1991

Intersection algorithms for nonlinear geometric modeling.

Jyun-Ming Chen
Carnegie Mellon University

Carnegie Mellon University.Engineering Design Research Center.

Intersection Algorithms for Nonlinear Geometric Modeling: Part II

J. Chen

EDRC 24-76-91

Intersection Algorithms for Nonlinear Geometric Modeling *

Part II

Jyun-Ming Chen
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

November 1, 1991

Abstract

The next generation geometric modeling systems have higher requirements on the reliability and the efficiency of Boolean operations. The two parts of these technical reports describe the details of the nonlinear intersection algorithms developed at EDRC. The prototype of the algorithms has been shown to be robust, accurate, and efficient, without the overhead of subdivision.

The second part of this report describes the intersection routines for two dimensional entities.

*This work has been supported by the Engineering Design Research Center, an NSF Engineering Research Center.

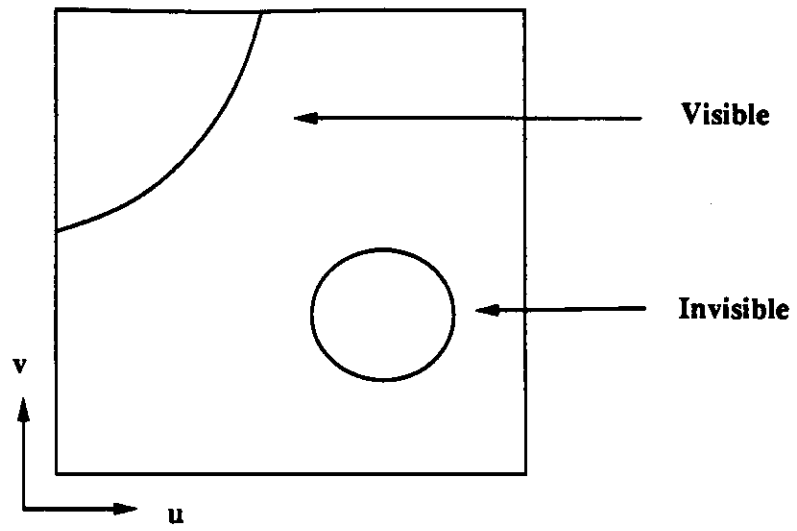


Figure 1: Examples of *visible* and *invisible* curves

1 Introduction

In the second part of this technical report, we will describe the intersection algorithms involving two dimensional entities (i.e., surfaces and planes). These problems are more complicated (than the ones described in part I) because the search spaces are two dimensional and more sophisticated algorithms are required to find the intersection. In this report, we will limit the discussion to *transversal* intersection [3] only. In a nut shell, transversal intersection excludes the cases of tangential contacts, which mathematically correspond higher order singularities. Non-transversal intersection is still under research and will be discussed in a later report.

The (transversal) intersection in SPI and SSI can be classified into two categories: *visible* and *invisible* curves (see figure 1). For the former, the intersection curve starts from and/or ends at the boundary of the domain. In other words, the intersection curve is *visible* from the boundary and can be obtained once the boundary intersection point is found. For the latter, the intersection curve is totally embedded in the domain, thus *invisible* from the domain boundary. In the literature, this problem is sometimes referred as *loop detection* ([14], [12]).

As mentioned in the part one of this report, here we only discuss the intersection algorithms for untrimmed entities. For example, a planar face, which can be a non-uniform multiply-connected polygon, is generalized to an infinite plane for the SPI operation. All the free-form surfaces are assumed to have the unit square as its domain. Post processing will be employed to adjust the intersection result to the appropriate domain. A more efficient way is to consider the effect of the trimmed domain in the preprocessing. The algorithms described in this paper can be easily modified if desired.

2 SPI

2.1 General Description

The problem is to find the intersection between a parametric surface and an infinite plane, $ax + by + cz + d = 0$. From the convex hull property of B-spline surfaces, we can easily exclude the non-intersecting case by inspecting the location of the control points of the surface. This inspection can be further simplified by applying a transformation to the surface and the plane such that the plane equation is equal to $z = 0$. As will be shown later, this simplification also unifies the treatment of rational and non-rational surfaces.

2.2 Mathematical Details

2.2.1 Transforming a plane to the *nominal* position (z=0):

This operation contains two transformations: translating the plane such that it passes the origin, and orienting the plane such that the normal is aligned with the Z-axis. This is exactly the same operation as in CPI. Please refer to part I of the report.

2.2.2 Derivation of the intersection curve $f(u, v) = 0$:

Plugging in the component functions of the parametric surface $(x(u, v), y(u, v), z(u, v))$ into the plane equation (the nominal formulation), we get:

$$z(u, v) = Z(u, v)/W(u, v) = 0$$

Therefore,

$$f(u, v) \equiv Z(u, v) = 0$$

Note that if the surface is rational, in which case $W(u, v)$ is no longer a constant function, the formulation is still the same. That is, Z-component is the only effective term.

From the above, we know that SPI has a close form solution $f(u, v) = 0$. However, in geometric modeling, it is more important to know the trace of the intersection curve than the exact algebraic representation. Since this is a problem with one equation and two variables, continuation method [11] can be applied to trace the solution set $f(u, v) = 0$.

The evaluations of function, Jacobian, and Hessian of the function $f(u, v)$ come from the Z-components of surface evaluation, first derivatives, and second derivatives respectively. However, there are some important issues:

- There may be more than one components in $f(u, v) = 0$. How to make sure that enough starting points have been located such that no component is missing.
- How to realize whether a starting point has been traced by other branches such that no repetitive effort is spent on that point.

For the former issue, our approach can find all the necessary points of the visible curves, thanks to the robust CPI solver. Starting points for invisible curves are also found using a robust algorithm, which will be discussed later (section 2.2.4). For the second issue, the intersection curves are separated at the *characteristic* points. The bookkeeping of tracing record at these points is the key of our method.

2.2.3 Numerical Computation of Characteristic Points:

Since the intersection branches start from characteristic points, accurate computation of the characteristic points is crucial to curve tracing. The accuracy is achieved by numerical computation at these points. In the following, we will describe the definition and the formulation used in Newton iteration at each type of points.

border points : the intersection points at the boundary of the surface. Numerical computation of these points is actually an one-dimensional problem. These points are calculated accurately using the CPI routine.

u-turning points : A turning point is a point at which the tangent vector is parallel to either axis in the parametric space. U-turning points (with tangent parallel to v-axis) are the key to our curve tracing processing (see section 2.2.4). V-turning points (with tangent parallel to u-axis) are only useful in the case of the complication of border points (section 4). The following two equations are satisfied at a u-turning point:

$$f(u,v) = 0 \tag{1}$$
$$f_v(u,v) = 0 \tag{2}$$

With two variables and two equations, we can compute a turning point accurately using the Newton iteration, provided that a good initial guess is given. Note that if we use Newton-Raphson iteration, the function, and therefore the surface is assumed to be at least twice differentiable (C^2).

Continuity in general will not be an issue, unless the point happens to be exactly located at the boundary of a component subpatch where the continuity condition is not guaranteed. In that case, we need to break the surface into its component patches such that the continuity condition is satisfied [1].

singular points (nodes) : Nodes are the points at which the intersection curve crosses itself. At these points, three conditions are satisfied: $f = f_u = f_v = 0$. We formulate the problem to be the gradient map of the objective function ${f}^2 + {f_u}^2 + {f_v}^2$. That is,

$$f\ f_u + f_u\ f_{uu} + f_v\ f_{uv} = 0 \tag{3}$$
$$f\ f_v + f_u\ f_{uv} + f_v\ f_{vv} = 0 \tag{4}$$

In this formulation, we have the same concern of continuity. Actually, this formulation requires one order higher of continuity (C^3).

[1] Note that in general continuity should also be a concern for curve tracing. However, in continuation method, the requirement is only C^1 (Jacobian is needed for correctors in curve tracing). Based on the properties of B-spline surface, we can have a simple preprocessing routine to detect the tangential discontinuous features.

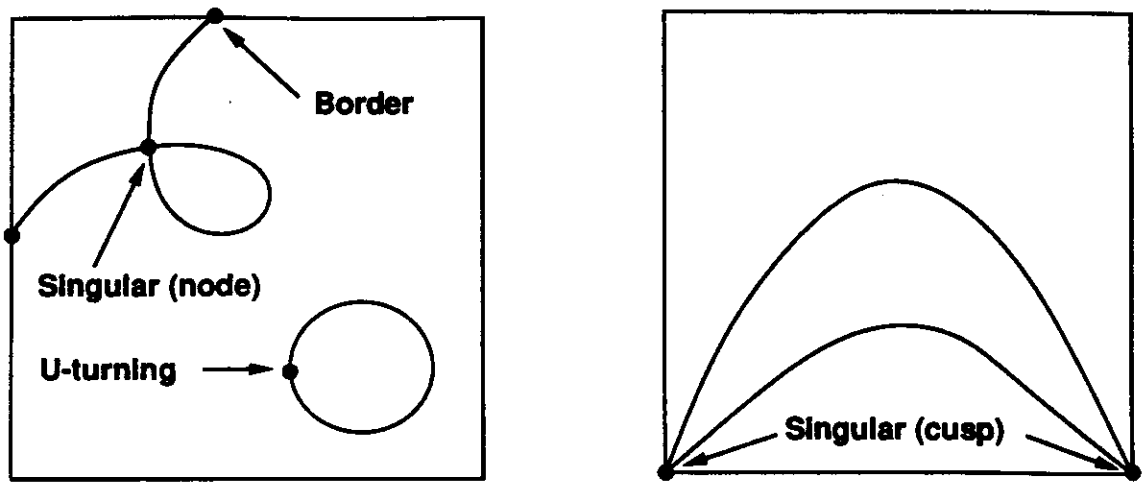


Figure 2: Schematic diagram of characteristic points

singular points (cusps) : the higher order singularity that forms a cusp. At these points, same conditions of nodes are satisfied: $f = f_u = f_v = 0$. Thus, the numerical formulation is the same as those of nodes. The important issue of cusp handling is to obtain the starting points for the branches emanating from the point. This will be discussed in a later section.

2.2.4 Finding all U-turning points:

As was previous mentioned, intersection curves are categorized into visible and invisible. Visible curves are easily discovered by solving the border points. For invisible curves, we utilize the following simple fact:

> *On every invisible curve, there is at least one u-turning point or one singular point.*

Thus, the solution strategy is to find all the u-turning points and singular points. That is, find all the points that satisfy $f = f_v = 0$. The nature of this problem is to find a methodology to solve *all* the roots in a system of equations. We have reviewed several methods: homotopy, interval arithmetic, and trajectory methods ([2], [5], [7], [9]). In the current implementation, we have experimented the following package: CONSOL developed by Dr. Alexander Morgan in General Motors Research Laboratory [9]. The package uses the homotopy concept and perfects its implementation in polynomial system. As is well known, finding the appropriate homotopy formulation is the most difficult problem. He has developed a systematic way of finding the homotopy equation that will solve the polynomial system.

To use this package, we need to formulate the problem as a polynomial system. If the surface is not a Bezier surface, we need to subdivide the surface into Bezier subpatches, which are by definition polynomials. Although the subdivision might create a lot more subpatches to be considered, many subpatches can be excluded because either they have no intersection with the plane, or it is possible to have invisible

curves in them. The time and storage spent on the overhead of subdivision is moderate, but the robustness is greatly enhanced.

The system is formulated as follows: This is a two-variable, two-equation problem. The equations are of the form:

$$f(u,v) = 0$$
$$f_v(u,v) = 0$$

By expanding the first one into monomial form [2], the first equation can be express as

$$f(u,v) = U_m(u) K V_n(v)^T = 0$$

where

$$U_m(u) = (1 \; u \; u^2 \; \cdots \; u^m),$$

$$V_n(v) = (1 \; v \; v^2 \; \cdots \; v^n),$$

and K is $(m+1) \times (n+1)$ coefficient matrix.

The efficiency of the homotopy method in this case is directly proportional to the number of paths explored, which is proportional to the total degree of the system. The following preprocessing routines are employed to reduce the degree, and thus to increase the efficiency.

- Delete the terms that are negligible because of small coefficients.
- Factor out any variable in each equation. Make sure that in every equation, the term of lowest degree in each variable is a constant.
- Introduce a new variable $x_3 = uv$, if this substitution can reduce the total degree. Notice that this substitution will increase the system size to three, but at this point, we assume that the total is the dominant factor of execution time. [3]

Interpretation of the results: CONSOL will calculate all the roots (real and imaginary) in the system. In general, we are only interested in the *real* roots that are *inside* the domain. However, the tolerance used for judging whether a root is real might vary depending on the type of the solution. For a regular simple solution (e.g. the u-turning point in a circle), the imaginary part is quite small (10^{-15}). As the degree of singularity gets larger (e.g, nodes and cusps), the imaginary part is also larger. It might be as large as 10^{-3}. Thus, the choice of the threshold of real roots needs to take the type of roots into consideration.

Infinite number of roots: CONSOL is designed to solve the system with (geometrically) isolated roots. Thus, we need to make an effort to exclude the case of

[2] We found that the numerical precision is not a big problem for this expansion if we use double precision arithmetics.

[3] Although this is the conclusion we drew from several experiments, this conjecture needs to confirmed.

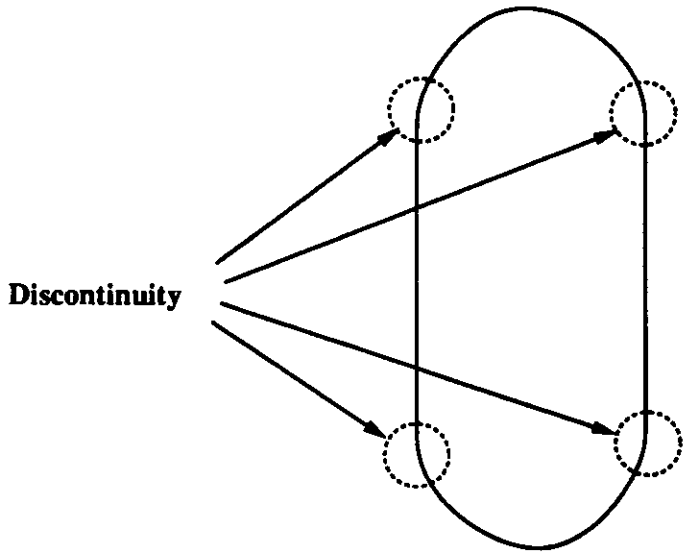


Figure 3: It is impossible to have a hidden straight segment in SPI.

non-isolated roots as best as we can [4]. In the current context, this effort is translated to extracting the presence of every non-isolated u-turning points. We claim that this task can be achieved if we can extract all the vertical straight component in the solution set. Furthermore, the extraction can be done by checking the location of the border points. Here are the explanations:

- The intersection between a Bezier surface and a plane is an algebraic curve. Thus, it is impossible to have a straight vertical segment connected to some curve branches, as shown in figure 3.
- If we ever discover some u-turning point belonging to part of vertical intersection line (with zero curvature), this point must belong to a visible vertical component of the intersection curve. And this intersection component must be of the form $u = u^*$, where u^* is a border point we discovered in CPI.

The extraction is implemented numerically. Our experience showed that with double precision arithmetic, it is quite reliable.

2.2.5 More on curve tracing:

There are two issues that we need to consider. First, usually we get more than enough starting points to complete all components of the intersection curve. In order to avoid tracing the same branch many times, we need to do some bookkeeping on the starting points (i.e., the characteristic points). That is, we need to *turn off* the appropriate

[4] In Chapter 4 of Morgan's book[9], it was mentioned that CONSOL will solve all other isolated roots, even with the presence of the non-isolated roots. However, this does pose some complexity of the solution process, which we want to avoid.

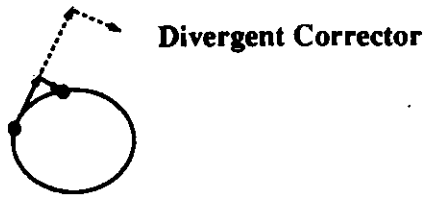


Figure 4: Inappropriate Predictor Size

starting directions during the curve tracing. The choice of the direction depends on the type of the characteristic point. For simple border points where only one branch can start, the choice is obvious. For the turning point, because its tangent is parallel to one of the coordinate axis, the choice of the branch is determined by comparing the position of previous points on this branch with the position of the turning point. For nodes, the choice is determined by comparing the *incoming* direction of the branch with the four branches emanating from the node. The choice for cusps is the most difficult. This will be discussed in a later section.

Second, there are some features that might hinder the efficiency of curve tracing. The curve tracing process employs the predictor-corrector method to follow the curve. If we use a big predictor to start off a small loop, the corrector step in continuation might fail to converge. In that case, we need to reduce the size of the predictor until the corrector behaves satisfactorily. This iteration process can be costly if we are dealing with high order surfaces. A better solution is to start off with the predictor of appropriate size determined from on the local geometry, e.g. the magnitude of the curvature at that point.

2.2.6 Current Development

In addition to CONSOL , we are experimenting another root solution package using interval arithmetic, GENBIS [7]. The initial experiments show that GENBIS is in general a lot more efficient. However, as reported in [7], GENBIS is poor in handling singular solutions. In our experience, for the configurations with nodes and cusps, the solution process will not stop until reaching the prescribed number of evaluations. Moreover, the robustness (in terms of the capability of solving every root) is sometimes worse than CONSOL . Since interval arithmetic method is used, it is especially vulnerable to the cases where there might be infinite number of roots. Thus, extracting non-isolated roots becomes crucial in applying this method. Because the efficiency gain of using GENBIS for the cases that worked is quite big, we are contemplating writing a high level decision routine to choose the appropriate solution methodology.

2.3 An Application — Slicing

Slicing is an operation to calculate a set of cross sections of a solid object. It is useful in many applications. For example, these cross sections are used in rapid prototyping technologies for constructing the layers of the model.

The criteria of a good slicing algorithm are twofold: The cross sections calculated must be accurate, and the operation must be reasonably efficient. Most of the parts that we want to deal with are described in free form geometry. We have two options of handling this kind of object: linearize the part (i.e., approximate the free form surface by linear facets) and employ linear intersection algorithms, or calculate the intersection directly for the free form geometry. For the former, the efficiency of intersection algorithm is in general better, but the accuracy depends on the approximation of the linear facets. To generate a good linear approximation of a free form object is quite costly, both in terms of creating and handling of the data. Thus, we have decided to tackle the problem by the latter option. With the SPI algorithm described in the previous section, we believe we can accomplish an accurate slicing result with reasonable efficiency.

There are two steps to generate a *slice* : Calculate the intersection curves between the plane and all the surfaces in the model, and then connect the intersection curves to form a loop[5]. In the following, we will discuss only the computation of intersection curves. Since we have to intersect the same surface with a set of parallel planes, we need to make some modification to the general SPI algorithm. Without the loss of generality, we assume that the plane and the surface have been transformed such that the plane is parallel to XY plane. That is, the cutting planes are of the equations $z = c_i$, where c_i is a distinct constant for each plane.

Plugging in the component functions of the surface $\mathbf{r}(u, v)$, we get:

$$z(u, v) = Z(u, v)/W(u, v) = c_i$$

Or,

$$f(u, v) \equiv Z(u, v) - c_i W(u, v) = 0$$

The critical points are the points at which the following equations are satisfied: $f_u = f_v = 0$. It is clear that within each invisible loop, there must be at least one critical point. If $\mathbf{r}(u, v)$ is a non-rational surface, $W(u, v)$ is a constant. Thus, the critical points for every *slice* will be the same (independent of c_i) . Thus, the time-consuming critical point calculation will only be necessary for the first slice. The slicing operation can be described in the following:

- Solve the visible curves using the same routines as in SPI.
- Trace the u-characteristic curve ($f_v = 0$) from each critical point computed in the first slice. The tracing stops when the point goes out of the boundary of the domain.
- Each intersection point ($f = 0$) discovered along the u-characteristic lines is a u-turning point. If this point has not been visited by the visible curves, start tracing the invisible segment of the intersection.

[5]Since the slices are taken from a solid, the contours for each slice are loops.

For rational surfaces where the homogeneous coordinate term W(u,v) is a function of u and v, the critical points for each *slice* will be different. The general SPI algorithm has to be used for each slice.

3 SSI

3.1 General Description

There are three requirements for a surface-surface intersection (SSI) algorithm: accuracy, efficiency, and robustness. Accuracy is measured by the deviation between the derived intersection curve and the actual curve lying on the participating surfaces. Efficiency is measured by the time and space needed for the operation. A robust algorithm should correctly identify every segment of the intersection curve. In the following, we review several types of SSI algorithms, and compare their achievements in these three requirements.

- **Analytic** approaches seek analytic expressions for the intersection of two algebraic surfaces [13]. Although it can reliably provide a closed form representation of the intersection, the complexity of the representation makes it impractical for the application of moderately high order parametric polynomial patches.
- **Subdivision** methods [6] decompose the surfaces into polygons or triangles and an approximation of the intersection is calculated from intersections of the linear facets. These methods may fail to compute all loops and branches for the intersections, due to the finite level of subdivision. Further increase of the subdivision level does not assure correct topology of the intersection, and can rather result in excessive computation. One advantage of the subdivision method is that bounding boxes for subpatches can be built on the basis of control polyhedra [15]. Further subdivision of subpatches can be eliminated if the bounding boxes do not intersect.
- The basic idea of the **lattice** method is to reduce the dimensions of the operand of the intersection ([1], [16]). A set of spatial curves lying on one surface (e.g., isoparametric curves) is defined and their intersection points with the other surface can be computed. These points are then sorted and connected to form an approximation of the intersection curve. The SSI problem is transformed into a series of curve-surface intersections, for which more robust algorithms are available. However, because of the discrete nature of the method, it is difficult to capture all small loops and singularities.
- By formulating the problem as a system of nonlinear equations, one can employ **marching** methods to obtain the intersection curve [4]. The critical issue is to provide all necessary starting points for every branch of the intersection curves. This is commonly done by intersecting linear approximations of the two surfaces. These methods are usually sensitive to singularities. Small numerical perturbation might result in the change of topology of the intersection curve. Nevertheless, since every point on the curve can be computed accurately by numerical techniques, marching methods are more accurate.

In this paper, we will describe another method we developed — the *Augmented Lattice* method. The basic idea is to discover all invisible intersection curves by locating the critical points of the distance function between the two surfaces. Since the distance function is quite complex, the powerful techniques used in SPI can no longer be applied. The strategy we use for exploring the nonlinear distance function is based on our

previous work [17]. The lattice evaluation is added to increase the robustness. Also, the path tracker is changed to a more accurate continuation solver [11]. In summary, the solution procedures for solving SSI is the following:

1. Locate all visible curves — solve the CSI subproblems. Since the operands are assumed to be untrimmed surfaces, eight CSI problems need to be solved. Bounding boxes are used to increase the efficiency of the operation.
2. Trace all visible curves. Record the u-turning points encountered on the paths.
3. *Augmented Lattice:*

   Step 1 : Generate a set of sampling lines on one surface. Explore the distance function along these sampling lines. Record the positions where the gradient vectors (of the distance function) are perpendicular to the sampling lines. The sampling lines are the isoparametric lines at the node position of the more complicated surface.

   Step 2 : For each interval between the sampling lines, trace the *U-characteristic* lines. The goal of this tracing is to find the u-turning points and singular points of the intersection curve.
4. Trace the invisible curves starting from the new u-turning points.

Comparing this approach with the criteria of a good SSI algorithm, we conclude:

Accuracy It is important to make sure that every intersection point is accurately computed, and that the deviation of the interpolation segment is within the tolerance. In the current approach, the interpolation points are calculated using continuation method. Thus, the accuracy is guaranteed to be satisfied (unlike the ODE tracing method). The deviation can be estimated using heuristic rules.

Efficiency The performance of our prototype implementation is similar to the data presented in [8]. It is our opinion that the execution time is not a good comparison, because the performance depends on the actual implementation. However, the algorithm does not use subdivision, which is always a benefit in saving the extra calculation and storage. Moreover, since the CSI subproblems (the border point computation and the sampling lines exploration) are done independently, this algorithm is suitable for parallel implementation.

Robustness We have successfully tested the implementations of the previous [17] and the current works over all the cases listed in [8]. Moreover, because of the modification of additional sampling lines, the current implementation is more robust than our earlier work.

Misc. Since we work directly on the distance function, it can be applied to the intersection of offset surfaces, without much modification.

The details of the implementation will appear in a later report. Here we only discuss the technical details of this problem.

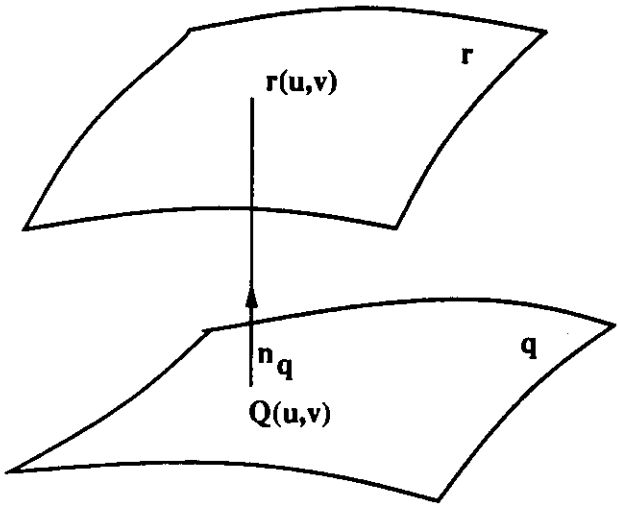


Figure 5: Orthogonal Projection from a point to the surface

3.2 Mathematical Details

3.2.1 Problem Formulation:

This problem can be formulated from several perspectives. Each of the following formulations is used in the appropriate context.

- Formulation 1 (equivalence of position in R^3):

$$\mathbf{r}(u,v) - \mathbf{q}(s,t) = 0 \tag{5}$$

  This formulation has three equations and four independent variables.

- Formulation 2 (distance function) ([10],[8]):

$$\phi(u,v) = \mathbf{n}_q(\mathbf{Q}) \cdot (\mathbf{r}(u,v) - \mathbf{Q}(\mathbf{r}(u,v))) \tag{6}$$

  where $\mathbf{Q}(\mathbf{r}(u,v))$ is a point on surface $\mathbf{q}(s,t)$ that is nearest to the point $\mathbf{r}(u,v)$, and $\mathbf{n}_q$ is the unit normal vector at $\mathbf{Q}$ on the surface $\mathbf{q}$, as shown in figure 5. The value of this function at a given point on surface $\mathbf{r}$ is the *signed* distance from that point to the corresponding $\mathbf{Q}$. Note that if $\mathbf{Q}$ is in the interior of the domain, it is also an orthogonal projection point and the following two equations are satisfied:

$$(\mathbf{r}(u,v) - \mathbf{q}(s,t)) \cdot \mathbf{q}_s(s,t) = 0 \tag{7}$$

$$(\mathbf{r}(u,v) - \mathbf{q}(s,t)) \cdot \mathbf{q}_t(s,t) = 0 \tag{8}$$

  And the partial derivatives of the distance function ϕ are

$$\phi_u = (\mathbf{n}_{q,s}\frac{\partial s}{\partial u} + \mathbf{n}_{q,t}\frac{\partial t}{\partial u}) \cdot (\mathbf{r} - \mathbf{q}) + \mathbf{n}_q \cdot (\mathbf{r}_u - \mathbf{q}_s\frac{\partial s}{\partial u} - \mathbf{q}_t\frac{\partial t}{\partial u}) = \mathbf{n}_q \cdot \mathbf{r}_u \tag{9}$$

  Similarly,

$$\phi_v = \mathbf{n}_q \cdot \mathbf{r}_v \tag{10}$$

  The formula for higher order partial derivatives can be found in [8].

- Formulation 3:

$$\mathbf{n}_q(s,t) \cdot (\mathbf{r}(u,v) - \mathbf{q}(s,t)) = 0 \tag{11}$$

$$(\mathbf{r}(u,v) - \mathbf{q}(s,t)) \cdot \mathbf{q}_s(s,t) = 0 \tag{12}$$

$$(\mathbf{r}(u,v) - \mathbf{q}(s,t)) \cdot \mathbf{q}_t(s,t) = 0 \tag{13}$$

This is a more complicated way of formulating a three-variable, four-equation problem (as compared to formulation 1). In this formulation, the parameters (u, v, s, t) are regarded as **independent** variables. Note that eq(11) gives the notion of distance function as described in the second formulation, and that eqs. (12) and (13) are the same orthogonal conditions as in the second formulation.

3.2.2 Continuation Formulation:

Among the formulations described above, it is important to choose the appropriate one for different context.

- For tracing SSI curves, it is advantageous to choose the least expensive formulation, equation (5).
- In the exploration of the distance functions, the second and third formulations are more appropriate.
- The *U-characteristic* lines mentioned in section 3. 1 are important in the tracing of invisible intersection curves. It is defined to be the solution set of $\phi_v = 0$. Between the two distance function definitions, we choose the third formulation for its simplicity (eqs. 11 to 13). That is,

$$\begin{aligned} f_1(u,v,s,t) &= \phi_v = \mathbf{n}(s,t) \cdot \mathbf{r}_v(u,v) = 0 \\ f_2(u,v,s,t) &= (\mathbf{r}(u,v) - \mathbf{q}(s,t)) \cdot \mathbf{q}_s(s,t) = 0 \\ f_3(u,v,s,t) &= (\mathbf{r}(u,v) - \mathbf{q}(s,t)) \cdot \mathbf{q}_t(s,t) = 0 \end{aligned}$$

3.2.3 Numerical Computation of Characteristic Points:

As described in the SPI section, characteristic points are used to facilitate the bookkeeping of the curve tracing process. Notice that we only need to define the characteristic points on the first of the two operands in the Boolean operation, since the connectivity information can be processed with just one set of points. In the following, we will describe the numerical formulations for different types of points in the SSI context.

u-turning points : The points on the intersection curve with vertical tangent vectors.

From the second formulation (6), the following equations are satisfied at a u-turning point:

$$\phi(u,v) = 0 \tag{14}$$

$$\phi_v(u,v) = 0 \tag{15}$$

subjected to

$$(\mathbf{r} - \mathbf{q}) \cdot \mathbf{q}_s = 0 \tag{16}$$

$$(\mathbf{r} - \mathbf{q}) \cdot \mathbf{q}_t = 0 \tag{17}$$

Note that *nested* Newton iterations is needed in this case. That is, the outer iteration is designed to converge to a u-turning point by adjusting the (u, v) (eqs(14) and (15)), while the inner iteration is designed to maintain the orthogonal conditions, (eqs(16) and (17)). Another set of simpler but equivalent conditions can be derived using the third formulation (eqs(11) to (13)):

$$\mathbf{r}(u, v) - \mathbf{q}(s, t) = 0 \tag{18}$$

$$\mathbf{n}(s, t) \cdot \mathbf{r}_u(u, v) = 0 \tag{19}$$

Here the problem is a system of four variables. No nested Newton is needed but the size of the system is larger than the previous one. Note that equation (19) is designed to converge to a u-turning point (see eqs. 15 and 10), and the first three (19) will guarantee to stay on the intersection curve (thus satisfying eqs (15), (16) and (17)).

Singular points (nodes) There is one more condition to be satisfied than the u-turning point, $\phi_u = 0$. This extra equation will make the system overdetermined. In the SPI case, we formulate the system to be the gradient map of a functional: $f^2 + f_u^2 + f_v^2$. In the current case, we can get away in a cheaper way by some not-very stringent assumptions.

We know the following equations are satisfied at a node:

$$\mathbf{r}(u, v) - \mathbf{q}(s, t) = 0$$

$$\mathbf{n}(s, t) \cdot \mathbf{r}_u(u, v) = 0$$

$$\mathbf{n}(s, t) \cdot \mathbf{r}_v(u, v) = 0$$

We can pick any two components of the first equation. Together with the remaining two equations, we can form a four-variable system. The assumption is that if the initial guess is accurate, in the neighborhood of the solution, there should not be a pair of points which have the same projection on z-plane (i.e. same x and y components), but with different z values. The heuristic used in choosing two out of three equations is to pick the two coordinates with larger component in $(\mathbf{r} - \mathbf{q})$.

Bifurcation on U-Characteristic Line On the U-characteristic line, $\phi_v = 0$. Thus, the nodes on the line satisfy $\phi_v = \phi_{vu} = \phi_{vv} = 0$. This problem can be solved using the same method as the singular point in SPI (forming the gradient map of a functional). But because of the expense of calculating the high order derivatives of the distance function, the current implementation uses the latter two equations in the numerical iteration and checks the equivalence of the first equation. The experimental results are encouraging. The reason is that since the initial point is not far away from the final solution, the converged point should also be on the U-characteristic curve.

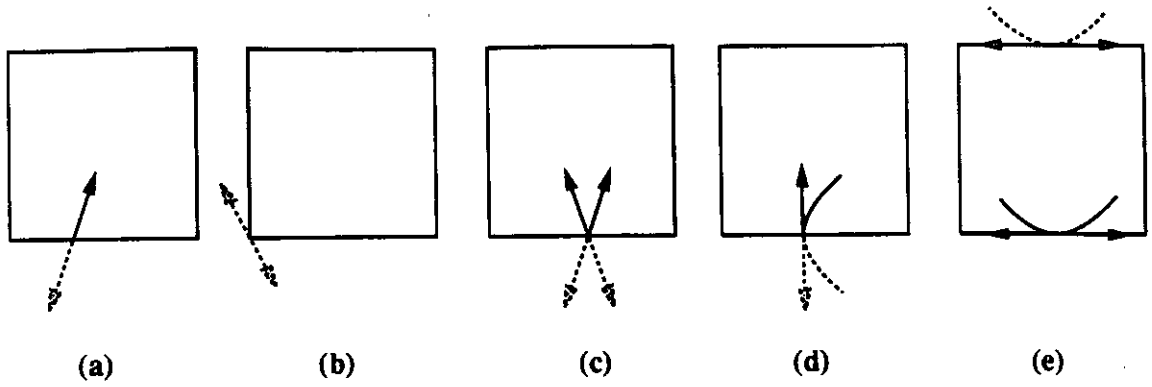


Figure 6: Validity of branches from a border point. Dashed lines (arrows) are invalid segments (directions).

4 Border Point Complication

4.1 Description

As mentioned earlier, border points are the intersection points calculated from the intersection of lower dimensional entities. It was also mentioned that the border points are the starting points of the visible intersection curve. Thus, it is important to know how many branches can emanate from a border point. First, some definitions: If a border point satisfies the condition of other types of characteristic point (e.g., turning points or singular points), we call it a *complicated* border point; otherwise, we call it a *simple* border point. Given a non-singular intersection point, we can always find the tangent direction along the intersection curve at this point. For a singular point, we can find the tangent directions for every branch emanating from that point.

The validity of a branch is determined by checking whether the tangent direction is pointing inside the valid domain. This check is useful for the *simple* border points and the singular points (nodes and cusps). However, if the border point is also a turning points whose tangent is parallel to the boundary line (case (e) in figure 6), the above information is not conclusive. Therefore we need to use the second order criteria to determine the validity of the branch. Note that if the branches are not valid at a turning point, this intersection point will be classified as an isolated intersection point.

4.2 The Turning Points

Turning points are the points where one of ϕ_u and ϕ_v is zero. but not both. That is, the gradient of the *distance* function at that point either parallel or perpendicular to the border line. If the gradient is parallel to (in the border point case, lying on) the border line (case (d) in figure 6), it is clear that only *one* of the branches goes into the domain, and the direction is perpendicular to the border line. On the other hand, if the gradient is perpendicular to the border line (case (e) in figure 6), the curve touches at the border and there are two cases: either the curve bends inward, or the curve bends outward. In the latter case, the touch point is the only valid intersection point in the domain.

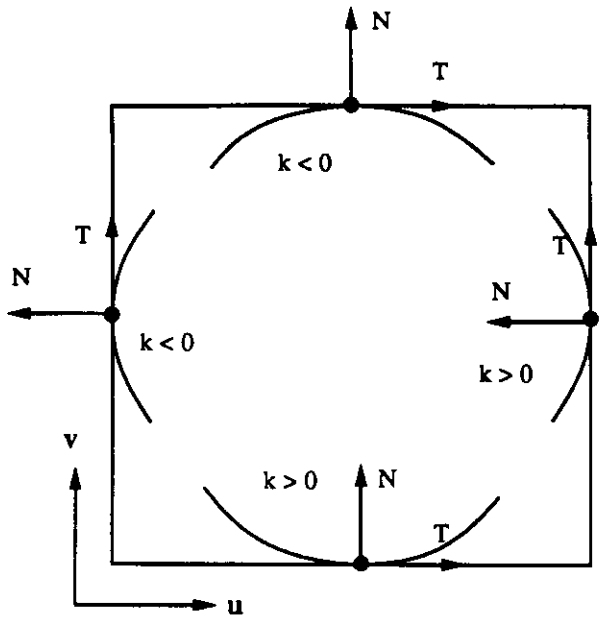


Figure 7: Complication of Border Points

To determine whether there are valid branches out of the turning point, we employ the local parameterization and the curvature information as follows.

We will explain the case of V-turning points in the following. The results of U-turning points can be derived in a similar fashion. In a planar parametric curve, the principle normal direction is defined to be 90 degrees counterclockwise from the tangent direction [3]. In the current case, there is no particular direction associated with the intersection curve. We choose, without loss of generality, that the curve goes in increasing u direction, as shown in figure 7.

The curvature for a planar parameterized curve $\gamma(t)(= u(t), v(t))$ is

$$k(t) = \frac{\dot{u}\ddot{v} - \ddot{u}\dot{v}}{(\dot{u}^2 + \dot{v}^2)^{\frac{3}{2}}}$$

It is sufficient to inspect the sign of curvature only. Thus,

$$Sign(k) = Sign(\dot{u}\ddot{v} - \ddot{u}\dot{v})$$

Since we take the local parameterization as $t = u$, and $\frac{du}{dt} = 1$, $\dot{u} = 1$, $\ddot{u} = 0$. Thus,

$$Sign(k) = Sign(\ddot{v})$$

Along the intersection curve,

$$f(u, v) = 0, \quad f_u du + f_v dv = 0, \quad \frac{dv}{du} = -\frac{f_u}{f_v}$$

$$\ddot{v} = \frac{d}{du}(\frac{dv}{du}) = -\frac{f_v \frac{df_u}{du} - f_u \frac{df_v}{du}}{(f_v)^2} = -\frac{f_{uu} f_v^2 - 2 f_{uv} f_u f_v + f_{vv} f_u^2}{f_v^3}$$

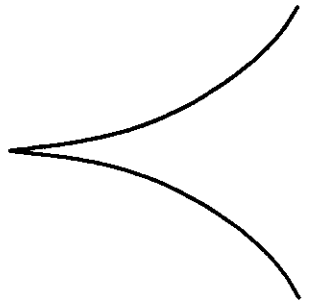
Thus,

$$Sign(k) = Sign(-\frac{f_{uu}f_v^2 - 2f_{uv}f_uf_v + f_{vv}f_u^2}{f_v^3}) \quad (V - turningPoint)$$

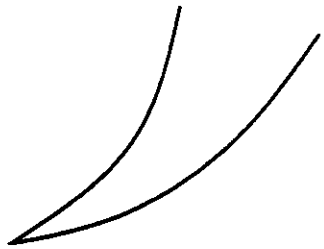
Similarly,

$$Sign(k) = Sign(\frac{f_{uu}f_v^2 - 2f_{uv}f_uf_v + f_{vv}f_u^2}{f_u^3}) \quad (U - turningPoint)$$

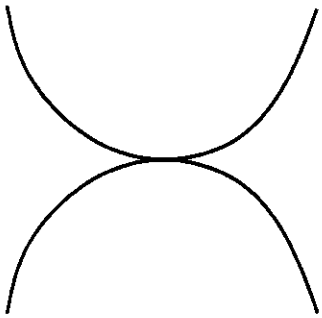
The function f is either the Z-component of the surface evaluation in SPI, or the distance function ϕ in SSI.



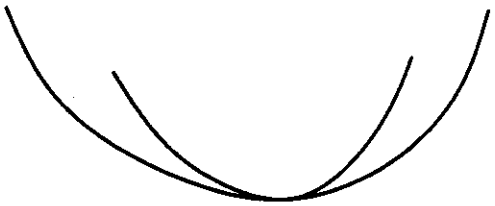
Cusp of 1st Kind



Cusp of 2nd Kind



Double Cusp of 1st Kind



Double Cusp (Tacnode)

Figure 8: Different Types of Cusp

5 Cusp

5.1 Description

Singularity can occur on the intersection curves of SPI and SSI. The dicussion of the self-intersecting points (nodes[6]) can be found in [8]. Another kind of singularity may occur — *cusps*. Cusps are the singular points where the limit of tangents on the branches coincide. It is called a *cusp of the first kind* if the two branches are at the opposite sides of the common tangent, and a *cusp of second kind* if they are on the same side. It is a *double cusp* if the curves extent on both sides of the cusp. We call a cusp point that is not a double cusp a *single* cusp.

Just like other types of characteristic points, two issues need to be addressed in curve tracing: The calculation of the branches, and the termination of certain branch directions (see section 2.2.5). These problems are more complicated at a cusp since the branches near the cusp share a common tangent. In the following sections, we will present the numerical solutions to the above issues.

[6] In some literature, the self-intersecting point is referred as the *crunode*; the cusp is referred as the *spinode*.

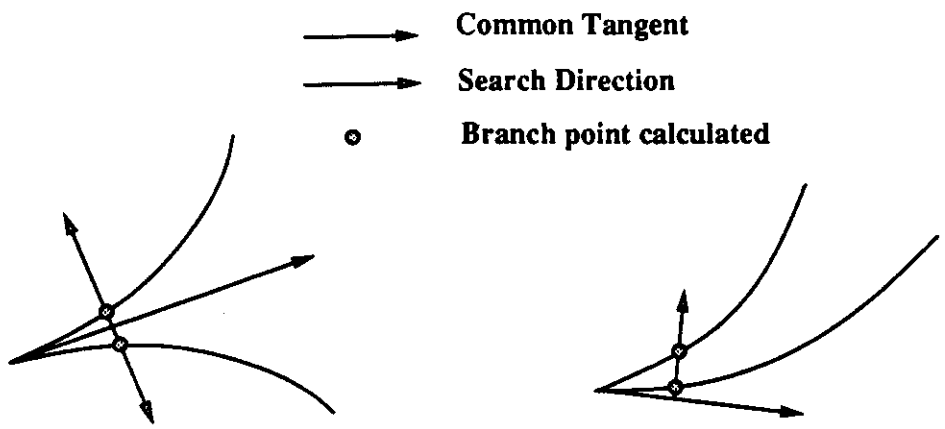


Figure 9: Calculate the branch points.

5.2 Algorithms

5.2.1 Calculate the branch points:

To start the curve tracing *from* a cusp, we need to obtain the starting points on every branch of the cusp. We develop a *numerical* method to calculate the branch point of a cusp as follows:

- Perturb along the common tangent of the cusp. Unless the cusp point is on the boundary of the domain, this perturbation is performed on both orientations of the tangent (the possibility of being a double cusp). It is theoretically difficult to distinguish a double cusp from a single cusp.
- For each orientation, starting from the perturbed point, search for the branches on both sides of the tangent along a direction perpendicular to the tangent. The branch points are the points that satisfy $f(u,v) = 0$, where $f(u,v)$, as defined in previous sections, is either the Z-component functions in SPI, or the distance function ϕ in SSI. This is a one dimensional search problem, since u and v are related by the search direction. The search implementation is similar to the one in CPI (see part I of the report). Note that the search is terminated once two branches are found.

The success of the implementation depends on the numerical values chosen for the perturbation along the tangent, and the step size used in the one dimensional search. The perturbation has to be significantly large such that the branches are numerically distinguishable. A large step size in the one dimensional search is prone to missing the branch point, especially for the case of cusp of second kind.

5.2.2 Determine the incoming branch:

As mentioned in section 2.2.5, in order to avoid duplicate tracing, we need to do the bookkeeping on starting points. The key to this bookkeeping is the comparison between

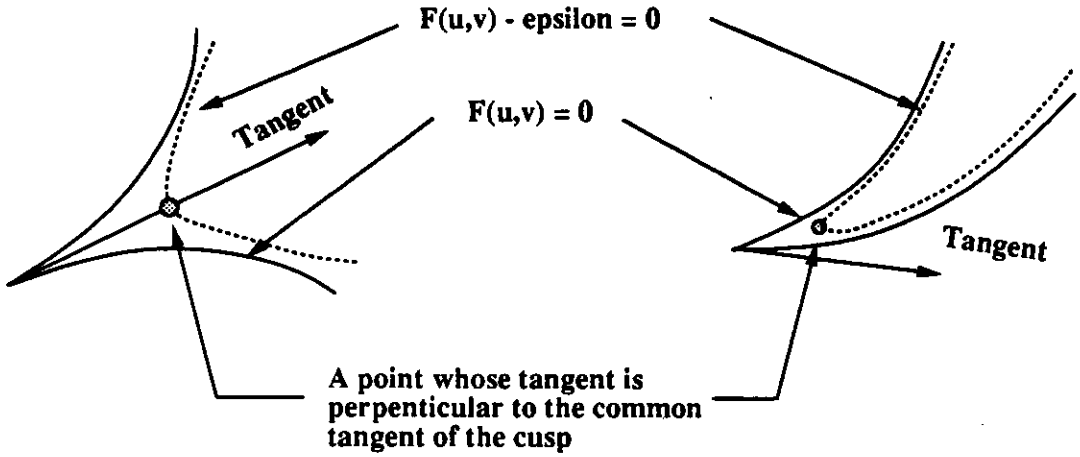


Figure 10: Determine the incoming branches of a cusp

the incoming direction and the branch direction. However, this is not applicable for single cusps, since both branches of a single cusp have the same tangent orientation [7]. Initial investigation suggests that the information of the curvature can be used to identify the branches for the cusp, but the usage is limited to cusps of the first kind. A more robust method is developed in the following such that the handling of both kinds of cusp is uniform.

The basic idea to handle the branch classification is the following fact:

> *For every cusp, there exists a separation line that passes through the cusp point, and separates the branches on both sides of the line.*

Let us discuss the concept more carefully. The intersection curve that contains the cusp is an instance of the level set of the function $f(u, v) = 0$. Observe the trace of the perturbed function, that is, the curve of the function $f(u, v) - \epsilon = 0$. It is always possible to find a point whose tangent is perpendicular to the common tangent of the cusp in the neighborhood of the cusp, as shown in figure 10. The branch on one side of the separation line differs from that of the other side by a sign, which denotes the orientation of the vector formed by the cross product of the tangent of the cusp and the local gradient vector. Thus, the sign for each branch can be determined by the branch points calculated in the last section. The incoming direction is determined by the sign of the last points in the curve tracing.

5.2.3 Cusp Branch Handling:

In section 2.2.5, we have mentioned the issue of branch management. The key idea is to avoid duplicate tracing by turning off the starting direction at the end of the branch. This method poses a problem for the cusps. The issue is that cusps are less desirable points to start off the curve tracing. Because the branches from a cusp are close, any over-sized predictor might go to another branch. On the other hand, to determine the incoming direction of a cusp is difficult. Although we have developed a robust method

[7] However, we can use this to distinguish the orientation of a double cusp.

in the last section, the continuation points might have already wandered to the other branch before we actually do this testing. There are remedies for both problems: If we want to start from a cusp, we need to pay extra caution on the size of the predictor at the beginning, until we are reasonably far away from the singularity. After we have decided the branch is to be terminated at a cusp, we need to check the incoming continuation points and decide which is actually the incoming branch that *most* of the points are following.

References

[1] R. E. Barnhill, G. Farin, M. Jordan, and B.R. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4:3–16, 1987.

[2] F.H. Branin. Widely convergent method for finding multiple solutions of simultaneous nonlinear equations. Technical report, IBM Journal of Research Development, September 1972.

[3] J.W. Bruce and P.J. Giblin. *Curves and Singularities, a geometrical introduction to singularity theory.* Cambridge University Press, 1984.

[4] 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, 1988.

[5] L.C.W. Dixon, J. Gomulka, and G.P. Szego. *Towards Global Optimisation*, pages 29–54. North-holland, Amsterdam, 1975.

[6] E. G. Houghton, R.F. Emnett, J.D. Factor, and C.L. Sabharwal. Implementation of a divide-and-conquer method for intersection of parametric surfaces. *Computer Aided Geometric Design*, 2:173–183, 1985.

[7] R.B. Kearfott. Some tests of generalized bisection. *ACM Transactions on Mathematical Software*, 13(3):197–220, September 1987.

[8] G. A. Kriezis. *Algorithms for Rational Spline Surface Intersections.* PhD thesis, MIT, 1990.

[9] A. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[10] J. Pegna and F.-E. Wolter. Designing and mapping trimming curves on surfaces using orthogonal projection. Chicago, September 1990. ASME.

[11] Werner C. Rheinboldt. *Numerical Analysis of Parameterized Nonlinear Equations.* Wiley, New York, 1986.

[12] T. W. Sederberg and R. J. Meyers. Loop detection in surface patch intersection. *Computer Aided Geometric Design*, 5:161–171, 1988.

[13] T.W. Sederberg, D.C. Anderson, and R.N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing*, 28:72–84, 1984.

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

[15] J.U. Turner. Accurate solid modeling using polyhedral approximations. *IEEE Computer Graphics and Applications*, pages 14–28, May 1988.

[16] K.-Y. Wang. Intersections of general parametric surfaces. Technical Report 89CRD003, Automation Systems Laboratory, GE Research and Development Center, January 1989.

[17] Y. Wang, E.L. Gursoz, J.-M. Chen, F.B. Prinz, and N.M. Patrikalakis. Intersection of parametric surfaces for next generation geometric modelers. In J. Turner, J. Pegna, and M. Wozny, editors, *Product Modeling for Computer-Aided Design and Manufacturing*, pages 75–96. North-Holland, 1991.