Proceedings Symposium on Interactive 3D Graphics, Providence, RI, April 27-30, 1997.

Direct Haptic Rendering Of Sculptured Models

Thomas V Thompson II David E Johnson Elaine Cohen

Department of Computer Science
University of Utah

Abstract

A new tracing algorithm is described that supports haptic rendering of NURBS surfaces without the use of any intermediate representation. By using this tracing algorithm in conjunction with algorithms for surface proximity testing and surface transitions, a complete haptic rendering system for sculptured models has been developed. The system links an advanced CAD modeling system with a Sarcos force-reflecting exo-skeleton arm. A method for measuring the quality of the tracking component of the haptic rendering separately from the haptic device and force computation is also described.

CR Descriptors: H.1.2 [Models and Principles] User/Machine Systems; C.3 [Special-Purpose and Application-Based Systems] Real-Time Systems; I.3.7 [Computer Graphics] Three-Dimensional Graphics and Realism; I.6.4 [Simulation and Modeling] Types of Simulation - *Distributed*; F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems; J.6 [Computer-Aided Engineering].

Additional Keywords: haptic, force feedback, NURBS evaluation, virtual environment, closest point calculation, distributed simulation.

1 Introduction

Current modeling systems offer limited feedback to a model designer. One promising approach for increasing the information available to a designer is haptic rendering. Haptic rendering is the process of simulating the forces generated by contact with a virtual model so that a person's sense of touch can be used to interrogate the model. In conjunction with visual feedback, haptic rendering can facilitate understanding of complex models and add a sense of realism to interactive systems [10].

In CAD/CAM design, parametric surfaces have become the surface representation of choice. Parametric surfaces such as NURBS have the advantage of compact representation, higher order continuity, and exact computation of surface normals. These properties are all useful in complex, realistic virtual environments [22]. Current haptic systems tend to use polygonal representations to calculate appropriate haptic responses [21], making correct conversion from parametric CAD models a non-trivial task.

We introduce direct haptic rendering of sculptured models. In order to support direct haptic rendering, we have developed and tested algorithms for surface proximity testing, fast updates to local closest point on a surface, and smooth transitions between surface patches. These algorithms are tested on a complete system that integrates *Alpha_1*, a research modeling package [18, 19], with a Sarcos Dextrous Arm Master [12]. We compare the results of these algorithms with an intermediate representation and demonstrate results of testing the algorithms against theoretically optimal methods.

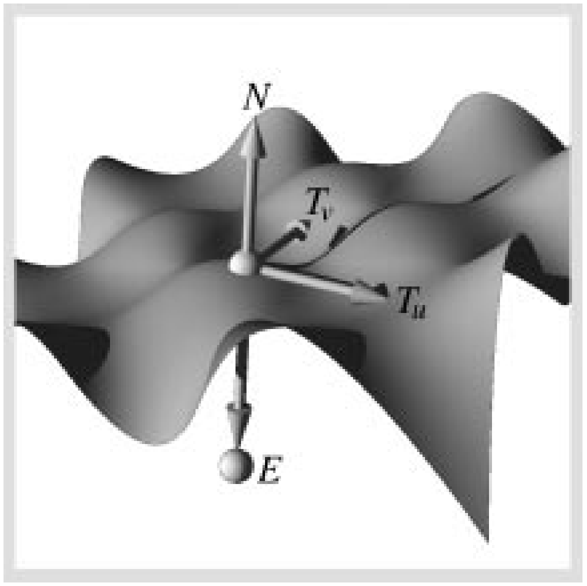

Figure 1: A NURBS surface showing the important state variables for direct haptic rendering.

2 Background

Haptic rendering systems generate forces that are applied to a person's hand or arm to create a sense of contact with a virtual model. The counterforce needed to prevent penetration into a virtual surface is called the restoring force, and is calculated using a wall model. Wall models often have a restoring force proportional [6] to the penetration depth and in the direction of the surface normal (Figure 1). The

[1] {tthompso,dejohnso,cohen}@cs.utah.edu
[2] Dept. of Computer Science, 3190 Merrill Engineering Bldg, University of Utah, Salt Lake City, UT 84112

wall model used in our experiments is a nonlinear damping model developed by Marhefka and Orin [13]. The wall model is defined as $F_v = k_p x^n + k_v x^n x'$ where x is the penetration depth, x' is the velocity of the end-effector, k_p is the spring coefficient, and k_v is the damping coefficient. For the Sarcos arm we found $n = \frac{1}{2}$ to be effective.

The penetration depth and surface normal to use in the wall model are found by tracking the closest point on the surface to the end-effector. The surface normal at the closest point provides the normal to use in the wall model. The penetration depth is calculated by projecting the end-effector onto the surface normal, with positive penetration defined as into the surface. In order to maintain the stiffness of the virtual surface, the force servo loop must calculate the closest point and response forces at several hundred Hz [15]. This fast update rate limits the complexity of algorithms that can be used to find the closest point on the surface.

In order to maintain fast force servo rates, current haptic rendering systems tend to use simple intermediate surface representations instead of sculptured surfaces in their force calculations. Adachi [1] and Mark [14] advocate the use of a sequence of relatively slowly changing planar approximations as an intermediate representation, since the closest point can be quickly computed with planes. However, planar representations are fundamentally limited when trying to approximate surfaces with high curvature [14]. In addition, the intermediate planar approximations are sampled in a time-dependent fashion, not by position, so that the surface "felt" by a user is not necessarily repeatable during multiple tracings.

We directly track the closest point on the parametric surface. Finding the closest point on a surface S to a point E is a fundamental query for a surface representation. A common approach [16] is to solve for the roots of

$$(S - E) \times (S^u \times S^v) = 0. \tag{1}$$

The roots of Eq. 1 can be found through iterative Newton methods [17], with various means of guaranteeing a globally correct solution. However, the system may involve high degree polynomials, making the solution difficult.

The CAD community has developed methods of tracking points on surfaces during intersection operations [11]. Many of these methods find the Euclidean closest point directly, requiring surface-plane intersections to be computed. These methods are too slow for haptic environments. Barnhill developed a parametric marching algorithm [3] for closest point tracking that minimizes error to a first order surface approximation. Snyder [22] uses Newton iteration to improve an approximation to the closest point on a surface during collision detection, as does Baraff [2].

3 System Overview

A useful computation model to come out of virtual environment research is decoupling the simulation and interactive processes. In haptic systems, this model has been applied to decouple the simulation and haptic processes [1, 14]. This decoupling allows the haptic loop to run on a real-time system, while the more computationally intensive simulation loop can run on a standard workstation. In our system (Figure 2), the simulation process runs on an SGI workstation and is used for visual display and global computations. The haptic process runs on Motorola 68040 microprocessors under the VxWorks real-time kernel and is used for haptic display and local computations. The two sides communicate over Ethernet using TCP/IP and UDP packets [9].

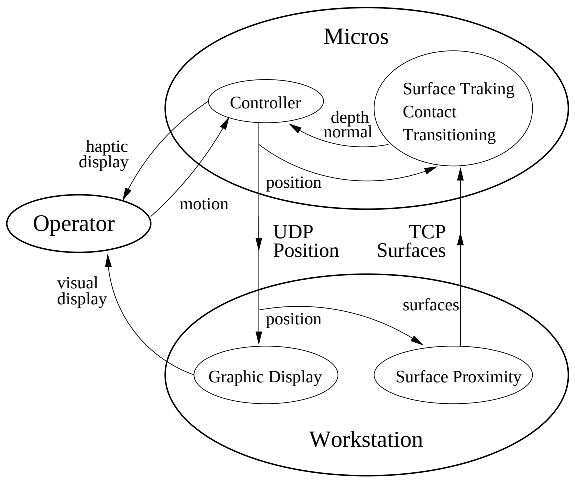

Figure 2: The system is divided into three portions: operator, micros, and workstation.

In haptic research, this system setup may vary in the type of workstation on the simulation side and with PC's often substituted on the haptic side. These systems share several characteristics. There is usually limited bandwidth between the simulation and haptic processes, the haptic process usually has limited memory (at least in relation to the simulation side) and the computational capacity of the simulation process is usually far superior to that of the haptic process. These constraints influence system design.

3.1 Haptic Process

With limited memory and computational power, the haptic process must be restricted to only those computations that must complete for each iteration of the force loop and to data that is necessary to perform those computations. To calculate the restoring force for the haptic display three main computations must take place within the haptic process: surface tracking, contact, and transitioning. The restoring force is then communicated to the Sarcos Dextrous Arm Master (Figure 3), an advanced hydraulic force-reflecting exoskeleton.

The local geometric environment is stored within the haptic process. This environment consists of all surfaces that are "active" (currently being tracked, see Section 4.2) and other surfaces that were active but since have become inactive. The contents of this environment are regulated by the simulation side since determining the contents involves global computations such as surface proximity. The restricted memory on the micros and the need for multiple surfaces to be cached within the haptic process again points out an advantage of a compact parametric surface representation over faceted models.

3.2 Simulation Process

The simulation process runs within the *Alpha_1* CAD modeling environment. *Alpha_1* is a research modeling package that aids in the creation, manipulation and display of NURBS models. A high-powered graphics engine, such as an SGI workstation, is required to both maintain high visual

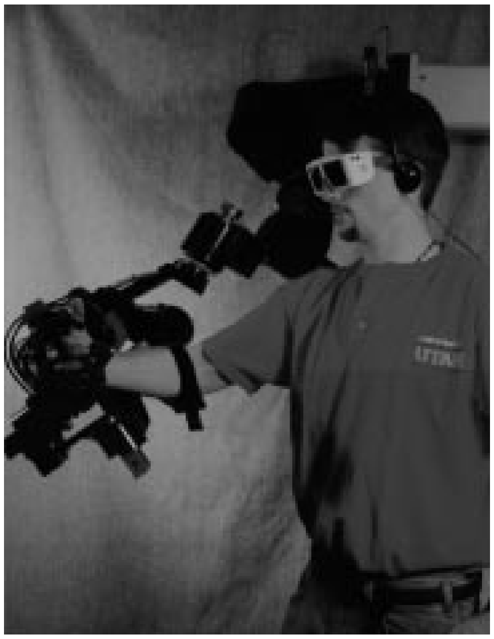

Figure 3: The Sarcos Dextrous Arm Master.

frame rates and perform the global computations necessary for the haptic simulation.

Maintaining the local environment of the haptic process is the main function of the simulation process. Using the arm location, the workstation does surface proximity checks to determine nearby surfaces (see Section 4.1).

When a surface is deemed "nearby" for the first time, the surface's control points, knot vectors, and approximate closest point are sent to the haptic process along with a unique surface number. Subsequent encounters with the surface result in a small activation record being sent. Conversely, a deactivation record is sent when proximity testing determines a surface is no longer near the arm.

3.3 Communications

The haptic process and simulation process are in constant communication over an Ethernet network (Figure 2). The simulation process must be continually updated with the current position of the arm in order to maintain synchronization between haptic and visual display as well as to perform correct global computations. The haptic process must have its local environment updated continually so that appropriate forces are calculated.

The arm controller continually sends filtered position information to the workstation via UDP packets. UDP is acceptable for this channel since the haptic loop runs much faster than the simulation process, and sends several packets per simulation time frame.

Transmissions to the haptic process are received by a dedicated networking board. The small size of activation/deactivation packets reduces network overhead. The surface patch information and activation/deactivation packets are sent by TCP/IP, since they must be guaranteed to arrive.

4 Direct Haptic Rendering

We break the problem of rendering sculptured surfaces into several phases. Surface proximity testing determines when the arm is near enough to a surface to potentially contact it. When a surface becomes proximal it is made active and the closest point on the surface is tracked along with arm movement. Contact occurs as the arm penetrates into the surface. Tracing is the result of lateral motion during contact, and the closest point must continue to be tracked in the haptic loop in order to compute restoring forces that create an accurate sense of touch. Transitions occur when tracking a point across surface patch boundaries, and must be determined in the haptic loop.

4.1 Surface Proximity Testing

Surface proximity testing approximates the distance from the end-effector point, E, to the nearest point on a surface, S. This information is used to maintain the proper local geometry within the haptic process. Since this testing must occur for all the models in the environment, it is done within the simulation process on the workstation.

A rough check for surface proximity is done using bounding boxes around each surface. The distance from E to a bounding box is a trivial computation. The majority of the surfaces in the environment are too distant to warrant a better distance approximation.

For the remaining surfaces, we use a method [22] we refer to as "nodal mapping" to find a first order approximation to the closest point on the surface (Figure 4). The end-effector point is projected onto the control mesh of the NURBS surface resulting in a point Q. Each vertex of the control mesh has an associated (u,v) parametric value that is called the "node" [5]. An approximate (u,v) for Q is determined by interpolating between node values using the barycentric coordinates of Q. The surface is evaluated at the interpolated (u,v) point and the distance between $S(u, v)$ and E is used as the surface proximity distance.

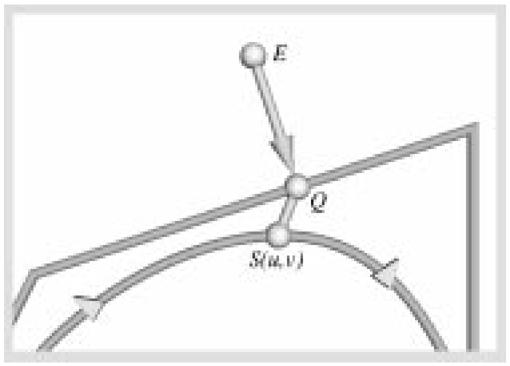

Figure 4: The projected distance along the control polygon is used as the parametric distance between associated nodes.

4.2 Tracking Phase

When a surface becomes active, the approximate closest point sent with the surface is used to initialize a local closest point tracking method (Figure 5a). Each active surface has its local closest point tracked until it is deactivated. This tracking method works directly on the parametric surface and is fast enough to track at force servo rates, making it suitable for direct haptic rendering of a surface.

For simplicity, we present the direct parametric tracing (DPT) method on a B-spline curve rather than a surface. Appendix A defines basic B-spline curves and surfaces and some of their properties. The DPT method uses the previous point on the curve $\gamma(u)$, the tangent vector at $\gamma(u)$, $\gamma'(u)$, and the current end-effector location, E, to determine a new approximate closest point on the curve (Figure 5b).

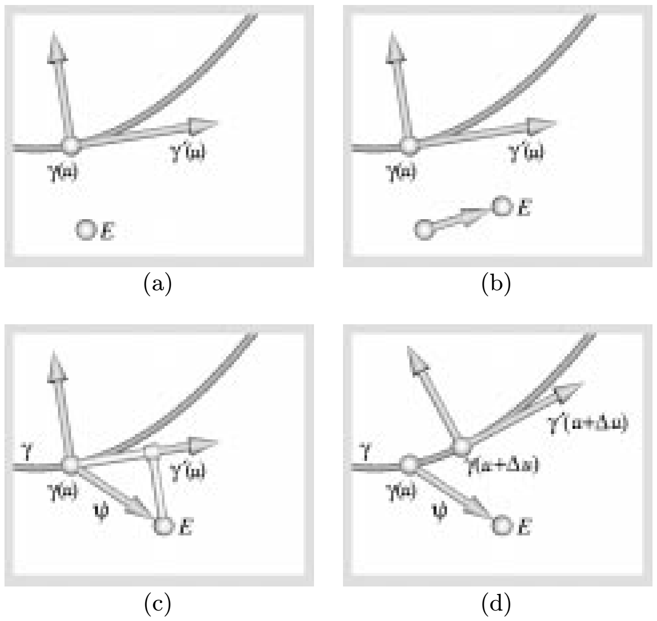

Figure 5: (a) Initial state. (b) End-effector moves. (c) Projection of position onto surface tangent plane. (d) New surface point and tangent plane found via parametric projection.

The velocity curve, $\gamma'(u)$, relates changes in position along the curve in Euclidean space to changes in position in parametric space (Eq. 2).

$$\gamma'(u) = \frac{d\gamma}{du} \approx \frac{\Delta\gamma}{\Delta u}. \tag{2}$$

Given an Euclidean movement along γ, the corresponding movement in the parametric space of the curve is calculated as

$$|\Delta u| \approx \frac{\|\Delta\gamma\|}{\|\gamma'(u)\|}. \tag{3}$$

In order to use Eq. 3 as a closest point tracking method, movement of the end-effector 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 end-effector 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 5c). Thus,

$$\Delta\gamma \approx \frac{\langle\, \psi \,,\, \gamma'(u) \,\rangle}{\|\gamma'(u)\|^2}\, \gamma'(u). \tag{4}$$

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

$$\gamma'(u) = (k-1) \sum_{i=1}^{n} \frac{(P_i - P_{i-1}) B_{i,k-1}(u)}{u_{i+k-1} - u_i}. \tag{5}$$

Eq. 5 can be simplified greatly if the curve is refined with $k-1$ knots at index i^* with the value u^*. The resulting simplified equation is

$$\gamma'(u^*) = \frac{(k-1)}{u_{i^*+k} - u_{i^*+1}} (P_{i^*+1} - P_{i^*}). \tag{6}$$

Since we wish to track points that are actually on the curve, Eq. 3 is used to convert back into parametric space. The key to efficient computation of Δu is Eq. 6. 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 $\Delta\gamma$ (Eq. 4), Eq. 3 can be expanded into

$$|\Delta u| \approx \frac{\|\Delta\gamma\|}{\|\gamma'(u^*)\|} \approx \left| \frac{\langle\, \psi \,,\, \gamma'(u^*) \,\rangle}{\|\gamma'(u^*)\|^2} \right|. \tag{7}$$

The sign of Δu is determined by the sign of the projection in Eq. 4. This is directly related to the dot product in the numerator of Eq. 7. Since the numerator is the only term in Eq. 7 that is signed, the absolute value signs can be removed. The constant term representing the parametric speed can be factored out leaving the result,

$$\Delta u \approx \frac{\langle\, \psi \,,\, (P_{i^*+1} - P_{i^*}) \,\rangle}{\|P_{i^*+1} - P_{i^*}\|^2} \left(\frac{u_{i^*+k} - u_{i^*+1}}{k-1} \right). \tag{8}$$

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 Eq. 8 at the next time step (Figure 5d).

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, this provides an excellent approximation.

For surfaces, the method is essentially the same, although the projection step now requires projection onto the tangent plane, $S'(u,v)$, of the surface. Barycentric coordinates are used to derive Δu and Δv. Our implementation of the DPT method, when used to trace a single surface, runs at 1400Hz on the Motorola 68040 processor that is used in the haptic process. As a basis for comparison of processor rates, on a SGI workstation with a R4400 processor the DPT method runs at 9000Hz.

4.3 Contact and Tracing

Contact is initiated when the penetration depth of the closest active surface becomes larger than zero. The penetration depth is calculated by projecting the arm location onto the surface normal. In our system, surface normals point out of a model so the negation of the projection results in a positive penetration depth when the end-effector is within the model. When a surface has been contacted it is considered "current".

During contact, the current surface's local closest point, C^*, is updated as during the tracking phase. However, the remaining active surface's have their local closest points, C_i, updated using C^* in the tracking algorithm instead of E. This is to allow for efficient surface transitioning calculations and is covered in more detail in Section 4.4.

Once a surface has become current it remains current until a transition either off the model or onto an adjacent surface occurs. [23] states several problems with methods that do not exhibit this characteristic. Among these problems are pushing through a model, force discontinuities, and inability to generate sufficient restoring forces due to lack of penetration depth.

All three of these problems are illustrated in Figure 6. The model in this figure is a narrow rectangle constructed from multiple surfaces. Figure 6a shows the end-effector entering the model through surface γ_1, which results in C^* being established as the current local closest point. The end-effector then continues to move into the model resulting in one of two possible new configurations.

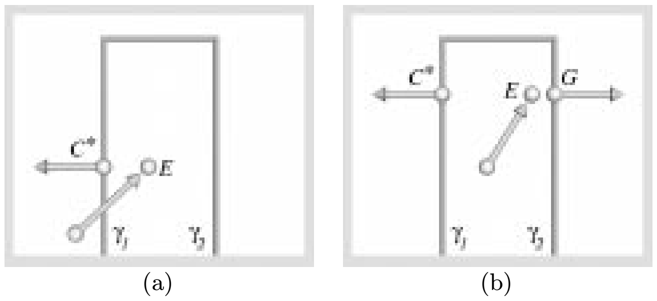

Figure 6: (a) Contact established. (b) Use of global closest point would accelerate the end-effector through the model.

Our method holds γ_1 as the current surface, therefore C^* stays on that surface resulting in a restoring force that is larger in magnitude but in the same direction as the previous iteration (Figure 6b). However, if a global closest point, G, was used then a force smaller in magnitude and opposite in direction than that of the previous iteration would result(Figure 6b). This clearly is not the characteristic one would want as the end-effector is able to push through the surface, the force becomes discontinuous, and the penetration depth does not grow high enough to generate a sufficient restoring force.

Generating a discontinuous force is not only possible when pushing through an object, it is possible any time the global closest point differs from the local closest point. Consider the case in Figure 7. In this example, contact has been established with γ_1 resulting in the given C^* (Figure 7a). The end-effector then moves to a position that results in a G that is not equal to C^* (Figure 7b). Our system would continue to generate restoring forces toward γ_1, but a system that uses G would end up pushing the end-effector in a direction that it is already traveling, accelerating the end-effector off the surface.

Another reason for using a current surface and its local closest point instead of G is the ability to trace out sharp edges (Figure 8). Consider a configuration where the end-effector has established contact with a model as in Figure 8a.

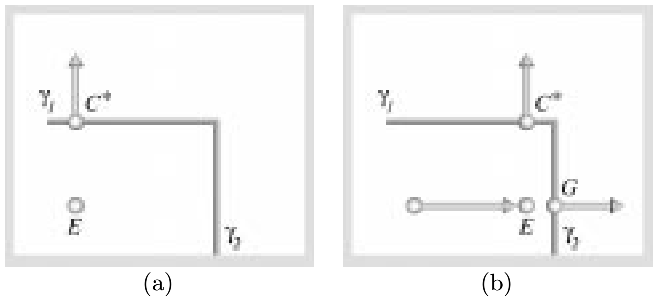

Figure 7: (a) Contact established. (b) Use of global closest point would cause a slip to be induced off the model.

The end-effector then moves out towards the edge (Figure 8b). C^* rests out near the edge of γ_1 while G not only is on γ_2 but has a negative penetration depth. This results in a zero restoring force, effectively portraying falling off an edge that is not present in the model. Our method would use C^* until the end-effector moved out past the edge of γ_1 and then transition off the edge, reentering free-flight.

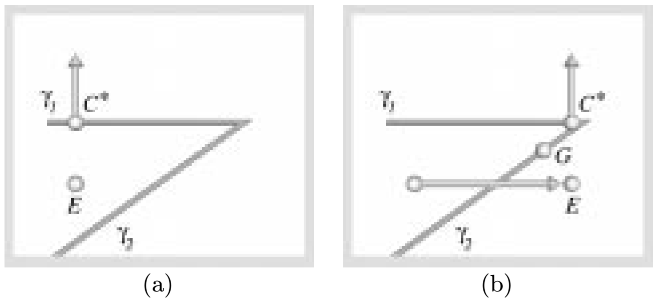

Figure 8: (a) Contact established. (b) Use of the global closest point would induce an artificial edge.

4.4 Transitions

Most sculptured models consist of multiple surfaces. It is necessary for the tracing algorithm to transition from one surface and onto another if the end-effector traces out such a path. This computation directly affects the resulting closest point, surface normal and penetration depth and therefore must be performed in the haptic process.

There are three main stages to the transitioning problem: edge detection, selection of an appropriate surface onto which to transition, and the calculation of an appropriate normal. One special form of transitioning is that of transitioning off the current surface and into free-flight. This type of transitioning is the opposite of the contact problem. If the penetration depth becomes negative then the current surface is returned to active status and tracking returns to nontracing mode (i.e. all active surfaces tracked from E).

All remaining forms of transitioning occur at an edge. Detecting an edge crossing on the current surface by C^* in Euclidean space would be a difficult problem. This problem would involve determining when a three-space point, C^*, crosses from one side of a three-space curve onto the other side, the curve being the extraction of the iso-curve along the boundary of the surface. However, this problem is greatly simplified since the surface has a parametric representation. Furthermore, the point C^* also has a parametric value on the surface. Edge detection is then reduced to detecting when the parametric value of C^*, (u, v), is on the boundary of the parametric domain (Figure 9). The domain of a B-spline surface is defined as $(u_{k_u-1} \cdots u_{m+1}, v_{k_v-1} \cdots v_{n+1})$.

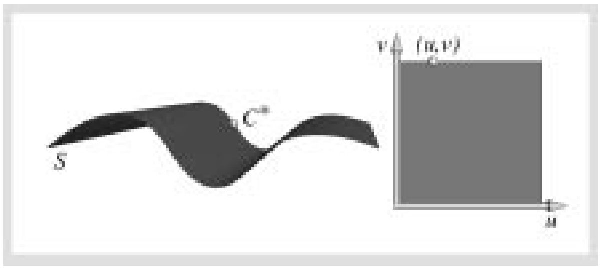

Figure 9: Edge detection performed in parametric domain.

Once contact with an edge has been confirmed, the next step is to determine the appropriate surface onto which to transition. Each of the active surfaces is tracked using C^* instead of E. Each C_i is therefore an approximation to the closest point on neighboring surfaces to C^*. When C^* lies an edge, there will be a corresponding C_j on the adjacent surface that equals (within some numerical epsilon) C^*. This surface is made current and the previous current surface is returned to active status. The tracking algorithm is applied to the new current surface using E so that C_j will represent the local closest point to E and can therefore be referred to as C^*.

If the new C^* is not on an edge, then the normal on the new current surface at C^* is used in the penetration depth and force calculations. If it is still on the edge special care must be taken in choosing the normal. Consider Figure 10a. If the point E lies in either shaded area then C^* will be on the edge between γ_1 and γ_2. If the normal for γ_1, n_1, is chosen and E is within the lower shaded area, a positive penetration depth would be calculated. But E is outside the model so this would be an incorrect answer. Similarly, if E is within the shaded area at the top and n_2, the normal from γ_2, is chosen, an incorrect penetration depth would be computed. Figure 10b illustrates the opposite case. In this figure if E lies within one of the shaded areas a negative penetration depth can be computed if the incorrect normal is chosen.

To solve this special case, and still keep the algorithm as computationally simple as possible, a new normal is computed when a transition results in a new C^* that lies on an edge. This new normal, N, is the normalized sum of n_1 and n_2 (Figure 10). This resulting normal solves the problem but also induces the side effect of beveling (Figure 11a) the interior of such trouble areas. Another choice for the normal would be a vector pointing along a line from E to C^* that is directed to be out of the surface. This normal is more difficult to compute and results in the side effect of the interior being rounded (Figure 11b). If penetration depth is kept small, neither side effect would be noticeable. Therefore, the choice of the computationally more efficient approach of normal summation was made.

This transitioning algorithm allows models constructed from multiple surfaces to be traced. It also allows these surfaces to be aligned in any fashion, as long as the edges

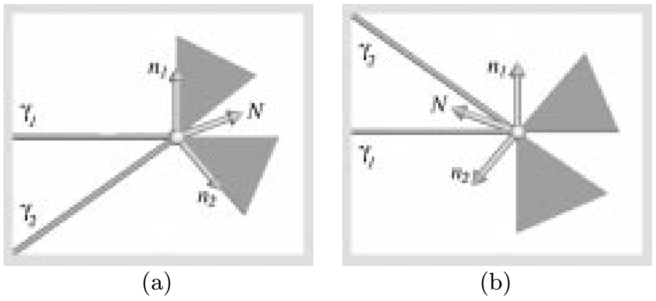

Figure 10: The choice of the normal when on an edge requires special care. Notice that in both (a) and (b) the normal N results in the correct classification of the shaded areas.

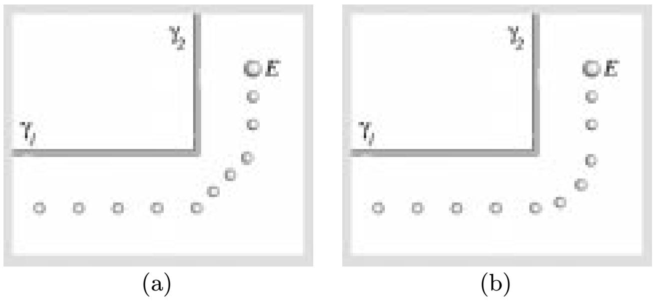

Figure 11: The choice for the new normal N results in two different side effects: (a) beveling and (b) rounding.

of the surfaces are adjacent. This means that a patchwork of dissimilar-sized surfaces can be used instead of surfaces that would have to share an entire edge. No intermediate iteration is necessary such as in [23] to solve special cases. However, there are pathological cases in which the algorithm will not return the best result. One such case would be a model that contains surfaces smaller that the distance the end-effector can travel in a single cycle. Transitioning would need to take place across these surfaces instead of onto them so the algorithm would fall behind. However, even under these circumstances the algorithm is designed to correct itself within a small number of cycles. Furthermore, since the algorithm runs at high cycle rates the erroneous results occur only briefly in time, most likely passing without notice.

5 Intermediate Representation

To help evaluate the quality of the direct parametric tracing method, we also implemented an intermediate geometry representation to use in the haptic loop. The intermediate representation chosen is a mesh of points and associated normals calculated by refinement techniques [4, 20]. The haptic loop determines arm penetration depth by projecting the arm location onto the closest mesh point's normal. The point normal is used to determine the direction of the restoring force. This model is equivalent to a set of disconnected planar facets whose edges are determined by the Voronoi regions of the point set. In our tests, we used a mesh of 12 by 12 points and updated the mesh at 10Hz. This update rate is comparable to that used in other haptic work [1][14], and the representation allows for curvature approximation, while simple intermediate planar representation do not.

6 Results

We tested the quality of the direct parametric tracing in two ways. Using the Sarcos arm, we traced a number of virtual models with both the direct parametric tracing method and the point mesh intermediate representation (Figure 12). In addition, simulations of a surface tracing were run and compared to optimal global methods.

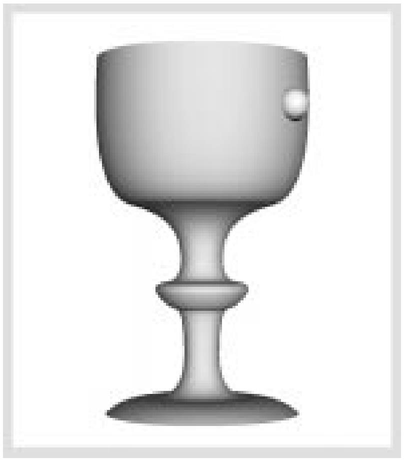

Figure 12: A goblet is traced by the user (represented as a small sphere). The goblet model consists of three parametric surfaces.

6.1 Experimental Results

During a tracing, one measure of the quality of a haptic rendering is the amount of arm penetration into the surface. For the same wall model, less penetration indicates better surface normal and penetration depth calculation.

We tested the direct haptic rendering method on several models. The models filled the usable workspace, about $1m^3$, of the Sarcos arm. The force servo loop ran at 330Hz for the mesh method and at 250Hz for the multiple surface goblet using the DPT method.

Model	Method	Sample Points	Average Depth (cm)
Flat	mesh	832	1.852
	DPT	1450	0.837
Bumpy	mesh	699	1.515
	DPT	2030	0.661
Cylinder	mesh	2541	1.787
	DPT	694	0.629
Goblet	DPT	1125	0.336

Table 1: Average penetration depth for mesh vs. direct parametric tracing of different models. Sample points is the number of distinct contact evaluations.

Table 1 shows that the average penetration depth for the parametric method was 1/3 to 1/2 the penetration depth of the point mesh method. There is a non-intuitive relationship in the results between penetration depth and model complexity, with smaller penetration depths produced by the more complex models. During the testing process, the person using the Sarcos arm tested the limits of the force response by punching and pushing as hard as possible against the simpler virtual models, skewing the collected penetration depths.

6.2 Simulation Results

In order to separate out error introduced by limitations in the Sarcos Arm and in the wall model, we also ran a number of simulated tracings on a bumpy surface. The direct parametric tracing method was compared to a hybrid symbolic/numeric solver [8] for global closest point on surface. The tracing path was generated by creating a non-isoparametric offset curve from the surface and evaluating the curve at fixed parametric steps. Figure 13 shows the difference in penetration depth found using the direct parametric tracing method with that found using the optimal solution.

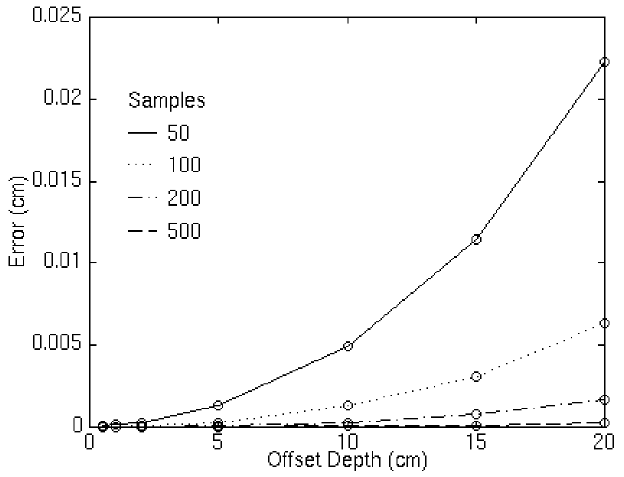

Figure 13: Average penetration depth error vs. trace curve offset depth. Each line represents a different sampling of the trace curve, with a higher sampling implying a smaller arm movement.

Note that the method was tested under a wide range of conditions. The largest offset curve depth corresponds to a tracking distance of $20cm$ and the largest step size (50 samples) corresponds to a Euclidean movement of roughly $4cm$ 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 [7], 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 2).

The Euclidean distance error (Table 2) shows that under the best conditions we measured, the parametric tracing was capable of resolving the closest point on the surface to within $0.0143cm$. This implies that the $4m^2$ surface would have to be tessellated into roughly a 7,000 by 7,000 mesh in order to maintain the same resolution of tracing. Similarly, for a 2 second trace across the surface, an intermediate planar representation would have to be updated at 3500Hz.

	Depth (cm)		
Error Metric	0.5	1.0	2.0
Penetration(cm)	0.0000	0.0000	0.0000
Normal (degrees)	0.0120	0.0145	0.0209
Parametric (percent)	0.0069	0.0080	0.0104
Euclidean (cm)	0.0143	0.0160	0.0200

Table 2: Error values as compared to an optimal solution when tracing at a step size of 3mm at three different depths beneath the surface.

7 Discussion

Intermediate surface representations have been used because of the difficulty in tracking the closest point on a sculptured surface. Direct parametric tracing allows fast tracking of an approximate closest point on a sculptured surface. 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 environments. In addition, some of the complications of using an intermediate representation, such as the force discontinuity artifacts mentioned in [14], do not appear in direct haptic rendering.

8 Future Work

Our goal is to create a haptic environment where complex models can be manipulated intuitively. In support of this goal, we need to add several capabilities to this system:

- Trimmed surfaces are common in realistic models. Transitioning over trimmed edges adds complexity to the transition phase.
- Moving surfaces are necessary to allow interesting model manipulation.
- Collision contact and response are difficult to compute, yet low latency methods need to be developed for realistic force response.
- Kinematics and dynamics add a sense of realism to a design environment. Again, how to deal with complicated global phenomena with the severe latency requirements of haptic systems is in need of study.

9 Conclusion

We have demonstrated a powerful new method for haptic rendering of sculptured models, the direct parametric tracing method. The direct parametric tracing method tracks the closest point on a surface at rates suitable for inclusion in a haptic controller, and provides excellent results on sculptured models when compared to simple intermediate representations. Tracking the closest point on multiple surfaces simplifies transitions between the surfaces and reduces artifacts during surface tracing. In addition, we hope the introduced method of comparing our algorithm against an optimal method provides a useful basis of comparison for future work on closest point tracking algorithms.

10 Acknowledgments

The authors would like to thank John Hollerbach, Don Nelson, Rod Freier, and Ali Nahvi for their help in operating and setting up the Sarcos Arm and its various control systems, as well as the networking software. We thank Bill Thompson for urging us to look more closely at direct tracking methods and Peter Shirley for helpful suggestions. Thanks also go to the students and staff of the *Alpha_1* project, within which this work was developed. Support for this research was provided by NSF Grant MIP-9420352, by DARPA grant F33615-96-C-5621, and by the NSF and DARPA Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219).

References

[1] Adachi, Y., Kumano, T., and Ogino, K., "Intermediate Representation For Stiff Virtual Objects," in *Proc. Virtual Reality Annual Intl. Symp.,* Research Triangle Park, NC, pp. 203-210, March 11-15, 1995.

[2] Baraff, David, "Curved Surfaces And Coherence For Non-penetrating Rigid Body Simulation," in *Proc. SIGGRAPH 90,* Dalls, pp. 19-28, Auguest 6-10, 1990.

[3] Barnhill, R.E., and Kersey, S.N., "A Marching Method For Parametric Surface/Surface Intersection," in *Computer Aided Geometric Design* 7 (1990), pp. 257-280.

[4] 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.

[5] Cohen, E., and Schumaker, L., "Rates Of Convergence Of Control Polygons," in *Computer Aided Geometric Design* 2 (1985), pp. 229-235.

[6] 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.

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

[8] Elber, G., *Free Form Surface Analysis Using A Hybrid Of Symbolic And Numeric Computation,* Ph.D. Thesis, University of Utah, Computer Science Department, December, 1992.

[9] 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 Manipulation Of Mechanical CAD Designs," *Submitted for publication.*

[10] Hollerbach, J.M., Cohen, E.C., Thompson, W.B., and Jacobsen, S.C., "Rapid Virtual Prototyping Of Mechanical Assemblies," *NSF Design and Manufacturing Grantees Conference,* Albuquerque, Jan. 3-5, 1996.

[11] Hoschek, Josef and Lasser, Dieter, *Fundamentals of Computer Aided Geometric Design,* Wellesley, Massachusetts, A K Peters, 1993.

[12] Jacobsen, S.C., Smith, F.M., Iversen, E.K., and Backman, D.K., "High Performance, High Dexterity, Force Reflective Teleoperator," in *Proc. 38th Conf. Remote Systems Technology,* Washington, D.C., pp. 180-185, November, 1990.

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

[14] 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 96,* New Orleans, pp. 447-452, August. 4-9, 1996.

[15] 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,* Snowbird, Utah, pp. 235-243, 1990.

[16] Mortenson, Micheal, *Geometric Modeling,* New York, John Wiley & Sons, 1985.

[17] Plass, Michael and Stone, Maureen, "Curve-Fitting With Piecewise Parametric Cubics," in *Proc. SIGGRAPH 83,* Detroit, pp. 229-236, July 25-29,1983.

[18] Riesenfeld, R., "Modeling With Nurbs Curves And Surfaces," in *Fundamental Developments of Computer Aided Geometric Design,* L. Piegl (ed.), Academic Press.

[19] 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

[20] 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 Modelling," in *Proc. National Computer Graphics Association,* 1981.

[21] Salisbury, J., Brock, D., Massie, T., Swarup, N., and Zilles, C., "Haptic Rendering: Programming Touch Interaction With Virtual Objects," in *Proc. Symposium on Interactive 3D Graphics,* Monterey, CA, pp. 123-130, 1995.

[22] Snyder, John, "An Interactive Tool For Placing Curved Surfaces Without Interpenetration," in *Proc. SIGGRAPH 95,* Los Angeles, pp. 209-218, August. 6-11, 1995.

[23] 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.

A B-spline 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,

$$\gamma(u) = \sum_{i=0}^{n} P_i B_{i,k}(u). \tag{9}$$

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,

$$B_{i,1}(u) = \begin{cases} 1, & u_i \le u < u_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

and for $k > 1$,

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

There is more than one way to evaluate a B-spline. The weighted combination of control points can be computed through evaluation of the basis functions as in Eq. 9, or curve refinement may be used [4, 20]. Inserting $k-1$ knots into $\mathbf{u}$ with the value u^* will create a new B-spline with a control point P_{i^*} that is the value of $\gamma(u^*)$. This point is called an evaluation point and results because only one basis function, $B_{i^*,k}$, has a value at time u^*. The value of i^* is one less than the index of the first new knot of value u^*. The curve refinement method is computationally efficient.

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

$$S(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j} B_{j,k_v}(v) N_{i,k_u}(u), \tag{10}$$

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.

B Curve velocity at an evaluation point

In this appendix a compact, computationally efficient equation for the velocity of a curve at an evaluation point is derived. The velocity of a B-spline (when it exists) is given by,

$$\gamma'(u) = \sum_{i=0}^{n} P_i B'_{i,k}(u), \tag{11}$$

where

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

Expanding Eq. 11 with the definition of $B'_{i,k}(u)$ yields,

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

Evaluation of this curve would be far from computationally efficient, therefore we try to eliminate one of the basis functions. By distributing the summation we get,

$$\gamma'(u) = (k-1)\left[\sum_{i=1}^{n} \frac{P_i B_{i,k-1}(u)}{u_{i+k-1} - u_i} - \sum_{j=0}^{n-1} \frac{P_j B_{j+1,k-1}(u)}{u_{j+k} - u_{j+1}}\right].$$

Notice that 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 $j = n$. 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,

$$\gamma'(u) = (k-1)\left[\sum_{i=1}^{n} \frac{P_i B_{i,k-1}(u)}{u_{i+k-1} - u_i} - \sum_{i=1}^{n} \frac{P_{i-1} B_{i,k-1}(u)}{u_{i+k-1} - u_i}\right],$$

and then combining the two summations we get,

$$\gamma'(u) = (k-1)\sum_{i=1}^{n} \frac{(P_i - P_{i-1}) B_{i,k-1}(u)}{u_{i+k-1} - u_i}. \qquad (12)$$

Consider now that $\gamma(u)$ has been refined with $k-1$ knots at index i^* with the value u^*. Then P_{i^*} is an evaluation point. In Eq. 12, only basis function $B_{i^*+1,k-1}(u)$ has a value at time u^*. Noting this we get the equation,

$$\gamma'(u^*) = \frac{(k-1)}{u_{i^*+k} - u_{i^*+1}}(P_{i^*+1} - P_{i^*}), \qquad (13)$$

for the velocity of the curve at time u^*. An important point to notice in Eq. 13 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, $\gamma'(u^*)$.