moves to a location resulting in the local closest point, C_L, becoming bound to the intersection of two surfaces (Figure 3.1a). This is the correct response: the probe is trying to move inside a second surface and should be restricted from movement in that direction. If the same algorithm was used for tracking, a similar scenario could occur (Figure 3.1b). Again the probe moves to a location resulting in the tracked point being bound to an edge. In this case, however, the closest point should not be bound since the tracked point is used to indicate the next possible point of contact. In fact, if the probe were to continue along the current path and intersect the model (Figure 3.1c), the contact would not be detected since the penetration would not occur at C_L. Instead, the global closest point, C_G, is the proper point of contact.

Once the model has been contacted, the surface intersected is deemed *current*. A virtual point of contact must be tracked that shadows the probe's movement locally on the current surface until either contact is lost or a transition occurs. Regardless of the haptic algorithm, a global closest point can not be used without producing several problems [66, 70]. Among these problems are pushing through a model, force discontinuities, and inability to generate sufficient restoring forces due to lack of penetration depth.



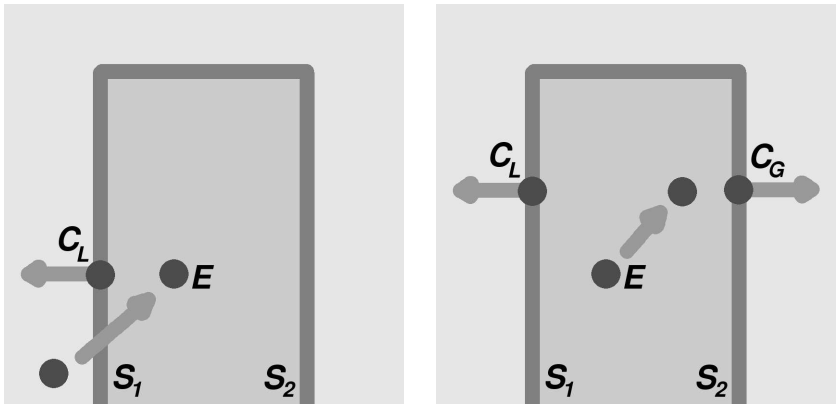
(a) (b) (c)

Figure 3.1. A local closest point, C_L, bound to a surface crease is desired for tracing (a) but not for noncontact tracking (b) where failing to use a global closest point, C_G, would result in a missed contact detection (c).

All three of these problems are illustrated in Figure 3.2. The model in this figure is a narrow rectangle constructed from multiple surfaces. Figure 3.2a shows the probe entering the model through surface S_1, which results in C_L being established as the current local closest point. The probe then continues to move into the model, resulting in one of two possible new configurations.

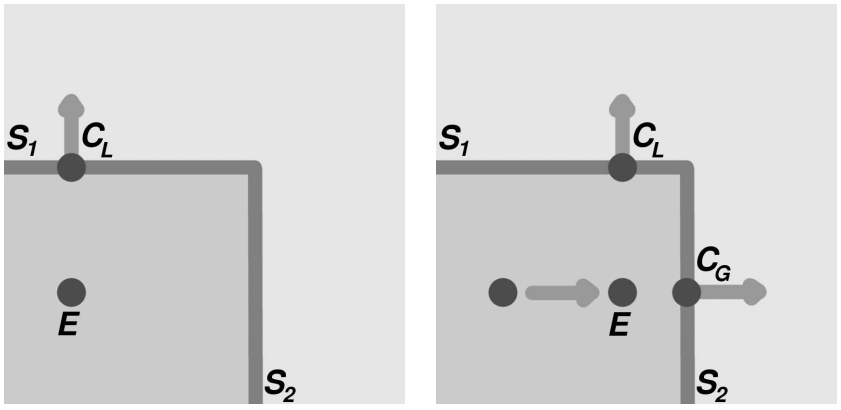
Our method holds S_1 as the current surface, therefore C_L stays on that surface resulting in a restoring force that is larger in magnitude but in the same direction as the previous iteration (Figure 3.2b). However, if a global closest point, C_G, was used then a force smaller in magnitude and opposite in direction than that of the previous iteration would result (Figure 3.2b). This clearly is not the characteristic one would want since it results in the probe pushing through the model, the force becoming discontinuous, and the penetration depth not growing high enough to generate a sufficient restoring force.

Generating a discontinuous force is possible not only when pushing through a model, but also at any time the global closest point differs from the local closest point. Consider the case illustrated in Figure 3.3. In this example, contact has been established with S_1 resulting in the given C_L (Figure 3.3a). The probe then moves to a position that results in a C_G that is not equal to C_L (Figure 3.3b). Our system would continue to generate restoring forces toward S_1, but a system that



(a) (b)

Figure 3.2. Contact established at point C_L (a). Use of global closest point, C_G (b), would accelerate the probe through the model.

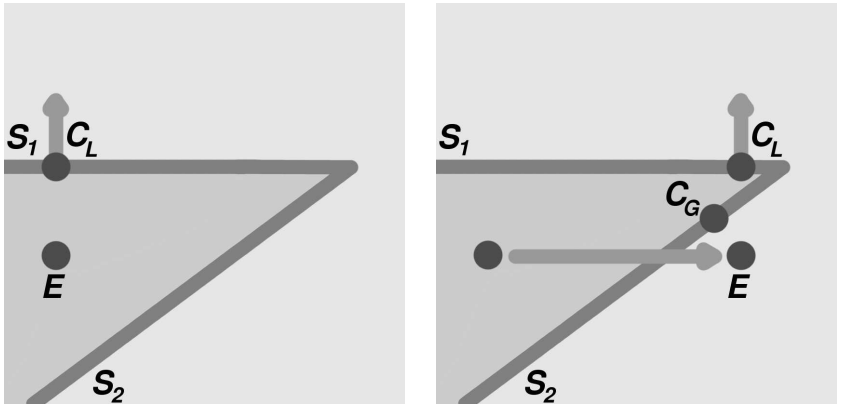


(a) (b)

Figure 3.3. Contact established at point C_L (a). Use of global closest point, C_G, would cause a slip to be induced off the model (b).

uses C_G would end up pushing the probe in a direction that it is already traveling, accelerating the probe off the model.

Another reason for using a current surface and C_L instead of C_G is to enable our system to trace out sharp edges (Figure 3.4). Consider a configuration where the probe has established contact with a model as in Figure 3.4a. The probe then moves out toward the edge (Figure 3.4b). C_L rests near the edge of S_1 while C_G not only lies on S_2 but would result in a negative penetration depth. Using C_G in this



(a) (b)

Figure 3.4. Contact established at point C_L (a). Use of the global closest point, C_G, would induce an artificial edge (b).

case would result in a zero restoring force, effectively portraying the probe falling off an edge that is not present in the model. Instead, our method would use C_L until the probe moved out past the edge of S_1 and then transition over the edge.

3.2 Proximity Testing

The haptic rendering algorithm presented needs to track only a single point per model per probe. However, it is still advantageous to keep the number of active models to a minimum. Doing so saves processor time which helps maintain high update rates as well as allowing for a more densely populated environment. To this end, our system checks the proximity of the probe to each model as the probe moves throughout the environment. We establish four levels of proximity: *distant*, *near*, *active* and *current*.

3.2.1 Four Level Control

By using four levels of proximity we can distribute the computational resources of our system to tasks that require highest priority. A model is termed *distant* if it cannot be reached by the haptic device (Figure 3.5). Such a model is of no interest to the haptic process since it is not a model that can be contacted by the haptic device. These *distant* models must still be checked for proximity since within

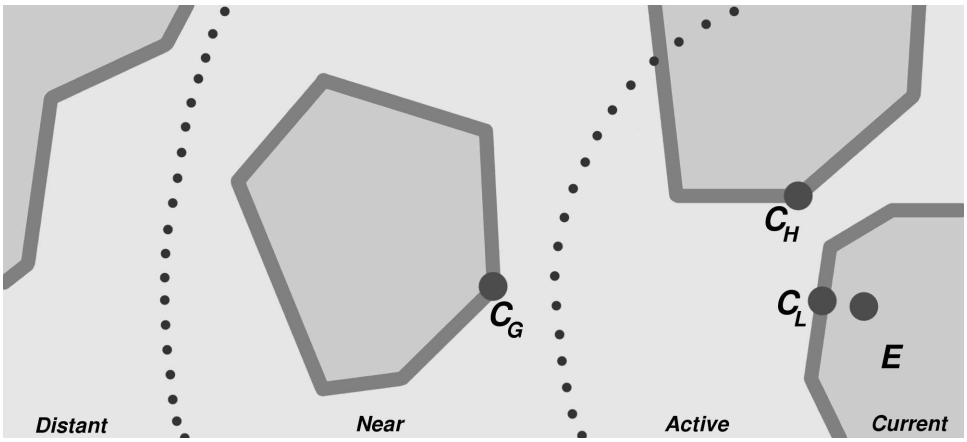


Figure 3.5. Model proximity for distant models is tested in the simulation process but no actual point need be determined. Near and active models require a global closest point with active models using a hybrid approach. The current model require a locally closest point.

a dynamic environment their status may change or the user may manipulate the view in such a way that the model moves to within the workspace of the haptic device. Since this computation is low in priority it is done within the simulation process. Neither an exact closest point nor an exact distance is required to make this determination. Using an algorithm that utilizes a LUB-Tree [27], we are able to determine whether a model is within a given distance and then abort the search without computing an exact answer. This approach makes it possible to test models more quickly, therefore allowing more models to be tested.

Once a model is proximal enough to be within the device's workspace the haptic process is signaled and the model is deemed *near*. At this point a global closest point, C_G, must be tracked on the model for the given probe within the haptic process (Figure 3.5). A model that is *near*, can be reached but is not a candidate for immediate contact. For this reason a lower priority is placed upon this computation than that for a model that is *active*.

An *active* model is one that is close enough to the probe that within the time it takes to compute a global closest point the probe may contact the model. Since the probe is still not in contact, a global closest point is required. However, the

tracked point needs to be computed at haptic rates. We use a hybrid approach that uses a local closest point between global closest point updates. The combination provides a good approximate to the true global closest point but at haptic rates, C_H, (Figure 3.5).

The final level of proximity, *current*, is when the probe is actually in contact with the model (Figure 3.5). This state requires the highest computation priority since it is in this state that forces are actually applied to the user. When *current*, no global closest point is tracked since a local closest point, C_L, is required for haptic tracing. A model remains *current* until contact is no longer present. At that point the model returns to being *active* and a global closest point is once again tracked along its surface.

3.2.2 Proximity Testing Results

The four level control methodology has proven to be highly effective at producing quality proximity testing. By default, the value for *distant* is computed as the size of the device workspace plus the distance a user can move the device within the amount of time it takes for a *near* message to be sent to the haptic process. This does not need to be exact but should err on the side of too loose rather than too tight so that in the worst case a model is deemed *near* too early.

The value for the *active* distance must be set to a value equal to or greater than the distance the user can move the device within the time it takes the haptic process to compute the next global closest point. This is not an easy number to establish as the more complex the model, or the more complex the scene within the haptic process, the slower the overall rate of the process. Therefore we chose a distance that assumes the worst case for our hardware and a hypothesized scene complexity, which would be $10cm$.

Perhaps the most important point to gather from this approach is that these proximity values are system specific. As hardware improves, given the same implementation, these value can be reduced. Further, the performance of the system

can be monitored and the values adjusted dynamically. Both approaches allow for more complex models and scenes.

3.3 Contact and Tracing

At the heart of a haptic system is its ability to detect the contact of the probe with a virtual model and to simulate the act of tracing along the surface of the model. Contact occurs when the probe first intersects the virtual model with a positive penetration depth. Similarly, if after contact the penetration depth becomes negative then contact is lost. The penetration depth is calculated by projecting the probe onto the surface normal. In our system, surface normals point out of a model. The projection must therefore be negated to result in a positive penetration depth when the probe is within the model.

Once contact has been established, any lateral movement along the model's surface indicates tracing. As the probe moves, a local closest point on the surface of the model is found to shadow the probe's movement. The accuracy of the computation for this closest point and its associated normal is essential since the restoring force calculation depends directly upon these results.

We relate movement of the probe to movement along the surface using direct parametric tracing (DPT) (Figure 3.6). This method works directly on the parametric surface and is fast enough to track a local closest point at haptic rates, making it suitable for direct haptic rendering [66]. For clarity, we derive the method on a NURBS curve and then show how the method is applied to NURBS surfaces. Appendix A defines NURBS curves and surfaces and derives some of their useful properties.

3.3.1 DPT on Curves

The DPT method is seeded with an evaluation point $\gamma(u^*)$, calculated using refinement [10, 52] where u^* is the coordinate for the point of contact (Figure 3.6a). At each time step as the probe moves (Figure 3.6b), the DPT method uses the previous point on the curve $\gamma(u^*)$, the tangent vector at $\gamma(u^*)$, $\gamma'(u^*)$, and the

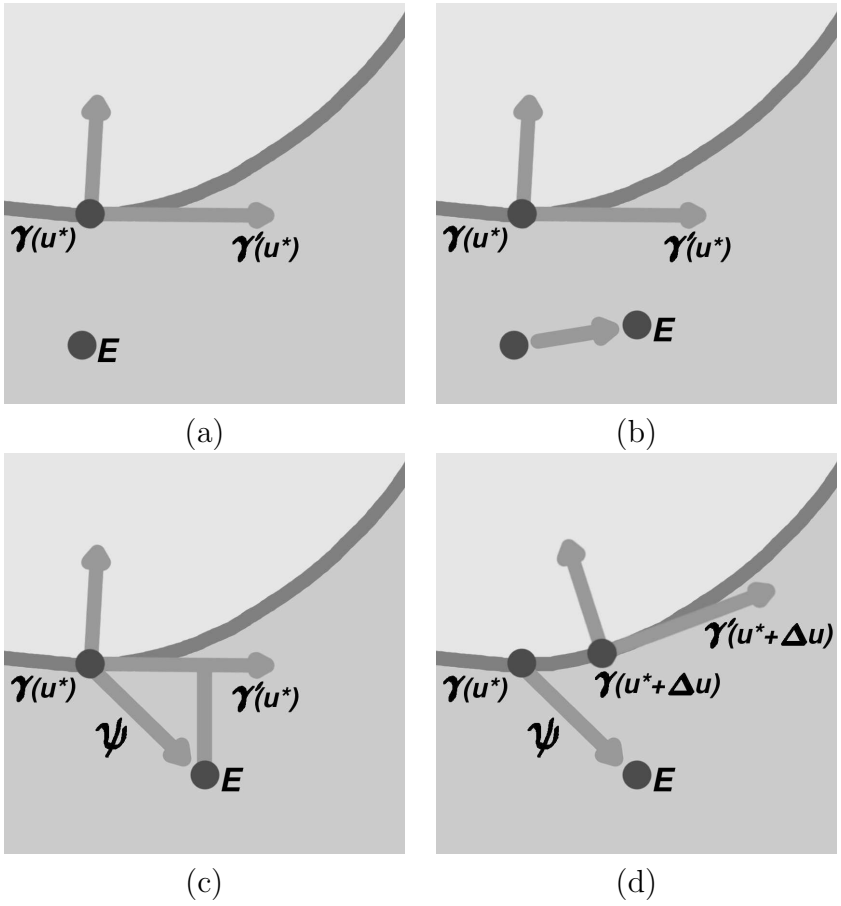


Figure 3.6. Direct Parametric Tracing. Initial state (a). Probe moves (b). Projection of probe onto tangent (c). New evaluation point and tangent plane via parametric projection (d).

current probe location, E, to determine a new approximate closest point on the curve.

The velocity curve, $\gamma'(u)$, relates changes in position along the curve in Euclidean space to changes in position in parametric space (Equation 3.1).

$$\gamma'(u) = \frac{d\gamma}{du} \approx \frac{\Delta\gamma}{\Delta u}. \tag{3.1}$$

Given an Euclidean movement along $\gamma(u)$, the corresponding movement in the parametric space of the curve is calculated as,

$$|\Delta u| \approx \frac{\|\Delta\gamma\|}{\|\gamma'(u)\|}. \tag{3.2}$$

In order to use Equation 3.2 as a closest point tracking method, movement of the probe needs to be related to movement of the closest point on the curve. The exact $\Delta\gamma$, corresponding to movement of the closest point along the curve, clearly involves finding the desired new closest point. Instead of finding an exact $\Delta\gamma$, a linear approximation to the curve, the tangent $\gamma'(u)$, is used to compute an approximate $\Delta\gamma$. The movement of the probe can now be related to movement of the closest point along the curve by projecting the offset vector, ψ , formed by subtracting $\gamma(u^*)$ from E, onto the curve tangent vector (Figure 3.6c). Thus,

$$\Delta\gamma \approx \frac{\langle\, \psi\ ,\ \gamma'(u)\,\rangle}{\|\gamma'(u)\|^2}\,\gamma'(u). \tag{3.3}$$

The equation for the velocity of a NURBS curve is derived in detail in Appendix B. The general form of the equation is given by,

$$\gamma'(u) = (k-1)\frac{\sum_{i=1}^{n}\sum_{j=0}^{n}\frac{w_i w_j(P_i-P_j)+w_j w_{i-1}(P_j-P_{i-1})}{u_{i+k-1}-u_i}B_{j,k}(u)B_{i,k-1}(u)}{\left(\sum_{j=0}^{n} w_j B_{j,k}(u)\right)^2}. \tag{3.4}$$

The contact point, $\gamma(u^*)$, is found by evaluating the curve with refinement by inserting $k-1$ knots into the knot vector with the value u^*. This results in a new knot vector $\{\hat{u}_i\}$, new weights $\{\hat{w}_i\}$, a new control polygon $\{\hat{P}_i\}$, and new basis functions $\{\hat{B}_{i,k}\}$ defined on the new knot vector. Further, $\gamma(u^*) = \hat{P}_{i^*}$ where i^*+1 is the location of the first knot in $\{\hat{u}_i\}$ with the value u^*. The properties of the refined curve result in a greatly simplified form for Equation 3.4. Only two basis functions remain active, $\hat{B}_{i^*,k}(u^*)$ and $\hat{B}_{i^*+1,k-1}(u^*)$, with both having a value of one. The resulting simplified equation is,

$$\gamma'(u^*) = \frac{(k-1)}{\hat{u}_{i^*+k}-\hat{u}_{i^*+1}}\frac{\hat{w}_{i^*+1}}{\hat{w}_{i^*}}(\hat{P}_{i^*+1}-\hat{P}_{i^*}). \tag{3.5}$$

Since we wish to track points that are actually on the curve, Equation 3.2 is used to convert back into parametric space. The key to efficient computation of Δu is Equation 3.5. The control polygon through $\gamma(u^*)$ lies in the tangent to the curve, and the parametric velocity is calculated using only the control polygon, knot vector, and curve order.

Using this simple relation and the linear equation for $\Delta\gamma$ (Equation 3.3), Equation 3.2 can be expanded into

$$|\Delta u| \approx \frac{\|\Delta\gamma\|}{\|\gamma'(u^*)\|} \approx \left| \frac{\langle\, \psi\,,\, \gamma'(u^*)\,\rangle}{\|\gamma'(u^*)\|^2} \right|. \tag{3.6}$$

The sign of Δu is determined by the sign of the projection in Equation 3.3. This is directly related to the dot product in the numerator of Equation 3.6. Since the numerator is the only term in Equation 3.6 that is signed, the absolute value signs can be removed. The constant term representing the parametric speed can be factored out leaving the result,

$$\Delta u \approx \frac{\langle\, \psi\,,\, (\hat{P}_{i^*+1} - \hat{P}_{i^*})\,\rangle}{\|\hat{P}_{i^*+1} - \hat{P}_{i^*}\|^2} \left(\frac{\hat{u}_{i^*+k} - \hat{u}_{i^*+1}}{k-1} \right) \left(\frac{\hat{w}_{i^*}}{\hat{w}_{i^*+1}} \right). \tag{3.7}$$

The new curve location, $\gamma(u^* + \Delta u)$, is a good approximation to the closest point to E. The new closest point is evaluated through multiple knot insertions at $u^* + \Delta u$, which maintains the conditions needed to use Equation 3.7 at the next time step (Figure 3.6d).

Essentially, we make a first order approximation of the closest point movement in Euclidean space with the tangent projection. The closest point movement is converted into parametric movement through a first order approximation to the parametric velocity at the previous closest point. The new closest point is then converted back into Euclidean space through curve refinement and evaluation. For small step sizes and penetration depths, which is the case for haptic rate computations, this provides an excellent approximation.

3.3.2 DPT on Surfaces

The spatial movement of the probe is related to movement along a surface by projecting onto the surface tangent plane at the previous tracked point. There are an infinite number of tangent vectors at any surface evaluation point. To form the tangent plane we require only two, but need the two selected to be linearly

independent in order to form a basis for the tangent plane. A good choice for these tangent vectors is the partial derivatives with respect to u and v,

$$T_u = \left.\frac{\partial S}{\partial u}\right|_{(u^*,v^*)}, T_v = \left.\frac{\partial S}{\partial v}\right|_{(u^*,v^*)}, \tag{3.8}$$

since they can be computed efficiently. This efficiency derives from using surface refinement to calculate the evaluation point $S(u^*, v^*)$. This refinement results in new knot vectors $\{\hat{u}\}$ and $\{\hat{v}\}$, new weights $\{\hat{w}_{i,j}\}$ and a new control polygon $\{\hat{P}_{i,j}\}$ where $\hat{P}_{i^*,j^*} = S(u^*, v^*)$. Furthermore, the i^* column and j^* row in the control mesh form control polygons for iso-curves that pass through the evaluation point $\hat{P}_{i^*,j^*}$. This allows the two tangent vectors to be calculated efficiently from the refined surface as if they were curve tangents,

$$\begin{aligned} T_{u^*} &= \frac{(k-1)}{\hat{u}_{i^*+k_u} - \hat{u}_{i^*+1}} \frac{\hat{w}_{i^*+1,j^*}}{\hat{w}_{i^*,j^*}} (\hat{P}_{i^*+1,j^*} - \hat{P}_{i^*,j^*}), \\ T_{v^*} &= \frac{(k-1)}{\hat{v}_{j^*+k_v} - \hat{v}_{j^*+1}} \frac{\hat{w}_{i^*,j^*+1}}{\hat{w}_{i^*,j^*}} (\hat{P}_{i^*,j^*+1} - \hat{P}_{i^*,j^*}). \end{aligned} \tag{3.9}$$

Since the tangent vectors need not be perpendicular to each other, it is incorrect to find the probe's coordinates within the tangent plane by projecting the offset vector onto the tangents using orthogonal vector projection. Instead, we relate the offset vector to a frame formed by the tangents and their cross product,

$$\psi = aT_u + bT_v + c(T_u \times T_v), \tag{3.10}$$

where a is the amount of movement along the surface in the direction of T_u, thus equal to Δu. Similarly, $b = \Delta v$. The value of c represents the penetration depth of the probe in relation to the closest point from the previous time step and is therefore not needed. To solve for a and b we form a system of two equations in two unknowns by taking the dot product of Equation 3.10 with each tangent in turn. The resulting system is then,

$$\begin{bmatrix} \langle\, \psi\,,\, T_u \,\rangle \\ \langle\, \psi\,,\, T_v \,\rangle \end{bmatrix} = \begin{bmatrix} \langle\, T_u\,,\, T_u \,\rangle & \langle\, T_v\,,\, T_u \,\rangle \\ \langle\, T_u\,,\, T_v \,\rangle & \langle\, T_v\,,\, T_v \,\rangle \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} \tag{3.11}$$

where the final term in each case has been eliminated since the resulting dot product will always equal zero. This system is solved efficiently using Cramer's Rule since the tangent vectors chosen are linearly independent,

$$\begin{aligned} \Delta u &= \frac{\langle\, \psi\,,\, T_u\,\rangle\langle\, T_v\,,\, T_v\,\rangle - \langle\, \psi\,,\, T_v\,\rangle\langle\, T_u\,,\, T_v\,\rangle}{\langle\, T_u\,,\, T_u\,\rangle\langle\, T_v\,,\, T_v\,\rangle - \langle\, T_u\,,\, T_v\,\rangle\langle\, T_u\,,\, T_v\,\rangle}\ , \\ \Delta v &= \frac{\langle\, \psi\,,\, T_v\,\rangle\langle\, T_u\,,\, T_u\,\rangle - \langle\, \psi\,,\, T_u\,\rangle\langle\, T_u\,,\, T_v\,\rangle}{\langle\, T_u\,,\, T_u\,\rangle\langle\, T_v\,,\, T_v\,\rangle - \langle\, T_u\,,\, T_v\,\rangle\langle\, T_u\,,\, T_v\,\rangle}\ . \end{aligned} \tag{3.12}$$

Notice that the 2×2 matrix in Equation 3.11 is actually the first fundamental form for surfaces. This quantity is invariant under coordinate transformation and derives from calculations that determine distances between points that lie on a surface.

3.3.3 Surface Evaluation

The DPT method is very efficient at finding the step size for the tracked point given a new probe position. This efficiency derives from the use of the control mesh and knot vectors of a surface evaluated at the tracked point from the previous time step. The majority of the time spent in the DPT method is spent performing this surface evaluation. It is possible to evaluate the surface by using the weighted combination of the control points computed through evaluation of the basis functions. This evaluation technique is not appropriate for DPT, however, since it also requires surface tangents.

A second approach to evaluating a surface is to use refinement [10, 52]. The process of surface refinement is to insert new knots into either knot vector. Multiple knots can be inserted in a single pass, but into only one of the knot vectors at a time. Depending on which knot vector is chosen, either a new row or column of control points is created for each knot inserted. When a new row is created the value of each new control point is calculated as the weighted sum of neighboring control points from the original mesh that lie in the same column. These surrounding control points also take on a new value based on a weighted sum of neighboring control points. The number of control points that contribute to the weighted sum is bounded by the order of the surface in that direction minus the number of knots

inserted with the same value. In the limit, as knots are inserted in both knot vectors this process produces a control mesh that tightly approximates the surfaces.

Given a surface that is of order k_u in one direction and k_v in the other, inserting $k_u - 1$ knots with the value u^* into the $\{\hat{u}\}$ knot vector and $k_v - 1$ knots with the value v^* into the $\{\hat{v}\}$ knot vector results in one of the new control points being an evaluation point $S(u^*, v^*)$. The control point in the new control mesh is $\hat{P}_{i^*,j^*}$, where the value of i^* and j^* are one less than the index of the first knot of value u^* and v^* in their respective new knot vectors. As a side effect of this process, tangent vectors can be efficiently calculated using the new control mesh and the knot vectors (Equation 3.9).

The process of refinement is accomplished via the use of an alpha matrix [10]. This matrix holds blending coefficients that, when applied to the original control mesh, produce the new refined control mesh. In the case of DPT, however, we do not require a complete new control mesh. DPT requires only the surface evaluation point and the two neighboring mesh points that determine the direction for the two tangents. The value of each entry in the alpha matrix is not dependent on any other, so we produce only those columns necessary for the needed points. Due to the property of local control in a NURBS surface, a point in the new control mesh will be the weighted sum of at most order number of points from the original mesh.

In the first pass, when refining the $\{\hat{u}\}$ knot vector, we have the complete original control mesh as input. The output from this pass is a control mesh that is k_v rows by two columns. The two columns correspond to i^* and $i^* + 1$ while the k_v rows are needed as input to the second pass for refining the columns. The second pass results in a two by two control mesh that holds the three requisite points plus one unused point. This is the minimum computation that can be performed to accumulate the necessary information through refinement. Further savings can be found through caching the alpha matrix for reuse in each pass. This computation is considered *local* since the order of the surface alone determines the necessary size for both the alpha matrix and the intermediate control mesh. The size of the surface's control

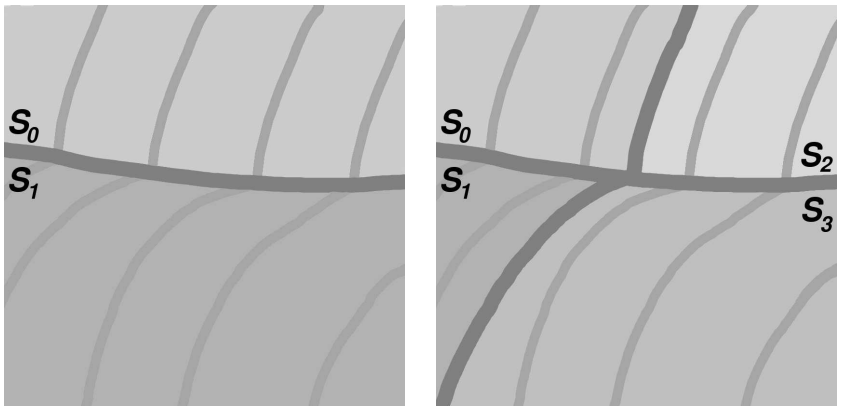
mesh does not matter. The overall complexity, in both memory and time, for this approach is therefore determined solely by the order of the surface.

3.3.4 Trim Tracing

Another form of tracing is that of tracing along the intersection between surfaces. This intersection is defined by the trimming information. The intersection between two surfaces is a single trim edge made up of perhaps multiple trim segments (Figure 3.7a). It is also possible for an intersection that appears continuous on the model to actually be a collection of edges in series (Figure 3.7b).

It is possible to build a separate data structure that represents all of the surface intersections as a collection of segments. However, doing so would significantly increase model preprocessing time when in fact such a structure is not necessary. Since adjacency information is available for all trim segments, moving along the intersection, from edge to edge, is not computationally intense (Section 3.5.5).

Trim tracing relates closely to DPT. The trim tracing algorithm must slide along the intersection in Euclidean space to a point locally close to the probes position. The algorithm projects the probe onto the current trim segment. If the projection is beyond either endpoint of the segment then we continue to project onto segments



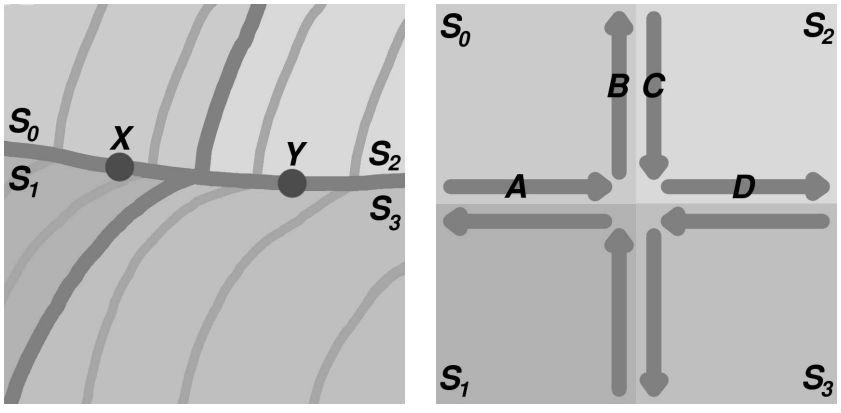
(a) (b)

Figure 3.7. Intersection of two surfaces formed by a single edge (a). Intersection along multiple surfaces requires multiple edges even though visually the intersection is the same (b).

along the loop in that direction until a local minimum is found. The final segment is the one that the projection results in a value bounded as $[0, 1)$ for that segment. This removes any ambiguity at segment endpoints.

If the final projection was internal to a segment, meaning not at its endpoint, then that projection is the final result. However, if the projection was at the segments endpoint then it is possible that the slide should continue along the intersection, but on an edge from an adjacent surface. For example, consider the intersection in Figure 3.8a. Given the probes movement from X to Y the slide along the trim will cease at the intersection of all four surfaces. Figure 3.8b illustrates the trim segments at this multiple surface intersection point. The initial slide results in trim segment B being the final result. Note that this is the case due to the constraint on the projection value being $[0, 1)$ for the segment, which rules out segment A. Since this result is at an endpoint the adjacent segment is found and trim tracing continues. In this case, the adjacent segment may appear to be segment C; however, this would not satisfy the bound constraint. The adjacent segment is actually segment D.

The algorithm terminates under two conditions. First, as mentioned previously, the algorithm halts if the projection lies internal to a trim segment. This should be



(a) (b)

Figure 3.8. Geometric view of trim trace(a). Multi-surface intersection point represented by their trim segments (b).

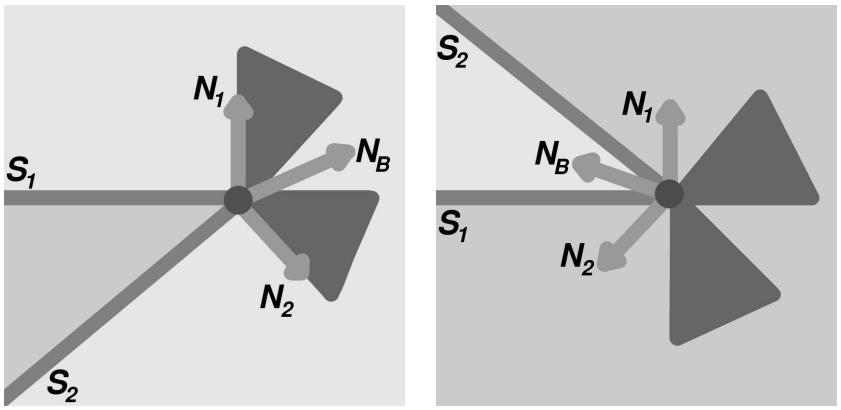
obvious as the result would not be different if the adjacent segment was used since it is identical to the final segment. The second case is that of the final result being at a segment endpoint. If a trace ends at a trim endpoint, the adjacent segment is found. If the trace for this, and each subsequent adjacent segment found, results in a halt at an endpoint, without any sliding to neighboring segments, the original segment will be found again, completing the cycle. This signals that the local closest point on the trim is at a multiple surface intersection point and that the algorithm should halt.

3.3.5 Boundary Normal

During haptic tracing along a surface, the normal used for both force reflection and contact detection is the actual surface normal of the tracked point. This normal is found by normalizing the cross product of the two tangent vectors used to form the tangent plane. However, if the tracing determines that the closest point lies on the boundary of two or more adjacent surfaces, such as during trim tracing, then a special vector must be constructed for use as the boundary normal. This vector, N_B would then be used for both contact detection and force calculation.

Consider Figure 3.9a. If point E lies within the shaded area then the closest point (either C_L or C_G) will be on the edge between S_1 and S_2. The entire shaded area is outside the model. If the normal for S_1, N_1, is chosen for N_B and E is within the lower shaded area, a positive penetration depth would result and indicate contact with the model. Since E lies outside the model, this result is deemed a *false contact*. Similarly, if E is within the shaded area at the top and N_2, the normal for S_2, is chosen for N_B an incorrect penetration depth would be computed and again a false contact would result. Figure 3.9b illustrates the converse where the entire shaded area lies within the model. In this case, if E lies within either the top or bottom shaded area a negative penetration depth would be computed if the incorrect normal is chosen. The result is an incorrect release from the model.

Consider first the case of the probe being in contact with the model and the tracked point lying on a surface intersection. In this case, a good solution for



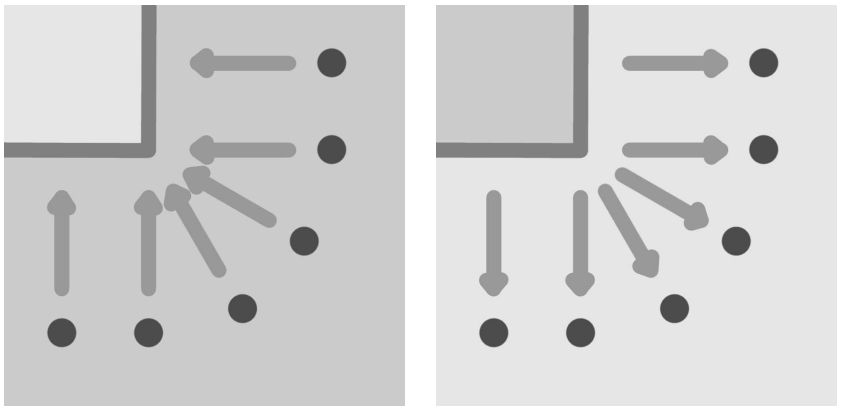
(a) (b)

Figure 3.9. The choice of normal at a surface boundary requires special care. In both (a) and (b) neither surface normal correctly classifies both shaded areas. However, N_B correctly classifies all areas.

N_B is the normalized vector formed by subtracting E from C_L. This vector points out of the model and can therefore be used in contact detection and force reflection. Further, it produces a smooth C^1 transition from one surface to the next (Figure 3.10a). Similarly, when the probe is not in contact with the model and the tracked point lies on a surface intersection, the negation of the above formulation can be used for N_B. Again, this vector will point out of the model and produce a C^1 transition across surface boundaries (Figure 3.10b). However, this raises the problem that the direction of the boundary normal requires previous knowledge of the probe's contact status.

The construction of the boundary normal is the same regardless of contact status up to the point of the possible final negation. For this reason, we form N_B under the assumption of no contact since this formulation is simply the normalized offset vector ψ. This vector is then negated if it points into the model.

At this point we make a couple of observations about the collection of surface normals, N_i, calculated from the adjacent surfaces, S_i. If there are only two adjacent surfaces then the two normals define the edge. If there are more than two normals then every pair of normals defines a *virtual edge* in the model. Given that there are n surface normals, there are $\binom{n}{2}$ virtual edges. The angle between any two normals



(a) (b)

Figure 3.10. The boundary normal, whether the probe is inside (a) or outside (b) the model points out of the model and produces a smooth transition.

must be less than π since any larger angle would define a negative volume between the associated adjacent surfaces. Further, since all N_i are less than π apart from all other N_i, all N_i must lie within a single hemisphere of the unit sphere.

Since all N_i exist in a single hemisphere, there exists a single vector that best represents the collection. This vector, N_A, is the axis of the tightest cone bounding all N_i. This cone also bounds all space points that map to the C_L lying along a surface intersection since the extremal N_i also represent edges of the open-ended polytope bounding this space. Therefore, if the dot product of N_B and N_A is negative then the boundary normal must be negated to point out of the model.

The problem of calculating a boundary normal is therefore reduced to finding N_A. However, it can be shown that if N_i are taken as points on the unit sphere then N_A must pass through the closest point from the origin to the 3D convex hull of that point set. A proof of this is presented in Appendix C. Further, the convex hull of N_i is exactly defined by the points N_i since any collection of points on a sphere define their convex hull. We use Gilbert's algorithm [18] to find this closest point, and therefore N_A, in linear time.

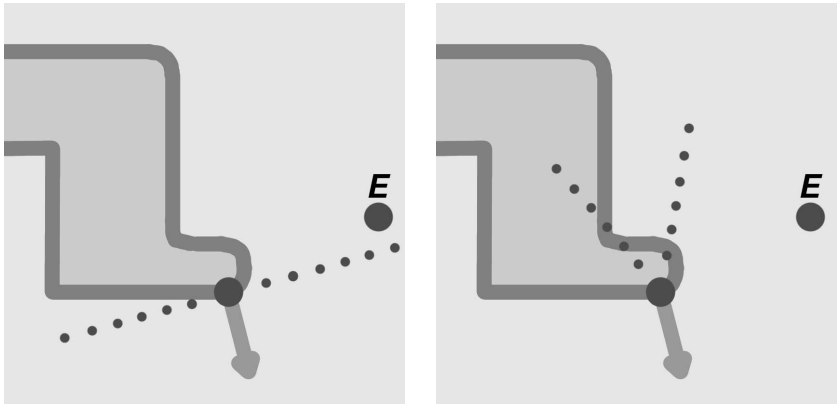
3.3.6 Eliminating False Contacts

Proper selection of a boundary normal eliminates one form of false contact; however, a second form can arise if the tracking algorithm produces sufficiently incorrect results in an area of high curvature (Figure 3.11a). While the error from the tracking algorithm is generally small, its use of the tangent plane as an approximation to the surface is less correct in areas of high curvature. This can result in larger errors, especially as the distance to the surface increases (Section 3.3.8).

To eliminate this form of false contact, after determining what side of the tangent plane the probe lies, we check the accuracy of the tracked point. We form a new offset vector ψ^* by subtracting the current tracked point, $\gamma(u^*)$, from the probe location, E, and then normalizing. If the tracked point is exact then the surface normal and ψ^* will be coincident. Therefore, a good measure of accuracy is the angle between these two vectors. Since the tracking algorithm produces an approximate closest point there will typically be some angle between these vectors. However, as shown in Section 3.3.8 the error is usually very small.

As a result, our solution is to construct a *contact cone* that ψ^* must lie within for contact to be detected (Figure 3.11b). Essentially, we are placing a tighter constraint on contact detection by selecting an angle of confidence that reflects the worst case we are willing to accept. The choice for this angle is dependent on the error of the tracking algorithm when a model is active. Since a model is deemed active at a specific distance the selection of the angle for the contact cone can be easily chosen. In practice, we have found that 25 degrees works well at eliminating false contacts while not discarding true contacts.

It is important to note that since the application of forces to the user during contact keeps the probe very near the surface, a false release is generally not a problem. For this reason, we do not use this accuracy based approach for release detection.



(a) (b)

Figure 3.11. High curvature and distance can result in false contact (a) unless a tighter bound on contact is used (b).

3.3.7 Internal Edges

It is necessary to use the boundary normal generation algorithm discussed above whenever there is an edge in the model. Edges can occur internal to a single surface as well as where multiple surfaces are adjacent to one another. If a surface knot vector of order k has $k-1$ knots of equal value the surface will be only C^0 continuous along the iso-curve at that value. It is possible for the surface to still be smooth across this iso-curve despite the drop in continuity, such as during knot insertion for surface evaluation. However, it is possible for the surface to form a sharp edge at any point along the iso-curve. Any sharp edge on a surface that is not formed by a trimming loop is called an *internal edge*.

At such a parametric location the tracing algorithm can not insert knots for refinement in order to produce a new control mesh. The original control mesh could still be used to form the surface tangents, but it is possible that the tangents on each side of the internal edge will be different. The tracing algorithm assumes that the tangent plane forms a good approximation to the surface, but at such an edge this is obviously not the case.

To alleviate the need for special purpose code to detect and properly handle these internal edges, we eliminate them. As a model is input into the system it is

checked for internal edges. Each surface that has an internal edge is subdivided into two surfaces with their shared edge being the previously internal edge. Thus, the internal edge is replaced by a trimming edge along the boundary of the two new surfaces. The resulting model will have more surfaces than the original; however, since the tracing algorithm is model based, the increase in surface count does not adversely effect the performance of the trace algorithm. In fact, it has the converse effect of maintaining the algorithm's performance.

3.3.8 Contact and Tracing Results

Using the Phantom, we traced a number of virtual models with direct parametric tracing and recorded the results. One measure of the quality of a haptic rendering is the amount of penetration depth into the model. A lower penetration depth indicates better surface normal and penetration depth calculation. However, having a small average penetration depth is not sufficient to demonstrate a smooth tracing experience. Penetration depth must also be shown to be consistently near the mean.

Table 3.1 shows that DPT easily meets the $500Hz$ goal for all models tested. The accuracy is approximately the same for all models. The mean is given to show the average penetration depth, but perhaps the better measure is the median. The median shows a lower value than the mean, which demonstrates that the depth produced by the trace is typically lower than the average. The maximum depth is near the mean which shows good contact response by DPT. The final two columns

Table 3.1. DPT tracing results

	Update	Trace		Depth		Percent Below	
Model	Rate	Samples	Mean	Median	Max	Mean	$3mm$
Cube	8426	1940	3.57	2.85	5.61	64.95	52.78
Goblet	1182	2310	3.30	2.01	6.27	81.00	78.53
Teapot	2142	2522	4.30	3.37	6.85	76.45	36.48
Brake	2451	2000	3.33	2.28	5.30	76.03	66.50
Gear	1671	2000	4.06	2.99	8.18	77.80	50.30

DPT update rate in Hz, mean, median, and maximum penetration depth in mm, and percent samples below the mean and $3mm$.

show the percentage of the trace that resulted in values below the mean and below a good $3mm$ trace depth.

To further illustrate the consistency of the trace algorithm, histograms are given in Figure 3.12 for the goblet, teapot, brake, and gear models. These histograms show a tight distribution of penetration depths with peaks at less than $4mm$. The combination of Table 3.1 and Figure 3.12 illustrate a consistent and smooth tracing experience.

Surface evaluation is a major component of the computation cost for DPT. Our method of surface evaluation computes only those elements of the alpha matrix necessary and stores them in cached memory. In comparison to full refinement and noncaching evaluation, this is significantly faster (Figure 3.13). Further, this evaluation technique is dependent only on the order of the surface and not the dimension whereas full refinement is dependent on both factors. For this reason the two evaluation groups, the top and middle, in the figure are near linear while the refinement approach degrades at increased control mesh dimensions.

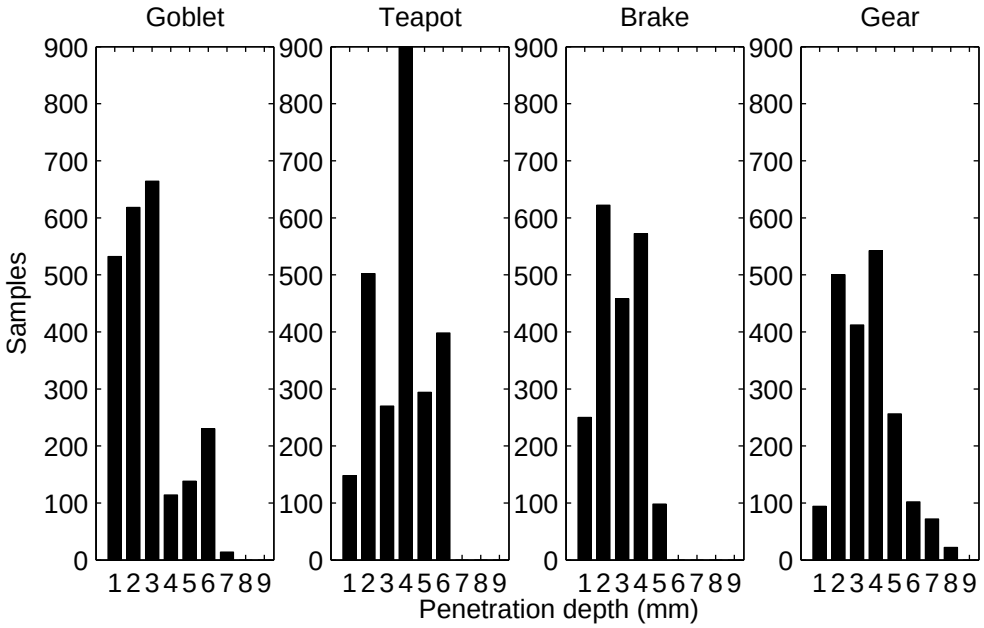


Figure 3.12. Histograms for penetration depth in mm of goblet, teapot, brake, and gear models.

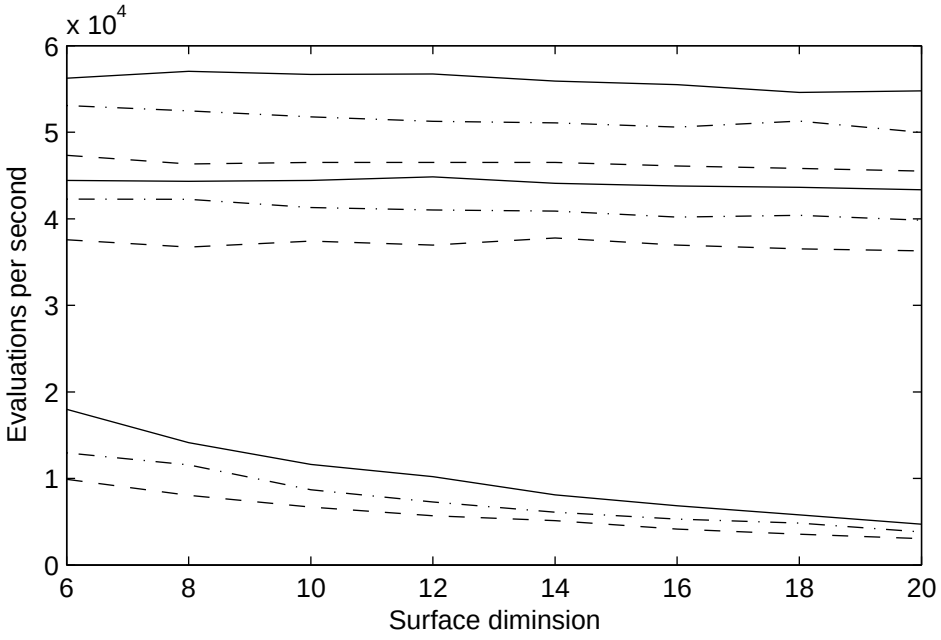


Figure 3.13. Speed of surface evaluation for three techniques. Top three lines are cached evaluation, next three are noncached evaluation, and the final three are full refinement. In each case the three lines represent order 3, 4, and 5 from top to bottom.

Note that the trace results given also include tracing along surface boundaries, which means both trim tracing and boundary normal calculation were required. Since both of these values are completely deterministic, no accuracy results need to be given. However, both computations contribute to the computational time and therefore must meet the same rigorous requirements. Our tests have shown that at $500Hz$ we can trace over 5000 trim segments. Similarly swift, given an intersection of 20 surfaces our algorithm can properly compute the boundary normal at over $67000Hz$. Neither of these cases is likely to occur, showing the approach easily satisfies the constraints.

In order to separate out error introduced by limitations in the Phantom and in the wall model, we also ran a number of simulated tracings on a bumpy surface. The results of direct parametric tracing were compared to those for global closest point on surface. The tracing path was generated by creating a nonisoparametric offset curve from the surface and evaluating the curve at fixed parametric steps.

Figure 3.14 shows the difference in penetration depth and surface normal found using the direct parametric tracing method with that found using the optimal solution. These two metrics are shown since they directly represent the resulting force that will be reflected upon the user.

Note that the method was tested under a wide range of conditions. The largest offset curve depth corresponds to a tracking distance of $10cm$ and a Euclidean movement of $2mm$ between each sample. Even under these extreme (and unlikely to be encountered) conditions, the algorithm performed reasonably well. This graceful degradation shows the algorithm has time-critical qualities [16], a useful property in real-time systems. In more typical cases, with small penetration and small step sizes, the penetration error was below our numerical precision and the difference in surface normals was in the hundredths of a degree (Table 3.2).

The Euclidean distance error for the surface point (Table 3.2) shows that under the conditions we measured, parametric tracing was capable of resolving the closest point on the surface to within $0.01032mm$. In combination with the error in the

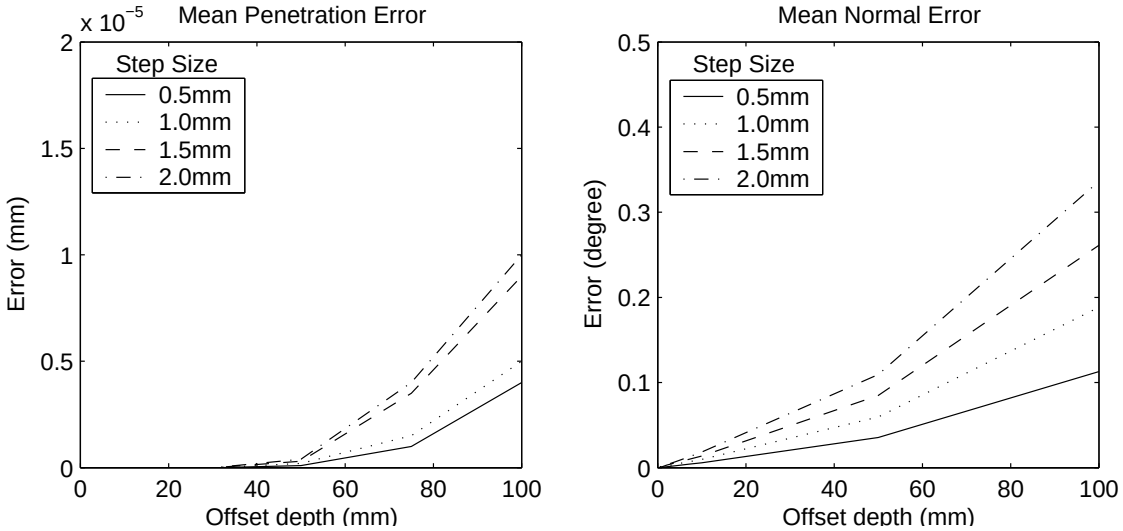


Figure 3.14. Mean penetration depth error (a) and mean normal error (b) vs. trace curve offset depth. Each line represents a different step size along the trace curve.

Table 3.2. DPT error results

Error	Depth (mm)				
Metric	1.0	2.0	4.0	7.0	10.0
Penetration	0.00000	0.00000	0.00000	0.00000	0.00000
Normal	0.00115	0.00286	0.00573	0.00974	0.01375
Surface Point	0.01032	0.01102	0.02721	0.04607	0.06130

Error values (in mm and degrees) as compared to an optimal solution when tracing at a step size of $1.5mm$ at five depths beneath the surface.

normal being below 0.1 degrees, the force vector as computed would be highly accurate giving a true representation of the underlying geometry. Further, the low error in the normal illustrates the validity of our cone approach to eliminating false contacts.

3.4 Tracking

Any movement around a model deemed near or active is referred to as tracking. Since tracking requires a global closest point be used in order to indicate the next possible contact point, the tracing algorithm cannot be used exclusively. The tracing algorithm correlates probe movement to movement of a local closest point bound to the surface of the model. The goal being to restrict the probe's movement to a path along the models surface. During tracking, however, there should be no restrictions on the probe's movement or the closest point which shadows it.

When a model is near, but not active, the global closest point tracked does not need to be updated at haptic rates since the model is not a candidate for imminent contact. In this case, the global point tracked is used to determine when the model should become active. Once active, however, the global closest point must be updated at haptic rates to ensure correct contact detection.

The method used to calculate the global closest point is based on a LUB-Tree approach [27]. In this approach a hierarchy is constructed that facilitates efficient upper and lower bound computations for the contained geometry. Using these bounds in conjunction with a breadth first search, nodes that cannot possibly hold

the answer are pruned away quickly without the need to recurse to the leaf nodes. This approach is modified to handle trimmed NURBS surfaces and to use a final numerical solver instead of a more time consuming subdivision technique.

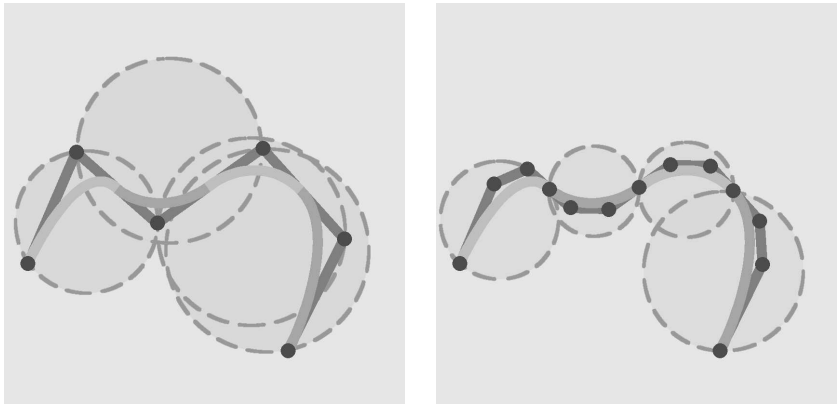
3.4.1 LUB-Tree Construction

There are many possible choices for the LUB-Tree bounding volumes. Since speed is a main concern, we choose the bounding volume with the lowest time cost: the sphere. The main disadvantage to using spheres as a bounding primitive is that in general they do not fit the underlying geometry tightly. However, its speed in checking distance allows for more levels in the hierarchy without loss of overall computation time when compared to tighter bounding volumes.

We build the hierarchy so that a leaf sphere bounds a surface patch represented by the parametric domain within each paired span of the knot vectors. One choice for the location and radius of the sphere is to build the sphere so that it tightly contains the convex hull for the surface patch under consideration. This convex hull is formed from the surfaces control points (Figure 3.15a). These convex hulls however do not necessarily fit the surface tightly and also overlap one another considerably as the order of the surface increases.

The overlap can be eliminated and the convex hull tightened simultaneously by converting the surface into a collection of Bezier patches through refinement. A single patch is built for each of the span regions. Each patch has its own convex hull for the underlying surface that is much tighter than the convex hull from the original surface (Figure 3.15b). Computations on the underlying geometry are more expensive than those performed on the bounding volume. Further, the bounding volume chosen allows for additional levels in the hierarchy without significant cost increase. Therefore, we insert an additional knot into the middle of each knot vector span prior to the Bezier conversion. This creates four times as many leaf nodes while only adding at most two levels to the binary tree hierarchy.

Before a sphere is built, we first check to see if the patch in question has valid surface domain. If the patch is completely trimmed away by a trimming loop then



(a) (b)

Figure 3.15. Using convex hulls from the original surface produces loose overlapping spheres (a). Converting to Bezier patches first produces tighter nonoverlapping convex hulls which results in tighter spheres (b).

no sphere is formed. If there is valid domain then a leaf node is constructed with a sphere fit to tightly bound the convex hull of the patch. Also stored in the patch is a single parametric point centered within the patch as well as a single valid Euclidean point on the surface patch that can be used for upper bound computations.

The remainder of the LUB-Tree is built by successively pairing the nodes. Each pair is bound by the tightest fitting sphere to contain the component spheres. The diameter of the new parent sphere is equal to the sum of the radii of the child spheres plus the distance between their mid-points. Its center point is located along a segment connecting the center points of the child spheres. Its parametric location from the center of the first sphere is the parent radius minus the radius of the first sphere divided by the distance between the child spheres. One special cases exists and that is the case where one of the two sphere's is contained within the other. In this case the new parent sphere will simply be a copy of the larger child sphere. A collection of the Euclidean surface points stored in the child nodes is gathered within the new node. The number of points in the collection is kept below six and randomly chosen to represent the underlying geometry. This process continues until the tree is fully constructed resulting in a single root node.

3.4.2 LUB-Tree Traversal

Finding the global closest point involves a breadth first search of the LUB-Tree. The main advantage of this approach is its use of bounds on the possible distance to the closest point. If a lower bound to a node is greater than the current upper bound the geometry within that node cannot contain the closest point and the node can be pruned away.

The search algorithm begins by finding the global closest point to the trim segments of the model. This result is stored as a possible solution and the distance to this point is used to set the initial value of the upper bound on the solution. Meaning, the global closest point can obviously be no further away than the closest point on the trim segments. However, the global closest point might be closer so the initial lower bound is set to zero. These bounds make all geometry part of the solution space.

Following the initial trimming segment check, the algorithm searches across each level of the binary tree. The distance to each sphere is used to find a new lower bound on the solution. Any node that has a lower bound greater than the current global upper bound is pruned away, along with its subtree. A new upper bound is computed using the node found with the least lower bound. The minimum distance from the probe to each of the points stored within the chosen node is the upper bound for the node. If this upper bound is smaller than the current global upper bound it becomes the new global upper bound. At this point the search drops to the next level in the tree and repeats the lower bound pruning process.

When a leaf node is reached its lower bound is first checked to see if the node can be pruned away. If it cannot be pruned, then an exact closest point must be found. The parametric point saved within the node is used as a seed point for a Newton search. The search is performed on the original surface with the search point not being constrained to be within the trims or within the defined bounds of the patch the node represents. Once the algorithm has converged, the point is checked against the trim loops to verify it is a valid surface point. Upon passing this test the point is compared against the current stored closest point. Even though

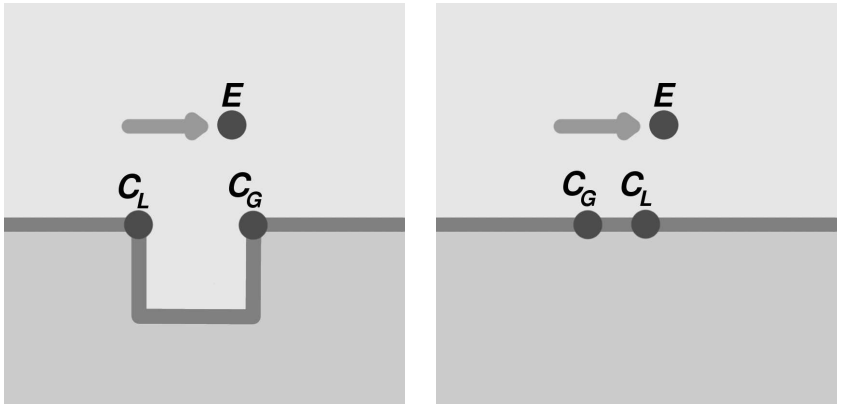
the point found may not lie within the patch, it is still used as it may be the correct answer. There is no reason to discard it only to search for it again later. Further, if it is not the final answer it still may provide a tighter upper bound that can eliminate other nodes from consideration. A point that does not pass the trim test is discarded.

This traversal algorithm continues until all nodes have been pruned. If no answer was found during the traversal then the point found on the trimming loops is the globally correct answer. Any answer found during traversal that is closer than the trim point will replace it as the global answer.

3.4.3 Hybrid Approach

It is currently not feasible to use this, or any other, global closest point routine for trimmed NURBS surfaces exclusively since the routines cannot run at haptic rates. Therefore we use a hybrid approach where the tracing algorithm for tracking a local closest point is used between global closest point updates. If upon the completion of the global closest point calculation the answer is found to be better than the current point found using the trace algorithm, it is used to reseed the trace algorithm (Figure 3.16a). Otherwise the local point is used until the next available global closest point can be found (Figure 3.16b). This comparison is necessary since the global closest point algorithm uses a probe position that is a few iterations old by the time the computation has completed. The trace algorithm updates its closest point every cycle so may have a more accurate result for the current probe position.

This periodic reseeding allows the tracing algorithm to form a good approximation to the global closest point between updates. We have found this to be very effective since the probe typically does not move very far between updates. Even though contact detection may be delayed for several cycles, the cycle rate is so high that the delay is, in practice, imperceptible to the user.



(a) (b)

Figure 3.16. Global closest point replaces trace point when more accurate (a). It is discarded when the trace point is better (b).

3.4.4 Tracking Results

Table 3.3 summarizes the important data concerning the use of a LUB-Tree representation as presented in the previous sections. The first column shows the time to setup the tree for the five test models. In all cases the setup took well under a second, not affecting user interaction.

The next two columns give results for searching the tree. When determining if a model should be considered *near* the actual closest point need not be computed. It is only important to know the relationship of the model to the distance separating *near* from *distant*. For this reason the rates in the *Near* column are better than

Table 3.3. Tracking results

Model	Setup	Near	Active	GCP	LCP	Mean	Used
Cube	0.0109	2802.40	1959.66	18058	54069	2.99	81.88
Goblet	0.1404	593.43	286.19	8082	27116	3.36	84.82
Teapot	0.1653	209.88	130.23	6972	25556	3.67	89.13
Brake	0.2970	351.87	166.38	7924	26506	3.35	77.03
Gear	0.3984	70.12	64.29	2664	12011	4.51	84.72

Time to setup LUB-tree in seconds, rate to determine *near* in *Hz*, rate to compute GCP when *active*, number of GCP and LCP computed, mean number of LCP between GCP updates, and percentage of GCP used to update tracking point.

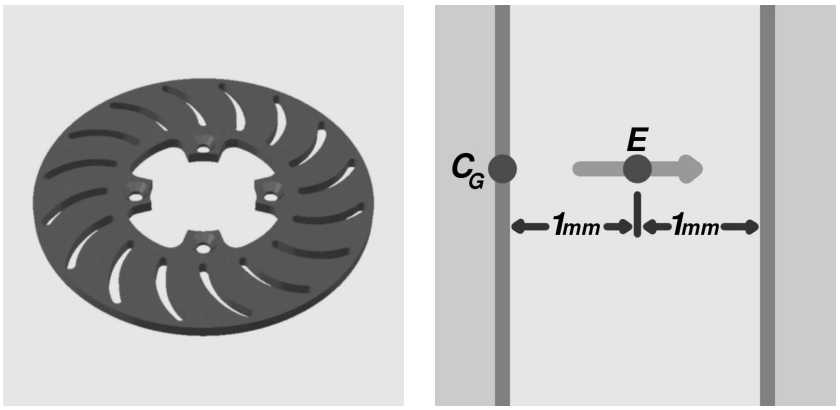
those in the *Active* column. An *active* model requires the final GCP be computed in order to update the tracked point using our hybrid algorithm.

The final four columns illustrate the use of the GCP as computed from the LUB-Tree within the hybrid tracking algorithm. The *GCP* and *LCP* columns give the number of values computed during a session where the probe circled the model. The *Mean* column shows the average number of LCP values that were needed between GCP updates. Finally, the *Used* column indicates the percentage of GCP values used to update the LCP to obtain our tracking point. This final column can be misleading as its results depend on how often the tracked point crosses either trimmed away regions or local concavities. It can also be affected by the distance the probe is from the surface as the direct parametric tracing becomes less accurate at greater distances. In this case the probe was circling just within the *active* distance of $10cm$ at a step size of near $2mm$. Table 3.3 shows direct parametric tracing to be fairly accurate for this combination, but an update by the GCP value may still be warranted.

It is important to note that the speed of the tracking algorithm, in particular the GCP update, directly determines the feature size that can be traced. Consider the grooved slots in the brake model (Figure 3.17a). If the user can move the probe $1mm$ between GCP updates, then the smallest width that any feature can have is $2mm$. This results from the possibility that the GCP is computed while the probe is directly in the center of the groove and results in a point on the opposite wall from the direction the probe is moving (Figure 3.17b). The next GCP needs to be computed within the time it takes the probe to traverse the remaining $1mm$ to contact the model.

3.5 Transitioning

Most CAD models consist of multiple surfaces. It is necessary for the tracing algorithm to allow smooth transitions (a) across, (b) onto, or (c) off of surface boundaries if the probe traces out such a path (Figure 3.18). This computation di-



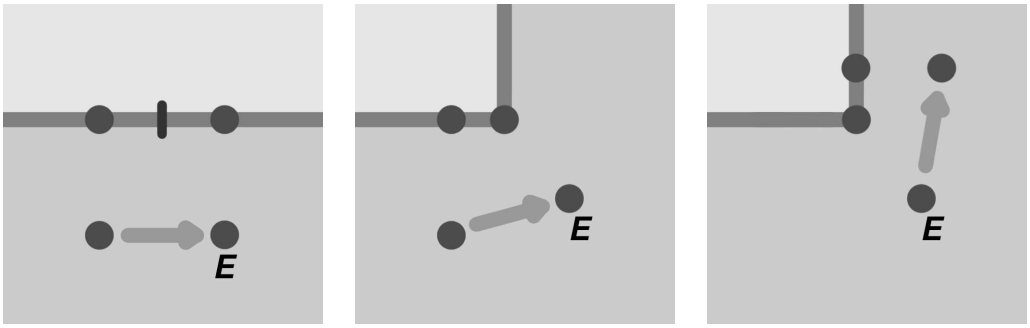
(a) (b)

Figure 3.17. The smallest feature size for a model (a) is determined by the distance a probe can move between GCP updates (b).

rectly affects the resulting closest point, normal, and penetration depth. Therefore, it must be efficient enough to operate at haptic rates.

The trimming loops of the current surface are of primary concern since all transitioning occurs at trimmed surface boundaries. In the first form of transitioning, across a trim boundary (Figure 3.18a), the algorithm must detect an intersection of the probe's path with a segment of the surface's trimming loops. Following this determination, the adjacent surface and an equivalent point on that surface must then be determined in order for the trace to continue.

The second and third forms can be thought of as a partitioning of the first form across multiple trace cycles. The second form, onto a trim boundary, requires the same trim intersection calculation, but if it is determined that the trace point cannot cleanly transition onto the adjacent surface, the transition is instead made to trim tracing (Figure 3.18b). The third form, off of a trim boundary, occurs when it is determined that the trace should leave the trim (Figure 3.18c). This ability to detect a release from the trim is the same functionality required by the first form when it must be determined where on the adjacent surface the trace should continue.



(a) (b) (c)

Figure 3.18. Transitioning across (a), onto (b), and off of a trim boundary (c).

3.5.1 Grid Overlay

A complex trimmed model can have thousands of trim segments on a single surface. Using a brute force intersection algorithm to check every trim segment on the current surface is not feasible, given the time constraints of a haptic algorithm. Our solution to this problem overlays each surface with a grid where each cell contains any trim segments that lie within or intersect it (Figure 3.19). Ideally, each cell would contain one segment and each segment would be contained in exactly one cell, resulting in the number of cells equaling the number of segments. Such a grid would result in heavy preprocessing overhead, and would preclude the use of a uniform grid. This is neither practical nor is it necessary.

The size of the grid is set such that it tightly bounds the outer most trimming loop for the surface. Since any domain outside the outer most trimming loop is not valid, the grid need not contain it. We then construct a square grid such that each dimension is twice the square root of the number of total trim segments for the surface. This results in $4n$ total cells and has proven an effective heuristic for maintaining low segment counts within each cell without introducing unnecessary complexity to the grid structure.

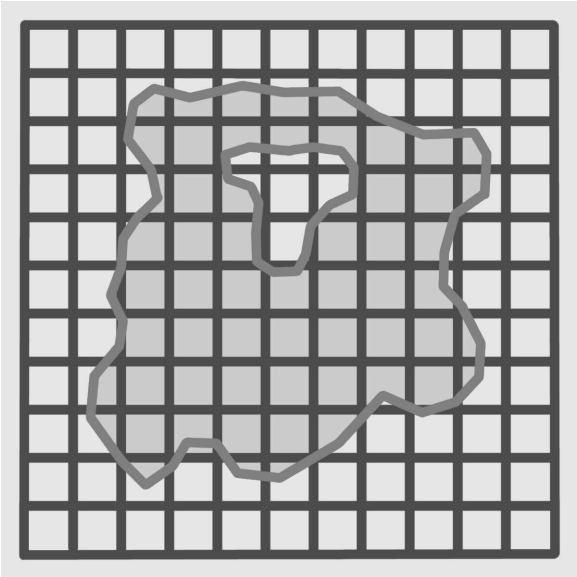


Figure 3.19. A grid overlay on a trimmed surface.

3.5.2 Grid Walking

Once the grid has been constructed, it is necessary to have an efficient method to move from cell to cell within the grid as the probe traces along the surface. We describe this movement through the grid as *grid walking*. The first step to solving this problem is locating where within the grid a point lies. Given a parametric point (u, v) the coordinates of the cell are given by

$$ci = \lfloor (u - u_{\mathsf{min}})/u_{\mathsf{dim}} \rfloor \ , \ cj = \lfloor (v - v_{\mathsf{min}})/v_{\mathsf{dim}} \rfloor, \tag{3.13}$$

where u_{min} and v_{min} are the minimum u and v values of the grid and u_{dim} and v_{dim} are the dimensions of each u column and v row. If either ci or cj is equal to *size*, the number of rows (or columns since the two are equal), then it is decremented by one. This is necessary in order for the last column and last row to contain the outer most edge of the cell.

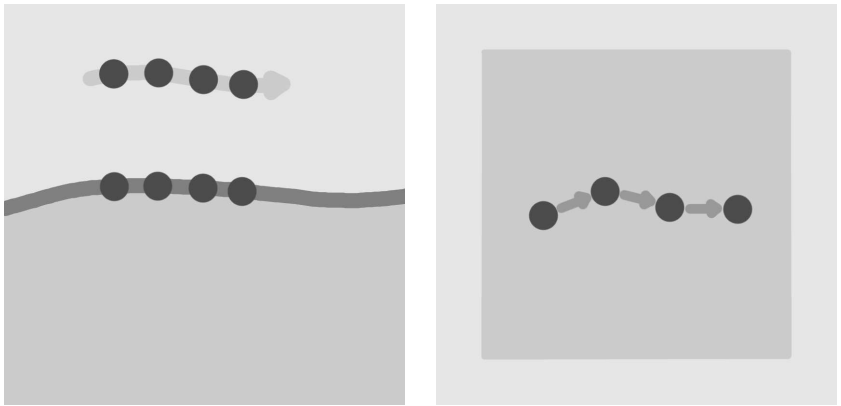
While the movement of the haptic device by the user is continuous, the movement of the probe is discrete since it is a time sampled version of the actual device (Figure 3.20a). Discrete motion along the surface corresponds to a directed line

segment in parametric space (Figure 3.20b). This segment, or *movement vector*, has as its start point the current contact point's parametric coordinates, (u^*, v^*). The end point is the next closest point computed using DPT, $(u^* + \Delta u, v^* + \Delta v)$. The resulting vector has a value $(\Delta u, \Delta v)$.

Given the location in the grid of the start point for the movement vector (first calculated when the probe is deemed *near* and subsequently known from the previous iteration), we wish to walk along the vector, stepping through each cell it intersects until the end point of the vector is reached. Define t to be the parameter along the movement vector, and *next_t* to be the value of t at the entry of the next cell the vector intersects. We can then define and initialize $next\_t_u$ and $next\_t_v$ as,

$$\begin{aligned} next\_t_u &= \begin{cases} (u_{\mathsf{min}} + (ci+1) \cdot u_{\mathsf{dim}} - u^*)/\Delta u & \Delta u > 0 \\ (u_{\mathsf{min}} + ci \cdot u_{\mathsf{dim}} - u^*)/\Delta u & \Delta u < 0 \\ \infty & \Delta u = 0 \end{cases}, \\ next\_t_v &= \begin{cases} (v_{\mathsf{min}} + (cj+1) \cdot v_{\mathsf{dim}} - v^*)/\Delta v & \Delta v > 0 \\ (v_{\mathsf{min}} + cj \cdot v_{\mathsf{dim}} - v^*)/\Delta v & \Delta v < 0 \\ \infty & \Delta v = 0 \end{cases}, \end{aligned} \tag{3.14}$$

where each is the next t value for an intersection with a u or v grid line respectively (Figure 3.21a). The four variables, $next\_t_u$, $next\_t_v$, ci, and cj, are all that is needed



(a) (b)

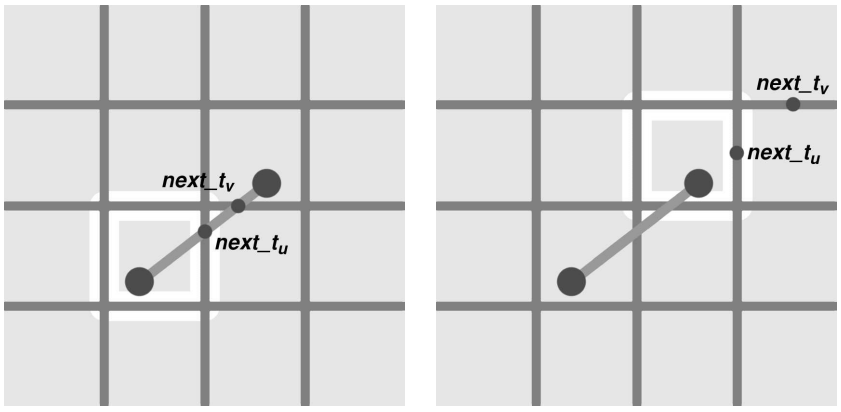
Figure 3.20. Time sampled movement of probe (a) and directed line segment in parametric space (b).

to walk the grid. The algorithm chooses the smaller of $next\_t_u$ and $next\_t_v$. If $next\_t_u$ is chosen then the walk steps into one of the adjacent column cells. Which cell to step into is determined by the sign of Δu. When positive ci is incremented by one and $next\_t_u$ is incremented by $u_{\mathsf{dim}}/\Delta u$. When negative both of the increments are negated. The same discussion holds when $next\_t_v$ is the smaller of the two $next\_t$ values.

The algorithm continues until the smaller of $next\_t_u$ and $next\_t_v$ is greater than one (Figure 3.21b). With both greater than one the next step would be beyond the end of the movement vector, therefore the algorithm halts. The cell in which the algorithm halts is the cell the end point is located within, satisfying the assumption for the next iteration when the current end point will become the next start point.

3.5.3 Trim Intersection

As stated above, the movement vector relates motion along the surface to motion in the parametric domain of the surface. Since surface boundaries are represented in parametric space using trimming loops, it is along the movement vector that intersections with the surface boundary must be detected. Further, since the trimming segments represent the common boundary of the current surface and



(a) (b)

Figure 3.21. The next cell is chosen using the smaller of $next\_t_u$ and $next\_t_v$(a). The algorithm halts when both values exceed one (b).

its adjacent surface, finding the first intersection along the movement vector (i.e., the one closest to the starting point) is all that is required. Any intersections past this initial one would represent an intersection with an edge that cannot be reached without traversing across invalid surface domain.

An efficient solution is to check only those segments lying within the cells the movement vector intersects. Further, these cells can be checked in the order the movement vector traverses through them. As the grid walking algorithm steps into a cell the movement vector is checked for intersections with all segments contained in that cell using an efficient parametric segment-segment intersection algorithm [3]. The intersection algorithm returns a value t when an intersection is found. If this t value is less than the $next\_t$ to be used in the grid walking algorithm, then the intersection is within the current cell. The smallest valid t found is the point of intersection. When no valid intersections are found the walk algorithm steps into the next cell and the algorithm repeats until either an intersection is found or the end point is reached.

3.5.4 Trim Release

There are two cases when it is necessary to determine if a tracked point should release from a trim intersection. First, when a trim intersection is detected, the transitioning algorithm must determine if the tracked point should release from the trim immediately to allow the trace to continue on an adjacent surface. Second, if the trace algorithm is tracing along a surface boundary (trim tracing) then the transitioning algorithm must determine when to release and onto which of the adjacent surfaces the transition should take place.

It is possible that a point on a trimming loop can be adjacent to multiple surfaces, such is the case at the corner of a cube where three surfaces intersect at a point. It should be noted that this can only occur at the endpoint of a trim segment and not interior to a trim segment. This comes straight from the definition of a trimming segment. A trimming edge is a collection of segments that represents a shared boundary of two surfaces. Only at the point where trimming edges connect

can multiple surfaces become adjacent. This connection point is the endpoint of two trimming segments from two neighboring edges.

If the tracked point is to release onto a surface it will do so onto only one of the adjacent surfaces. For this reason, the trim release algorithm checks each surface adjacent to the tracked point in turn. When the adjacent surface is found the parametric value for the tracked point on that surface is determined directly from the trimming segment information. Using this value, the surface is evaluated to permit DPT to be used on this new surface. While the Euclidean value of the tracked point will not change (actually the value may change slightly since the trimming information does not exactly express the intersection of the surfaces) the normal and tangent vectors may differ depending on the continuity of the surface boundary. Using this information and the probe location, DPT can determine a candidate location for the next tracked point. If this candidate point has parametric coordinates that are valid, then the trace is released from the trim. If the point does not release then the algorithm continues with the next adjacent surface. In the case that the tracked point does not release onto any of the adjacent surfaces, trim tracing is indicated.

3.5.5 Adjacency

In order to smoothly transition from one surface to another it is necessary to calculate an accurate transition point on the neighboring surface. Our system maintains an *edge adjacency table* (EAT) for each surface. A row in the table represents an edge and consists of four items: the index of the end of that edge, pointer to the adjacent surface, pointer to the adjacent trim loop, and the index of the start of the adjacent edge. These four items, along with the percent along the trim segment where the intersection occurred, is sufficient for determining the exact point on the adjacent surface.

As an example consider Figure 3.22. In this case assume that surface two is current and the intersection occurred on trimming loop one, edge three, and segment six. The EAT entry for edge three indicates that its last segment has an

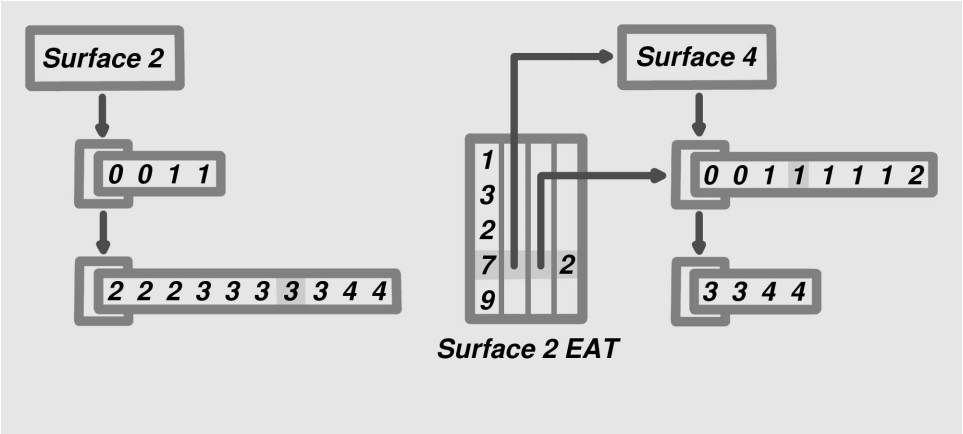


Figure 3.22. Edge adjacency table example illustrating that edge three of surface two is adjacent to edge one of surface four.

index of seven, surface four is adjacent, trimming loop zero contains the proper edge, and that the edge starts at segment index two. Since adjacent trimming edges run in opposite directions, we can determine that the intersected segment is at index $7-6+2$ within surface four's number zero trimming loop. The exact point on the new segment which corresponds to the intersection point is found at $1-p$ along the new segment, where p is the percent along the original segment that the intersection occurred. The one special case here is when $p = 0$. In this case the newly found segment index is incremented by one so that its intersection point will also reside at the start point (maintaining our $[0, 1)$ constraint).

3.5.6 Time Critical Operation

As discussed earlier, there is a budget that must be adhered to within the haptic process. Each cycle must complete in under 1/500sec in order to maintain our targeted $500Hz$ haptic rate. Given this budget, we can analyze each component of the direct haptic rendering algorithm and determine how much time it takes to complete. For example, when tracing a surface, DPT must perform a surface evaluation and check for trim intersections. If a trim is intersected then a number of surface evaluations are done in order to determine if the trace can release from the trim.

By collecting data on the amount of time for each operation, we can assign a maximum number of evaluations allowed. Our trace routine specifies a cap to be placed on the number of evaluations. If at any point the number of evaluation exceeds our max, the routine can abort out. This is not the usual case, and experimentally has not produced ill effects in the tracing experience. As machine performance improves, this cap can be raised.

Another factor in maintaining a budget is accuracy. A computer is only so precise when calculating trim intersections and surface evaluations. For this reason, we break the basic trace algorithm into two parts. If a trace intersects a trim we determine if it can release onto the neighboring surface. If it cannot release we stop for that cycle (i.e., we do not proceed with trim tracing until the next cycle). Continuing would allow the possibility of the trace sliding down the trim a short distance, releasing, then actually intersecting the same trim again. This should not occur, but it does since the trims are piecewise linear and therefore not an exact seam between surfaces. This one cycle delay has proven to inhibit this problem and has not been noticed by users.

3.5.7 Error Management

Since nearly all computations are either floating point or performed at very high rates, or both, it is important to maintain control over any errors in our results. In many areas this amounts to judicious use of epsilons. For example, when generating the grid to bound all of the trimming loops a small epsilon is applied so that the boundary trim segments fall within a cell instead of along its boundary. Nearly every stage of the haptic rendering process has some form of error management.

In the pre-processing stage we take a few steps to improve our results when dealing with trims. Foremost of these is reparameterizing the knot vectors. Both knot vectors are shifted so that they are centered about zero since that is where floating point numbers are most accurate. Furthermore, the knot vectors are scaled so that the node distance is no less than five times the control point distance. This relates the parametric movement more closely to the Euclidean. This is especially

important when a model has been scaled up in size. Recall that a scaled up model is actually handled by scaling down the probe movement. Therefore, a rather normal movement by the user could equate to a very small movement on the original model. This would in turn have produced an extremely small movement in parametric space. By altering the knot vectors, the model can be scaled upwards of 100 times its original size before movement vector lengths equate to those that would be present on the original model with a knot vector ranging from zero to one.

During tracing we take steps to not only reduce error, but also improve stability. Even if the user holds the probe perfectly still, the device may have small positional errors. This would end up being interpreted by the trace algorithm as user movement, producing slightly different contact points and restoring forces. Therefore, the trace algorithm accepts a noise threshold for each probe. If the reported movement is less than this distance, then no new point is calculated. In addition to the noise threshold, the users true movement is forced to exceed a small epsilon before a new trace point will be found. This cuts down on possible chatter due to the slight error in DPT itself. Similarly, if there has been a sufficient euclidean movement but DPT reports the next parametric point is within some small epsilon of the previous, no new point is calculated.

3.5.8 Transitioning Results

Trimmed NURBS models vary greatly in the number of surfaces and trim segments required to represent them. Table 3.4 lists a sampling of models against which we tested the system. The *Surfaces* column indicates the total number of surfaces for the model. *Segments* indicates the average number of trim segments per surface. Grid statistics are represented in the final four columns. The *Grid* column relays the average number of segments per grid overlay. The column labeled *Empty* gives a percentage for the number of cells in a surfaces grid that contain no trim segments. Empty cells translate into essentially zero work for the transitioning algorithm. *Max* gives the maximum number of segments in any one cell. This number represents the worst case for the transitioning algorithm for the

Table 3.4. Statistics on models used in system testing

Model	Surfaces	Segments	Grid	Empty	Max	Mean
Cube	6	4.00	16.00	25.00	2	1.33
Goblet	3	236.00	342.00	89.57	13	3.33
Teapot	7	370.29	547.71	89.47	29	3.32
Brake	28	168.14	332.57	77.64	6	2.08
Gear	198	168.26	260.57	87.11	13	2.84

given model. Finally, the *Mean* column shows the average number of segments in cells that actually contain segments. This number indicates the amount of work the transitioning algorithm can be expected to perform when near a surface boundary. Note that both the *Max* and the *Mean* columns contain very small numbers in comparison to the *Segments* column. These numbers indicate the drastic reduction in work the transitioning algorithm must perform when compared to an algorithm that would check every segment. Setting up the grid overlay for each model took under three seconds on average.

Our grid walking algorithm has constant time complexity in the number of steps. Table 3.5 shows that during tracing sessions with tens of thousands of cells being traversed the mean number was always very near one. The maximum number of cells traversed for each is small, limiting the worse case performance. The largest component of cost for transitioning, much as it is for tracing, is the surface evaluation. As shown in Table 3.5 the number of evaluations per walk also was very near one. More importantly, the maximum number was always recorded at five or less. This demonstrates predictable performance for the transitioning algorithm.

There are three other contributors to the computational cost for the transitioning algorithm: computing a trim intersection, determining the adjacent surface, and gathering surface normals for use in the boundary normal computation for trim tracing. Since our implementation of the grid overlay references directly into the trimming loops, the EAT can be used to determine the adjacent surface in

Table 3.5. Grid walking results

		Walks			Evals	
Model	Cells	Mean	Max	Total	Mean	Max
Cube	56623	1.00	2	58267	1.02	3
Goblet	55938	1.05	15	53137	1.00	3
Teapot	51151	1.04	13	57931	1.11	5
Brake	45918	1.03	4	49992	1.08	4
Gear	36215	1.04	19	41603	1.12	4

Number of cells walked, mean walked per trace call, and max cells walked in any one call. DPT update rate in Hz, mean and maximum penetration depth in mm, and mean and maximum surface evaluations.

constant time. The gathering of surface normals is also essentially free since the normal is a side effect of the surface evaluation. As each neighboring surface is evaluated to test for a release the resulting normal is collected and cached. The cost is therefore already included in the surface evaluation accounting. Finally, we have computed that we can check the movement vector against over 3000 segments at $500Hz$. In the worst case, our results showed a walk of 19 cells and a maximum of 29 segments in any cell. In this scenario our algorithm would need to check only 551 segments, illustrating it easily fits within our time constraints.

3.6 Extensions

Using the direct haptic rendering algorithm as a black box entity, several worthwhile extensions are possible. Three extensions have been implemented: model manipulation, dimensioned probes, and multiprobe contact. While these are but a few of the potential improvements and extensions, they demonstrate the power and flexibility of the algorithm.

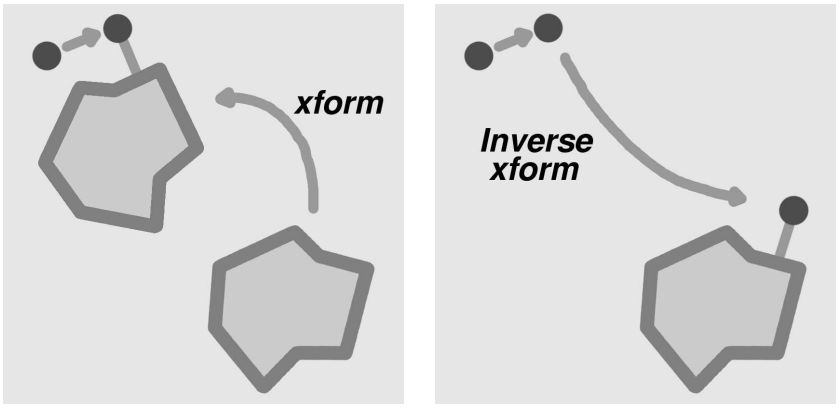
3.6.1 Models in Motion

Whether through manipulation, animation, or dynamic properties, mobile models are a fundamental property of virtual environments. The direct haptic rendering algorithm presented is designed for probe movement with static models, but can

be extended so that both the probe and the model can be moving. Basically, three approaches can solve this problem. First, the models can be physically transformed as they are moved. There are several drawbacks to this approach. Incremental object transformation has been shown to induce numerical errors over time. In addition, the transform calculations are all done when the processing power of the haptic processors is most needed – when the dynamics and tracing are both being calculated.

The second and third approaches involve storing a transformation matrix, *xform*, for each model. By using an xform, these methods avoid the problems found in incremental updates. A new absolute xform can be computed at each time step. The second approach is to use an xform to transform the current model and perform DPT on it as usual. This approach has a similar drawback as the first method in that the current model is transformed during tracing. In fact, transformations must be calculated even when the probe is not in contact with a model so that the tracking of near and active models can still be successful.

We have adopted a third approach (Figure 3.23) which involves using DPT on the original (nontransformed) model with a probe that has been transformed into model space [67].



(a) (b)

Figure 3.23. Model movement (a) transformed into probe movement (b) through the inverse model xform.

For each model being traced or tracked, the probe is transformed through the inverse of the model's xform. This process transforms the movement of the model into a component of the probe's movement. The resulting closest point and normal are then transformed from model space into world space.

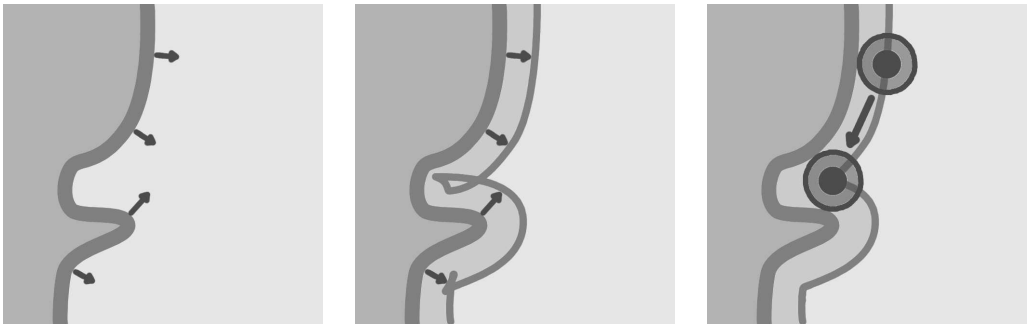
One obvious advantage to this approach is efficiency. If the combined number of models being traced and tracked is n then there will be $n \times 3$ vectors transformed (probe, closest point and normal). Like the second method, these transformations must be computed for each cycle through DPT. However, since $n \times 3$ is far less than the number of control points in all of the surfaces for the n models, this approach is more efficient than either of the other two approaches. This embedding of the tracing algorithm requires minimal overhead and does not affect the update rate of the haptic process.

Given this extension, we were able to perform several dynamic simulations. Among these are simple movement through force application, push buttons with detent, and a dynamic pendulum [67]. Further extension allowed kinematic chains where a collection of models connected by either spherical, universal, prismatic, or revolution joints. These assemblies are solved through an efficient "self-assembly" technique developed specifically for haptics and CAD [41].

3.6.2 Dimensioned Probe

One of the drawbacks to point probe methods is the dimensionless nature of the probe. If two models are placed directly adjacent to one another, a probe without finite size could still move between the two models without making contact. To eliminate this possibility we compute a model that is projected outward by the radius of the desired probe. A model of this type is often referred to as an offset model [22]. Our system uses the offset model in the haptic rendering process while using the original model for the visual display. Figure 3.24 illustrates the construction and use of an offset model.

The original model in Figure 3.24a is offset by the radius of the probe in the direction of the surface normal resulting in the model in Figure 3.24b. Isolated



(a) (b) (c)

Figure 3.24. Actual model (a). Initial offset model (b). Final offset model with possible trace positions (c).

regions are trimmed away, producing the offset model in Figure 3.24c. Contact with the surface of this offset model represents contact with the original model with a dimensioned probe (Figure 3.24c). Notice that any part of the offset model that is trimmed away represents a portion of the original model that could not be contacted with the dimensioned probe. Tracing with a point probe along an edge created by trimming away a region corresponds to tracing multiple contact points of the original model with a dimensioned probe (Figure 3.24c).

It is important to note that this process depends on trimming and adjacency information. Further, while this approach uses an "auxiliary" representation it is not a simplifying "intermediate" representation, since the offset model exactly represents the parts of the original model that can be contacted by the dimensioned probe. Producing the offset model adds significant preprocessing, but it does not affect performance of the tracing algorithm as long as the model geometry does not change during the trace.

3.6.3 Multiple Probe Rendering

Our current implementation of direct haptic rendering of trimmed models runs at over $1000Hz$. However, when using the Sarcos arm we notice no improvement when running at any rate over $500Hz$. By running at this lower rate, there is extra time within each cycle of the haptic process. Since the Sarcos device can reflect

forces to multiple end-effectors, we make two calls to the trace algorithm using the location of the finger and thumb as the probe locations.

Similarly, the CyberTouch glove provides six vibrotactile stimulators. Each of these can be tracked individually. Since this device is not haptic, it does not require the same high update rates. We have found rates as low as $100Hz$ to be very effective for this device.

To implement this feature, the trace state information for each probe (current surface point, tangents, and proximity) is stored as an attribute of each model. With this minimal data overhead and negligible impact on the tracing algorithms performance, multiple probe rendering is possible. This has been found to significantly add to the overall tracing experience. Further, the combination of multiple probes with movable models has been used to demonstrate object grasping and manipulating [34].

3.7 Summary

This chapter presented direct haptic rendering of trimmed NURBS models. We established two primary goals for the algorithm: efficient enough for inclusion within a haptic controller and accurate enough to faithfully reproduce the CAD design haptically. Each of the results sections above demonstrated success with a particular component of our complete algorithm. This section summarizes those results in order to show that both goals were met.

Section 3.3.8 showed that for all of our test models DPT performed at over $1000Hz$. This included both surface tracing and boundary tracing. In no small part, this speed derives from our cached alpha matrix surface evaluation technique. Since we want to prove that our algorithm fits our budget in the worst case, and not just on average, we add up the cost of tracing from one surface, across a trim boundary, and onto a second surface. In this case we essentially have two trace calls, one for each surface with the added cost of detecting the trim intersection and determining the neighboring surface.

The transitioning results presented in Section 3.5.8 showed that the cost of grid walking, segment intersection, and adjacency were negligible in comparison to the actual surface tracing. Therefore, since the tracing algorithm was over twice as fast as required, and the problem breaks down to tracing twice, we have met our efficiency goal. Further, the power of the machines on which we tested our systems has been surpassed more that two fold during the writing of this document.

The results given in section 3.3.8 illustrate the accuracy of DPT. Both the surface normal and the local closest point had low computational errors. Additionally, in simulation we found that the penetration depth computed produced an error to small to measure. The combination of these three factors illustrate the high degree of accuracy of DPT, and therefore the ability of the algorithm to faithfully reproduce the underlying CAD model.

CHAPTER 4

APE: TARGET APPLICATION

As a target application for Direct Haptic Rendering we present *APE*, the Active Prototyping Environment. The goal of *APE* is to allow the haptic rendering results presented in this document to be applied to real models from a real CAD system. The CAD system chosen is *Alpha_1*, a research system developed at the University of Utah. There are several subproblems to the design and development of this target application, including the user environment, managing models, managing multiple devices, user safety, distributing computation, and of course the actual CAD integration.

4.1 User Environment

Most, if not all, CAD systems provide at least one form of graphical user interface. Whether it be as simple as a wire-frame display or as complex as a 3D display that permits translucency and arbitrary cutting planes the goal is the same; help the user to better understand the model being designed. There is a fine line between providing the most information possible to the designer and complicating the interaction with information overload.

The virtual environment of *APE* is a stand-alone fish-tank display. There are three reason for this choice. First, the GUI of the design system is unchanged and therefore remains familiar to the designer. Second, *APE* need not be integrated tightly within any one design system. This allows *APE* to be used stand-alone or loosely integrated with any design system. Third, the *APE* system requires significant computational resources. The design chosen permits *APE* to run on a separate machine, most likely a more powerful multiprocessor system, than the design system.

4.1.1 Interface

Most haptic devices require some amount of user attachment. For instance, the Sarcos arm requires a user to be strapped into the arm and stand on a fixed platform. It is simply not convenient to use a keyboard or mouse at such times. For this reason mouse and keyboard use is kept to a minimum and most interaction has an alternate form that can be performed using the haptic device in conjunction with a 4 button device (Figure 4.1) we have built for the user to hold in his off hand. Table 4.1 shows the set of actions and bindings of *APE*.

The substitution of the button box for "center view," "default view," and "quit" are obvious. In the case of "translate" and "rotate" it is not. When using the mouse the two actions are decoupled. This is not the case when using the device substitute. When button 1 is pressed the user has effectively "grasped space." Subsequent movement of the device causes the entire view to translate and rotate about the probe.

Some interaction is, despite the device attachment, still more convenient from the keyboard. An example of this is loading a new model into the environment.

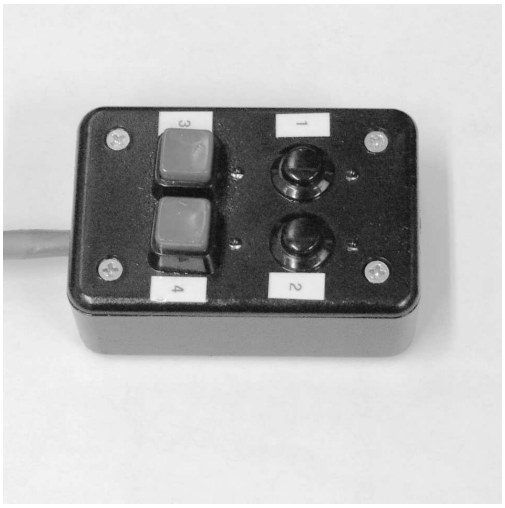


Figure 4.1. Four button device for off hand.

Table 4.1. User interface bindings and device substitutes

Action	Key/Mouse Binding	Device Substitute
Center View	C	Button 2
Default View	D	Button 3
Load Model	L	
New Session	N	
Quit	Q	Button 4
Rotate	Button2	Button 1
Scale	Button3	
Translate	Button1	Button 1

This currently can be accomplished in two ways: opening a file or receiving a model from a CAD package.

4.1.2 Graphical Viewer

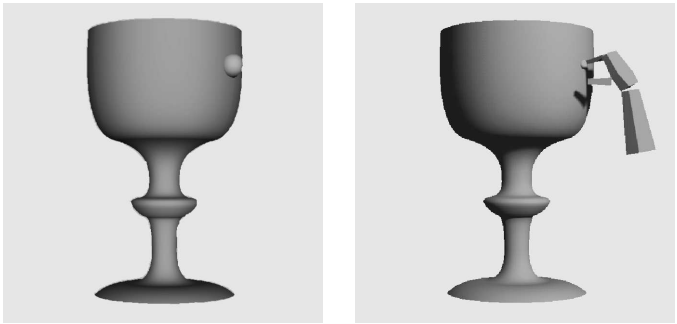
The graphical display for the environment uses OpenGL. This open standard provides for a wide variety of effects such as texture mapping, anti-aliasing and fog for depth cuing. More importantly, *APE* was designed and tested on SGI workstations that have highly optimized hardware implementations of OpenGL. Later porting to Linux and advanced graphics processors proved even further performance gains. *APE* also takes advantage of any options OpenGL provides that will further accelerate its display such as one sided polygons, backface culling, display lists, and materials properties for shading. All tested models were able to be displayed at rates well over the 20 frames per second required.

Several types of objects can be displayed within the virtual environment. Most important among these are the model being traced and the graphical representation of the haptic device being used. In all cases, object-oriented design was used such that an object has a method to draw itself. Each model is drawn using the OpenGL NURBS tessellator. However, since this is not the greatest tessellator the display system has been designed to allow alternate renderers to be used. The haptic devices are rendered in a fashion that illustrates the degrees of freedom inherent to

the device. For example, a single point device like the Phantom is represented by a single sphere while the Sarcos arm has a more complex representation (Figure 4.2).

There are three different *spaces* within which objects in *APE* can be displayed (Figure 4.3). The first is *view space* and is the default. In view space objects are transformed by the view matrix and then the perspective projection matrix. The second space is *object space* and in this space objects are transformed by the projection matrix alone. An example of an object that would be placed in this space is the graphical representation of the haptic device. The display of the device must remain outside of the view matrix so that movement of the physical device is mirrored within the graphical display. The third space is a special view called *sticker space* and is provided for placing graphics directly on the screen similar to old style pixel displays. For example, the *APE* logo is placed in the lower right hand corner of the window using this space. Another use for this space is to show context sensitive help relative to the probes location. Only 2D objects make sense in sticker space and therefore the projection matrix is simply orthogonal with a viewport set to the size of the window in pixels.

The quality of the haptic experience is greatly increased with the addition of a stereo display. By showing the model and the device in 3D stereo the haptic rendition fuses with the visual and the object becomes more real to the user. This



(a) (b)

Figure 4.2. Graphical representations of Phantom (a) and Sarcos arm (b).

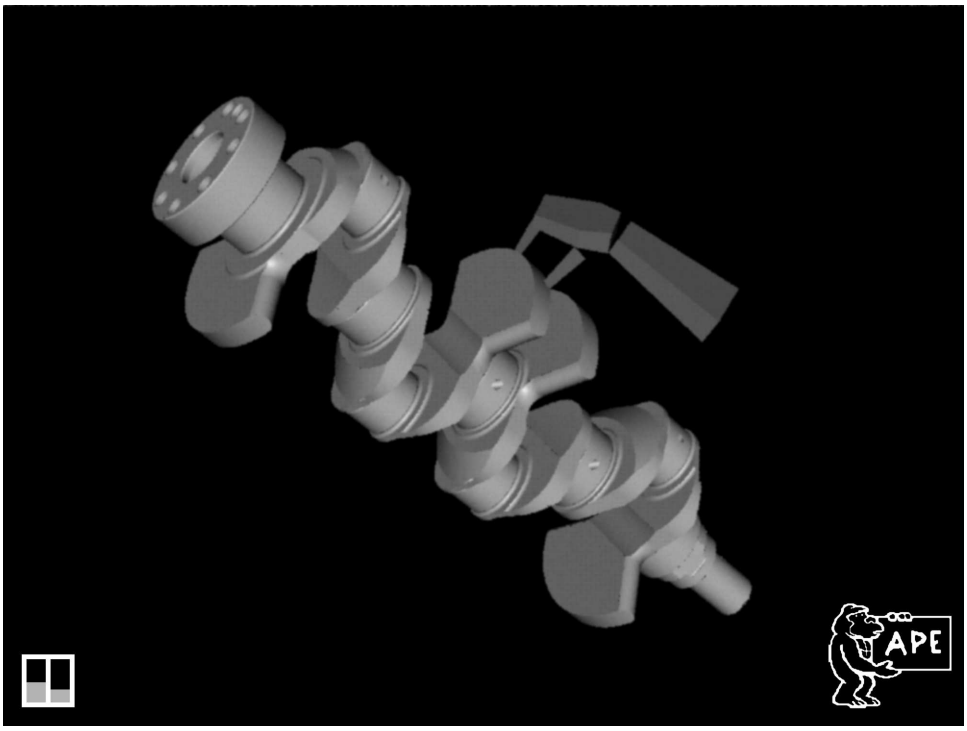


Figure 4.3. The graphical display space.

experience is heightened with the inclusion of head tracking. *APE* provides the ability to display the environment in mono or stereo with or without head tracking.

4.2 Model Manager

Central to the *APE* environment are the models under investigation. Several types of queries, from separate threads of computation, must be facilitated without delaying the processing of these time critical threads. In *APE* there is a central master repository of models within the user environment that is controlled by the *Model Manager*. Models can be loaded either through the user interface (Section 4.1.1) or through a connection with a CAD system (Section 4.4). It is the responsibility of this master model manager to distribute the models to the various device controllers, allow controlled access from different processing threads, and determine when a model is near a device.

4.2.1 Distributed Managers

The distributed design of *APE*, and the computational needs of a haptic system, often results in each device controller running on a separate machine than that of the user environment. This model of computation requires that the model cache of the master model manager be mirrored into a model manager residing within each device controller. Mirroring the models in this way permits the device ready access to the models without having to traverse a network connection.

It is important that these model caches be kept consistent or the visual display and the haptic display will not correspond. As a model is loaded into the environment a copy is sent to each device controller. The position and orientation of each model is determined by an xform from model space to world space. The master manager and all devices initialize this xform to the same value. In actuality, it is the xforms that must be synchronized at all times.

While the submanagers are very similar in design to the master manager, there are a few differences. Chief among these is the requirement that the manager hold a tracing *state* for each model. This state indicates if the model is near, active, or current to any probe of the device. When active or current, the state will hold all geometric data necessary for direct haptic rendering. This data is used by the trace routine during each cycle of the haptic controller, allowing the algorithm to be implemented without knowledge of the type of device or significant local data caching.

Another important item stored by the submanagers is the inverse view matrix for the graphical viewer. As discussed in Section 3.6.1 this matrix is required to map the probe movement into model space, thus saving the computational cost of transforming the models. For efficiency, the view matrix is also stored as it is required to transform the resulting contact point and normal from model space into world space.

4.2.2 Model Locker

Each model manager controls access to its model cache through the use of a model *locker*. Basically, models are stored within a locker with each model being accessed through its own *key*. When a process requires the use of a model it requests a key for either read or read/write access. This key is unique per check-out, ensuring a model cannot continue to be accessed after it has been checked-in.

Writing to a model requires exclusive access whereas multiple readers can share the model at once. To prevent starvation of processes requiring write access, readers are only allowed access to a model if there are no waiting writers. Therefore, when a key is checked in waiting writers are awakened first, but only if there are no more readers holding a key. If there are no waiting writers then all waiting readers are awakened. While this approach could potentially result in processes requiring read access to starve, in practice this is not a problem. The locker provides an iterator that can be used by the read process to checkout the "next available model," which is most often what is desired.

Programmatically this control is granted through the use of a single data *mutex* and two *conditionals*, one for the readers and one for the writers. The mutex is used to control access to the lockers key control data. The conditionals allow processes that are willing to be blocked until the model is available to wait without occupying the data mutex. Essentially, the conditional is a waiting area for processes. All readers share a conditional, regardless of what model they are waiting for. The same is true of writers. This separation of wait areas allows us to signal only the appropriate waiting pool when a model is checked-in, if any should be awakened at all.

In addition to the data mutex, the locker contains a mutex for complete locker control. This mutex is used when an operation will change the contents of the locker, for example when a model is loaded or removed. Locking the entire locker is equivalent to locking all members for writing with the difference being that a single key can then be used to access all models. Unlocking is made more efficient in this

case as a single broadcast to wake-up the waiting readers and writers can be made once all models have been checked-in.

4.2.3 Tracking Deamons

Each model manager spawns a tracking deamon. Since all models must be constantly checked for their near proximity status, the master model manager deamon is used for this task. This proximity thread loops over each probe for each device and checks its proximity to all models. Determining if a model is near enough to be of interest to a device does not require an exact closest point be found. For this reason we provide a cutoff distance to the LUB-tree search equal to our near distance. This allows the search for the global closest point to abort if either the distance from the probe to the models lower bound is greater than the cutoff or the upper bound is less than the cutoff. Only when the search aborts from the upper bound being less than our near distance do we deem the model near and send a message to the device. Similarly, if the model is not near, and previously was, then a message would be sent to the device as well.

Once a device is signaled that a model is near, its submanager deamon will begin tracking a global closest point. Unlike in the proximity thread, this tracking requires that the LUB-tree search actually converges to a solution. The distance from the probe to the tracked point is used to determine if the model should be made active. Once active, the tracked point is used, in conjunction with a local closest point, to determine a potential contact point. When a model is deemed current, therefore being traced and requiring only a local closest point, the deamon will skip the model for the probe in contact.

The master manager's deamon works with all models in the scene, but doesn't require exact results. The submanager's deamon does require exact results, but it works on only a subset of the scenes models. In practice this division of labor has been found to be very effective in distributing the computation costs as well as producing better tracking results for direct haptic rendering.

4.3 Device Manager

While the most common usage of *APE* is with a single device, perhaps with multiple end-effectors, *APE* can also handle multiple devices through its *Device Manager*. At startup the *APE* system is given a config file (Appendix E) which indicates the devices for the session along with any device specific information (e.g., the machine on which the controller should run, the network ports to use, etc.). The device manager reads the config file and creates each device passing on the rest of the information in the config file to the devices constructor. This allows the device to parse the data and complete its initialization. This same config file is later read by the device controller program when it is started to ensure that both processes are initialized identically.

Each device has specific requirements and as such they contain the necessary code to communicate with their controller. That said, all communications from the user environment to the devices are funneled through the device manager. This common interface makes possible communication to individual devices, broadcasting to the whole collection, and broadcasting to all but a given device. This final feature is helpful when one device is controlling the environment and the other devices need to be placed into a different safety level.

Another responsibility of the device manager is arbitrating device control over the view and models. If multiple devices request to manipulate the view or move a model, the device manager will chose a winner and then notify the losers to cease. This approach allows all devices immediate feedback and access without first having to request permission from the device manager. If multiple devices begin to modify a models position, one will win and have continuous real-time haptic feedback, while the others would simple be placed into a safety level that would prevent further manipulation.

4.3.1 Devices

APE provides an abstract device class with which the rest of the system can interface. Derived from this abstract class are type specific classes for each type of

device. These subclasses communicate with their particular controller in whatever fashion is optimal for that device while still providing a single interface to the rest of the system. Due to this design choice *APE* supports three different haptic devices that communicate with type specific controllers.

Communicating with the controller is the singular responsibility of the device deamon. Spawned at the time of creation by the device manager, the deamon continuously reads device configuration packets. The polling technique is specific to the device and may involve querying a port or communicating across a network. Regardless, the results are cached using a scheme called *n+2 buffering* [50], where n is the number of readers. This scheme is guaranteed to always provide the most recent data to the readers and never block readers or writers.

In *APE* there are two readers: the UI and the master model managers proxy thread. As described in Section 4.2.3, the proximity thread requires only the location of the probe for its distance tracking. Conversely, the UI needs the full device configuration in order to display each device within the graphical viewer. For each cycle of the visual display each device will redraw itself using its configuration data. This method ensures that the visual display is in sync with that of the haptic.

4.3.2 Safety Levels

There are three different safety levels that a device can be placed into during a haptic session. The first, level 0, is deemed *safe* which means forces can be applied when the device is in contact with a model. The remaining modes indicate the possibility of an unsafe condition and result in no forces being applied to the user.

Level 1 is indicated when a transient event has occurred. The device should not produce forces until the probe enters a confirmed safe area in the environment. Each probe has its own safety level so each can move into a safe area individually. A safe area is defined as a location close enough to the model to accurately determine that the probe is above, and not within, the model.

Finally, level 2 signals an ongoing event within the *APE* viewer that is outside the control of a device. The device must not generate forces until the event is

completed. An example is active view manipulation. When a device is modifying the view it does not want to have objects that are moving about the view cause impact, and force generation, with the device. For this reason all devices are enter level 2 when the view is modified. Another example is model manipulation. While the device doing the manipulation certainly does want forces applied, the movement may be unexpected to other devices within the environment and therefore they are placed into level 2. Unlike level 1, the device cannot remove itself from this level. Instead, it must rely on the environment to signal it to drop back to level 1 from where it can follow the level 1 protocols to return to safety.

These three levels ensure that all force generation is under the control of the user (assuming no mechanism animation). This eliminates jolting forces without having to wait for control to be granted by the environment.

4.3.3 View Manipulation

Users can manipulate the view via the mouse or through any device. A device initiates manipulation of the view by sending a button status (Section 4.1.1) from the controller to the device deamon. Manipulation itself will take the form of the user "grabbing" the view with the device with the objects in the view then being moved about the device as if attached through an either. The device manager will detect the request and determine if the request will be granted. The only cause for refusal would be another device already manipulating the view.

When the request is granted the device manager will send a safety level 2 message to all devices, including the controlling device. This will prevent any wildly moving models from colliding with a probe and therefore exerting undue force upon the user. The position and orientation of the device and the current view matrix are recorded as a baseline for computing the new view matrix.

Each subsequent call to update the view will result in a view matrix computed relative to the original view and device orientation. This prevents errors from creeping in through matrix operations. The matrix operation to compute the new view matrix is defined as:

$$\hat{V} = [\hat{T}\hat{X}X^{-1}\hat{T}^{-1}][T^{-1}\hat{T}]V, \tag{4.1}$$

where $\hat{V}$ is the new view matrix, V is the view matrix when the view was grabbed, $\hat{T}$ is the current position as a translation matrix, T is the position when grabbed, $\hat{X}$ is the current orientation, and X is orientation when grabbed.

Only when the device releases control over the view matrix does the device manager update the devices. First, the updated view matrix, its inverse, and scale component are sent to the controllers. These values need to be present before the device can be placed into safety level 1. The devices can then resume haptic interaction once they find a safe configuration.

4.3.4 Moving Models

The device managers function during model manipulation is similar to that of view manipulation. Primarily it is to arbitrate control over the manipulation. At current, when any device requests to manipulate a model the other devices are placed into safety level two. However, unlike view manipulation, the active device is left to continue receiving force feedback.

When a device moves a model, whether it be through direct force or from an assembly simulation, that models new xform is sent to the device deamon. The device manager recognizes the change and makes note of the devices request. The device does not wait for the managers approval, instead it begins the simulation with the assumption it will be approved. If multiple requests are received the manager will decide who should be granted control. Those devices that do not receive permission will be placed into safety level 2 and consequently their simulation will cease.

The device that is granted control will continue with its simulation. The models xform will be streamed from the controller to the device deamon where it will be read by the graphical viewer. The result is the visualization of the model will match that of the model being haptically rendered.

Finally, when the device has completed its simulation the models final xform will be transmitted to the device deamon. The device manager receives this event

and turns the xform around to all other devices. This returns all devices to a state of synchronization to the state in the model manager. Once this has completed, a safety level 1 message is sent to all of other devices so that they can resume their haptic interaction.

4.4 CAD System Integration

One of the primary goals in the design of *APE* was to integrate it within a CAD system, but do so loosely to allow maximum freedom in balancing computational resources. Further, I decided to not support specific application formats within *APE* but instead provide an exchange format that other applications could export, or to which other formates could be converted. This format is called Utah NURBS (UTN) and is described in Appendix D.

The *APE* system can load UTN files directly via a file dialog. To do so a user hits the "L" hotkey and then browses for their UTN file. Alternatively, the model can be sent from *Alpha_1*. Within *Alpha_1* a user can work on a design using all of the commands afforded to them by the design environment. At any point they can send their model into the *APE* environment through the `apeModelMsg` command.

This command will verify the model is correct and compatible with *APE*. For example, it checks that all surfaces have at least one trimming loop and that all edges have an adjacency defined. Once the model passes these checks, it is dumped to disk onto a server accessible by the running *APE* process. Upon successful completion of the models export, the command will write a *message file* onto the tmp drive of the machine running *APE*. This file signals *APE* of a waiting model and also includes where the file is located. Once a second the *Message Manager* in *APE* searches for this file. When found the model is immediately loaded by the Model Manager. Both the message file and the model are then cleaned from the disk as they were intended to be transient data.

The UTN converter allows any shell (with the exception of those that don't have the full adjacencies defined) to be used in the *APE* system. This makes it an integral part of the design environment, even though it is not part of the same code

base. All translation between formats is hidden allowing the design environment to run on a different machine than the *APE* system, further increasing performance of both applications.

4.5 Distributed Computation

In its most complete configuration, *APE* is a multiapplication, multiplatform, multiprocessor, and multitreaded system with multiple network channels and ttys open for communicating between the main application and the devices (Figure 4.4). Minimally, in its simplest configuration, *APE* is multithreaded with local tty device communication. In all cases careful consideration was given to try and balance the needs of the various components against the available system resources.

Consider for example the system configuration given in Figure 4.5. In this configuration a single Phantom device is being used in conjunction with the *Alpha_1* modeling system. *Alpha_1* is run on a separate machine from the *APE* user interface and in fact could actually be executing under a different operating system. The only requirement is that it have write access to the machine on which *APE* is executing. This allows *Alpha_1* to send the model messages to *APE* loosely binding the two systems.

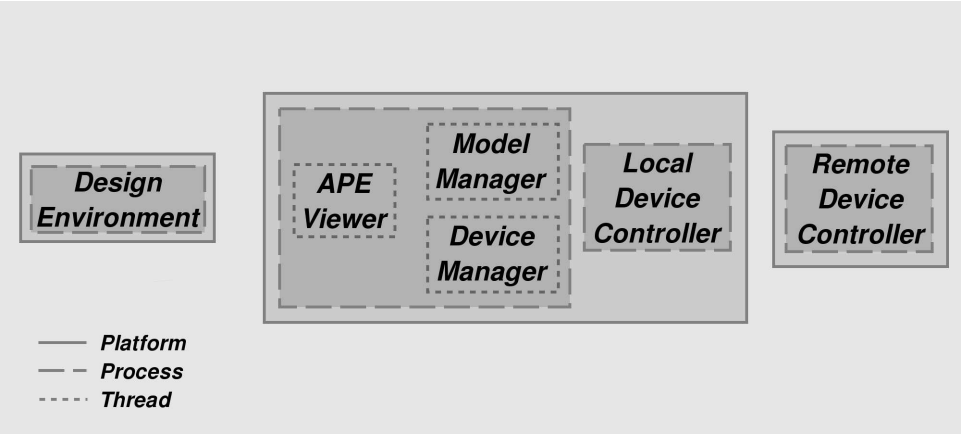


Figure 4.4. *APE* is multiplatform, multiprocessor, and multithread capable.

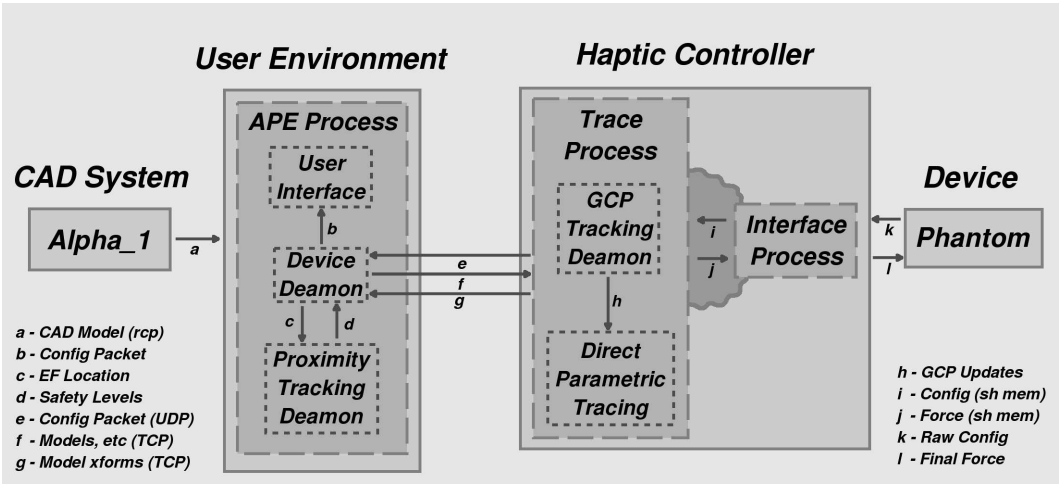


Figure 4.5. System configuration for Phantom device.

The Phantom requires real-time access to its controlling computer to maintain device stability. As such, *APE* runs the device controller on a separate machine than the user environment (note that this is not a requirement but does produce better results). The device controller for the Phantom is broken into two processes that communicate via shared memory. One process actually interfaces with the device and must maintain $500Hz$. In practice, the Phantom consistently will maintain near $1000Hz$. This interface process queries the device for its probe location and writes it into shared memory using triple-buffering. The second process contains the model manager and the tracking thread. It is responsible for taking the probe location and performing DPT. If the probe is in contact then the appropriate force vector is computed and placed into shared memory, again using triple-buffering. The result is a completely decoupled computation model. The trace process can, if necessary, run slower than the interface process without causing missed cycles in the controller. It can also, if possible, run faster than the interface and produce better results in its tracking deamon.

Within *APE* the Phantom device communicates with the controller via multiple network channels. Acting as a server, the Phantom device controller continuously gathers, packages, and sends device configuration data across a UDP connection.

The device deamon, using the *n+2 buffering* algorithm, reads and caches this data, making it immediately available to the system readers.

The configuration packet for the Phantom contains the position and contact information for the end-effector; button status and finally, if a model is being moved, the contacted models id, position and orientation. All told this amounts to 33 floating point numbers. As a comparison, the Sarcos Dextrous Arm configuration packet contains 60 floating point numbers including the elbow, wrist, finger and thumb joint locations and orientations. The end-effector positions are read by the master model manager for use in its proximity detection thread. The user interface uses all remaining data to graphically display the device.

Two other TCP network channels connect *APE* to the device controller. The device uses one channel for sending all messages, such as models, safety levels, near proximity notification, xforms, and view matrix changes, to the controller. The second is used by the controller to send the xform changes of models being moved through assembly manipulation but that are not in direct contact with the probe. By using the UDP packet for the contacted model the display of this model remains completely in sync with the haptic display, the remaining models can better handle any lag that might result from having to use the slower TCP protocol.

This simple single device setup amounts to a rather distributed computation model. There are two applications: CAD system and *APE*. Three machines are used to logically distribute core components: CAD system, *APE* user environment, and the haptic controller for the Phantom. Both the *APE* user environment and the device controller are multithreaded. The user environment has three threads: GUI main loop, master model manager proximity deamon, and the device configuration deamon. Finally, the device controller is both multiprocess and multithreaded. There are two processes: the tracing process and the arm interface process. The tracing process has a thread for the trace loop and one for its model manager deamon. In total, this particular *APE* setup uses three machines, two applications, four processes, and seven threads of computation.

As a result of this distribution model, all core components can maximize their computational potential. In practice we have found that the display frame rate is not affected by device connections. The haptic display is not affected by either the user environment or the design system. Given enough processors, the system scales naturally because of it multithreaded design. All tested configurations ran smoothly with no one component dominating the computational resources or otherwise delaying the processing of any other component.

CHAPTER 5

CONCLUSIONS

We have presented a powerful algorithm that supports the direct haptic rendering of models constructed from trimmed NURBS. Direct Parametric Tracing computes a local closest point that shadows the movement of a haptic probe a rates suitable for inclusion in a haptic controller. Haptic rendering is improved because the DPT method supports exact computation of surface normals as well as higher order continuity of surface representation. The parametric surfaces being rendered have compact representations, allowing for haptic rendering of complex models within the same memory as would be used by an approach utilizing a secondary representation. In addition, some of the complications of using an intermediate representation, such as force discontinuity artifacts, do not appear in direct haptic rendering.

A grid overlay and efficient grid walking algorithm allow our approach to be model based with only a single point being tracked per model. This model-based grid approach permits highly trimmed models to be included without loss of efficiency while also permitting more complex scenes to be haptically rendered. Additionally, we have demonstrated the ability of the haptic algorithm to be extended in order to permit model manipulation, tracing by a dimensioned probe, and multiprobe contact.

As a test-bed system, we developed an Active Prototyping Environment. *APE* integrates our haptic algorithm with a fish-tank virtual environment and loosely couples it to a CAD system. The result is a complete system that supports multiple haptic, and nonhaptic, devices. The distributed system design allows high update rates on both sides of the system, resulting in an interactive visual

display coupled with an accurate haptic rendition produced from the actual model. This combination greatly improves the amount of information a designer can gather about a design while he is immersed within the design environment.

APPENDIX A

NURBS CURVES AND SURFACES

A B-spline curve, $\gamma(u)$, of order k is determined by a set of points, $\mathbf{P} = \{P_i\}_{i=0}^{n}$, its knot vector, $\mathbf{u} = \{u_i\}_{i=0}^{k+n}$, and its basis functions, $\mathbf{B} = \{B_{i,k}\}_{i=0}^{n}$. The definition of the curve is given by,

$$\gamma(u) = \sum_{i=0}^{n} P_i B_{i,k}(u). \tag{A.1}$$

The basis functions have a nice recursive form and are a generalization of the Bernstein/Bezier blending functions. The definition for the basis functions is given by,

$$B_{i,1}(u) = \begin{cases} 1, & u_i \leq u < u_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{A.2}$$

and for $k > 1$,

$$B_{i,k}(u) = \begin{cases} \frac{u-u_i}{u_{i+k-1}-u_i} B_{i,k-1}(u) \ + \frac{u_{i+k}-u}{u_{i+k}-u_{i+1}} B_{i+1,k-1}(u), & u_i < u_{i+k} \\ 0, & \text{otherwise} \end{cases} \tag{A.3}$$

where each successive basis function is a convex combination of two basis functions of a lower order. The $B_{i,1}$ define piecewise constant polynomials, $B_{i,2}$ piecewise linear, $B_{i,3}$ piecewise quadratic and so forth. This definition ensures a C^{k-2} curve when the $B_{i,k}$ basis functions are used. Also, each piecewise segment of the curve is within the convex hull of the control points defining it.

A powerful extension to standard B-Spline curve is the Non-Uniform Rational B-Spline (NURBS) curve. As the name indicates, NURBS provide the ability to exactly represent rational curves like circles. The definition of a NURBS curve is given by,

$$\gamma(u) = \frac{\sum_{i=0}^{n} P_i w_i B_{i,k}(u)}{\sum_{i=0}^{n} w_i B_{i,k}(u)}, \tag{A.4}$$

where $\mathbf{w} = \{w_i\}_{i=0}^{n}$ are the rational weights for each control point in $\mathbf{P}$. A NURBS curve has all of the same properties that a standard B-Spline exhibits. Indeed,

if the weights are all defined to be equal to one then Equation A.4 reduces to Equation A.1. Conversely, all B-Spline curves can be written as NURBS curves by assigning all of the weights the value of one.

There is more than one way to evaluate a NURBS curve. The weighted combination of control points can be computed through evaluation of the basis functions as in Equation A.1, or curve refinement may be used [10, 52]. Inserting $k-1$ knots into $\mathbf{u}$ with the value u^* will create a new representation of the NURBS curve with a control point $\hat{P}_{i^*}$ that is the value of $\gamma(u^*)$. This point is called an evaluation point and results because only one basis function, $\hat{B}_{i^*,k}$, is nonzero at u^*. The value of i^* is one less than the index of the first knot of value u^* in the new knot vector. The curve refinement method is computationally efficient when only the necessary values are computed.

The tensor product NURBS surface has a similar definition. The surface $S(u,v)$ with the collection $\mathbf{P} = \{P_{i,j}\}$ and $\mathbf{w} = \{w_{i,j}\}$ as it's control mesh is defined as

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} P_{i,j} w_{i,j} B_{j,k_v}(v) N_{i,k_u}(u)}{\sum_{i=0}^{m}\sum_{j=0}^{n} w_{i,j} B_{j,k_v}(v) N_{i,k_u}(u)}, \tag{A.5}$$

where k_u is the order, $\mathbf{u} = \{u_i\}_{i=0}^{k_u+m}$ is the knot vector, and $\mathbf{N} = \{N_{i,k_u}\}_{i=0}^{m}$ are the basis functions for the rows of the control mesh. Similarly, k_v is the order, $\mathbf{v} = \{v_j\}_{j=0}^{k_v+n}$ is the knot vector, and $\mathbf{B} = \{B_{j,k_v}\}_{j=0}^{n}$ are the basis functions for the columns of the control mesh.

A NURBS surface, with its compact parametric representation, efficient exact evaluation techniques, convex hull property, and higher order continuity, is a powerful tool for design and has made it the de facto standard in the design community.

APPENDIX B

NURBS CURVE VELOCITY AT EVALUATION POINT

In this appendix a compact, computationally efficient equation for the velocity of a curve at an evaluation point is derived. For clarity, and to allow the equations to fit within the bounds of the page, the derivation will be done in parts. Recall the definition of a NURBS curve,

$$\gamma(u) = \frac{\sum_{i=0}^{n} P_i w_i B_{i,k}(u)}{\sum_{i=0}^{n} w_i B_{i,k}(u)} = \frac{N(u)}{D(u)}, \tag{B.1}$$

where N and D are functions representing the numerator and denominator respectively.

The velocity of a NURBS curve (when it exists) is given by the quotient rule as,

$$\gamma'(u) = \frac{N'(u)D(u) - N(u)D'(u)}{D(u)^2} \tag{B.2}$$

where

$$N'(u) = \sum_{i=0}^{n} P_i w_i B'_{i,k}(u), \tag{B.3}$$

$$D'(u) = \sum_{i=0}^{n} w_i B'_{i,k}(u), \tag{B.4}$$

and by definition,

$$B'_{i,k}(u) = (k-1)\left[\frac{B_{i,k-1}(u)}{u_{i+k-1} - u_i} - \frac{B_{i+1,k-1}(u)}{u_{i+k} - u_{i+1}}\right]. \tag{B.5}$$

We now continue with Equation B.3 noting that Equation B.4 will follow a similar path. Expanding Equation B.3 with the definition of $B'_{i,k}(u)$ yields,

$$N'(u) = (k-1)\sum_{i=0}^{n} P_i w_i \left[\frac{B_{i,k-1}(u)}{u_{i+k-1} - u_i} - \frac{B_{i+1,k-1}(u)}{u_{i+k} - u_{i+1}}\right], \tag{B.6}$$

$$= (k-1)\left[\sum_{i=1}^{n}\frac{P_i w_i B_{i,k-1}(u)}{u_{i+k-1}-u_i} - \sum_{j=0}^{n-1}\frac{P_j w_j B_{j+1,k-1}(u)}{u_{j+k}-u_{j+1}}\right]. \qquad \text{(B.7)}$$

Notice that after distributing the summation the bounds of the two summations have changed. This is because the basis function $B_{i,k-1}(u)$ evaluates to zero at $i = 0$ and basis function $B_{j+1,k-1}(u)$ evaluates to zero at $j = n$. In both cases this is true since the basis functions exist outside the domain of the original curve. The two summations bounds can be made to once again agree by variable substitution. Replacing j in the second summation with $i - 1$ the equation becomes,

$$N'(u) = (k-1)\left[\sum_{i=1}^{n}\frac{P_i w_i B_{i,k-1}(u)}{u_{i+k-1}-u_i} - \sum_{i=1}^{n}\frac{P_{i-1} w_{i-1} B_{i,k-1}(u)}{u_{i+k-1}-u_i}\right], \qquad \text{(B.8)}$$

and then combining the two summations we get, for both N' and D',

$$\begin{aligned} N'(u) &= (k-1)\sum_{i=1}^{n}\frac{(P_i w_i - P_{i-1} w_{i-1})}{u_{i+k-1}-u_i}B_{i,k-1}(u), \\ D'(u) &= (k-1)\sum_{i=1}^{n}\frac{(w_i - w_{i-1})}{u_{i+k-1}-u_i}B_{i,k-1}(u). \end{aligned}$$

Returning to Equation B.2, we can now construct the two products of the numerator,

$$\begin{aligned} N'(u)D(u) &= (k-1)\sum_{i=1}^{n}\frac{P_i w_i - P_{i-1} w_{i-1}}{u_{i+k-1}-u_i}B_{i,k-1}(u)\sum_{j=0}^{n} w_j B_{j,k}(u), \\ &= (k-1)\sum_{i=1}^{n}\sum_{j=0}^{n}\frac{P_i w_i w_j - P_{i-1} w_{i-1} w_j}{u_{i+k-1}-u_i}B_{j,k}(u)B_{i,k-1}(u), \\ N(u)D'(u) &= (k-1)\sum_{j=0}^{n} P_j w_j B_{j,k}(u)\sum_{i=1}^{n}\frac{w_i - w_{i-1}}{u_{i+k-1}-u_i}B_{i,k-1}(u), \\ &= (k-1)\sum_{i=1}^{n}\sum_{j=0}^{n}\frac{P_j w_i w_j - P_j w_{i-1} w_j}{u_{i+k-1}-u_i}B_{j,k}(u)B_{i,k-1}(u), \end{aligned}$$

and finally write the final general form of the velocity curve,

$$\gamma'(u) = (k-1)\frac{\sum_{i=1}^{n}\sum_{j=0}^{n}\frac{w_i w_j (P_i - P_j) + w_j w_{i-1}(P_j - P_{i-1})}{u_{i+k-1}-u_i}B_{j,k}(u)B_{i,k-1}(u)}{\left(\sum_{j=0}^{n} w_j B_{j,k}(u)\right)^2}. \qquad \text{(B.9)}$$

Consider now that $\gamma(u)$ is the result of previous refinement whereby $k-1$ knots of value u^* have been introduced at location $i^* + 1$ within the curves knot vector.

Then P_{i^*} is an evaluation point. In Equation B.9, only basis functions $B_{i^*,k}(u^*)$ and $B_{i^*+1,k-1}(u^*)$ exist at u^* and have a value of one. Noting this we reduce the equation and get,

$$\gamma'(u^*) = (k-1)\frac{w_{i^*+1}w_{i^*}(P_{i^*+1} - P_{i^*}) - w_{i^*}w_{i^*}(P_{i^*} - P_{i^*})}{(w_{i^*}w_{i^*})(u_{i^*+k} - u_{i^*+1})}, \tag{B.10}$$

$$= \frac{(k-1)}{u_{i^*+k} - u_{i^*+1}}\frac{w_{i^*+1}}{w_{i^*}}(P_{i^*+1} - P_{i^*}), \tag{B.11}$$

for the velocity of the curve at time u^*. An important point to notice in Equation B.11 is that the velocity at time u^* can be determined by two control points and a scaling factor. This relates the Euclidean tangent direction to the parametric speed.

Therefore, refining the original curve with $k-1$ knots of value u^* yields both the evaluation point, P_{i^*}, and the tangent vector, $\gamma'(u^*)$.

APPENDIX C

MINIMAL BOUNDING CONE AXIS PROOF

This appendix gives a proof based on one originally by X. Gu and modified by S. Gortler [19]. The version shown below has been further modified for readability, consistency with this documents notation, and completeness. We set the stage with lemmas that aid in the development of the final proof.

Lemma 1 *Given finite point set* $\mathbf{N} = \{N_i\}$*, let HULL*($\mathbf{N}$)*=boundary of the convex hull of* $\mathbf{N}$*. If point O is not in HULL(N), then there is a* single *closest point P on HULL*($\mathbf{N}$) *to point O.*

Lemma 2 *Given finite point set* $\mathbf{N} = \{N_i\}$*, any plane that intersects HULL*($\mathbf{N}$) *and is not coincident with any face of HULL*($\mathbf{N}$) *will divide HULL*($\mathbf{N}$) *such that at least one* N_i *is on each side of the plane.*

Lemma 1 is true since all convex hulls are convex and compact by definition. Lemma 2 derives from the construction of a convex hull from planes defining half spaces and basically states that any plane that intersects the interior of a convex hull will "clip away" at least one of the defining vertices of the convex hull.

Theorem 1 *Given finite point set* $\mathbf{N} = \{N_i\}$ *lying within a single hemisphere of the unit sphere, then if P is the closest point from HULL*($\mathbf{N}$) *to the sphere origin O, then P is the center of C, the smallest circle on the sphere bounding the points.*

Proof:

I.

The plane orthogonal to $\vec{OP}$, through P, cuts the sphere in a circle, C, and P is the center of that circle.

II.

Suppose there is another circle C^*, centered at P^*, that bounds the point set more tightly than C. The plane of C^* must separate P from O. If this were not the case then P and O would lie on the same side of the C^* plane, resulting in the C^* plane intersecting HULL($\mathbf{N}$). This would imply that C^* would not bound all of N_i (Lemma 2). Let the intersection of $\vec{OP}$ with the plane of C^* be the point D.

III.

$\|\vec{OP}\| > \|\vec{OD}\|$ since the plane of C^* intersected $\vec{PO}$ at D.

$\|\vec{OD}\| > \|\vec{OP^*}\|$ since P^* is the closest point on the plane of C^* to O.

$\|\vec{OP}\| > \|\vec{OP^*}\|$ by transitivity.

Therefore, The radius of C^* must be greater than the radius of C, so C is the minimum radius bounding circle of N_i.

Since the theorem is proven, we can now state the relevant Lemma needed for the computation of the boundary normal in Section 3.3.5.

Lemma 3 *Given point set* $\mathbf{N} = \{N_i\}$*, lying within a single hemisphere of the unit sphere, and circle* C *with center* P*, the smallest circle on the sphere bounding the points, then the tightest fitting bounding cone for the point set has* C *as a cross section and* $N_A = P - O$ *as its axis (Figure C.1).*

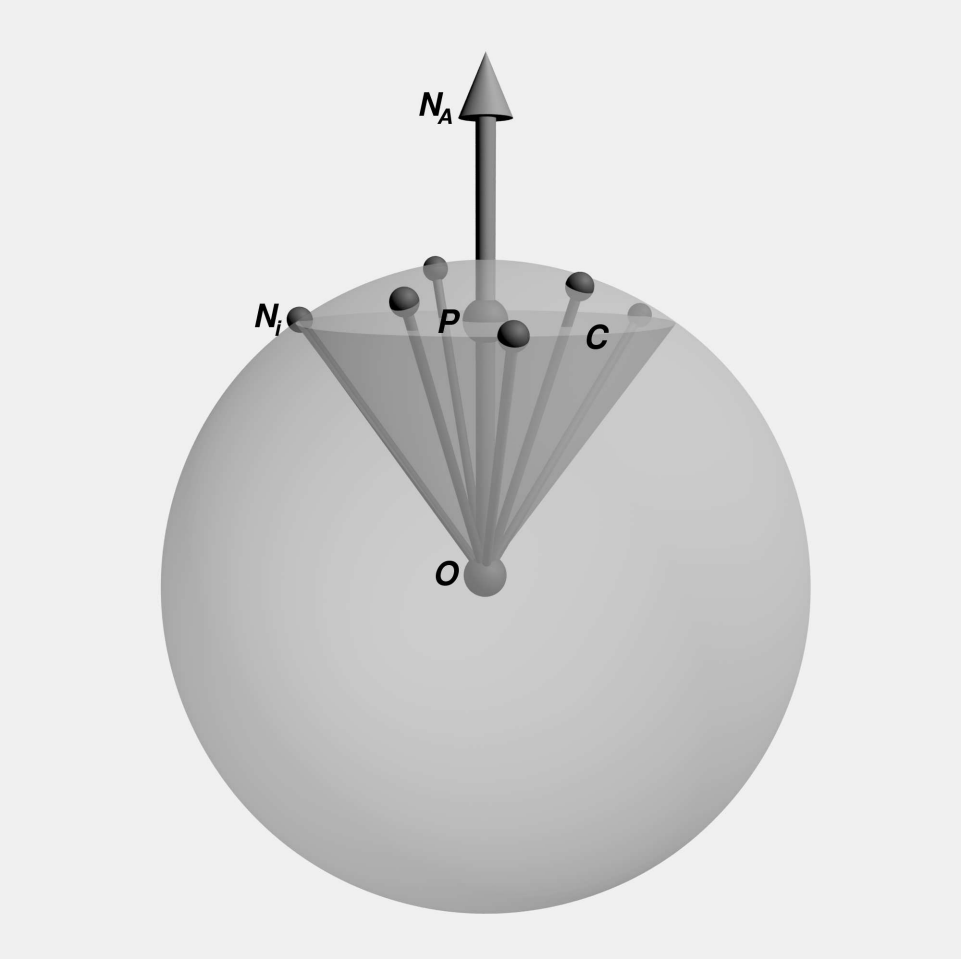


Figure C.1. The minimum bounding cone axis N_A is formed from the closest point, P, on the convex hull of $\{N_i\}$ minus the sphere origin O.

APPENDIX D

UTN FILE FORMAT

In order to facilitate the transfer of models between the CAD environment and *APE*, I have developed a simple file format called Utah NURBS (UTN). The main goal of this format was to provide a common ground for converting third party CAD application data into a format that *APE* could process. Secondary to that was the ability to hand encode models for testing purposes. It was not feasible to use the *Alpha_1* file format (binIO) as that would have required supporting all native types within *Alpha_1* (this is truly a proprietary format). The use of a custom format better facilitates the use of *APE* in conjunction with other third party CAD systems.

The format, a combination of TAG and STATE based stream IO, is described using a unit cube as an example. Any line beginning with a "#" is deemed a comment line and ignored. The first token expected within the file is the Model token and that is indicated by a "M" in column one. Following the token is a model ID, center of mass as a 3d point, and the mass of the model. Both the center of mass and mass are used when dealing with the model in a dynamic simulation (such as an assembly).

```
M  0  6  0.0000  0.0000  0.0000  1.0000
```

Each model is constructed from one or more surfaces. The start of a surface is signaled when the "S" token is read in column one. Following this token is the surface ID, number of knots in the U and V knot vectors, number of rows and columns in the control mesh, and the order of the surface in U and V.

```
S  0  6  6  3  3  3  3
```

Directly following the surface information line are the U and V knot vectors on separate lines.

```
-1.0000  -1.0000  -1.0000  0.0000  0.0000  0.0000
 0.0000   0.0000   0.0000  1.0000  1.0000  1.0000
```

The control mesh follows the knot vectors in row major order. Each element of the mesh is a rational space point with the rational component premultiplied.

```
0.5000   0.5000   0.5000  1.0000
0.5000   0.0000   0.5000  1.0000
0.5000  -0.5000   0.5000  1.0000
0.5000   0.5000   0.0000  1.0000
0.5000   0.0000   0.0000  1.0000
0.5000  -0.5000   0.0000  1.0000
0.5000   0.5000  -0.5000  1.0000
0.5000   0.0000  -0.5000  1.0000
0.5000  -0.5000  -0.5000  1.0000
```

Any trimming loops associated with the surface will follow the control mesh and precede the next surface token. The trimming token is a "T" in the first column. If a trimming loop is "simple" (on the boundary of the patches parameterization) then the token will have a "2" appended, otherwise a "1" is appended. This is just for internal optimization and not strictly required.

```
T2
```

The trimming data itself is then laid out such that the process of reading, and constructing, the internal structures is optimized. The total number of trim edges for all loops comes first. This is followed on a new line by the number of edges in a given loop and the number of points in that same loop.

```
4
4  5
```

In this case the surface has four total edges (which is expected since it is a simple trim). The next line starts the loop and indicates four edges in that loop represented by five points. Each edge of the loop is then specified by a line holding the adjacent surface's ID, the adjacent edge ID on that surface, and the number

of points representing the edge. Trailing this header information are the points making up the segments for the edge. A shared point between edges is not doubled (except for the final point of the loop) but is instead listed in the second segment to contain it.

```
  5 2 1
-1.0000  0.0000    0.5000   0.5000   0.5000
  4 0 1
 0.0000  0.0000    0.5000  -0.5000   0.5000
  2 0 1
 0.0000  1.0000    0.5000  -0.5000  -0.5000
  1 0 2
-1.0000  1.0000    0.5000   0.5000  -0.5000
-1.0000  0.0000    0.5000   0.5000   0.5000
```

The file reader will continue until enough edges have been read as specified on the first trimming data line. Therefore, if further loops need to be listed they can be appended here by simply specifying the number edges in the loop and the number of points in that new loop. Processing would then continue with the edges as above.

The remaining five surfaces would follow, each with its own ID and trimming loops. Optionally, a color token, "C", can be specified. If one follows the model token then the color will be assigned to the model as a whole. If it follows any given surface then that surface will have a color assigned that overrides that of the model. This token simply supplies the red, green, and blue color values as normalized components.

```
C  1.0  0.7412  0.3121
```

APPENDIX E

DEVICE CONFIG FILES

APE uses configuration files to define which devices will be used per session. The config file also contains any device specific setup data. As an example, if two active devices require TCP ports the config file would allocate them such that they do not conflict.

The format for the config file defines any line beginning with a "#" as a comment. All other lines must supply a valid device name as the first word. The device manager uses this to determine which device class to construct. Any remaining test on the line is passed onto the device class to be interpreted, therefore can be different for each device. The specific requirements for each device configuration line are as follows:

```
sarcos <mach> <tcps-port> <tcpr-port> <udpb-port>
phantom <mach> <tcps-port> <tcpr-port> <udpb-port>
bird <bird-ttyd> <button-ttyd>
glove <mach> (ttys) <glove> <bird> <button> (ports) <tcps> <tcpr> <udpb>
```

As an example, consider the follow config file that initializes a session to use all of the currently supported devices. In practice, the config file would define one or two devices to be used by a single user, but the system does allow for as many devices as requested, as well as duplicate devices.

```
#
# This device config file causes the device manager to setup
# the sarcos arm, phantom arm, and the glove/bird combination.
#
sarcos    vxw0-gw  5050 5060 5070
phantom   surreal  5000 5001 5002
glove     surreal  /dev/ttyd2 /dev/ttyd3 /dev/ttyd058 5003 5004 5005
```

REFERENCES

[1] Adachi, Y., "Touch And Trace On The Free-Form Surface Of Virtual Object," in *Proc. Virtual Reality Annual International Symposium,* pp. 162-168, Seattle, WA, September, 1993.

[2] Adachi, Y., Kumano, T., and Ogino, K., "Intermediate representation for stiff virtual objects," in *Proc. Virtual Reality Annual International Symposium,* pp. 203-210, Research Triangle Park, NC, March 11-15, 1995.

[3] Antonio, F., "Faster Line Segment Intersection," in *Graphics Gems III,* David Kirk (ed.), Academic Press, p. 199-202, 1992.

[4] Appino, P.A., Lewis, J.B., Koved, L., Ling, D.T., and Codella, C.F., "An Architecture for Virtual Worlds," in *PRESENCE: Teleoperators and Virtual Environments,* Vol. 1, No., 1, pp. 1-17, Winter, 1991.

[5] Ascension Bird, 6-dof magnetic tracking device from Ascension Technology.

[6] Brooks Jr., F.P., "Walkthrough - A Dynamic Graphics System for Simulating Virtual Buildings," in *Proc. Workshop on Interactive 3D Graphics,* pp. 9-21, Chapel Hill, NC, 1986.

[7] Bryson, S., and Levit, C., "The Virtual Wind Tunnel: An Environment for the Exploration of Three Dimensional Unsteady Flows," in *Proc. Visualization '91,* San Diego, CA, October, 1991.

[8] Bryson, S., and Johan, S., "An Extensible Interactive Visualization Framework for the Virtual Windtunnel," in *Proc. Virtual Reality Annual International Symposium,* pp. 106-113, Albuquerque, NM, March 1-5, 1997.

[9] Card, S.K., Moran, T.P., and Newell, A., "The Psychology of Human-Computer Interaction," *Lawrence Erlbaum Associates,* Hillsdale, NJ, 1983.

[10] Cohen, E., Lyche, T., and Riesenfeld, R., "Discrete B-Splines and subdivision techniques in computer aided geometric design and computer graphics," *Computer Graphics and Image Processing,* Vol 14, Number 2, October, 1980.

[11] Colgate, J.E., and Brown, J.M., "Factors Affecting the Z-Width of a Haptic Display," in *Proc. IEEE 1994 International Conference on Robotics & Automation,* pp. 3205-10, San Diego, CA, 1995.

[12] CyberTouch, 18-sensor CyberGlove with vibrotactile feedback option from Immersion Corporation.

[13] Dachille IX, F., Qin, H., Kaufman, A., and El-Sana, J., "Haptic Sculpting of Dynamic Surfaces," in *Proc. Symposium on Interactive 3D Graphics,* pp 103-110, Atlanta, GA, April 26-29, 1999.

[14] Dachille IX, F., Qin, H., and Kaufman, A., "A Novel Haptics-Based Interface and Sculpting System for Physics-Based Modeling and Design," in *Computer Aided Design,* Vol. 33, No.5, pp. 403-420, April, 2001.

[15] Duan, Y, Hua, J., and Qin, H., "HapticFlow: PDE-based Mesh Editing with Haptics," in *Proc. International Conference on Computer Animation and Social Agents,* Geneva, Switzerland, July 7-9, 2004.

[16] Durlach, N.I., and Mavor, A.S. Editors, *Virtual Reality Scientific And Technological Challenges,* Washington, D.C., National Academy Press, 1995.

[17] Funkhouser, T.A., and Séquin, C.H., "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments," in *Proc. SIGGRAPH,* ACM, pp. 247-254, Anaheim, CA, August, 1993.

[18] Gilbert, E.G., Johnson, D.W., and Keerth, S.S., "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," in *Proc. IEEE Journal of Robotics and Automation,* Vol. 4, No. 2, April, 1988.

[19] Gu, X., Snyder, J., and Gortler, S., "Central Axis for Silhouette Clipping," E-Mail Correspondence, September 20, 2000.

[20] Hatvany, J., "Can Computers Compete with Used Envelop?," in *Proc. IFAC* 9^{th} *Triennial World Congress,* pp. 3373-3375, Budapest, Hungary, 1984.

[21] Hua, J., Duan, Y., and Qin, H., "Design and Manipulation of Polygonal Models in a Haptic, Stereoscopic Virtual Environment," in *Proc. International Conference on Shape Modeling and Applications,* pp. 145-154, MIT, Cambridge, MA, June 15-17, 2005.

[22] Ho, C., *Feature-Based Process Planning and Automatic Numerical Control Part Programming,* Ph.D. Thesis, University of Utah, Computer Science Department, December, 1997.

[23] Hollerbach, J.M., Cohen, E.C., Thompson, W.B., Freier, R., Johnson, D., Nahvi, A., Nelson, D., Thompson II, T.V., and Jacobsen, S.C., "Haptic Interfacing for Virtual Prototyping of Mechanical CAD Designs," in *Proc. Design for Manufacturing Symposium,* ASME, Sacramento, CA, September, 1997.

[24] Hollerbach, J.M., Cohen, E.C., Thompson, W.B., Freier, R., Johnson, D., Thompson II, T.V., "Virtual Prototyping for Human-Centric Design," in *Proc. NSF Design and Manufacturing Research Conference,* Vancouver, January, 2000.

[25] Hong, L., Muraki, S., Kaufman, A., Bartz, D., and He, T., "Virtual Voyage: Interactive Navigation in the Human Colon," in *Proc. SIGGRAPH,* ACM, pp. 27-34, Los Angeles, CA, August 3-8, 1997.

[26] Jacobsen, S.C., Smith, F.M., Iversen, E.K., and Backman, D.K., "High performance, high dexterity, force reflective teleoperator," in *Proc. 38th Conference Remote Systems Technology,* pp. 180-185, Washington, DC, November, 1990.

[27] Johnson, D.E. and Cohen, E.C., "A Framework For Efficient Minimum Distance Computations," in *Proc. International Conference on Robotics and Automation,* IEEE, pp. 3678-3684, Leuven, Belgium, May, 1998.

[28] Johnson, D.E. and Cohen, E.C., "An Improved Method For Haptic Tracing Of Sculptured Surfaces," in *Proc. Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems,* ASME, DSC-Vol. 64, pp. 243-248, Anaheim, CA, November, 1998.

[29] Johnson, D.E., Thompson II, T.V., Kaplan, M., Nelson, D., and Cohen, E.C., "Painting Textures with a Haptic Interface," in *Virtual Reality '99*, pp. 282-285, Houston, Texas, March 13-17, 1999.

[30] Johnson, D.E., and Willemsen, P., "Accelerated haptic rendering of polygonal models through local descent," in *Proc. Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems,* ASME, pp. 18-23, March 2004.

[31] Johnson, D.E., Willemsen, P., and Cohen, E., "6-DOF Haptic Rendering Using Spatialized Normal Cone Search," in *Transactions on Visualization and Computer Graphics,* 2005.

[32] Kalawsky, R.S., *The Science of Virtual Reality and Virtual Environments,* Addison-Wesley, Cambridge, 1993.

[33] Kim, Y., Otaduy, M., Lin, M.C., and Manocha, D. "Six degree-of-freedom haptic display using localized contact computations," in *Proc. Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems,* ASME, pp. 209-216, March 2002.

[34] Maekawa, H., and Hollerbach, J.M., "Haptic Display for Object Grasping and Manipulating in Virtual Environment," in *Proc. IEEE International Conference Robotics & Automation,* pp. 129-134, Leuven, Belgium, May 16-21, 1998.

[35] Marhefka, D.W., and Orin, D.E., "Simulation Of Contact Using A Nonlinear Damping Model," in *Proc. International Conference on Robotics and Animation,* pp. 1662-1668, Minneapolis, MN, April 1996.

[36] Mark, W.R., Randolph, S.C., Finch, M., Van Verth, J.M., and Taylor III, R.M., "Adding Force Feedback to Graphics Systems: Issues and Solutions," in *Proc. SIGGRAPH,* ACM, pp. 447-452, New Orleans, LA, August. 4-9, 1996.

[37] Massie, T.M. and Salisbury, J.K., "The PHANToM Haptic Interface: A Device for Probing Virtual Objects," in *Proc. Symposium Haptic Interfaces for Virtual Environment and Teleoperator Systems,* DSC-Vol 1, pp. 295-301, Chicago, IL, November 6-11, 1994.

[38] McNeely, W., Puterbaugh, K., and Troy, J., "Six degree-offreedom haptic rendering using voxel sampling," in *Proc. SIGGRAPH,* pp. 401-408, Los Angeles, CA, August 8-13, 1999.

[39] Mine, M., "Virtual Environment Interaction Techniques," University of North Carolina Computer Science Technical Report TR95-018, 1995.

[40] Minsky, M., Ouh-Young, M., Steele, M., Brooks, F.P. Jr., Behensky, M., "Feeling and Seeing: Issues in Force Display," in *Proc. Symposium on Interactive 3D Graphics,* pp. 235-243, Snowbird, Utah, 1990.

[41] Nelson, D.D. and Cohen, E.C., "Interactive Mechanical Design Variation for Haptics and CAD" in *Proc. EUROGRAPHICS 1999,* September, 1999.

[42] Nelson, D.D., Johnson, D.E., and Cohen, E.C., "Haptic Rendering of Surface-to-Surface Sculpted Model Interaction," in *Proc. Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems,* ASME, Nashville, TN, November, 1999.

[43] Otaduy, M.A., and Lin, M.C., "Sensation preserving simplifcation for haptic rendering," in *Proc. SIGGRAPH,* ACM, pp. 543-553, San Diego, CA, July 27-31, 2003.

[44] Piegl, L. and Tiller, W., *The NURBS Book,* Springer, Berlin, 1995.

[45] Patoglu, V. and Gillespie, B., "Extremal Distance Maintenance for Parametric Curves and Surfaces," in *Proc. International Conference on Robotics and Automation,* IEEE, pp. 2817-2823, 2002.

[46] Patoglu, V. and Gillespie, B., "Haptic Rendering of Parametric Surfaces using a Feedback Stabilized Distance Tracking Algorithm," in *Proc. International Conference on Haptic Interfaces for Virtual Environment and Teleoperator Systems,* IEEE, Vol. 3, pp. 391-399, 2004.

[47] Patoglu, V. and Gillespie, B., "A Closest Point Algorithm for Parametric Surfaces with Global Uniform Asymptotic Stability," in *Proc. World Haptics Symposium,* Pisa Italy, March 2005.

[48] Pausch, R., Burnatte, T., Brockway, D., and Weiblen, M.E., "Navigation and Locomotion in Virtual Worlds via Flight into Hand-held Miniatures," in *Proc. SIGGRAPH,* pp. 399-400, Los Angeles, CA, August 6-11, 1995.

[49] Pierce, J.S., Forsberg, A., Conway, M.J., Hong, S., Zeleznik, R., and Mine, M.R., "Image Plane Interaction Techniques in 3D Immersive Environments," in *Proc. Symposium on Interactive 3D Graphics,* pp. 39-43, Providence, RI, April 27-30, 1997.

[50] Peterson, G.L., "Concurrent Reading While Writing," *ACM Transactions on Programming Languages and Systems,* Vol. 5, No. 1, pp. 46-55, January, 1983.

[51] Poupyrev, I., Billinghurst, M., Weghorst, S., and Ichikawa, T., "The Go-Go Interaction Technique: Non0linear Mapping for Direct Manipulation in VR," in *Proc. UIST,* 1996.

[52] Riesenfeld, R., Cohen, E., Fish, R., Thomas, S., Cobb, E., Barsky, B., Schweitzer, D., and Lane, J., "Using the Oslo algorithm as a basis for CAD/CAM geometric modeling," in *Proc. National Computer Graphics Association,* 1981.

[53] Riesenfeld, R., "Design tools for shaping spline models," in *Mathematical Methods in Computer Aided Geometric Design,* (Edited by T. Lyche and L. Schumaker), Academic Press, 1989

[54] Riesenfeld, R., "Modeling with NURBS curves and surfaces," in *Fundamental Developments of Computer Aided Geometric Design,* L. Piegl (ed.), Academic Press, 1993.

[55] Robertson, G.G., Card, S.K., and Mackinly, J.D., "The Cognitive Coprocessor Architecture for Interactive User Interface," in *Proc. UIST,* pp. 10-18, November 13-15, 1989.

[56] Ruspini, D.C., Koloarov, K., and Khatib, O., "The Haptic Display of Complex Graphical Environments," in *Proc. SIGGRAPH,* pp. 345-351, Los Angeles, CA, August 3-8, 1997.

[57] Sachs, E., Roberts, A., and Stoops, D., "3-DRAW: A Tool For Designing 3D Shapes," in *IEEE COmputer Graphics and Applications,* Vol. 11, pp. 18-24, November, 1991.

[58] Salisbury, K. and Tarr, C., 1997, "Haptic Rendering of Surfaces Defined by Implicit Functions," in *Proc. Symposium Haptic Interfaces for Virtual Environment and Teleoperator Systems,* DSC-Vol. 61, pp. 61-67, Dallas, TX, November 15-21, 1997.

[59] Shaw, C., Green, M., Liang, J., and Sun, Y., "Decoupled Simulation in Virtual Reality with The MR Toolkit," in *ACM Transactions on Information Systems,* Vol. 11, No. 3, July, 1993.

[60] Sheridan, T.B., "Supervisory Control of Remote Manipulators, Vehicles, and Dynamic Processes: Experiments in Command and Display Aiding," in *Advances in Man-Machine Systems Research,* Vol. 1, pp. 49-137, 1984.

[61] Snyder, J., "An Interactive Tool For Placing Curved Surfaces Without Interpenetration," in *Proc. SIGGRAPH,* pp. 209-218, Los Angeles, CA, August 6-11, 1995.

[62] State, A., Livingston, M.A., Garett, W.F., Hirota, G., Whitton, M.C., Pisano MD, E.D., and Fuch, H., "Technologies for Augmented Reality Systems: Realizing Ultrasound-Guided Needle Biopsies," in *Proc. SIGGRAPH,* pp. 439-446, New Orleans, LA, August 4-9, 1996.

[63] Stewart, P., Buttolo, P., and Chen, Y., "CAD Data Representations for Haptic Virtual Prototyping," in *Proc. ASME Design Engineering Technical Conference,* Sacramento, CA, September 14-17, 1997.

[64] Stoakley, R., Conway, M.J., and Pausch. R., "Virtual Reality on a WIM: Interactive Worlds in Miniature," in *Proc. CHI '95,* ACM, pp.265-272, Denver, CO, 1995.

[65] Sutherland, I.E., "A Head-Mounted Three Dimensional Display," in *FJCC,* Thompson Books, Vol. 757, Washington DC, 1968.

[66] Thompson II, T.V., Johnson, D.E., Cohen, E., "Direct Haptic Rendering Of Sculptured Models," in *Proc. Symposium on Interactive 3D Graphics,* pp. 167-176, Providence, RI, April 27-30, 1997.

[67] Thompson II, T.V., Nelson, D.D., Cohen, E., and Hollerbach, J.M., "Maneuverable NURBS Models within a Haptic Virtual Environment," in *Proc. Symposium Haptic Interfaces for Virtual Environment and Teleoperator Systems,* DSC-Vol. 61, pp. 37-44, ASME, Dallas, TX, November 15-21, 1997.

[68] Thompson II, T.V. and Cohen, E.C., "Direct Haptic Rendering Of Complex Trimmed NURBS Models," in *Proc. Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems,* ASME, Nashville, TN, November, 1999.

[69] Yoshikawa, T., Yokokohji, Y., Matsumoto, T., Zheng, X., "Display of Feel for the Manipulation of Dynamic Virtual Objects," in *Journal of Dynamic Systems, Measurement, and Control,* Vol. 117, December, 1995.

[70] Zilles, C.B., and Salisbury, J.K., "A Constraint-based God-object Method For Haptic Display," in *Proc. IEE/RSJ International Conference on Intelligent Robots and Systems, Human Robot Interaction, and Cooperative Robots,* Vol 3, pp. 146-151, 1995.