Intersection Algorithms for Nonlinear Geometric Modeling: Part I

J. Chen

EDRC 24-75-91

Intersection Algorithms for Nonlinear Geometric Modeling *

Part I

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 first part of this report describes the intersection routines of lower dimensions.

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

1 Introduction

The next generation computer aided geometric modelers will combine the paradigms of wireframe, surface and solid modeling. A designer will be able to incrementally convert a wireframe sketch into a solid representation. In order to achieve this goal, the underlying modeling system must be able to automatically classify the modeling space as the geometric entities are updated. Systems with such capabilities have been developed recently ([12, 2]), but the domains are limited to objects bounded by planar facets and natural quadrics (e.g., cylinder, cone and sphere). This limited geometric coverage is not sufficient for more sophisticated applications, such as automobile design (e.g., designing the hood of a car). One of the major obstacles for extending the geometric coverage to include parametric curves and surfaces is the computation of the intersections among entities of different dimensionalities.

Before going into the details of these algorithms, we would like to discuss the similarities and the differences between the nonlinear and the linear algorithms. In a B-rep geometric modeler, an object is represented by vertices, edges, faces, solids, and the connections among the entities. In order to perform the basic Boolean operations between two objects, one needs to carry out the intersection tasks on the entities of different dimensionality from the two objects. It has been shown that it is more desirable to perform the intersections in a bottom-up order [3]. The reason is that the operations in lower dimensions are faster than those in higher dimensions. Also, the results in lower dimensional intersection is essential for the intersection of higher dimensional intersection routines. This principle is true for both linear and nonlinear algorithms. However, the difference between the linear and nonlinear algorithms is that the intersection of nonlinear entities is less predictable than that of the linear ones. For instance, in linear geometry, if two edges have the same end vertices, we can deduce that the two edges coincide. However, this is not true in nonlinear geometry. Furthermore, the result of face/face intersection is deductible from the results of lower dimensionality (edge/face, vertex/face, etc.). But, nonlinear face/face intersection is much more difficult, and less predictable. In fact, it is due to this very fact that people have devoted their efforts on the problem of *loop detection*, which deals with the intersection curves that are not predictable from the lower dimensional results ([7], [10]).

In these two technical reports, we will describe the intersection routines developed at Engineering Design Research Center of Carnegie Mellon University. The need for extending the geometric coverage of the non-manifold geometric modeler *Noodles* [2] initiates the development of this work. We choose the Non-Uniform Rational B-Spline (NURBS) to be the major representation for the nonlinear geometry. The reasons of this choice are the completeness and the popularity of this representation. Frequently used entities, such as the natural quadrics and the torus, will be approximated using NURBS. Note that the NURBS representation of those entities can be exact if rational arithmetic is used. The linear entities (i.e., line segments and polygons) still remain their linear representations. In part one of the report, we will concentrate on the intersection of dimensions lower than one. In part two, we will discuss the intersection of two dimensional entities.

General Comments:

- Trimmed entities are those which have the same mathematical mapping between the parametric space and R^3 , but only have a subset of the regular parametric domain as their domain. This type of entity is commonly used in design, and is also used to represent the result of the Boolean operations. The intersection algorithms presented in these two reports are for *untrimmed* entities only. Post processing needs to be employed to adjust the results to the appropriate domain. The handling of trimmed curves is trivial. The handling of trimmed surfaces is more complicated. One needs to classify the intersection curves obtained in the untrimmed surface with respect to the trimmed domain.
- Continuation method, in these reports, refers to the method for solving a series of solution points to a problem with n variables and $n-1$ equations [6]. This type of formulation occurs in several places of the work (e.g., CCI, CSI, SPI and SSI).
- The references mentioned in these reports are for comparisons only. The more complete literature survey will appear in a later report.

In the following two sections, we will describe the intersection algorithms involving one-dimensional entities only, namely CLI and CCI. These problems are in general easier than the higher dimensional ones (e.g., CSI and SPI), since the search spaces are only one-dimensional. Two types of intersection can occur in this category: (geometrically) isolated intersection points, or tangentially contact segments. In the literature, most of the work is devoted to finding solutions of the former type. However, in developing a nonlinear geometric modeler, the capability of handling tangential contact is equally important.

2 CLI

2.1 General Description

If the curve and the line are not tangentially contact, the solution of CLI is geometrically isolated. For this type of solution, the CLI problem can be solved as the following:

- The line segment is first extended to an infinite line.
- Find two planes such that the line corresponds to the intersection of the planes.
- Solve the CPI problem with one of the planes obtained in the previous step.
- Check the result against the second plane.

The advantage of this approach is that the CPI subproblem has the most robust and efficient algorithm among all the intersection algorithms (see section 4). However, some other useful information is not available in this efficient implementation, e.g., the closest point (and distance) between the line and the curve. If such information is desired, one can transform the line into a parametric curve and use the more elaborate (and less efficient) CCI algorithm in the next section.

To find the embedding planes for CPI subproblem, we want to avoid picking a plane which the curve lies exactly on. Since we have already excluded the case of tangential contact between the line and the curve, it is always possible to find such a plane.

The tangential contact in CLI can be easily detected if the curve is represented in NURBS. In general, a Bezier curve can not be linear unless all the control points are aligned with some certain line. However, since a B-spline curve is composed of several pieces of curve, it is possible to have linear components within its domain. The detection of linearity can be done by decomposing a NURBS curve into its Bezier components, and then apply the same check as in the Bezier curve.

2.2 Mathematical Details:

2.2.1 Generating the embedding planes:

A plane can be defined by a face normal and a point. In the present case, we know that the plane must pass through one end point of the line segment, and the face normal must be perpendicular to the tangent of the line. To ensure that the embedding plane will not be in tangential contact with the curve, we obtain the face normal by

$$normal = CrossProduct(TangentOfLine, RandomVector)$$

By the independence principle mentioned in [4], a random vector is not possible to align with the line; thus the normal is valid. Using the same argument, we have assured that that plane will not include the curve.

2.2.2 Post Processing:

After the CPI subproblem is solved, the results are tested to see whether it is in the given line segment. This is done by checking the intersection point against the parametric equation of the line segment.

3 CCI

3.1 General Description

Let us first discuss the intersection of planar curves. Since the problem forms a system of two variables, one might consider using the linear approximation method: calculate the initial guesses of the intersection from the intersection of the linear approximations, and then use numerical method to obtain more accurate solutions. However, this is not reliable as shown in figure 1. Recently, there is another approach proposed in the literature, *Bezier Clipping* [8]. The basic idea is to use the divide-and-conquer scheme with the geometric insight of convex hull property of the Bezier curves. The performance is quite impressive for solving regular intersection points, but is not extendible to higher order tangential contact.

For the case of space curves, one solution is to solve the intersection of the three planar projected curve, and post-process the results. This is a feasible solution because of the following fact: *if two space curves intersect, the two projected curves pairs must*

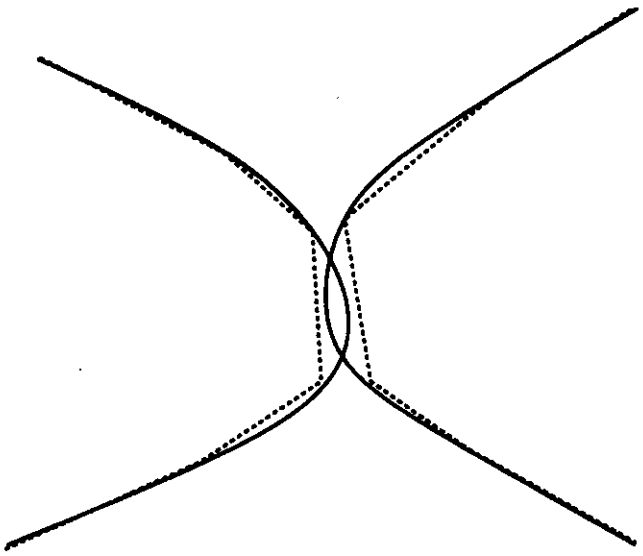

Figure 1: Linear approximation may miss the intersection.

also intersect. Mathematically, this corresponds to solving an overdetermined system by ignoring one of the equations. However, there are two disadvantages to this method. First, the projected curve might have undesirable features, e.g., self intersection [1]. Second, it is difficult to address the problem of tangential contacts of higher order by the projected curves. In the following, we will propose a *projection* method to solve the CCI problem, for both planar and space curves. The key ideas are the following:

- The first curve is projected onto the second one.
- The intersection point is detected by the appropriate *local projection* and then accurately calculated with the initial guess.

The tangential contact between two curves are not as well defined as the case of CLI. Thus, we use the following *numerical* definition: After the projecting the first curve onto the second one, if the distance between a series of projection pairs are smaller than some tolerance, then we can say that the two curves are tangentially contact in this interval. If this happens, we need to further determine the start and the end of this (tangentially) intersecting segment.

3.2 Mathematical Details:

3.2.1 Continuation equation:

The orthogonal projection can be formulated by the following equation:

$$(\mathbf{r}(u) - \mathbf{q}(s)) \cdot \mathbf{q}_s(s) = 0 \tag{1}$$

where $\mathbf{q}$ is the curve to be projected onto. This is an one parameter problem and we use the standard continuation technique to solve this problem [6]. In order not to miss

[1] For example, the projection of a spiral onto a plane can become a circle.

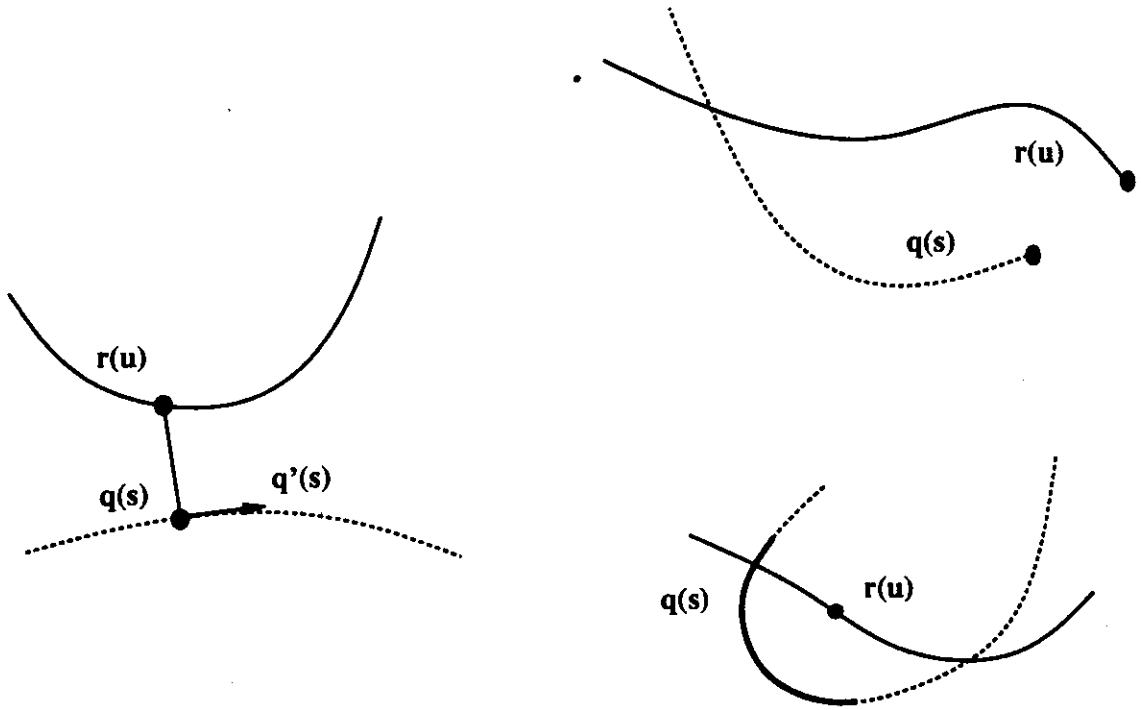

Figure 2: CCI projection cases: (a) orthogonal projection, (b) *ill* projection, and (c) multiple projection

any intersection, we dynamically adjust the step size in the continuation based on the distance between the curves.

We use the term *projection pair* to denote the point on the first curve and its projected image on the second curve. The projected point on the second curve is always the closest point to the point on the first curve. In some cases, the projected point might not be an orthogonal projection point. That is, eq(1) is not satisfied at this projection pair. We call this pair the *ill projection* pair. In this situation, we will continue to march on the first curve until the projection *status* changes. Note that during ill projection status, the *projection* point will be on either end of the second curve. There is another exception of the projection pair. Sometimes the point on the first curve marches to a point where there are multiple points on the second curve that have the same distance. We claim that this can be detected in the continuation formulation. The solution to this situation is to perturb the point on the first curve slightly, find a new projected point on the second curve, and restart the continuation process.

Although the design of the algorithm is not symmetric, the choice of the first curve (of the two) should not affect the validity of the algorithm. However, due to the factor of efficiency, the more complicated one (the one with higher order) will be chosen as the first curve.

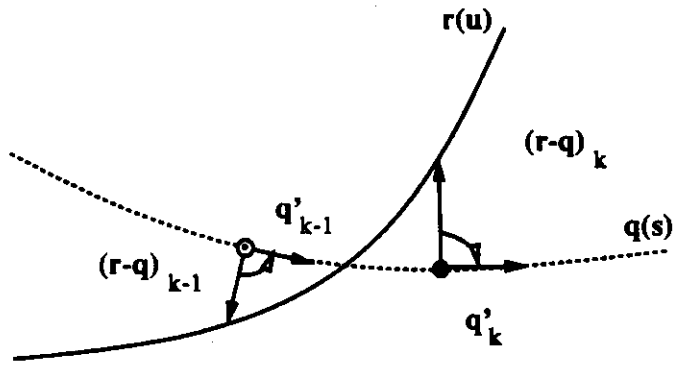

Figure 3: Root Detection

3.2.2 Root Detection:

In this problem, mathematically we have three equations and only two unknowns. Thus, the intersection can be detected using two out of the three equations. Geometrically, this corresponds to projecting the two curves onto one of the coordinate planes (XY, YZ, or ZX). The choice of the equations (or the projection plane) is determined dynamically based on the values of the components at that time.

For simple roots, we record the values of the following expressions for every pair of projection points:

$$((\mathbf{r}-\mathbf{q})\times\mathbf{q}_s)\cdot e_i,\quad i=0,1,2$$

Once a sign change is detected, the projection plane is determined, and we can use typical root finding techniques (e.g., bisection and Newton) to obtain the accurate solution. To catch higher order roots (tangential touching points), we need more sophisticated schemes (similar to the sampling method in section 4). Note that the solution needs to be verified for the equivalence of all three coordinates.

3.2.3 Tangential contact of curves:

The result of the tangential contact, as defined in the last section, should be represented as point-*pairs*, indicating the start and the end of the tangential segment. It is important to calculate the *branching* positions of the tangential segment. This computation is done by adding an additional equation specifying the *branching* tolerance. Together with the condition orthogonal projection (eq 1), we can solve the determined two-variable system, using the current ending projection pairs as initial guesses.

In the following two sections, we will describe the intersection algorithms involving one-dimensional and two-dimensional entities.

4 CPI

4.1 General Description

This is probably the most trivial intersection of all nonlinear algorithms. The component functions of the curve are plugged into the plane equation. The search is then performed in the one-dimensional interval (the parametric space of the curve).

A NURBS curve can be decomposed into several Bezier curves. Since Bezier curves are (rational) polynomial curves, it is possible to solve the problem in the most robust way, using the theories of roots of a polynomial equation [5]. Here we propose another *sampling* method to solve this one-dimensional root finding problem. If the samples can be created *intelligently*, all the roots can be isolated and computed. Thus, the same robustness can be achieved without the cost of subdivision.

4.2 Mathematical Details:

Mathematically, this problem is equivalent to finding all roots in an one-dimensional interval. We will use the root isolation technique.

4.2.1 Transforming a plane to the *nominal* position:

To simplify the calculation, the first step is to find a transformation such that the plane will be transformed to $z = 0$. 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.

The distance from a plane $ax + by + cz + d = 0$ to origin is given by:

$$distance = \frac{d}{\sqrt{a^2 + b^2 + c^2}}$$

Thus the transformation matrix for the first part is an identity orientation matrix, with the translation vector equals

$$[\frac{da}{a^2 + b^2 + c^2} \quad \frac{db}{a^2 + b^2 + c^2} \quad \frac{dc}{a^2 + b^2 + c^2}]^T$$

To rotate a vector into a desired orientation can be achieved by appropriate roll-pitch-yaw angles. The mathematics can be summarized in the following:

$$RPY(\phi, \theta, \psi)\ [a\ b\ c]^T = [0\ 0\ 1]^T$$
$$Rot(z, \phi)Rot(y, \theta)Rot(x, \psi)\ [a\ b\ c]^T = [0\ 0\ 1]^T$$

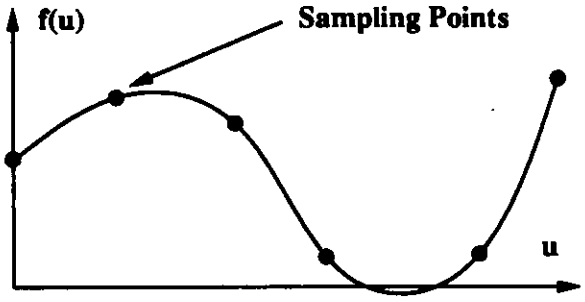

Figure 4: One dimensional root isolation method

After some calculation,

$$\begin{aligned}[a\ b\ c]^T &= Rot(x,-\psi)Rot(y,-\theta)Rot(z,-\phi)\ [0\ 0\ 1]^T \\ &= \cdots \\ &= (-\sin\theta \quad \sin\psi\cos\theta \quad \cos\psi\cos\theta)^T\end{aligned}$$

Thus, the orientation matrix for this part is the roll-pitch-yaw matrix,

$$RPY(0.0, \sin^{-1}(-a), \tan^{-1}(b/c))$$

Exception: When b and c are both zero, $\tan^{-1}(b/c)$ is set to zero.

Combining the results from the two parts, that is **premultiply** the first part by the second part, we have got the transformation matrix to put the plane to the *nominal* position.

4.2.2 Root Isolation:

The next step is to plug in the transformed curve into the plane equation. Since the plane has been transformed into $z = 0$, we just need to look at the z-component of the curve, for *both* rational and non-rational curves. Note that at this point, the problem is equivalent to an one-dimensional root finding problem: $f(u) = 0$, as shown in figure 4. The standard root isolation technique is to detect the sign change of the $f(u)$ for simple roots, and the sign change of $f'(u)$ for singular roots. The sampling points are generated based on the node vector [1]. The reason of this choice is that the node points are located at strategic positions and the number of nodes reflects the complexity of the curve. The success of this root isolation technique depends on the sampling process. We have found the samples described above to be satisfactory and we are investigating the general robustness of this method.

5 CSI

5.1 General Description

The problem of intersecting a curve and a surface is a three-variable, three-equation system. Traditionally, this problem has been solved by subdivision with numerical correction [11]. However, subdivision is not reliable, especially when the curve and the

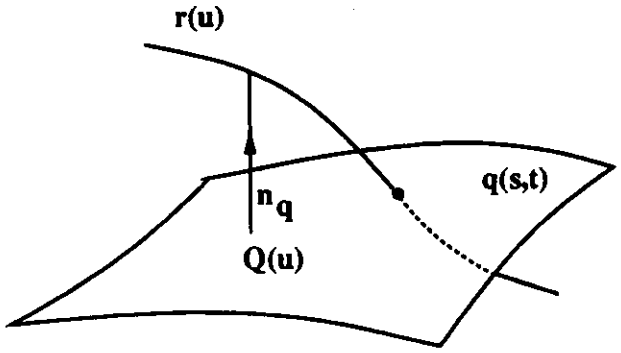

Figure 5: Orthogonal Projection from a point to the surface

surface are closely spaced. In this paper, we propose a method based on the idea of *generalized* orthogonal projection. The basic idea is to project the curve on the surface. The projected point is defined to be the closest point on the surface with respect to a given point (see figure 5). When the projected point is in the interior of the domain, it can be shown that it is also an orthogonal projection. Thus the following two equations are satisfied:

$$(\mathbf{r}(u) - \mathbf{q}(s,t)) \cdot \mathbf{q}_s(s,t) = 0 \tag{2}$$

$$(\mathbf{r}(u) - \mathbf{q}(s,t)) \cdot \mathbf{q}_t(s,t) = 0 \tag{3}$$

The surface normal at the projected point $\mathbf{n}_q$ and the vector $(\mathbf{r} - \mathbf{q})$ will contribute a sign, indicating whether that point is above or below the surface. Thus, with this signed distance function (distance is the magnitude of $(\mathbf{r} - \mathbf{q})$), we can use the same one-dimensional root finding techniques as we mentioned in CPI.

The intersection between a line segment and a surface is also implemented in the same routine, after converting the line segment into a parametric curve.

5.2 Mathematical Details:

5.2.1 More on Generalized Orthogonal Projection:

In the previous section, we have described the case where the projected point is in the interior. However in some cases, the closest point on the surface to a point will not satisfy the orthogonal projection conditions (eqs (2) and (3)). The reason is that one of the variables is constrained by the domain of the surface. Thus, we need to generalize the notion of orthogonal projection. We divide the domain into several regions of different *projection status*, as shown in figure 6. In ST_BOTH, both parameters are free to vary; thus, orthogonal projection can be achieved. In S_ONLY, only S can vary, thus only the first condition (2) can be satisfied. In T_ONLY, only T can vary, thus only the second condition (3) can be satisfied. In ST_NONE, the projected point will stay on the corner; thus, neither condition is satisfied.

The change of the projection status is determined by the location of the projection point and the value of the projections (left hand side of eqs(2 and 3)). When the

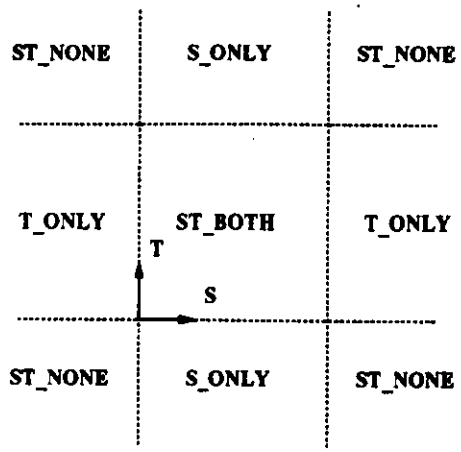

Figure 6: Projection Status Definition in CSI

projected point moves out of the domain at the four sides of the domain (excluding the four corners), we know that the projected status will be *downgraded* from ST_BOTH to X_ONLY (S_ONLY or T_ONLY). If the projected image moves out directly at the corner, the closest point will be the corner point and the projection status will be downgraded to ST_NONE. The *upgrade* of the projection status is determined by the sign change of the projection values. That is, when the projection status is X_ONLY, one of the orthogonal projections must have a nonzero value. When the status is upgradable, this will be indicated by a sign change.

Whenever the projection status changes, we need to correct the starting position to the boundary of the domain. The reason is that the curve might intersect at the boundary. Although this should be classified as a lower dimensional subproblem, it is more efficient in a nonlinear algorithm that the higher dimensional routines can resolve all the lower dimensional cases.

Some implementation details are worth mentioned:

- When the projection remains orthogonal and stays on the border, we need to prevent the projection status from changing back and forth due to floating point calculation. This will cause infinite looping. The solution is to detect such phenomenon and classify it as a special case.
- When the projection curve happens to go out of the domain from the corner, the point actually goes into an ST_NONE region. In the current implementation, we continue to move along the curve, until the point starts to have a valid projection (S_ONLY, T_ONLY, or ST_BOTH).

5.2.2 Continuation equation:

We need to set up two classes of continuation formulations for both orthogonal projection, and *partial* projection. The former is a two-equation three-variable problem, and the latter is an one-equation, two-variable problem.

ST_BOTH

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

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

S(T)_ONLY

$$(\mathbf{r}(u) - \mathbf{q}(s,t_i)) \cdot \mathbf{q}_s(s,t_i) = 0 \quad (S\_ONLY)$$
$$(\mathbf{r}(u) - \mathbf{q}(s_i,t)) \cdot \mathbf{q}_t(s_i,t) = 0 \quad (T\_ONLY)$$

5.2.3 Finding intersection point:

To solve CSI, we can find the intersection points along the curve. With the above definition of the signed distance function, we can use the same one dimensional root isolation and calculation technique as in CPI. But instead of the samples created by the node points in CPI, here we generate the sampling projection pairs between the curve and the surface. The sampling process is dynamically adjusted by the magnitude of the distance. That is, as the point on the curve moves closer to the surface, more samples will be created, and vice versa. This is done by controlling the size of the predictor in the continuation solver. It is clear that if the intersection were to happen between the curve and the surface, it will occur in the interior and the boundary of the surface domain. Thus, we only need to pay attention when the projection status is ST_BOTH, and when the projection status changes.

References

[1] E.S. Cobb. *Design of Sculptured Surfaces Using the B-spline Representation.* PhD thesis, University of Utah, 1984.

[2] E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. In M. J. Wozny, J. U. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 107 - 130. North-Holland, New York, 1990.

[3] E.L. Gursoz, Y. Choi, and F.B. Prinz. Boolean set operations on non-manifold boundary representation objects. *Computer Aided Design*, 23(1):33–39, January/February 1991.

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

[5] L. Redei. *Algebra*, volume 1. Pergamon Press, 1967.

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

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

[8] T.W. Sederberg and T. Nishita. Curve intersection using Bezier clipping. *Computer Aided Design*, 22(9):538–549, November 1990.

[9] T.W. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 22(8):97–114, November 1991.

[10] 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.

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

[12] K. J. Weiler. Edge based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.