A Surface Intersection Algorithm Based on Loop Detection

Michael E. Hohmeyer
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720

Abstract

A robust and efficient surface intersection algorithm that is implementable in floating point arithmetic, accepts surfaces algebraic or otherwise and which operates without human supervision is critical to boundary representation solid modeling. To the author's knowledge, no such algorithm has been developed. All tolerance-based subdivision algorithms will fail on surfaces with sufficiently small intersections. Algebraic techniques, while promising robustness, are presently too slow to be practical and don't accept non-algebraic surfaces. Algorithms based on loop detection hold promise [45,46,48]. They don't require tolerances except those associated with machine arithmetic, and can handle any surface for which there is a method to construct bounds on the surface and its Gauss map. Published loop detection algorithms are, however, still too slow and do not deal with singularities. We present a new loop detection criterion and discuss its use in a surface intersection algorithm. The algorithm, like other loop detection based intersection algorithms, subdivides the surfaces into pairs of sub-patches which do not intersect in any closed loops. This paper presents new strategies for subdividing surfaces in a way that causes the algorithm to run quickly even when the intersection curve(s) contain(s) singularities.

1 Introduction

A robust and efficient surface intersection algorithm is crucial in the implementation of a boundary representation (B-rep) solid modelling software system. When performing Boolean operations on B-rep solids it is necessary to calculate the intersection of the boundary surfaces. The correctness of the model depends on the correct functioning of the surface intersection algorithm.

2 Existing Techniques

A good overview of the various techniques used for the surface intersection problem is given in Pratt [41]. One can divide surface intersection algorithms into two major groups: algebraic and numerical. Algebraic approaches naturally depend on the surfaces being algebraic and must generally be implemented in exact arithmetic. Numerical approaches are generally less dependent on the algebraic nature of the surfaces and can be implemented in approximate arithmetic. Both approaches consist of two steps: decomposing the intersection curve, and generating large numbers of points along the intersection (tracing).

2.1 Decomposition

The first step in a surface intersection algorithm is the decomposition of the curve. Exactly what this means varies from algorithm to algorithm. It might include, for example, determining the number of components, a point on each component, the location of component crossings or perhaps the location of certain other critical points on the intersection curve. In the algebraic approach this step can be accomplished in one of two ways. For restricted classes of surfaces such as quadrics, exhaustive case analysis can be performed [30,32,33,34,38,40,44,49]. Alternatively, various techniques including elimination theory and substitution can be employed to map the intersection curve into the plane [5,12,17,18,19]. The critical points of the resulting planar algebraic curve can be found via a cylindrical decomposition [2].

If an algorithm is to determine reliably the decomposition of the surface without using the surfaces' algebraic structure it cannot base its decision on a finite number of surface evaluations as is done in [3,8,15,21,27]. Bounds on the surface must be used for this determination. This is typically accomplished by subdividing the surface into smaller sub-surfaces that can be shown to be relatively flat. The intersection of planar approximations to these subsurfaces is taken as an outline of the intersection [1,4,9,11,10,16,20,21,27,

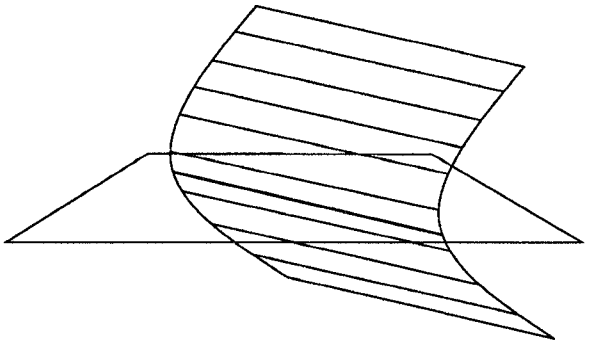

Figure 1: Unnecessary Subdivision of a Surface.

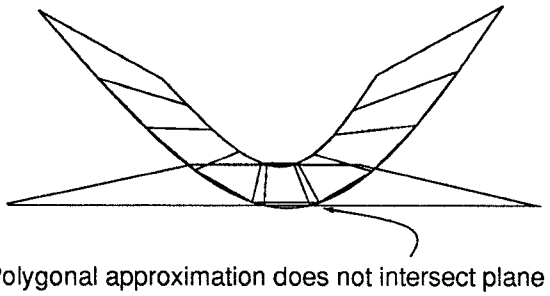

Figure 2: Insufficient Subdivision of a Surface.

29,35,36,37,50]. This flatness criterion is the Achilles' heel of all of these methods. Figure 1 depicts an intersection problem in which the flatness tolerance is very small and considerable work is performed. The correct outline of the curve could have been determined with a coarser tolerance. Figure 2, on the other hand, depicts a situation in which the flatness tolerance is too large and a nearly tangential intersection will be missed. The fundamental problem is that the flatness of a pair of surfaces has nothing to do with how they intersect.

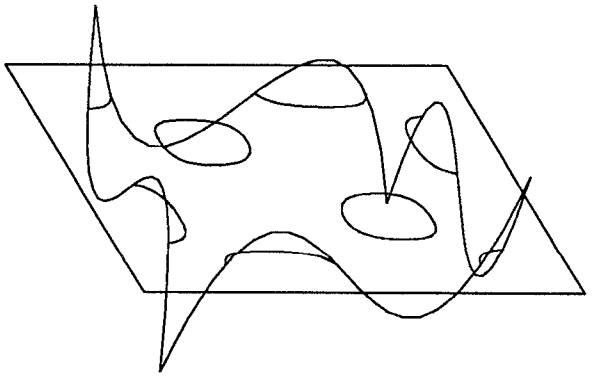

Figure 3: An intersection consisting of eight components.

Figures 3 and 4 depict intersection problems in which the surfaces intersect in identical sets of curves. One

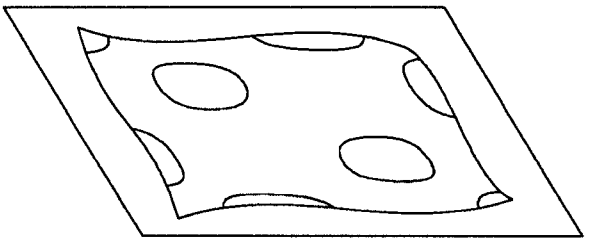

Figure 4: An intersection also consisting of eight components.

surface in Figure 3 is not at all flat. Both surfaces in Figure 4 are relatively flat. The surfaces could be made as flat as one pleases and the intersection would remain the same.

One would like to be able to make a qualitative, as opposed to a quantitative, statement about the intersection. Recent research by Sinha [48] and Sederberg [45,46] indicates that the correct thing to establish is that the intersection curve(s) form(s) no loops. (A loop in this context is a closed circuit, i.e. a component that does not intersect the boundary of a patch.)

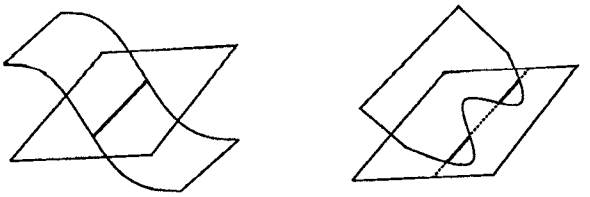

Figure 5: If there are no loops only edge intersections need to be checked.

If no loops exist in a subdivided patch, as depicted in Figure 5, then all intersection curves must intersect the boundary of the patch. The intersection of the surface boundaries with the intersection curves can be more easily managed and thus the surface intersection problem reduces to intersecting curves and surfaces.

In this paper we will describe a new test that determines if it is possible for two surfaces to intersect in a loop. The criterion presented here is similar to the results of Sinha and Sederberg. Two surfaces that satisfy the criteria of Sinha or Sederberg will satisfy the criterion presented here. However, there are a large number of situations in which two surfaces will satisfy the criterion presented here but will not satisfy either the the criterion of Sinha or of Sederberg. Before presenting the criterion itself, consider the tracing step of an intersection algorithm.

2.2 Curve Tracing

The second step of the intersection process for both algebraic and numerical methods is curve tracing [6,7,24,

25,26,28]. The difference is that if elimination theory has been used the curve will be traced in R^2, while in the numerical approaches, the curve is traced in the direct product of the parameter spaces of the two surfaces, R^4. The transformation from R^4 to R^2 via elimination of variables is not necessarily numerically stable; small changes in the data defining the surface may cause large changes in the curve in R^2, hence the need for exact arithmetic, or at least very careful analysis. Additionally, the transformation will raise the degree of the curve and introduce singularities. Although one can trace these curves [26], researchers have observed that it is easier, at least in approximate arithmetic, to trace curves that are the intersection of many low degree surfaces in a higher dimensional space than to trace the intersection of higher degree surfaces in a lower dimensional space [23]. The price of using the cylindrical decomposition is the increased difficulty involved in tracing the curve.

The tracing of curves in three dimensions has been discussed often in the literature [3,8,13,14,39,42,43]. Two problems that must be addressed when tracing a curve are *jumping components* and *backtracking*. Jumping components occurs when the tracing algorithm takes a step that is too large and proceeds on a different component of the curve. Backtracking occurs when for some reason the tracing algorithm generates points out of order. While numerical techniques exist to prevent either of these occurrences, it is difficult to verify in practice that the algorithm has not actually made these mistakes. The loop detection criterion presented in this paper allows one to devise a very simple curve tracing algorithm that is guaranteed to generate points in order and on a single component of the intersection curve.

3 Loop Detection Criterion

The loop detection criterion is geometric and requires only simple algebra to prove.

Lemma 1 *Let N_1 and N_2 be two sets of vectors and $\mathbf{P}_1$ and $\mathbf{P}_2$ be two vectors satisfying*

$$\mathbf{P}_1 \cdot \mathbf{n}_1 > 0, \quad \mathbf{P}_1 \cdot \mathbf{n}_2 < 0 \tag{1}$$

$$\mathbf{P}_2 \cdot \mathbf{n}_1 > 0, \quad \mathbf{P}_2 \cdot \mathbf{n}_2 > 0 \tag{2}$$

for all $\mathbf{n}_1 \in N_1$ and $\mathbf{n}_2 \in N_2$, as in Figure 6. Choose any pair of vectors $\mathbf{n}_1 \in N_1$ and $\mathbf{n}_2 \in N_2$. Then the cross product $\mathbf{T}$ given by

$$\mathbf{T} = \mathbf{n}_1 \times \mathbf{n}_2 \tag{3}$$

satisfies

$$\mathbf{T} \cdot (\mathbf{P}_1 \times \mathbf{P}_2) > 0. \tag{4}$$

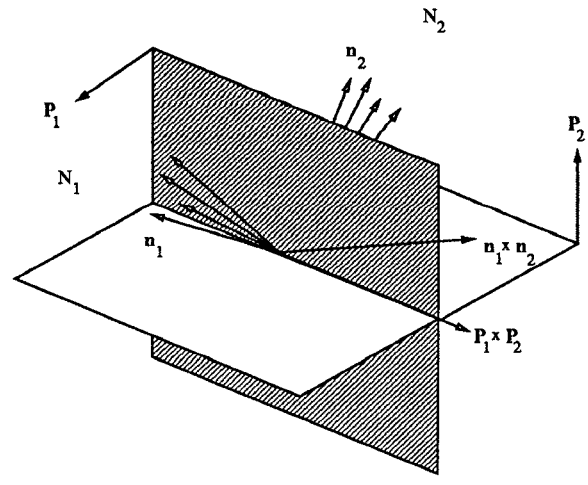

Figure 6: Geometry of the Subdivision Criterion.

Note that the hypothesis precludes the possibility that $\mathbf{n}_1 \times \mathbf{n}_2 = \mathbf{0}$.

Proof:

$$\begin{aligned}
&(\mathbf{P}_1 \times \mathbf{P}_2) \cdot (\mathbf{n}_1 \times \mathbf{n}_2) && (5)\\
&= ((\mathbf{P}_1 \times \mathbf{P}_2) \times \mathbf{n}_1) \cdot \mathbf{n}_2 && (6)\\
&= -(\mathbf{n}_1 \times (\mathbf{P}_1 \times \mathbf{P}_2)) \cdot \mathbf{n}_2 && (7)\\
&= (-(\mathbf{n}_1 \cdot \mathbf{P}_2)\mathbf{P}_1 + (\mathbf{n}_1 \cdot \mathbf{P}_1)\mathbf{P}_2) \cdot \mathbf{n}_2 && (8)\\
&= -(\mathbf{n}_1 \cdot \mathbf{P}_2)(\mathbf{P}_1 \cdot \mathbf{n}_2) + (\mathbf{n}_1 \cdot \mathbf{P}_1)(\mathbf{P}_2 \cdot \mathbf{n}_2) && (9)
\end{aligned}$$

Thus

$$(\mathbf{P}_1 \times \mathbf{P}_2) \cdot (\mathbf{n}_1 \times \mathbf{n}_2) > 0. \tag{10}$$

■

The Gauss map of a surface is the map that takes points on the surface to the surface normal. Here we use the term to mean the image of the Gauss map, the set of *all* normals to the surface. The following theorem allows one to make conclusions about surface intersections based on the Gauss maps of the surfaces.

Theorem 1 *Let S_1 and S_2 be two C^1 surfaces whose Gauss maps are contained in sets N_1 and N_2, respectively. If there exist vectors $\mathbf{P}_1$ and $\mathbf{P}_2$ such that*

$$\mathbf{P}_1 \cdot \mathbf{n}_1 > \mathbf{0}, \quad \mathbf{P}_1 \cdot \mathbf{n}_2 < 0 \tag{11}$$

$$\mathbf{P}_2 \cdot \mathbf{n}_1 > \mathbf{0}, \quad \mathbf{P}_2 \cdot \mathbf{n}_2 > 0 \tag{12}$$

for all $\mathbf{n}_1 \in \mathrm{N}_1$ and $\mathbf{n}_2 \in \mathrm{N}_2$, as in Figure 6, then the intersection of the two surfaces is a curve, a point, or a set of curves and points. Furthermore, all isolated point intersections are at the boundaries of the surface patches, the curves do not contain singularities, and no intersection curve forms a loop.

Proof: If the surfaces intersected anywhere in a surface then the normals to S_1 and S_2 would be parallel or anti-parallel at each point of the intersection, in violation of equations (11) and (12).

If $\mathbf{p}$ is a point of intersection and is interior to both patches, then the normals to the two surfaces at $\mathbf{p}$ are not collinear by equations (11) and (12). In a neighborhood of $\mathbf{p}$ the surfaces are very nearly a pair of planes intersecting in a curve that is very nearly a line perpendicular to the normals to both surfaces at $\mathbf{p}$. Thus $\mathbf{p}$ is not an isolated point, but rather lies on a curve.

Let each intersection curve be oriented so that the tangent to the curve at a point $\mathbf{x}$ is:

$$\mathbf{T}(\mathbf{x}) = \frac{\mathbf{n}_1(\mathbf{x}) \times \mathbf{n}_2(\mathbf{x})}{|\mathbf{n}_1(\mathbf{x}) \times \mathbf{n}_2(\mathbf{x})|}, \tag{13}$$

where $\mathbf{n}_i(\mathbf{x})$ is the normal to surface S_i at the point $\mathbf{x}$. Note that the denominator in equation (13) cannot be zero since equations (11) and (12) preclude the normals from being parallel or anti-parallel. By Lemma 1,

$$\mathbf{T}(\mathbf{x}) \cdot \mathbf{P} > 0. \tag{14}$$

Where $\mathbf{P}$ is given by

$$\mathbf{P} = \mathbf{P}_1 \times \mathbf{P}_2. \tag{15}$$

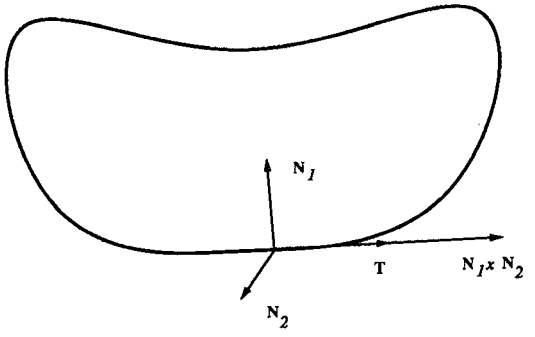

Figure 7: A Hypothetical Intersection Loop.

Suppose that there were a closed loop in the intersection of the two surfaces, as depicted in Figure 7. The fact that the normals to the surfaces are never collinear precludes any singularities in the intersection curve. Thus, the curve tangent $\mathbf{T}$ is well defined (up to a choice of orientation). If we were to parametrize the loop by arc length, s, from an arbitrary point on the loop then

$$\int_{s=0}^{s=l} \mathbf{T}(s) ds = \mathbf{0} \tag{16}$$

where l is the length of the curve. More specifically,

$$\int_{s=0}^{s=l} \mathbf{T}(s) \cdot \mathbf{P} ds = 0. \tag{17}$$

The tangent $\mathbf{T}(s)$ is a scalar multiple of the the cross product of the normals to the surface. This scalar varies continuously and is never zero. Thus, the sign of this scalar is constant (positive by convention) throughout the loop. Equations (17) and (14) are clearly inconsistent; thus there can be no loop. ∎

Theorem 1 is much like that in [48], with two differences. First, the proof is simpler. Second, we know the direction in which the curve is monotonic. Thus, two points on the intersection curve, $\mathbf{x}_1$ and $\mathbf{x}_2$, can be ordered by ordering the scalar values $\mathbf{x}_1 \cdot \mathbf{P}$ and $\mathbf{x}_2 \cdot \mathbf{P}$. Finally if there are only two patch/edge intersections then there can be only one intersection curve and the intersection points are its end points. In this case all points are guaranteed to lie on the same component. If there are more edge-surface intersection points, then the surfaces must be further subdivided.

Note that we do not make any assumptions about the form of the surfaces. In fact, parametric surfaces are allowed to contain holes in their domains. This is in contrast to Sederberg's criterion [45] which requires that the interior of the loop be homeomorphic to the unit disc.

4 A Simple Tracing algorithm

If one has subdivided a pair of parametric patches $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$ so that they pass the loop detection criterion presented in the last section, obtained the vector $\mathbf{P}$ given by equation (15), and calculated the two points $\mathbf{p}_0$ and $\mathbf{p}_1$ where the edges of the surfaces intersect each other, then a simple tracing algorithm is as follows:

Trace

$t_0 = \mathbf{P} \cdot \mathbf{p}_0$
$t_1 = \mathbf{P} \cdot \mathbf{p}_1$
for $t = t_0$ *to* t_1 *stepping by* δt
    solve the system of equations
        $\mathbf{F}(s,t) = \mathbf{G}(u,v)$
            and
        $\mathbf{F}(s,t) \cdot \mathbf{P} = t$
    using Newton iteration and the previous solution as a starting iterate
endfor

Having passed the loop detection criterion the system of equations will never be singular. Since there are only two points of intersection of the curve with the surface edges there can only be one intersection curve. Since the curve tangent always makes a positive inner product with $\mathbf{P}$, it can only intersect a plane perpendicular

to **P** once. Since the curve must proceed from $\mathbf{p}_0$ to $\mathbf{p}_1$, it must pass through an intermediate plane. Thus, the system of equations has exactly one solution in the region specified by the subdivision of **F** and **G**. No other checks need to be made. Experience has shown that the Newton iteration generally converges. If it does not, one can simply reduce δt until it does. There is no possibility of jumping components since there is only one curve component in the range. There is no possibility of generating points out of order since the series of planes automatically orders them. Thus the value δt does *not* need to be chosen carefully.

5 Implementation

In order to make this criterion useful one must be able to construct the Gauss maps N_1 and N_2 and determine if vectors $\mathbf{P}_1$ and $\mathbf{P}_2$ exist. If the vectors do exist it would be desirable to know them so that points on the intersection curve can be ordered. In the following section we discuss how this can be done for certain classes of surfaces. Although we do not present methods to do this for all forms of surfaces employed in modeling systems, the development of such methods appears straightforward.

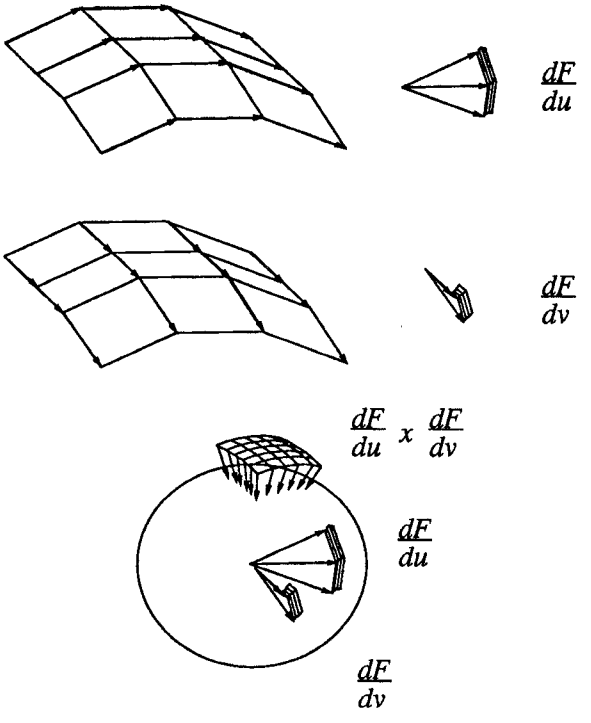

Figure 8: Computation of bounds on normal.

5.1 Computing the Gauss Map for Some Common Surfaces

For a parametric surface $\mathbf{F}(u,v)$ in Bézier form it is possible to compute the Bézier surface:

$$\mathbf{N}(u,v) \equiv \frac{\partial \mathbf{F}}{\partial u} \times \frac{\partial \mathbf{F}}{\partial v} \tag{18}$$

depicted in Figure 8. Let us call $\mathbf{N}(u,v)$ the *pseudo-normal surface* of **F**. The true normal surface is obtained from $\mathbf{N}(u,v)$ by projecting it onto the unit sphere. The convex hull of the projection of the control vectors of $\mathbf{N}(u,v)$ is a bound on the Gauss map.

B-Spline surfaces, on the other hand, can be broken into Bézier surfaces and the above technique applied. If a surface were given in implicit form, say as the zero set of a trivariate scalar valued polynomial $f(x,y,z)$, one could again obtain a pseudo-normal function (in fact the gradient function)

$$\mathbf{N}(x,y,z) \equiv \nabla f. \tag{19}$$

The convex hull of its three dimensional lattice of Bernstein control vectors would form a bound on the gradient function and thus the Gauss map.

5.2 Linear Programming

Once one has obtained the discrete sets N_1 and N_2 whose convex hulls are bounds on the Gauss maps of the surfaces $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$, respectively, one must find $\mathbf{P}_1$ and $\mathbf{P}_2$ satisfying equations (1) and (2):

$$\mathbf{P}_1 \cdot \mathbf{n}_1 > 0, \quad \mathbf{P}_1 \cdot \mathbf{n}_2 < 0 \tag{20}$$

$$\mathbf{P}_2 \cdot \mathbf{n}_1 > 0, \quad \mathbf{P}_2 \cdot \mathbf{n}_2 > 0 \tag{21}$$

for all $\mathbf{n}_1 \in N_1$ and all $\mathbf{n}_2 \in N_2$.

This is almost a pair of linear programming (LP) problems in two variables. For instance, to compute $\mathbf{P}_1$ is to find x and y so that

$$A_i x + B_i y + C_i > 0 \tag{22}$$

where

$$(A_i, B_i, C_i) = \mathbf{n}_{1i} \tag{23}$$

for $i = 1, ..., |N_1|$ and

$$(A_{i+m}, B_{i+m}, C_{i+m}) = -\mathbf{n}_{2i} \tag{24}$$

for $i = 1, ..., |N_2|$, where $m = |N_1|$. The solution would then be

$$\mathbf{P}_1 = (x, y, 1). \tag{25}$$

Unfortunately, this approach cannot find $\mathbf{P}_1$'s with non-positive z components. We can remedy this by solving a second problem if the first were infeasible. In this second problem we attempt to satisfy

$$A_i x + B_i y - C_i > 0 \tag{26}$$

and the solution would be

$$\mathbf{P}_1 = (x, y, -1) \tag{27}$$

However a more fundamental problem remains. LP algorithms solve constraints of the form

$$L_i(x, y, ..., z) \geq 0 \tag{28}$$

and not

$$L_i(x, y, ..., z) > 0. \tag{29}$$

In general, the solution to a LP problem in d dimensions will satisfy d of the constraints with equality. Equations (22) and (26) are of the form (29) and not (28). This is important since, if equality is ever achieved, inequality 4 will be an equality and the tangent $\mathbf{T}$ might be perpendicular to the computed vector $\mathbf{P}$ allowing the system of equations in the tracing algorithm to become singular. We can rectify this by solving a higher dimensional LP problem.

The LP problem we will solve will be

$$A_i x + B_i y + C_i z \geq \epsilon > 0 \tag{30}$$

maximizing $\epsilon/(x^2 + y^2 + z^2)$. If we re-write this as

$$A_i x + B_i y + C_i z - \epsilon \geq 0. \tag{31}$$

One can see that this is a LP problem in projective three-space. This problem can be solved with a LP algorithm that runs in time linear in the number of constraints [47] . Although the LP algorithm runs in linear time, it is slow enough to dominate the running time of the surface intersection algorithm unless other steps are not taken.

First, the spherical analogs of bounding boxes are used as a cheap test. These bounding boxes are a set of three pairs of planes as depicted in Figure 9. Each pair of planes contains one of the principal axes. A pair of planes defines a wedge-shaped region of the sphere. The intersection of the three wedges forms a hexagon on the sphere. Two such regions can be tested for intersection in constant time.

Secondly, the LP algorithm is constructed in such a way that it terminates as soon as it determines that there is no feasible region. Thus, if two regions on the sphere overlap considerably, the algorithm will only have to look at a handful of vectors before returning the answer that they intersect. However, if the bounding box test fails and the regions do not intersect, the LP algorithm will have to examine every vector.

If the Gauss maps of two sub-patches have been constructed to intersect at a point on the sphere the intersection test becomes much faster. In that case, the separability test is only a matter of finding a plane containing the intersection point with the two regions on opposite sides. Such a plane will only have one degree

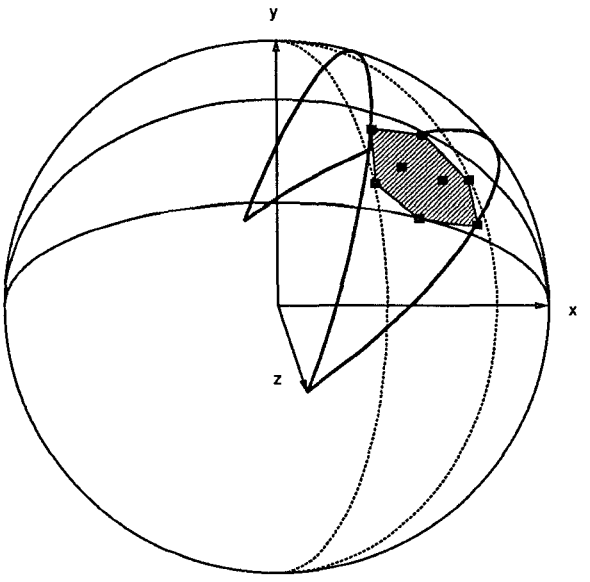

Figure 9: Bounding boxes on the sphere.

of freedom. Such a vector will be known when the surfaces have been subdivided at a point where they share a common normal. This method of subdivision is explored in the next section.

6 Subdivision Algorithm

In this section we describe how one can use the loop detection criterion in a surface intersection algorithm. The input to the algorithm will be two surfaces. It is assumed that one is able to calculate bounding volumes around the surfaces and around their Gauss maps. It is further assumed that it is possible to subdivide the surfaces and that as one subdivides the surfaces the bounding volumes approach the surface and the bounds on the Gauss map approach the Gauss map.

Intersect

if *the bounding volumes of the surfaces intersect* **then**
  if *the Gauss maps satisfy the loop detection criterion* **then**
    Intersect the edges of the first surface with those of the second and those of the second surface with those of the first.
    if *there are two intersections* **then**
      Trace *the intersection curve*
    else
      special case code
    endif
  else

Subdivide each surface.
Intersect *all pairs.*
endif
endif

The special case code deals with non-generic intersections such as edge-edge intersections or interior-corner intersections. Although implemented, the discusion of the special case code and the surface edge intersection is beyond the scope of this paper. In the simplest approach, the surface with the largest Gauss map is subdivided in the middle to make four subpatches. The results of **Intersect** are displayed in Figure 10.

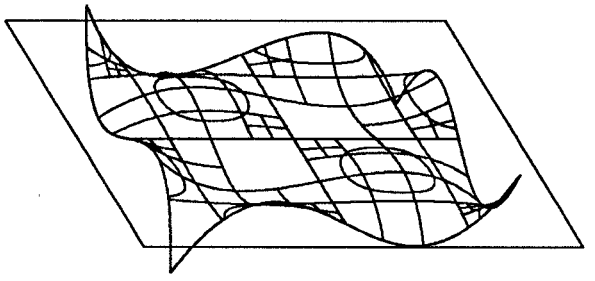

Figure 10: Results of a simple subdivision strategy.

The two surfaces being intersected are a plane and a bi-cubic patch. At each level of recursion in the algorithm, the Gauss map of the bi-cubic surface grows smaller. If the recursion continued indefinitely the Gauss maps would, for each surface, converge to points on the sphere representing the infinitesimal surfaces' normals. Thus the loop detection criterion must eventually be satisfied unless the two surfaces share the same normal at the same point. This example took approximately 7.6 seconds (not including the curve tracing) on a Silicon Graphics 12 MHz Personal Iris. It takes about 3 seconds to trace all the curves but this is very dependent on the number of points generated and thus was not included in the previous time.

Consider the surface intersection problem shown in Figure 10: if we lower the plane slightly the intersection will consist of two sets of three parallel lines. These lines will intersect in nine points. At these points the surfaces have the same normal. Such a point is a *singular point* of the intersection curve, (see [22], page 32). If the algorithm were run on this example with the simple subdivision strategy, it would recurse indefinitely at all of these singular points.

Even for examples where the curve nearly contains a singular point, the amount of subdivision grows without bound. In Figure 11 we have taken the example shown in Figure 10 and lowered the plane closer to the position where it would form intersection curves with singularities. It can be seen that the amount of subdivision has

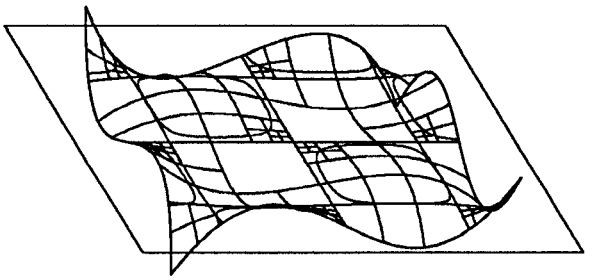

Figure 11: The results of the simple subdivision strategy when the intersection curve nearly contains a singular point.

increased near the points where the singularities nearly occur. This example took 11.8 seconds to solve under the same conditions.

We would like a strategy that causes the subdivision to terminate quickly in the presence of singularities and near-singularities. We propose the following alternative to subdividing the surfaces in the middle. If the bounding volumes of the two surfaces intersect and their normals do not pass the loop detection criterion then Newton iteration is performed in an attempt to find a solution to the system of equations

$$\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t) \cdot \mathbf{G}_u(u,v) = 0 \quad (32)$$
$$\mathbf{F}_s(s,t) \times \mathbf{F}_t(s,t) \cdot \mathbf{G}_v(u,v) = 0 \quad (33)$$
$$(\mathbf{F}(s,t) - \mathbf{G}(u,v)) \cdot \mathbf{F}_s(s,t) = 0 \quad (34)$$
$$(\mathbf{F}(s,t) - \mathbf{G}(u,v)) \cdot \mathbf{F}_t(s,t) = 0. \quad (35)$$

While we have only given a system of equations here for a pair of parametric surfaces, a similar system of equations exists when one or both of the surfaces are implicit. The significance of these equations is that they are satisfied at a singularity, and additionally, if either of the surfaces is perturbed by a slight amount the solution of this system of equations does not change too much. Points on the surface that satisfy this set of equations have been called *magic points* in [31] and also *critical points.*

If a solution to equations (32) - (35) is found the surfaces are subdivided there. If the iteration does not converge to a solution then we simply subdivide the surfaces as before or apply another subdivision strategy.

In Figure 12 we have taken the intersection problem depicted in Figure 10 and used the subdivision strategy described above instead. For this pair of surfaces there are 13 solutions to equations (32) - (35). Nine are at the intersection of the two pairs of three lines and the remaining four are at the tops of the two "hills" and the bottoms of the two "valleys". When there was a solution to equations (32) - (35) in the regions of the surfaces being subdivided, the Newton iteration found

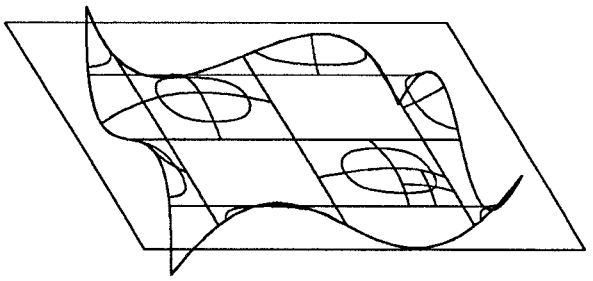

Figure 12: Subdivision resulting from more sophisticated subdivision strategy.

it in every case except one. The number of patches in the final subdivision is 31 (compared to 88 when the simple subdivision strategy is used) and the running time decreases to 3.8 seconds.

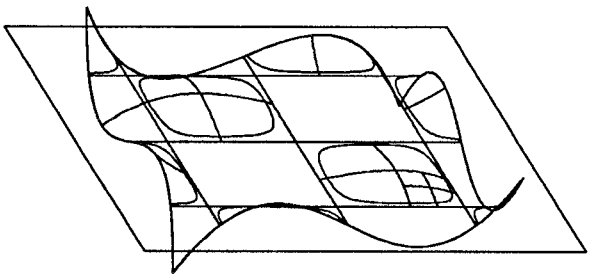

Figure 13: The number of patches in the subdivision is unchanged when the surfaces are near the degenerate position.

More importantly, as the two surfaces get closer to intersecting in singular curves, the amount of subdivision does not change. In Figure 13 we have taken the surfaces from Figure 11 and applied the the more sophisticated subdivision strategy. The number of patches and the running time remain unchanged at 31 and 3.8 seconds, respectively, while in Figure 11 these numbers have increased to 11.8 seconds and 110 patches.

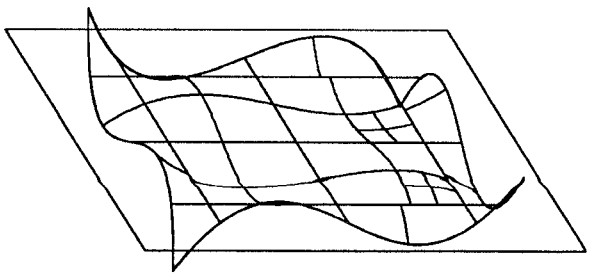

Figure 14: The number of patches in the subdivision is moderate even with degenerate intersections.

The number of patches increases only slightly when the intersection is actually in the degenerate position as depicted in Figure 14 (The running time is not available since the surface-curve intersection algorithm has not been modified to handle the degeneracies encountered here).

The process of finding singular points is not as susceptible to the unreliability of the Newton iteration as one might suspect. Suppose that there were a singular point on the intersection curve. The loop detection criterion would fail since the Gauss maps would intersect at the corresponding normal. The Newton iteration might or might not find the singularity at that level of subdivision. If the singularity is not found the patches are subdivided and the pair containing the singularity will again fail the loop detection criterion. When the subdivided surfaces become smaller than the region of convergence of the Newton iteration, the iteration will succeed in finding the singularity. Thus, the singularity will eventually be found.

7 Subdividing at Singularities

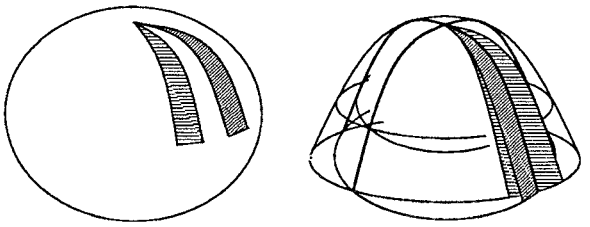

Figure 15: A pair of intersecting wedges

The examples discussed in the previous section are rather special because one of the surfaces is a plane. The Gauss map of the plane is only a single point on the sphere. Thus, when the other surface has been broken up into pieces that do not have this point in the convex hull of their Gauss maps, each piece will pass the loop detection criterion regardless of its spatial position. When both surfaces are curved, the singular points are still the correct place to subdivide, but the subdivision pattern must be more complex.

Using a second order Taylor series we can determine the directions of the intersecting curve branches. If the curve branches are not parallel at the singularity, the singularity is a *node*, a transversal crossing of two curves. Other types of singularities that might arise are tacnodes, triple points, or curves where each point is singular [22]. We only consider the case of the node. Our objective is to break up the surface intersection problem into pairs of sub-surfaces containing no singularities and where the intersection is a single curve. In the case depicted by Figure 15 we must subdivide the surface into eight sub-patches all meeting at the singularity. The patches will be the image of triangular regions of the domains of the surfaces. The eight

subpatches will be of two types, four of each.

The first type will be called an *intersecting wedge.* A pair of intersecting wedges is shown in Figure 15. On each surface there will be four intersecting wedges, one corresponding to each intersection curve branch radiating from the singularity. The subdivision process terminates only when a pair of surfaces fail to intersect in either R^3 or on the Gaussian sphere. Since this pair clearly intersects in R^3 they must be constructed not to intersect on the Gaussian sphere. Note that as one traces the intersection curve away from the singularity, the normals to the two surfaces separate linearly with the distance from the singularity. Thus, the Gauss maps of two sufficiently narrow wedges will intersect at the singularity, and will diverge from one another linearly as depicted in Figure 15. While the two Gauss maps may overlap further away from the singularity, there is some neighborhood of the singularity in which they intersect only at the singularity. Thus, some truncation of these wedges will pass the loop detection criterion.

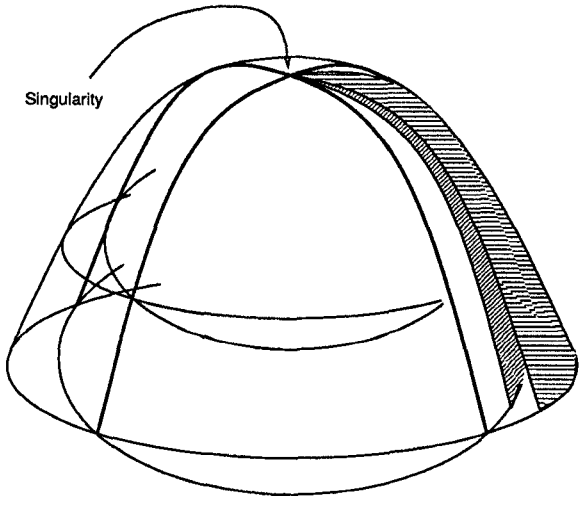

Figure 16: A pair of disjoint wedges.

The second type of subpatch will be called a *disjoint wedge.* There will be four disjoint patches per surface, one for each inter-branch angle. The pair of disjoint wedges depicted in Figure 16 intersect on the Gaussian sphere. However, to a second order approximation they do not intersect in R^3. The two surface patches are tangent at the singularity. Because the second order terms in the surfaces do not cause them to draw away from the tangent plane in opposite directions, it will not be possible to find a separating plane. A higher order separating element must be used. A quadric surface represented as $h(x, y, z)$ called a *quadric separating function* can be determined that will separate the surfaces to the second order. The disjoint wedges $\mathbf{F}(s, t)$ and $\mathbf{G}(u, v)$ can be substituted into h yielding

$$h \circ \mathbf{F}(s,t) = h(F_x(s,t), F_y(s,t), F_z(s,t)), \quad (36)$$

$$h \circ \mathbf{G}(u,v) = h(G_x(u,v), G_y(u,v), G_z(u,v)). \quad (37)$$

The zeroth and first order terms of these two functions will be zero at the singularity. However, in some sufficiently small neighborhood of the singularity, but not at the singularity itself, $h \circ \mathbf{F}(s, t)$ and $h \circ \mathbf{G}(u, v)$ will be positive and negative respectively.

A similar quadric separating function must be constructed to demonstrate that a disjoint wedge doesn't intersect a neighboring intersecting wedge. When the wedges have been truncated by later subdivision if necessary, each of the wedges will pass either the R^3 separability test or the Gaussian sphere separability test and the subdivision will terminate.

8 Conclusion

In this paper we have introduced a new subdivision criterion. A pair of surface patches that has passed this criterion and have only two surface/edge intersections is guaranteed to intersect in a single curve whose tangent makes a positive inner product with a calculable vector. This results in a simplified curve tracing algorithm. A strategy was discussed for subdividing the surfaces in such a way that the recursion terminates quickly even in the presence of singularities and near-singularities. Algorithms for intersecting curves and surfaces were not discussed in this paper, and warrant further research. Although a method was discussed for dealing with nodes, methods for triple points, cusps, tacnodes and double curves have not yet been developed and also warrant further research.

References

[1] P. R. Arner. *Another Look at Surface/Surface Intersection.* PhD thesis, University of Utah, Salt Lake City, 1987.

[2] Dennis S. Arnon, George E. Collins, and Scott McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal of Computation,* 13(4):865–889, November 1984.

[3] C. Asteasu. Intersection of arbitrary surfaces. *Computer-Aided Design,* 20(6):533–538, July 1988.

[4] D. Ayala. Boolean operations between solids and surfaces. *Computer-Aided Design,* 20(8):452–465, October 1988.

[5] Chanderjit Bajaj, Thomas Garrity, and Joe Warren. On the applications of multi-equational resultants. Technical Report TR88-83, Rice University,, Houston, Texas, November 1988.

[6] Chanderjit L. Bajaj, Christoph M. Hoffmann, and John E. Hopcroft. Tracing planar algebraic curves. Technical Report CSD-TR-637, Department of Computer Science, Purdue University, West Lafayette, Indiana 47907, September 1987.

[7] Chanderjit L. Bajaj, Christoph M. Hoffmann, John E. Hopcroft, and Robert E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5(4):285–307, November 1988.

[8] Robert Barnhill, Gerald Farin, and Bruce Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4:3–16, 1987.

[9] Wayne E. Carlson. An algorithm and data structure for 3D object synthesis using surface patch intersections. In *SIGGRAPH '82 Conference Proceedings*, pages 255–259, July 1982.

[10] M. S. Casale and J. E. Borrow. A set operation algorithm for sculptured solids modeled with trimmed patches. *Computer Aided Geometric Design*, 6:235–247, June 1989.

[11] Malcolm S. Casale. Free-form solid modeling with trimmed surface patches. *IEEE Computer Graphics and Applications*, 7(1):33–43, January 1987.

[12] Vijay Chandru and Bipin S. Kochar. Analytic techniques for geometric intersection problems. In Gerald E. Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 305–318. SIAM, Philadelphia, 1987.

[13] J. J. Chen and T. M. Ozsoy. An intersection algorithm for C^2 parametric surfaces. In *Knowledge Engineering and Computer Modelling*, pages 69–77. 1986.

[14] J. J. Chen and T. M. Ozsoy. Predictor-corrector type of intersection algorithm for C^2 parametric surfaces. *Computer-Aided Design*, 20(6):347–352, July 1988.

[15] Koun-Ping Cheng. Using plane vector fields to obtain all the intersection curves of two general surfaces. In Wolfgang Strasser and Hans-Peter Seidel, editors, *Theory and Practice of Geometric Modeling*, pages 188–204. Springer-Verlag, 1989.

[16] T. Dokken. Finding intersections of B-spline representations using recursive subdivision. *Computer Aided Geometric Design*, 2:189–196, 1985.

[17] R. Farouki. Trimmed surface algorithms for the evaluation and interrogation of solid boundary representations. *IBM Journal of Results and Developements*, 31, 1986.

[18] R. T. Farouki. The characterization of parametric surface sections. *Computer Vision, Graphics and Image Processing*, 33:209–236, 1986.

[19] Rida T. Farouki. Direct surface section evaluation. In Gerald Farin, editor, *Geometric Modeling: Algorithms and New Trends*, pages 319–334. SIAM, Philadelphia, 1987.

[20] Daniel Filip, Robert Markot, and Robert Madgedson. Using bounds on derivatives in computer aided geometric design. Submitted for publication.

[21] S. L. Hanna, J. F. Abel, and D. P. Greenberg. Intersection of parametric surfaces by means of look-up tables. *IEEE Computer Graphics and Applications*, 3(3), October 1983.

[22] Robin Hartshorne. *Algebraic Geoemtry*. Springer-Verlag, New York, 1977.

[23] Christoff M. Hoffmann. A dimensionality paradigm for surface interrogations. Technical Report CER-87-45, Department of Computer Science, Purdue University, December 1988.

[24] Christoph M. Hoffmann. Algebraic curves. Technical Report Technical Report, Computer Science Department, Purdue University, West Lafayette, Indiana 47907, May 1987.

[25] Christoph M. Hoffmann. Applying algebraic geometry to surface intersection evaluation. Technical Report CSD-TR-772, Computer Science Department, Purdue University, West Lafayette, Indiana 47907, April 1988.

[26] Christoph M. Hoffmann and Robert E. Lynch. Following space curves numerically. Technical Report CSD-TR-684, Computer Science Department, Purdue University, West Lafayette, Indiana 47907, May 1987.

[27] E. G. Houghton, R. F. Emnet, J. D. Factor, and C. L. Sabharwal. Implementation of divide-and-conquer method for intersection of parametric sirfaces. *Computer Aided Geometric Design*, pages 173–183, 1985.

[28] John K. Johnstone. *The Sorting of Points Along An Algebraic Curve*. PhD thesis, Cornell University, Ithica, New York, 1987.

[29] Dieter Lasser. Intersection of parametric surfaces in the Bernstien-Bézier representation. *Computer-Aided Design*, 18(4):186–192, May 1986.

[30] Joshua Z. Levin. Mathematical models for determining the intersections of quadric surfaces. *Computer Graphics and Image Processing*, 11:73–87, September 1979.

[31] R. P. Markot and R. L. Magedson. Solutions of tangential surface and curve intersections. *Computer-Aided Design*, 21(7):421–427, September 1989.

[32] James R. Miller. Geometric approaches to nonplanar quadric surface intersection curves. *ACM Transactions on Graphics*, 6(4):274–307, October 1987.

[33] James R. Miller. Sculptured surfaces in solid models: Issues and alternative approaches. *IEEE Computer Graphics and Applications*, 6(12):33–43, 1987.

[34] James R. Miller. Analysis of quadric-surface-based solid models. *IEEE Computer Graphics and Applications*, 8(1):28–42, 1988.

[35] B. K. Natarajan. On computing the intersection of B-splines. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*. ACM, June 1990.

[36] J. Owen and A. Rockwood. Intersection of general implicit surfaces. In Gerald Farin, editor, *Geometric Modeling: Algorithms and Trends*. SIAM Publications, Philadelphia, 1987.

[37] Qun Sheng Peng. An algorithm for finding the intersection lines between two B-spline surfaces. *Computer-Aided Design*, 16(4), July 1984.

[38] H. U. Pfeifer. Methods for intersecting geometrical entities in the gpm module for volume geometry. *Computer-Aided Design*, 17(7):311–318, September 1985.

[39] M. B. Phillips and G. M. Odell. An algorithm for locating and displaying the intersection of two arbitrary surfaces. *IEEE Computer Graphics and Applications*, 4(9):48–58, 1984.

[40] Leslie Piegl. Geometric method of intersecting natural quadrics represented in trimmed surface form. *Computer Aided Design*, 21(4):201–212, May 1989.

[41] M. J. Pratt and A. D. Geisow. Surface/surface intersection problems. In Gregory J. A., editor, *The Mathematics of Surfaces*, pages 117–142. Claredon Press, Oxford, 1986.

[42] Malcolm A. Sabin. *Interrogation Techniques for Parametric Surfaces*, pages 1095–1118. Plenum Press, London and New York, 1971.

[43] Malcolm A. Sabin. A method for displaying the intersection curve of two quadric surfaces. *The Computer Journal*, 19:336–338, November 1976.

[44] Ramon F. Sarraga. Algebraic methods for intersections of quadric surfaces in gmsolid. *Computer Vision, Graphics and Image Processing*, 22(2):222–238, May 1983.

[45] Thomas W. Sederberg, H. N. Christiansen, and S. Katz. Improved test for closed loops in surface intersections. *Computer-Aided Design*, 21(8):505–508, October 1989.

[46] Thomas W. Sederberg and Ray J. Meyers. Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5:161–171, 1988.

[47] Raimund Seidel. Linear programming and convex hulls made easy. In *ACM Symposium on Computational Geometry*, pages 211–215. ACM Press, 1990.

[48] P. Sinha, E. Klassen, and K. K. Wang. Exploiting topological and geometric properties for selective subdivision. In *ACM Symposium on Computational Geometry*, pages 39–45. ACM Press, 1985.

[49] Jack Scott Snoeyink. Intersecting trimmed quadric surface patches: a geometric method using parametric functions. Submitted for publication, 1990.

[50] Spencer W. Thomas. Modelling volumes bounded by B-spline surfaces. Ph.d. thesis, University of Utah, Department of Computer Science, Salt Lake City, Utah, June 1984.