Salisbury, J.K. and Tarr, C., ``Haptic Rendering of Surfaces Defined by Implicit Functions," Proceedings of the ASME 6th Annual Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Dallas, TX, November 1997, pp. 61-68.

HAPTIC RENDERING OF SURFACES DEFINED BY IMPLICIT FUNCTIONS

Kenneth Salisbury
Christopher Tarr

MIT AI Lab and SensAble Technologies, Inc.
jks@ai.mit.edu and ctarr@sensable.com

ABSTRACT

This paper presents a method for computing the contact forces which must be imposed on a human (via a haptic interface) to evoke the sensation of touching surfaces described by implicit equations of the form $S(\mathbf{p}) = 0$. The algorithm computes in real-time the forces of interaction which would arise from a point interacting with a real surface of the given shape. It models the forces of interaction which would arise when contacting linearly elastic material with surface friction. At each instant, a tangent plane to the surface at the contact point is computed and used to track the contact point's motion and prevent it from passing though the surface. A method using cutting planes to limit the extent of these surfaces is described. Though the rendering techniques are presented in the context of implicit surfaces, they can be applied to other classes of surfaces.

NOMENCLATURE

S, $S(\mathbf{p})$	implicit function defining a surface
T	tangent plane to S
N	normal to S
$\mathbf{p}$	point in 3D space, (x, y, z)
$\mathbf{p}_0$	the haptic interface point
$\mathbf{p}_1$	the surface contact point (SCP)
$\mathbf{p}'_0$	first approximation to $p0$, on T
S_{name}	a named surface or cutting plane
$\mathbf{p}_{i,k}$	point i at time k
T_k	tangent plane at time k

1. INTRODUCTION

Haptic interaction is the process by which humans touch, explore, perceive and manipulate objects. It is the bilateral nature of this interaction, in which we both receive stimulation from and do work upon the environment, that makes the haptic modality unique among our senses. As a relatively new area of study in human-computer interaction it offers types of perception and degrees of expressiveness previously unavailable.

Recent advancements in display technology have made it possible to produce compelling computer-generated haptic images that observers can feel and manipulate. In (Salisbury, et al., 95) we described our preliminary efforts in enabling people to feel computer simulated objects via use of the PHANToM haptic interface. An enormous amount of new research activity has begun to focus on how to better generate haptic illusions and how to utilize this new interaction modality (Salisbury & Srinivasan, 96).

In order for "observers" to feel simulated objects, three components are necessary: 1) a means for tracking motion of the observer's interaction point and imposing haptic stimuli (force and other haptic effects) on the observer, 2) a model of the simulated object, and 3) a "haptic rendering algorithm" to compute the appropriate stimulation from consideration of observer motion and the object model.

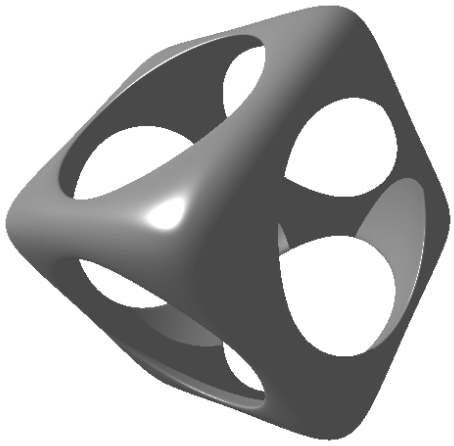

Figure 1: A whiffle cube that can be felt. The implicit function for this surface is found by subtracting a sphere from a cuboid with the resulting equation: $S(x, y, z) = \left(x^8 + y^8 + z^8\right)^8 + \left(x^2 + y^2 + z^2 - 0.44\right)^{-8} - 1$.

Previous haptic rendering algorithms fall into two categories, stateless potential field algorithms and surface contact point algorithms. The first category encompasses algorithms which simulate volumes that generate haptic interaction forces based only on the current position of the user interaction point (i.e. the tip

of the PHANToM). These volumes can be described as density fields or potential fields and only approximate interacting with a surface by generating forces toward the surface from any location internal to the volume. The potential method discussed in by Massie & Salisbury (1994) and several voxel based haptic algorithms (Avila 1996) fall into this category. Unfortunately, surfaces with small features relative to the distance traveled between interaction points can be lost or passed through without notice, since past history is not taken into account when generating forces and determining the type of contact.

To solve this problem, a second class of algorithms called surface contact point algorithms was created. These algorithms are strictly intended to model the surface geometry of an object as opposed to a volume or density properties. These algorithms take into account the path traveled between each discrete location of the interaction point. They can determine the direction from which a surface was penetrated and whether or not the interaction point should be allowed to pass through or be forced toward an appropriate location on the surface. The first algorithm taking advantage of this approach was described by Zilles (Zilles, 1995; Zilles & Salisbury, 1995). His algorithm permits rendering the important class of polyhedral modeled shapes and prevents the surface contact point from passing through the object even in the case of small features. Ruspini et al. (1996) also addresses this problem in the rendering of polyhedral shapes. Zilles (1995), Morgenbesser and Srinivasan (1995), and Ruspini also investigated the use of "force shading" techniques to alter apparent surface normals to give the sensation of curvature to the polyhedon's facets (much in the spirit of Phong shading). Techniques for rendering intrinsically curved object models have been addressed by Thompson et al. (1997) for NURBS surfaces and in this paper for surfaces described by implicit equations.

2. SURFACE DESCRIPTION BY IMPLICIT FUNCTIONS

Implicit functions take the form $S(\mathbf{p}) = 0$. Points $\mathbf{p}$ which satisfy this equation are defined to be *on* the surface S and define the boundary of the enclosed volume. For example, the surface of a sphere is represented by the equation $x^2 + y^2 + z^2 - 1 = 0$. The implicit function representation provides a compact, analytic, continuous surface that can represent many shapes, provide well defined surface normals and derivatives, and affords simple collision detection tests.

Although implicit surfaces are commonly used to represent simple geometric shapes such as quadric and super-quadric surfaces, there is an abundance of exciting work in the 3D graphics field using implicit functions to model complex surfaces. Blinn (1982) used them for molecular modeling to provide artistic variety and scientific accuracy. Recent efforts have developed to reconstruct surfaces in implicit form from scattered points (Irfan 1992; Savchenko 1995). Others have found techniques to use them for implicit solid modeling (ISM) for engineering design (Storti, et al. 1992).

Additionally, it has long been recognized that useful and logical shape description can be accomplished through combination of primitive shapes. Such constructive solid geometry (CSG) techniques perform regularized Boolean set operations between shapes. These techniques can equally be applied to implicitly defined shapes. One of the simplest versions of this approach, developed by Ricci (1973), permits pseudo-Boolean operations to be performed between shapes with the useful side effect of providing controllable blending. This method actually combines the function equations to form a new composite equation that represents the new surface after performing the boolean operation. The object shown in Figure 1, known as a whiffle cube, can be modeled with an implicit equation derived using this technique. The shape is found by subtracting a sphere from a cuboid with a blending exponent of 8.

We have found that CSG can also be performed without explicit derivation of the equations for the combined objects by real-time consideration of the individual objects and between objects of mixed polyhedral and implicit representation. This has been implemented in the GHOST Toolkit described in Section 7 and permits haptic perception of combined, even independently moving, objects.

In the following sections we discuss how to compute, in real-time, the forces necessary to simulate feeling the rich class of shapes representable by implicit equations.

3. PROPERTIES OF THE IMPLICIT REPRESENTATION

Property 1: Inside-outside function for $S(\mathbf{p})$

> If a S represents an orientable surface then we may detect whether an arbitrary point $\mathbf{p}$ is inside, on, or outside the volume enclosed by the surface by checking the value of $S(\mathbf{p})$. If $S(\mathbf{p}) < 0$, $\mathbf{p}$ is inside the volume enclosed by surface S, if $S(\mathbf{p}) = 0$, $\mathbf{p}$ is on the surface S, otherwise $\mathbf{p}$ is outside the volume enclosed by surface S. Further the magnitude of $S(\mathbf{p})$ is related to the distance from S. It also is interesting to note that S can be inverted by simply reversing the sense of the above tests, permitting one to feel the inside of surfaces as easily as the outside.

Property 2: The gradient of $S(\mathbf{p})$

> The gradient of $S(\mathbf{p})$, $\nabla S(\mathbf{p})$, is defined as $\nabla S(\mathbf{p}) = dS(\mathbf{p})/d\mathbf{p} = (\partial S(\mathbf{p})/\partial x, \partial S(\mathbf{p})/\partial y, \partial S(\mathbf{p})/\partial z)$. It is a three-element vector which points in the direction which $\mathbf{p}$ should change to maximize change in the (scalar) value of $S(\mathbf{p})$ for incremental changes in $\mathbf{p}$. For points $\mathbf{p}$ on S its components are proportional to the outward pointing surface normal at $\mathbf{p}$.

4. RENDERING A SURFACE $S(\mathbf{p}) = 0$, FRICTIONLESS CASE

The goal of the following algorithms is to compute the forces which would be exerted on a rigid probe point interacting with an object surface of which is given by $S(\mathbf{p}) = 0$. In order to prevent penetration of the surface S we will solve for and track a surface contact point $\mathbf{p}_1$. A tangent plane T will be associated with each successive value of the point $\mathbf{p}_1$ and used to preserve the state of contact in a physically meaningful way.

Finding a point on S nearest to given point p

We first present an algorithm that, given a seed point $\mathbf{p}_{\text{seed-point}}$ in the neighborhood of S, will find the nearest point on S, $\mathbf{p}_{\text{surface-point}}$. This algorithm will be used in two ways as described in later subsections and is derived in detail in Appendix 1.

The nearest point could be found by using Lagrangian multipliers to find $\mathbf{p}_{\text{surface-point}}$ that minimizes $||\mathbf{p}_{\text{seed-point}} - \mathbf{p}_{\text{surface-point}}||$ subject to the constraint $S(\mathbf{p}_{\text{surface-point}}) = 0$. For non-trivial (i.e. non-planar) S this would involve finding the roots (of possibly large) equations and inspecting candidate solutions for the minimal result. Instead, we exploit the fact that $\mathbf{p}_{\text{seed-point}}$ is in the neighborhood of S and that the gradient of $S(\mathbf{p}_{\text{seed-point}})$ (∇S) points in a direction that is a good approximation to the direction toward the closest point on S. A prescribed step from the seed point is taken in the gradient direction. The size of the step is a function of the $S\,\nabla S$. This process is repeated until the step size becomes sufficiently small. Terminating when the step size becomes small does not directly indicate we have reached our goal, however it works well in the neighborhood of S. Checking for $S(\mathbf{p}_{\text{surface-point}}) = 0$ would be more exact but is unnecessary here. Thus, given a seed point $\mathbf{p}_{\text{seed-point}}$, we perform the following computation:

Algorithm 1

$\mathbf{p} = \mathbf{p}_{\text{seed-point}}$
do
  $\delta\mathbf{p} = -\frac{S(\mathbf{p})\nabla S(\mathbf{p})}{\nabla S(\mathbf{p})\cdot\nabla S(\mathbf{p})}$
  $\mathbf{p} = \mathbf{p} + \delta\mathbf{p}$
until $(||\delta\mathbf{p}|| < \epsilon)$
$\mathbf{p}_{\text{surface-point}} = \mathbf{p}$

The resulting $\mathbf{p}_{\text{surface-point}}$ has been found to be good approximation of the nearest point on the surface to the initial seed value of $\mathbf{p}_{\text{seed-point}}$ when the seed point is near S and ϵ is sufficiently small. Typically, in our experiments only 1 to 3 iterations of the above algorithm are required.

Finding a surface contact point on S at first contact

The haptic interface point, $\mathbf{p}_0$, is defined to have the coordinates of the tip of the haptic interface (e.g. the user's fingertip, stylus point, the tip of the PHANToM haptic interface, etc.) when this point is outside the surface, $S(\mathbf{p}_0) > 0$ and no interaction force is exerted on the user. The first time $\mathbf{p}_0$ penetrates the surface (i.e. $S(\mathbf{p}_0)$ goes negative), we declare the state of S to be TOUCHED. Using **Algorithm 1** with a seed value of $\mathbf{p}_{\text{seed-point}} = \mathbf{p}_0$, we compute the first *surface contact point*, $\mathbf{p}_1 = \mathbf{p}_{\text{surface-point}}$. We then compute the force, $F = k*(\mathbf{p}_1 - \mathbf{p}_0)$, to be applied to the user. To prepare for the next tick, we define the outward pointing surface normal N to be a unit vector in the direction of the gradient of $S(\mathbf{p}_1)$ (i.e. $N = \nabla\hat{S}(\mathbf{p}_1)$) and define the associated tangent plane T for this surface contact point. Figure 2 illustrates the geometry.

Finding subsequent surface contact points on S

During subsequent ticks (times steps) the haptic interaction point $\mathbf{p}_0$ will penetrate further into (or even beyond) the volume enclosed by S. Thus, following the gradient from $\mathbf{p}_0$ directly to the nearest point on S as above will not be appropriate. Instead, we use the tangent plane T computed above to constrain successive values of $\mathbf{p}_1$ from moving inside S. T will be updated at each tick as the surface contact point $\mathbf{p}_1$ moves across S. In an important way, tracking T as it glides over S permits us to capture the state of the contact interaction in a physically meaningful sense. We can imagine the plane T being pulled through space by the point $\mathbf{p}_0$ to which T is attached by an elastic band. When $\mathbf{p}_0$ passes through the surface S, the plane is prevented from passing through. The plane's contact point with the surface defines $\mathbf{p}_1$ and the stretching of the elastic band gives rise to force F. In the absence of friction, this plane moves to locally minimize the energy stored in the elastic connection.

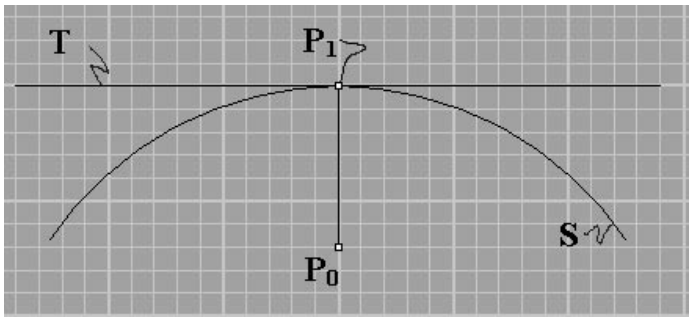

Figure 2: Illustration of surface contact point $\mathbf{p}_1$ found via **Algorithm 1** after first entry into the volume S by $\mathbf{p}_0$.

Thus, during subsequent ticks (until contact is broken) we first drop a perpendicular from $\mathbf{p}_0$ to T to define point $\mathbf{p}_0'$, a first approximation to the surface contact point. We then use the iterative procedure of **Algorithm 1**, with $\mathbf{p}_0'$ as the seed value, to find $\mathbf{p}_1$ on S as the closest point to $\mathbf{p}_0'$. This is a key point in the algorithm in that it uses the state of T to permit us to rapidly find the nearly correct surface contact point, and then refine it by iteration over a short distance. At the end of the iteration a new normal, N, and new tangent plane, T are computed at $\mathbf{p}_1$, readying us for the next tick. This process of dropping a perpendicular from each new value of $\mathbf{p}_0$ to the previous T and then using **Algorithm 1** to find a new $\mathbf{p}_1$ on S and new T continues so long as contact is maintained. If, after computing a new T, we find the next $\mathbf{p}_0$ to be outside of it, then contact with S is broken (S is no longer in the TOUCHED state), no interaction force is imposed on the user, and we revert to looking for the next incursion of $\mathbf{p}_0$ inside S. See Figure 3.

5. RENDERING A SURFACE $S(\mathbf{p}) = 0$, FRICTION CASE.

A simple modification of the above algorithm permits modeling of surface friction on S. When we drop the perpendicular from a new $\mathbf{p}_0$ to the last T to find $\mathbf{p}_0'$ we are implicitly assuming that there is no friction. When friction is present the value of $\mathbf{p}_0'$ will require modification; depending on friction and force directions, it may stick in place or may move to the edge of the *friction cone* described below. Thus, we first construct the cone shown in Figure 4 with it's apex at the new $\mathbf{p}_0$ and central axis

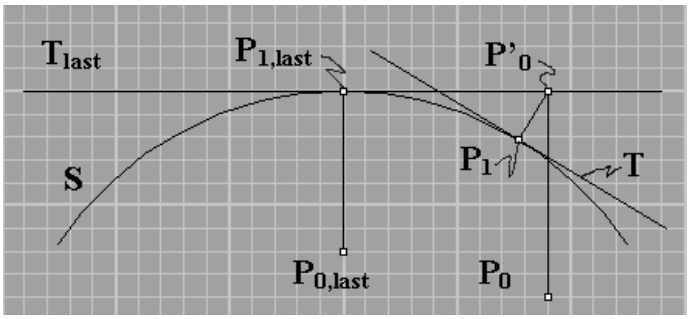

Figure 3: Finding subsequent surface contact point, $\mathbf{p}_1$, by dropping perpendicular to last tangent plane to find $\mathbf{p}'_0$. $\mathbf{p}'_0$ is then used to seed **Algorithm 1** to find $\mathbf{p}_1$ on S.

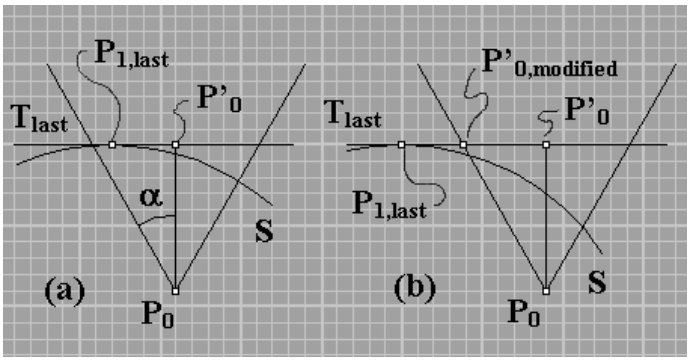

Figure 4: Effect of friction modification causing surface contact point to (a) stick or (b) slip.

normal to and passing through the last T at $\mathbf{p}'_0$. This is known as the *friction cone*. The angle α is function of the coefficient of friction μ between the surface S and our surface contact point ($\alpha = \arctan(\mu)$).

Then, if the old surface contact point $\mathbf{p}_{1,\text{last}}$ is inside the circle found by intersecting T with a friction cone, it sticks in place and becomes the current $\mathbf{p}_1$, and T_{last} becomes the current T. However, if $\mathbf{p}_{1,\text{last}}$ is outside this circle on the tangent plane, we set $\mathbf{p}'_0$ to be the point on the circle closest to $\mathbf{p}_{1,\text{last}}$ (it slides along T to the edge of the friction cone). From here we find the actual point on S by using the modified $\mathbf{p}'_0$ as the seed for **Algorithm 1** and follow the gradient to S to find the new $\mathbf{p}_1$ and T at $\mathbf{p}_1$.

While this computation is an approximation to finding the actual point $\mathbf{p}_1$ on S, it avoids the complex computation of finding the intersection of S with the friction cone. In practice the algorithm is run at around 1 kHz and the errors are not palpably apparent. It should also be noted that if S is itself a plane (such as the cutting planes used in Section 6) then the iterative steps of **Algorithm 1** are unnecessary for both the frictionless and friction case, and we can directly set $\mathbf{p}_1 = \mathbf{p}'_0$. Further it should be noted that different values of μ are used depending on whether

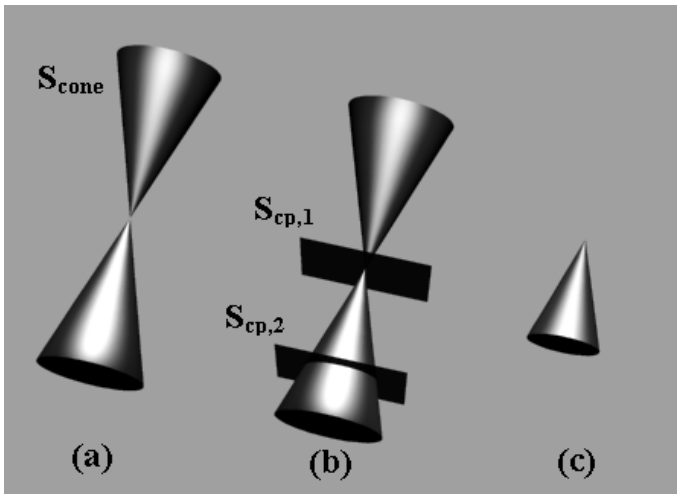

Figure 5: Truncating an infinite cone with cutting planes.

$\mathbf{p}_1$ is currently moving or stationary on S, corresponding to the cases of dynamic or static friction, respectively.

6. TRUNCATING OBJECTS WITH CUTTING PLANES

In order to render more familiar objects common in 3D graphics, such as cones and cylinders, it is necessary to truncate, or cut away, parts of these objects. This could be done by directly modifying the implicit equations by the Boolean operations described by Ricci (1973) or others, or by performing the truncation in realtime using the techniques described below. For example, the equation for a cone, $x^2 + y^2 - z^2 = 0$, represents the surface of infinite extent shown in Figure 5(a). In order to use this equation to represent the familiar cone so often used in 3D graphics, Figure 5(c), we must eliminate the surface above the top plane and below the lower cutting plane intersecting the cone in Figure 5(b). In addition, the exposed bottom circular disk of the cone must be covered to prevent the interaction point from passing into the cone.

This use of cutting planes is actually an intersection of the cone object with two half-spaces defined by the two cutting planes. The intersection or union of two volumes is considered a standard Boolean operation, more commonly referred to as constructive solid geometry or CSG. Although the proper set of CSG operations on implicit surfaces is beyond the scope of this paper, we discuss here the specific problem of intersecting half-spaces with an object described by an implicit equation to allow the creation of finite cones and cylinders. Refer to Section 7 for information on current work in this area.

Thus, we seek a method for taking the intersection of the volume defined by $S_{obj}(\mathbf{p}) = 0$ (such as a cone) and the half-spaces defined by the cutting planes, $S_{cp,i}(\mathbf{p}) = 0$. Cutting planes are represented in the same implicit form as other objects. The i-th cutting plane is given simply as $S_{cp,i}(\mathbf{p}) = (\mathbf{p} - X_i)N_i$, where X_i is a an arbitrary point on the plane and N_i is the outward pointing normal to the plane. We use the same inside-

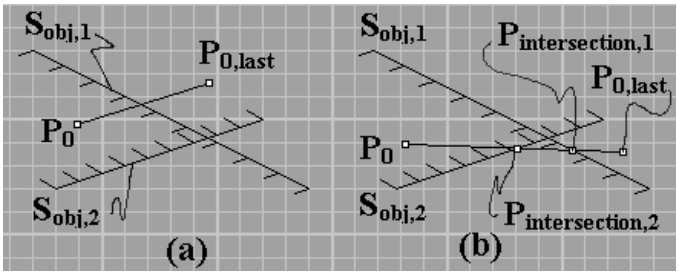

Figure 6: Illustration of first contact through (a) one, and (b) two surfaces.

outside function as in Property 1 of Section 3 to determine if $\mathbf{p}$ is inside or outside the half-space defined by $S_{cp,i}(\mathbf{p})$.

For the present description we simplify the problem by restricting cutting planes to be parallel or perpendicular to each other. The reason for this simplifying assumption is twofold. First, cutting planes were introduced to make familiar geometric shapes with implicit equations, such as cones and cylinders, which only require parallel cutting planes. Second, concave intersections of multiple cutting planes generate edges and corners similar to faceted surfaces. Faceted surfaces, though addressed by other rendering techniques developed by us and others, are beyond the scope of the current presentation.

Formally, portions of surface S_{obj} occupying the volume outside the half-space defined by the cutting plane $S_{cp,i}$ are eliminated. Accordingly, the top cutting plane of Figure 5(b) has the surface normal $(0,-1,0)$ and the lower cutting plane has the surface normal $(0,1,0)$. The following section details how we modify the original rendering method to deal with the desired intersection of object geometry S_{obj}, and cutting planes, *Scpi*.

Resulting Object In/Out Function

The volume resulting from the intersection of the volumes defined by the cutting planes with the volume of S_{obj} has an interior defined by the set of points $\mathbf{p}$ which simultaneously satisfy the set of inequalities $(S_{obj}(\mathbf{p}) < 0, S_{cp,1}(\mathbf{p}) < 0, \cdots S_{cp,\mathrm{N}}(\mathbf{p}) < 0)$, given N cutting planes. Thus, the interaction point is interior to the resulting surface when inside S_{obj} and inside all cutting planes $S_{cp,i}$. Otherwise, the interaction point is exterior to the resultant volume.

Rendering with Cutting Planes

Initial contact is detected, as in the original method, when the haptic interaction point $\mathbf{p}_0$ first passes from the exterior to interior of the surface. When first entering the interior volume from outside, the interaction point must pass through one, or more than one, of the surfaces S_{obj} and $S_{cp,i}$ since the last tick. When contact is established, one surface is determined to be the TOUCHED surface and is used exactly as in the previous method, to calculate and track $\mathbf{p}_1$. Contact is maintained or relinquished as before with one additional caveat. Once contact has been established it is now possible to move from one surface to another without releasing contact. This occurs when the line segment connecting $\mathbf{p}_{1,\mathrm{last}}$ to the current $\mathbf{p}_1$ intersects a surface other than the TOUCHED surface. The subsections below outline these steps in greater detail.

1st Contact: For the case when *one* surface is crossed by $\mathbf{p}_0$ as it passes into the resultant volume, that surface, S_{obj} or $S_{cp,i}$, becomes the active TOUCHED surface and is subsequently used to calculate and track $\mathbf{p}_1$ as in the original rendering method. See Figure 6(a).

When more than one surface, S_{obj} and/or $S_{cp,i}$, is crossed when $\mathbf{p}_0$ first passes into the resultant volume, as shown in Figure 6(b) one of the crossed surfaces must be determined to be the TOUCHED surface and subsequently used to calculate and track $\mathbf{p}_1$. To determine which surface becomes the TOUCHED surface, it is necessary to find the *last* surface crossed on the line from $\mathbf{p}_{0,last}$ to the current $\mathbf{p}_0$. This is done by finding the intersection point for each crossed surface, $\mathbf{p}_{\mathrm{intersection},i}$, with the line from $\mathbf{p}_{0,last}$ to the current $\mathbf{p}_0$. The surface associated with the intersection point nearest $\mathbf{p}_0$ becomes the TOUCHED surface and we proceed as in the original rendering method to find and track $\mathbf{p}_1$ on the surface. Finding the intersection of this line with a cutting plane $S_{cp,i}$ is done quite easily in closed form. Finding the intersection point with S_{obj} is not easily done in closed form. Instead, a binary search on the line from $\mathbf{p}_{0,\mathrm{last}}$ to the current $\mathbf{p}_0$ is performed to locate the intersection to a specified accuracy. This is shown below:

```
insidePoint = P0
outsidePoint = p1,last
midPoint = (p1 + p1,last)/2
    while (|S(midPoint)| > ε) do
        if (S(midPoint) < 0)
            midPoint = (outsidePoint + midPoint)/2
        else
            midPoint = (midPoint + insidePoint)/2
intersectionPoint = midPoint.
```

Changing Contact from One Surface to Another: Once contact has been established and one surface is determined to be the TOUCHED surface, $\mathbf{p}_1$ is calculated just as in the original rendering algorithm. After $\mathbf{p}_1$ is calculated, the line segment connecting $\mathbf{p}_{1,\mathrm{last}}$ to the new $\mathbf{p}_1$ may cross a surface other than the TOUCHED surface. If this occurs, the crossed surface instantly becomes the current TOUCHED surface and $\mathbf{p}_1$ must be recalculated. Unfortunately, $\mathbf{p}_0$ may be too far from the surface of the new TOUCHED surface to use in **Algorithm 1**. Instead, the just calculated $\mathbf{p}_1$ serves as an acceptable seed point to find a first surface contact point using **Algorithm 1**.

7. APPLICATION TO THE *GHOST(tm)* TOOLKIT

The above algorithms have been used in the SensAble Technologies Inc. Ghost Software Toolkit (SensAble, 96) to render shapes such as ellipses, toroids and cones. It has also been used to render other more complex shapes such as the whiffle cube shown in Figure 1. In addition, GHOST supports implicit surfaces within a mixed environment of representational types that currently includes implicit and polygonal representations.

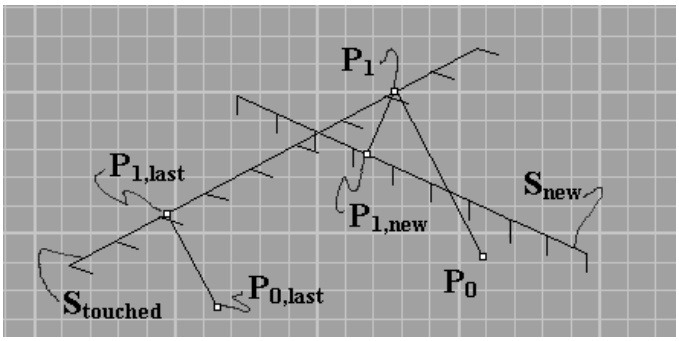

Figure 7: Changing contact from one surface to another.

A more general technique for taking the union of objects is intrinsically supported and is handled in realtime. This enables overlapping objects in the haptic environment to be automatically combined. The algorithm is fast enough to support calculating these unions even as shapes move dynamically in the scene at rendering rates up to 1kHz. By the time of printing, intersection and difference operations on all objects should be supported under GHOST SDK as well. The API is also open to allow developers to create their own primitives, such as custom implicit surfaces.

8. CONCLUSIONS

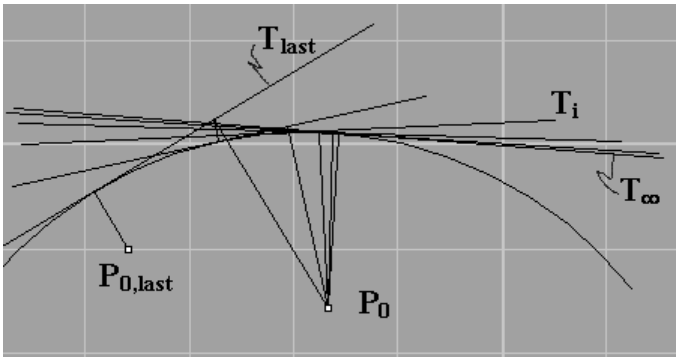

Figure 8: Typical convergent solution for $\mathbf{p}_1$. With $\mathbf{p}_0$ held stationary successive tangent planes (and therefor values of $\mathbf{p}_1$) can be seen to rapidly converge to T_∞.

After running the algorithm with a number of surfaces under different conditions we were pleased to note that most surfaces required less than than 20 microseconds per tick. For example, performance tests found that the sphere equation took approximately 10microseconds to haptically render on a Pentium 166Mhz machine. In comparison, the wifflecube took approximately 15 microseconds on the same machine. These results indicate that the algorithm is well suited for the 1-2kHz simulation rates needed for haptics.

Most surface curvatures required only 1 or 2 iterations of the surface finding algorithm to get sufficiently close. Only in high curvature concavities did these numbers change. By negating an ellipsoid equation we defined an interior volume of dimensions $40 \times 40 \times 0.4$ millimeters to test stability at sharp concavities. Though the surface finding algorithm never iterated more than 10 times, another problem became apparent. Small concave features with high curvature can cause a stable limit cycle in the surface contact point solution such that for a fixed $\mathbf{p}_0$ we found values of $\mathbf{p}_1$ to oscillate in the neighborhood of the correct solution. Figure 8 shows an example of a converging case and illustrates the rapid tracking nature of the algorithm; Figure 9 shows a case where the surface contact point oscillates in a stable limit cycle.

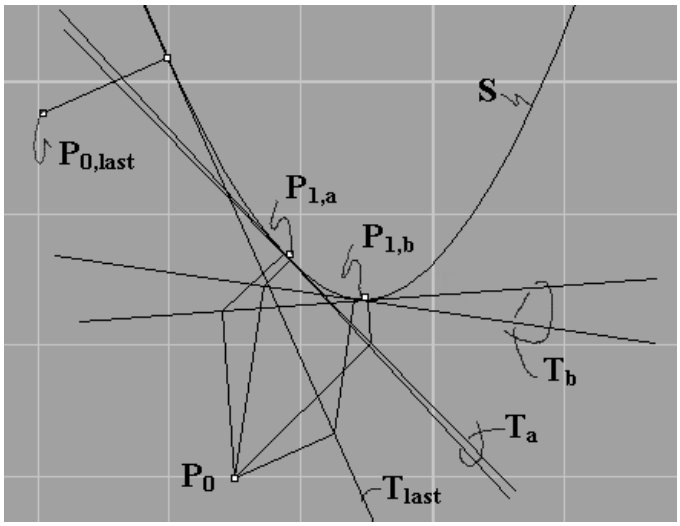

Figure 9: Example showing limit cycle in $\mathbf{p}_1$ for highly curved concave S. Despite stationary $\mathbf{p}_0$, the tangent planes can be seen to oscillate between the bundles designated T_a and T_b.

Fortunately, this problem seems to be limited to concave features with a small radius of curvature relative to the penetration depth; it can be reduced or eliminated by one of several adaptations to the algorithm.

Adaptation 1: Just before calculating $\mathbf{p}_1$ If $S(\mathbf{p}_0') < 0$ then let $\mathbf{p}_0' = (\mathbf{p}_0' + \mathbf{p}_{1,\text{last}})/2$.

Adaptation 2: Replace calculation of $\mathbf{p}_1$ and tangent plane T with the following:

maxDist $= ||\mathbf{p}_0' - \mathbf{p}_{1,\text{last}}||$
do
  $\mathbf{p}_1$ = solution to **Algorithm 1** using $\mathbf{p}_0'$ as seed
  T = plane with normal $\nabla\hat{S}(\mathbf{p}_1)$, passing through $\mathbf{p}_1$
  $\mathbf{p}_2$ = projection of interaction point onto T
  CurDist $= ||\mathbf{p}_2 - \mathbf{p}_{1,\text{last}}||$
  $\mathbf{p}_0' = (\mathbf{p}_0' + \mathbf{p}_{1,\text{last}})/2$
while (curDist > maxDist)

Both of these adaptations tend to move $\mathbf{p}_1$ closer to $\mathbf{p}_{1,\text{last}}$ so that a better tangent plane will be found and $\mathbf{p}_1$ will eventually converge to the correct solution. These algorithms have some shortcomings. The first adaptation effectively adds a small viscous force to flat surfaces. The second adaptation takes more time to compute and causes some stickiness when moving along a concave trough. Modification to deal with fine detail and extremes of curvature are currently being investigated.

9. ACKNOWLEDGMENTS

The authors would like to gratefully acknowledge SensAble Technologies, Inc. for their support of the work reported herein.

10. REFERENCES

Avila, R.S. and L.M. Sobierajski, "A Haptic Interaction Method for Volume Visualization," *Visualization '96 Proceedings,* October, 1996.

Blinn A. H. 1982. "A generalization of algebraic surface drawing." *ACM Trans. on Graphics*, 1 (July): 235-256.

Irfan A.E., S. Sclaroff and A.P. Pentland, "Physically-based Modeling for Graphics and Vision," MIT Media Lab TR184, 1992. Also appears in *Directions in Geometric Computing*, R. Martin (Editor), Information-Geometers, UK., 1993.

Mark, W. R., Randolph, S. C., Finch, M., Van Verth, J. M., & Taylor, R. M. I. "Adding Force Feedback to Graphics Systems: Issues and Solutions." Computer Graphics Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 447-452.

Massie, Thomas H. and K. Salisbury, "The PHANToM Haptic Interface: A Device for Probing Virtual Objects," Proceedings of the ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, Chicago, IL, November 1994.

Morgenbesser H.B. and Srinivasan M.A., "Force shading for haptic shape perception," Proc. ASME Winter Annual Meeting, 1996 (in press).

Ricci, A. 1973. "A constructive geometry for computer graphics," *The Computer Journal*, v. 16, no. 2, 157-160.

Ruspini, D., Kolarov, K. and Khatib, O. "Robust Haptic Display of Graphical Environments," in J.K. Salisbury and M.A. Srinivasan, editors, Proc. of the First PHANToM Users Group Workshop, M.I.T. Artificial Intelligence Laboratory Technical Report AITR-1596, 1996.

Salisbury, Kenneth, D. Brock, T. Massie, N. Swarup and C. Zilles, "Haptic Rendering: Programming Touch Interaction with Virtual Objects," Proceedings of 1995 ACM Symposium on Interactive 3D Graphics, Monterey, California, April 1995.

Salisbury, J.K and Srinivasan, M.A. (Eds), "Proceedings of the First PHANToM Users Group Workshop," MIT AI Lab Technical Report No. 1596 and RLE Technical Report No. 612, Dec 1996.

Savchenko V.V., Pasko A.A., Okunev O.G., Kunii T.L. "Function representation of solids reconstructed from scattered surface points and contours," Computer Graphics Forum, vol.14, No.4, 1995, pp.181-188.

SensAble Technologies, Inc. Ghost Toolkit preliminary product literature, August 1996.

Storti, D. W., Ganter, M. A., and Nevrinceanu, C., "A tutorial on implicit solid modeling." *The Mathematica Journal* 2, 3 (1992), 70–78.

Thompson II, T.V., Johnson, D.E., and Cohen, E.C., "Direct Haptic Rendering Of Sculptured Models," Proc. Symposium on Interactive 3D Graphics, (Providence, RI), pp. 167-176, April 27-30, 1997.

Zilles, C, "Haptic Rendering with the Tool-Handle Haptic Interface," S.M. Thesis, MIT Dept. of ME, May 1995.

Zilles, C. and K. Salisbury, "A Constraint-Based God Object Method for Haptic Display," proceedings of IROS-95, Pittsburgh, Aug 6-9, 1995.

APPENDIX 1. DERIVIATION OF ALGORITHM 1

The purpose of **Algorithm 1** is that, given a point $\mathbf{p}$ inside or outside a volume defined by S, find the closest point $\mathbf{p}'$ on the surface. It accomplishes this by following the gradient of S at successive points from the initial seed point until it reaches the surface. The method derived below determines the step size $\delta\mathbf{p}$ along this direction which should be taken.

We define define $\delta\mathbf{p}$ so that $\mathbf{p}' = \mathbf{p} + \delta\mathbf{p}$. Noting that $S(\mathbf{p}') = 0$ because $\mathbf{p}'$ is assumed on the surface S we can write

$$0 = S(\mathbf{p}') = S(\mathbf{p} + \delta\mathbf{p}) \approx S(\mathbf{p}) + \nabla S(\mathbf{p}) \cdot \delta\mathbf{p}.$$

Though $\delta\mathbf{p}$ is a vector of indeterminant direction in the above equation, we will define it to be in the direction of the gradient of S at $\mathbf{p}$ and with an unknown magnitude d. Thus we let $\delta\mathbf{p} = \nabla\hat{S}(\mathbf{p})\, d$, where $\nabla\hat{S}$ represents the normalized gradient of S. Substituting this into the above equation gives:

$$\begin{aligned} 0 &= S(\mathbf{p}) + \nabla S(\mathbf{p}) \cdot \nabla\hat{S}(\mathbf{p})\, d \\ &= S(\mathbf{p}) + ||\nabla S(\mathbf{p})||d. \end{aligned}$$

Solving for d yields $d = \frac{-S(\mathbf{p})}{||\nabla S(\mathbf{p})||}$. Thus,

$$\delta\mathbf{p} = -\frac{S(\mathbf{p})\nabla S(\mathbf{p})}{\nabla S(\mathbf{p}) \cdot \nabla S(\mathbf{p})}$$

.

$\delta\mathbf{p}$ is the increment that must be added to $\mathbf{p}_0$ to find the best first-order approximation to $\mathbf{p}'$. If S is a plane, this equation is exact and finds $\mathbf{p}'$ in one step; for non-planar S we repeat the procedure letting $\mathbf{p} = \mathbf{p}'$ and continue until the step size $||\delta\mathbf{p}||$ is sufficiently small.