Algebraic Loop Detection and Evaluation Algorithms for Curve and Surface Interrogations*

Shankar Krishnan
Univ. of North Carolina
CB #3175, Sitterson Hall,
Chapel Hill, NC 27599-3175 USA
krishnas@cs.unc.edu

Dinesh Manocha
Univ. of North Carolina
CB #3175, Sitterson Hall,
Chapel Hill, NC 27599-3175 USA
manocha@cs.unc.edu

Abstract

Evaluating curves on surfaces is a frequently occurring operation in a number of applications involving surface interrogations. For example, evaluating the intersection curve of two surfaces is critical to boundary (B-rep) computation, and in applications involving visibility and rendering, the ability to evaluate the *silhouette* curve of surfaces is important. While dealing with high degree surfaces, these curves usually consist of a number of components, including *loops*. We present a new algebraic loop characterization algorithm that can be applied in a number of applications. In particular, we discuss its application to the intersection curve of two surfaces and the silhouette curve of a surface. Unlike some other loop detection algorithms, our method can be applied even when the curve contain(s) singularities.

*Supported in part by a Alfred P. Sloan Foundation Fellowship, ARO Contract P-34982-MA, NSF Grant CCR-9319957, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization

Algebraic Loop Detection and Evaluation Algorithms for Curve and Surface Interrogations

1 Introduction

Current geometric and solid modeling systems use rational parametric and algebraic curves and surfaces for representing curved models. Many fundamental problems related to curve and surface interrogations in these systems require robust techniques for curve evaluation. These include intersection of surfaces for boundary evaluation, silhouette curve on a surface for visibility computations and rendering, and offset curves for toolpath generation. The resulting curves typically correspond to *high degree algebraic degrees*[1] with multiple components. These components can be classified into *open* components and *loops*. The open components intersect with the boundary of one of the surfaces or the end-point of a curve and a point on such components can be computed using algorithms for evaluating zero-dimensional algebraic sets. The rest of the curve components that do not intersect the boundary of the surface or the curve end-points are called loops. For example, the intersection of two Bézier surfaces in Figure 1 has one component corresponding to a loop. In general it is hard to come up with a tight bound on the number of components and development of general purpose algorithms for robust, efficient and accurate evaluation of these curves continues to be a major challenge. In this paper, we present efficient and accurate algorithms to evaluate the loops using a combination of algebraic and numeric methods.

The problem of evaluating all the curve components has been extensively studied in the literature. In the last decade, a number of algorithms have been proposed to test for closed loops for intersection curve of two surfaces [SKW85, SM88, THS89, Che89, Hoh91, KPP90, KPW90, ZS93, FS90] and offset curves and surfaces [FN90, Hof90, MP93]. These techniques are based on symbolic methods, evaluation of Gauss maps, subdivision techniques, differential methods and vector field approaches. In practice, they can be slow or restrictive. For example, none of these techniques are efficient for evaluating all the loops of a silhouette of a bicubic tensor product Bézier patch from a given viewpoint.

Main result: We present a new algebraic loop detection and evaluation algorithm for curve and surface interrogations. The algorithm uses the analytic equation and lower dimensional formulation of the curve and performs tracing in real and complex space to identify at least one *turning point* on each loop component. Unlike some of the previous loop detection algorithms, our method can

[1] There are a few exceptions with respect to offset curves.

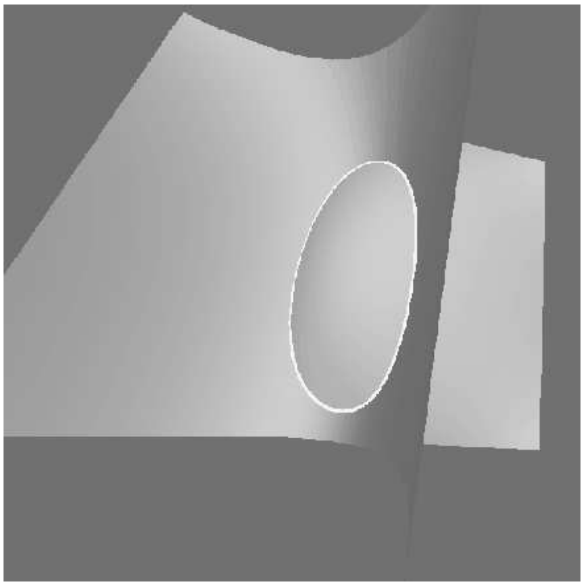


Figure 1: Two surfaces intersecting in a loop

be applied even when the curve contain(s) singularities. In practice, it can be applied to evaluate any one-dimensional algebraic set. We use methods from classical elimination theory (in particular, *resultants*) to project the curve onto a plane (say, onto the domain of the parametric patch). The equation of this curve is represented as the singular set of a bivariate matrix polynomial. We use curve tracing methods (based on eigenvalue methods and inverse power iterations) to follow the curve from the boundary of the domain to a turning point of any loop in *complex* space. The resulting algorithm has been implemented and we discuss its performance on evaluating the intersection of two surfaces, boundary computation and evaluating the silhouettes of a Bézier patch from a given viewpoint. We compare its performance with the earlier approaches as well.

1.1 Previous Work

The problem of evaluating all the curve components has been extensively studied in the literature and a number of techniques based on subdivision methods, marching methods, algebraic and symbolic techniques and lattice evaluations [Hof89, RR92]. In particular, the problem of determining all the loops of an algebraic curve can be solved robustly using symbolic methods. One such method is the cylindrical algebraic decomposition [ACM84], which can determine the topological type of a curve.

Other techniques are based on posing the problem as computing roots of two algebraic equations in two unknowns and solving them using zero-dimensional solvers. These include Gröbner basis [Buc85], resultants and eigenvalue methods [Man94b], homotopy methods [Mor92] or interval arithmetic [Moo79]. However, the complexity of the resulting algebraic systems is quadratic in the degree of the curve. The algebraic degree of the intersection curve of two bicubic Bézier patches can be as high as 324 in space and the corresponding curve in the domain can be of degree 108. The approaches based on symbolic methods and algebraic solvers are not practical for such high degree curves.

The subdivision based algorithms subdivide the domain up to a user-specified tolerance and evaluate the curves accordingly [Gei83, LR80, MP93]. In general, the two components of a curve can be very close and no good methods are known for computing a good tolerance value. As a result, most implementations use a conservative value for the tolerance. The resulting methods can be slow and lead to data proliferation. Some of the other approaches are based on *lattice evaluation* where the surface-surface intersection problem is simplified to a set of curve-surface intersection problems. The curves were obtained by evaluating the surface patch at a number of constant parameter values. The biggest drawback in this approach is the lack of robustness. Small loops could easily be missed depending on the frequency with which the curves are evaluated.

In the last decade, techniques based on curve tracing have been widely used to evaluate high degree curves [BFJP87, BHHL88, KPP90, MC91]. The main idea is to compute at least one point on every component of the curve and use the local geometry of the curve to evaluate successive points. In these class of methods, identifying a point on every loop is significantly harder than that on open components. As a result, simultaneously with the development of new ideas for evaluating such curves, number of techniques for loop detection have been proposed [SKW85, SM88, THS89, Che89, Hoh91, KPP90, KPW90]. However, most of these efforts were targeted towards developing loop detection methods for a special type of curve, the intersection curve of two surfaces. All the loop detection criteria are based on bounds on the *Gauss map* of the surfaces being intersected. Sinha et. al. [SKW85] had shown that if two (at least C^1) surfaces intersect in a closed loop, there exists a normal vector on one surface that is parallel to a normal vector of the other surface. Sederberg et. al. [THS89, SM88] strengthened the above work by proving that if two (at least C^1) surfaces intersect in a closed loop, there exists a line which is perpendicular to both surfaces (collinear normal vectors), provided the inner product between any normal on one surface and any other normal on the other surface is never zero. Patriakalakis et. al. [KPP90] precomputed the most significant points of the intersection curve between an algebraic surface and a parametric patch to identify the main features of the curve. Sederberg et. al. [SM88, ZS93] developed an efficient way to bound the normals and tangents of a surface using (*bounding cones*) and *pyramidal surfaces*, thereby giving a faster way to achieve the no loop condition. Hohmeyer [Hoh91] bounded the Gauss maps using pseudo-normal patches and used an efficient algorithm for linear programming [Sei90] to test the separability criterion. In these algorithms, if the loop detection criterion is not satisfied, each surface is divided into a pair of sub-patches and the criterion is recursively tested on each pair combination. This is continued until all patch pairs fail the test. The number of levels of subdivision depends on how tightly the Gauss maps [EC94] are bounded. For example, application of Hohmeyer's [Hoh91] loop detection criterion on the surfaces in Fig. 1 takes 8 levels of subdivision. Furthermore, these algorithms may not work well if the intersection curve is self-intersecting.

The algorithms based on Gauss maps are generic. However, they become quite inefficient when applied to other surface interrogations. Consider, for example, the application of algorithms based on Gauss maps to detect loops on the silhouette curve of a rational parametric patch of degree $m \times n$. Let's assume that the viewing direction is along the positive z-axis. Hohmeyer uses a pseudo-normal patch to bound the Gauss map of the patch [Hoh91]. The parametric degree of the pseudo-normal patch can be as high as $3m \times 3n$. We can pose the silhouette curve as the intersection curve of the Gauss map with the plane $z = 0$ for orthographic projections. While applying the loop detection criterion, we have to compute the Gauss map of the pseudo-normal patch which would be a $9m \times 9n$ rational parametric patch. Manipulating and subdividing such high degree parametric patches can be inefficient and inaccurate in practice. As a result, there is a need to develop general-purpose and efficient loop detection and evaluation algorithms.

Organization: The rest of the paper is organized in the following manner. Section 2 describes our formulation of the intersection curve between two parametric surfaces and the silhouette curve of a parametric surface. Section 3 discusses the loop detection algorithm. The implementation of the algorithm and its performance on various applications are highlighted in section 4, and we conclude in section 5.

2 Algebraic Formulation of the curve

In the most general setting, a curve in $\mathcal{R}^n$ can be expressed as a solution of $(n-1)$ polynomial equations in n unknowns.

$$F_1(u_1, u_2, \ldots, u_n) = 0$$

$$
\begin{aligned}
F_2(u_1, u_2, \ldots, u_n) &= 0 \\
&\vdots \\
F_{n-1}(u_1, u_2, , \ldots, u_n) &= 0.
\end{aligned}
$$

Moreover, we are only interested in evaluating all the components of the curve inside the region $D = [U_{11}, U_{12}] \times [U_{21}, U_{22}] \times \ldots \times [U_{n1}, U_{n2}] \in \mathcal{R}^n$. Formally, the functions F_i, $i = 1, 2, \ldots, n-1$, are the components of a vector function $F : D \rightarrow \mathcal{R}^n$, $D \subset \mathcal{R}^{n+1}$. The solution to the problem are elements of D that map to the *zero* vector under F. This can be illustrated by taking the example of parametric surface intersection. Given two Bézier surfaces, $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$ and $\mathbf{G}(u,v) = (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v))$ represented in homogeneous coordinates, their intersection curve is defined as the set of common points in 3-space and is given by the vector equation $\mathbf{F}(s,t) = \mathbf{G}(u,v)$. This results in the following set of three equations in four unknowns:

$$
\begin{aligned}
F_1(s,t,u,v) = X(s,t)\overline{W}(u,v) - \overline{X}(u,v)W(s,t) &= 0 \\
F_2(s,t,u,v) = Y(s,t)\overline{W}(u,v) - \overline{Y}(u,v)W(s,t) &= 0 \\
F_3(s,t,u,v) = Z(s,t)\overline{W}(u,v) - \overline{Z}(u,v)W(s,t) &= 0,
\end{aligned}
\tag{1}
$$

and the domain of the intersection curve is $(s,t,u,v) \in [0,1] \times [0,1] \times [0,1] \times [0,1]$. Existing numerical methods for curve evaluation perform tracing in these dimensions using techniques such as quasi-Newton's iteration. However, the convergence of these methods may not be good in higher dimensions [FF92].

Our approach is based on a classical result in algebraic geometry that states that any algebraic space curve has a one-to-one correspondence with an algebraic plane curve, after suitable linear transformations. The plane curves with one-to-one correspondence with the intersection curve in space are shown in Fig.2. Essentially, our evaluation algorithm is now restricted to the plane curve. Another advantage of this approach is that applications (like solid modelers), that use curve evaluation methods, need to provide simpler supporting algorithms. We represent the plane curve as the singular set of a bivariate matrix polynomial.

2.1 Intersection curve between parametric surfaces

In this section, we present our method of formulating the intersection curve. Particularly, we apply it to the intersection of two rational Bézier surfaces. However, our method can be easily applied to

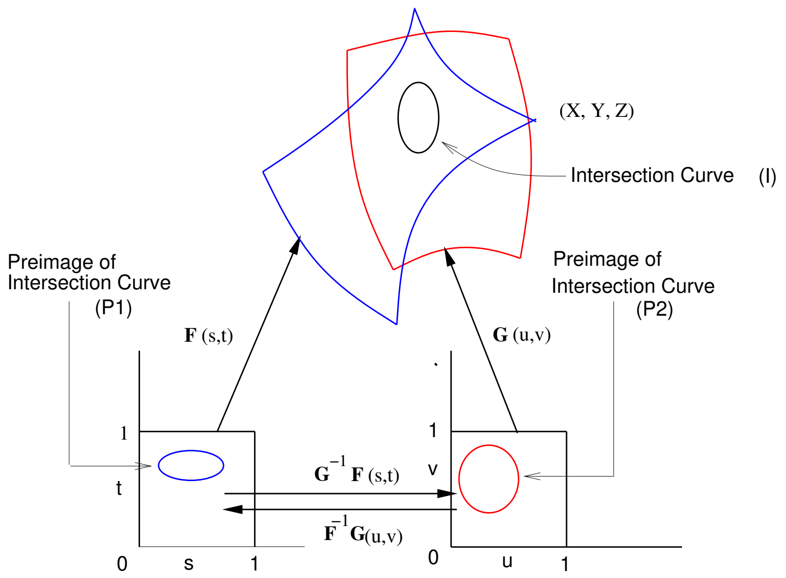


Figure 2: Intersection curve and its planar preimages

implicit algebraic surfaces as well. We represent the plane curve as an unevaluated matrix determinant [MC91].

Matrix Formulation: In this paper, we shall assume that the parametric surface is given in the form of a *tensor product Bézier patch.* A tensor product patch is of the form

$$F(s,t) = (\sum_{i=0}^{m}\sum_{j=0}^{n} V_{ij} B_{i,m}(s) B_{j,n}(t)),$$

where $V_{ij} = (x_{ij}, y_{ij}, z_{ij}, w_{ij})$ are the control point coordinates and $B_{i,m}(s) = \begin{pmatrix} m \\ i \end{pmatrix} s^i(1-s)^{m-i}$ is the Bernstein polynomial.

Given two Bézier surfaces, $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$ and $\mathbf{G}(u,v) = (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v))$ in homogeneous coordinates, implicitize $\mathbf{F}(s,t)$ to the form $f(x,y,z,w) = 0$ [Sed83, Hof89] and substitute the parameterization of $\mathbf{G}(u,v)$ into f to get an algebraic plane curve of the form

$$f(\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)) = 0.$$

The implicit representation of the patch is obtained by eliminating s and t from the equations

$$xW(s,t) - X(s,t) = 0$$

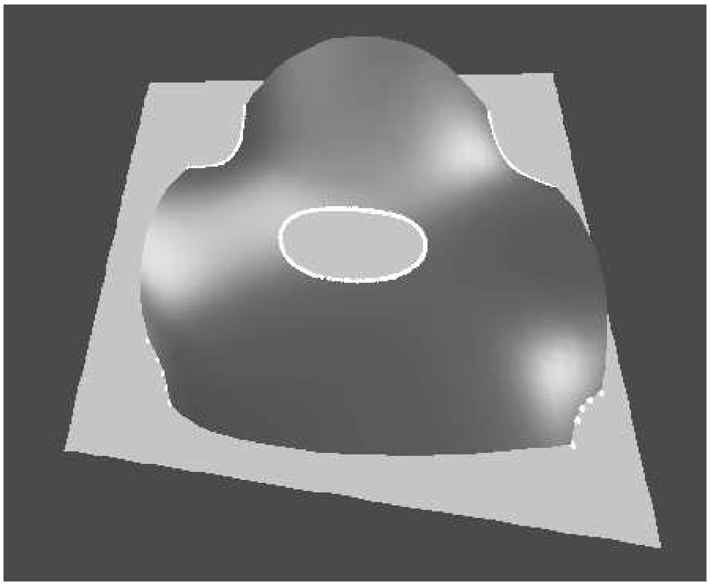


Figure 3: A pair of intersecting surfaces

$$yW(s,t) - Y(s,t) = 0$$

$$zW(s,t) - Z(s,t) = 0$$

using *resultants* [Sed83]. There are different formulations of resultants for tensor product surfaces and triangular surfaces. It turns out that the resultant of these three equations can always be expressed as the determinant of a matrix [Dix08]. Let us denote that matrix as $\overline{\mathbf{M}}(x, y, z, w)$. Furthermore, each entry of the matrix is of the form $a_{ij}x + b_{ij}y + c_{ij}z + d_{ij}w$. The order of $\overline{\mathbf{M}}(x, y, z, w)$ is a function of the degrees of the equations. For tensor product surfaces of the form $s^m t^n$, the order of the matrix is $2mn$. The determinant of the resulting matrix corresponds to the implicit representation of the parametric surface. We substitute the parameterization of $\mathbf{G}(u, v)$ into this matrix and obtain a representation of the form $\mathbf{M}(u, v)$, where each entry is a polynomial in u and v. This substitution is very simple because every entry of the matrix is just a linear term. The degree of each polynomial corresponds to the degree of $\mathbf{G}(u, v)$.

Base Points: The base points of a parameterization are the common solutions of the equations: $X(s,t) = 0$, $Y(s,t) = 0$, $Z(s,t) = 0$, $W(s,t) = 0$. Typically, a generic parameterization does not have base points. However, in the presence of base points the resultant of the parametric equations is identically zero. The implicit representation in such cases is a factor of the determinant of the maximum rank preserving minor of $\mathbf{M}(x, y, z, w)$. We use that minor for the representation and substitute the parameterization of $\mathbf{G}(u, v)$ to obtain the planar projection of the intersection curve. We handle patches with base points specially.

2.2 Silhouette curve of a parametric surface

Silhouette computation forms an important part of visibility and rendering (for radiosity applications) algorithms for curved surfaces. We shall restrict our discussion to surfaces whose silhouette (from a given viewpoint) is a curve on the surface. The property of the silhouette curve is that it subdivides the surface into front and back facing regions. In this section, we describe our formulation of the silhouette curve on a parametric (represented as a tensor product Bézier [Far93]) patch.

We assume for the sake of simplicity that the viewpoint is located at $(0, 0, -\infty)$. It is easy to see that even if this is not the case, one can always achieve it by applying an appropriate perspective transformation to the parametric surface $\mathbf{F}(u, v)$. We also require that all the surfaces are at least C^1 everywhere. We formulate the silhouette curve as an algebraic plane curve in the domain of $\mathbf{F}(u, v)$.

2.3 Formulation of the Silhouette Curve

Let $\mathbf{F}(u, v)$ denote the parametric (differentiable) surface and let $\phi_1(u, v), \phi_2(u, v)$ and $\phi_3(u, v)$ denote the mappings from the parametric space to (x, y, z) space.

$$\mathbf{F}(u, v) = \langle X(u, v), Y(u, v), Z(u, v), W(u, v) \rangle$$

$$\phi_1(u, v) = \frac{X(u, v)}{W(u, v}, \quad \phi_2(u, v) = \frac{Y(u, v)}{W(u, v}, \quad \phi_3(u, v) = \frac{Z(u, v)}{W(u, v}$$

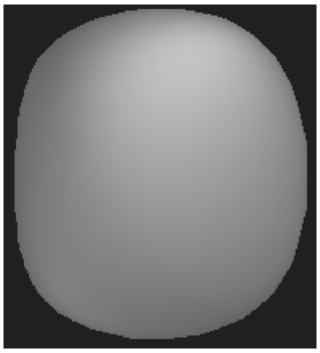
In the rest of this section, we shall drop the (u, v) suffixes from all the functions for more concise notation. The $z-$component of the normal at an arbitrary point on the surface is given by the determinant

$$N_z = \begin{vmatrix} \phi_{1_u} & \phi_{1_v} \\ \phi_{2_u} & \phi_{2_v} \end{vmatrix} \tag{2}$$

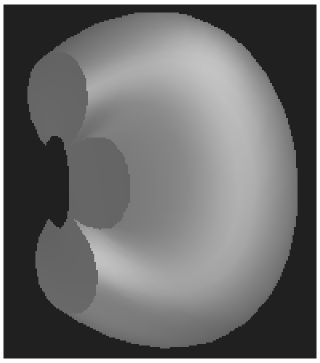
where ϕ_{i_u} and ϕ_{i_v} denote the partial derivatives of the appropriate function ϕ_i with respect to u and v.

$$\phi_{1_u} = \frac{(WX_u - W_uX)}{W^2} \qquad \phi_{1_v} = \frac{(WX_v - W_vX)}{W^2}$$

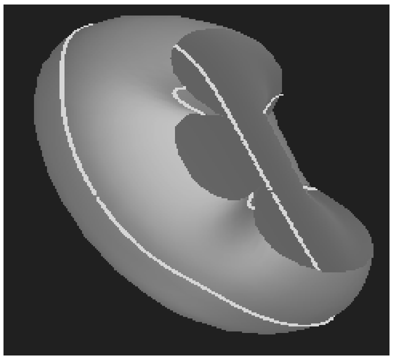
$$\phi_{2_u} = \frac{(WY_u - W_uY)}{W^2} \qquad \phi_{2_v} = \frac{(WY_v - W_vY)}{W^2}$$



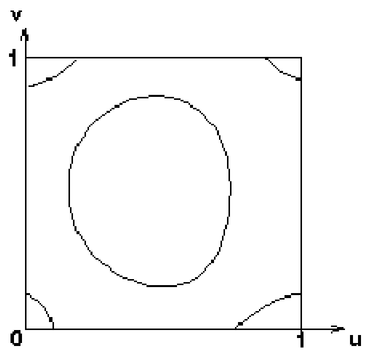
(a) Silhouette finding view



(b) Another view



(c) Surface with silhouettes



(d) Silhouette curve in domain

Figure 4: Loop as part of a silhouette curve

On the silhouette curve, $N_z = 0$. Since $W(u, v) \neq 0$, we can express the plane curve representing the silhouette as the determinant

$$N_z = \begin{vmatrix} (WX_u - W_uX) & (WX_v - W_vX) \\ (WY_u - W_uY) & (WY_v - W_vY) \end{vmatrix} = 0 \tag{3}$$

Expanding the determinant and rearranging the terms, we can express it as the singular set of the matrix $\mathbf{M}(u, v)$

$$\mathbf{M}(u,v) = \begin{pmatrix} X(u,v) & Y(u,v) & W(u,v) \\ X_u(u,v) & Y_u(u,v) & Z_u(u,v) \\ X_v(u,v) & Y_v(u,v) & Z_v(u,v) \end{pmatrix} = 0 \tag{4}$$

The singular set of $\mathbf{M}(u,v)$ are the values of u and v which make it singular.

3 Loop Detection

We apply our loop detection algorithm to find all the loops of an algebraic plane curve. We use a matrix determinant representation to deal with high degree curves, but any general form (like power or Bernstein basis) is sufficient for our algorithm. In this section, we shall describe our loop detection algorithm.

The curve we are interested in is an algebraic plane curve in the complex projective plane defined by u and v. We are, however, interested only in finding the part that lies in the portion of the real plane defined by $(u,v) \in [0,1] \times [0,1]$. If we relax this restriction so that one of the variables, say v, can take complex values, this curve is defined as a continuous set consisting of real and complex components. Before we give our algorithm, some basic notational definitions have to be introduced.

Definition 1 **Turning points** *are points on the curve where the tangent vector, as projected in the (u,v) space, is parallel to the u or v parameter axes. In other words, one of the partial derivatives (with respect to u or v) of the intersection curve is* 0*.*

We classify u-turning points into *left u-turning points* and *right u-turning points.* A point (u_1, v_1) is a *left u-turning point* if the curve goes into the complex domain in the left neighborhood of u_1 ($u = u_1 - \delta$, where δ is a small positive value). A point (u_1, v_1) is a *right u-turning point* if the curve goes into the complex domain in the right neighborhood of u_1 ($u = u_1 + \delta$).

Definition 2 **Isoparametric curves** *are curves lying on a parametric patch (surface) where one of the parameters of the patch (u or v) remains constant.*

The main idea behind our loop detection algorithm is based on the following lemma.

Lemma 1 *If the curve in the real domain $[0,1] \times [0,1]$ consists of a closed component, then two arbitrary complex conjugate paths meet at one of the real points (corresponding to a turning point) on the loop.*

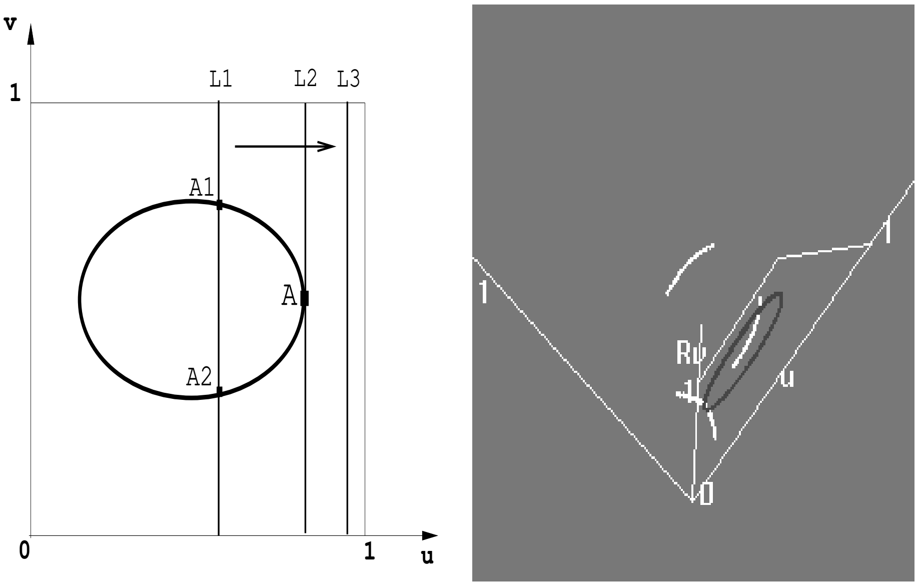


Figure 5: Characterization of loops based on complex tracing

Proof: The proof is based on *Bezout's theorem* which states that *if f and g are two algebraic curves of degree m and n respectively, then f and g intersect in exactly mn points in the complex domain counted properly, or they have a common component.* We use Bezout's theorem and the fact that the curve forms a continuous set in the complex domain to prove the result.

Let us consider an algebraic curve that forms a loop in the real domain, like the one shown in fig.1. All isoparametric curves on a surface have the same degree, namely the degree of the other parameter defining the surface. Therefore, the number of intersections of the algebraic curve with any isoparametric curve equals equals the Bezout bound in complex space. Fig.5 (left) illustrates the argument. The line $L1$ intersects the curve at two different real points. As we move the line continuously from $L1$ to $L2$, the two intersection points come closer, and at line $L2$, both of them coincide to form a double root maintaining the intersection count constant. This double root also corresponds to a u-turning point. As $L2$ approaches $L3$, all the real intersections vanish. Since the algebraic curve is continuous in complex domain, the double root must now take complex values and occur in conjugate pairs (real algebraic curve).

Now if the sweep is started from $L3$ towards $L2$, the complex conjugate components come closer together, and at line $L2$, their imaginary part vanishes to yield a double root. This argument shows that the complex conjugate pairs meet the real plane at some turning point of every component. Observing that every loop component must have at least two turning points completes the proof.

□

The domain of the intersection curve in the complex space is shown in fig.5 (right). The third axis corresponds to the imaginary components of v. It represents a continuous component of the intersection curve. The green curve is the intersection curve in the complex space and the blue curve is the part of the curve that lies in the real plane.

We need only one start point on each loop to trace it completely. So we restrict ourselves to u-turning points. Henceforth, we shall use *turning points* to denote u-turning points. Our domain has changed from the real plane to a three dimensional space formed by u, v_r and v_i, where v_r and v_i are the real and imaginary values of v. To compute the turning points on the curve, we combine boundary computations with complex tracing.

Boundary intersections: Boundary intersections refer to the portions of the curve that lie along the boundary of the surface (in our case, when $u = 0$, $u = 1$, $v = 0$ or $v = 1$). This corresponds to substituting one of these values into the equation $\mathbf{M}(u, v) = 0$. Let us assume without loss of generality that we substitute $u = 0$. This results in a matrix polynomial $\mathbf{M}(v)$ of the form

$$\mathbf{M}(v) = v^d M_d + v^{d-1} M_{d-1} + \ldots + v M_1 + M_0 = 0. \quad (5)$$

where M_i's are numeric matrices (of order $2mn$ for the intersection curve and 3 for the silhouette curve). The solution of this matrix equation can be reduced to the eigenvalues of an associated companion matrix.

$$C = \begin{bmatrix} 0 & I_n & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & I_n \\ -\overline{\mathbf{M}}_0 & -\overline{\mathbf{M}}_1 & -\overline{\mathbf{M}}_2 & \ldots & -\overline{\mathbf{M}}_{d-1} \end{bmatrix} \quad (6)$$

where $\overline{\mathbf{M}}_i = \mathbf{M}_d^{-1}\mathbf{M}_i$. In case $\mathbf{M}_d$ is singular or ill-conditioned, the intersection problem is reduced to a generalized eigenvalue problem [Man94a]. Algorithms to compute all the eigenvalues are based on QR orthogonal transformations [GL89]. They compute all the real and complex eigenvalues.

Tracing: Given the starting complex on the boundary of the surface, we use tracing in the complex domain to reach the turning points on every loop. The general tracing step proceeds as follows. Given a point on the curve, an approximate value of the next point is obtained by taking a small step size in a direction determined by the local geometry of the curve (tangent or curvature information). This approximate value is then refined using iterative techniques. We use *inverse power iterations* to trace the curve. Inverse power iterations are used to compute selected eigenvectors and eigenvalues of a matrix.

Let us assume that we are currently at a point (u_1, v_1) on the curve $(\mathbf{M}(u_1, v_1) = 0)$. Based on the local geometry of the curve, let the estimate to the next point be (u_2, v_2). Using v_2 as a guess we want to find the closest point (u_2, v) such that $\mathbf{M}(u_2, v) = 0$. We proceed by computing the companion matrix $\mathbf{C}$ from $\mathbf{M}(u_2, v)$ (see eq. (6)). This reduces the problem to finding the eigenvalue of C closest to v_2 (or smallest eigenvalue of $C - v_2\mathbf{I}$). The smallest eigenvalue of $\mathbf{C} - v_2\mathbf{I}$ corresponds to the largest eigenvalue of $(\mathbf{C} - v_2\mathbf{I})^{-1}$. Instead of computing the inverse explicitly (which can be numerically unstable), we use inverse power iterations. Given an initial unit vector $\mathbf{q}_0$, we generate a sequence of vectors $\mathbf{q}_k$ as

$$\textit{Solve } (\mathbf{C} - v_2\mathbf{I})\mathbf{z}_k = \mathbf{q}_{k-1}; \quad \mathbf{q}_k = \mathbf{z}_k / \parallel \mathbf{z}_k \parallel_\infty; \quad s_k = \mathbf{q}_k^T \mathbf{C} \mathbf{q}_k;$$

To solve the matrix system efficiently, we use LU decomposition of the matrix $(\mathbf{C} - v_2\mathbf{I})$ using Gaussian elimination. We also make use of the structure of the matrix to reduce the complexity of LU decomposition. Given s, let $\mathbf{B} = \mathbf{C} - v_2\mathbf{I}$. $\mathbf{B}$ is of the form:

$$\mathbf{B} = \begin{pmatrix} \alpha_1\mathbf{I}_n & \mathbf{I}_n & \mathbf{0} & \dots & \mathbf{0} \\ \ddots & \dots & & & \\ \mathbf{0} & \mathbf{0} & \dots & \alpha_1\mathbf{I}_n & \mathbf{I}_n \\ \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3 & \dots & \mathbf{P}_m \end{pmatrix}$$

where α_1 is a function of s, and $\mathbf{P}_i$'s are $n \times n$ matrices which are functions of $\mathbf{M}_i$'s and v_2. The LU decomposition of $\mathbf{B}$ has the form:

$$\mathbf{B} = \begin{pmatrix} \alpha_1\mathbf{I}_n & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \alpha_1\mathbf{I}_n & \dots & \mathbf{0} \\ \ddots & \dots & & \\ \mathbf{R}_1 & \mathbf{R}_2 & \dots & \mathbf{L}_m \end{pmatrix} \begin{pmatrix} \mathbf{I}_n & \frac{1}{\alpha_1}\mathbf{I}_n & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_n & \dots & \mathbf{0} \\ \ddots & \dots & & \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{U}_m \end{pmatrix}$$

where $\mathbf{L}_m$ and $\mathbf{U}_m$ correspond to the LU decomposition of $\mathbf{R}_m$. $\mathbf{R}_i$'s can be easily computed from the $\mathbf{P}_i$'s. The structure of the matrices are used in performing the triangular decomposition efficiently. LU decomposition can be inaccurate when the matrix $\mathbf{B}$ is ill-conditioned. In such cases, we perform an LQ factorization (lower triangular and orthonormal matrix decomposition) to improve the stability.

The basic technique of obtaining all the turning points is to evaluate the starting complex points on one of the boundaries and follow all these paths until they either leave the domain or meet the real plane using tracing. Unfortunately, these are not the only complex paths that could lead to a turning point. There could be complex paths starting from *right turning points* of some other component of the intersection curve. This can be illustrated by considering the intersection between a bicubic patch and a plane (see fig.3). The curve $\mathbf{M}(0, v) = 0$ is a cubic curve with two real solutions. This implies that there cannot be any complex solution to this equation. Therefore, the left turning point on the loop is connected in complex space to the right turning point of another component. So we use the following strategy to complete a sweep of the complex paths from $u = 0$ to $u = 1$.

Since complex solutions occur in conjugate pairs for real algebraic equations, we restrict ourselves to complex paths whose imaginary parts are strictly positive. When a complex path touches the real plane the imaginary part (of v) must reach some small constant value ϵ before reducing to zero. These are precisely the common points of the curve with the plane $v_i = \epsilon$. In other words, we are trying to find all the real solutions to the equation $\det \mathbf{M}(u, v_r + i\epsilon) = 0$ $(i = \sqrt{-1})$. Expanding out the expression and collecting the real and imaginary terms we can write

$$\det(\mathbf{M_r}(u, v_r) + i\mathbf{M_i}(u, v_r)) = 0 \tag{7}$$

It is easy to show that the solutions (u, v_r) satisfying equation (7) also satisfy the solution of $\det \mathbf{P}(u, v_r) = 0$, where

$$\mathbf{P}(u, v_r) = \begin{bmatrix} \mathbf{M_r}(u, v_r) & -\mathbf{M_i}(u, v_r) \\ \mathbf{M_i}(u, v_r) & \mathbf{M_r}(u, v_r) \end{bmatrix} \tag{8}$$

As before, the solutions to (8) can be posed as the singular set of matrix $\mathbf{P}(u, v_r)$. The singular set of $\mathbf{P}(u, v_r)$ is a discrete point set. The order of the matrix $\mathbf{P}(u, v_r)$ is twice that of $\mathbf{M}(u, v)$. Therefore, there are twice as many paths to trace in general. For an intersection curve, if the patches are of degree $m \times n$ and $p \times q$, then at most $2mn \min(p, q)$ paths have to be traced, and for the silhouette curve of a patch of degree $m \times n$, at most $3(m + n)$ paths have to be traced.

Initially we form the companion matrix of $\mathbf{P}(u, v_r)$, C_p, similar to the one in Eq.(6). We compute all the eigenvalues of C_p at $u = 0$ (we expect all of them to be complex). We use them as starting points and trace all the paths in increasing u direction until it either crosses the $u = 1$ plane or become real. All the real values of v_r are points lying very close to the turning points of the intersection curve. The corresponding point on the real plane is (u_r, v_r). This is used as an initial guess to converge to the turning point using inverse power iterations.

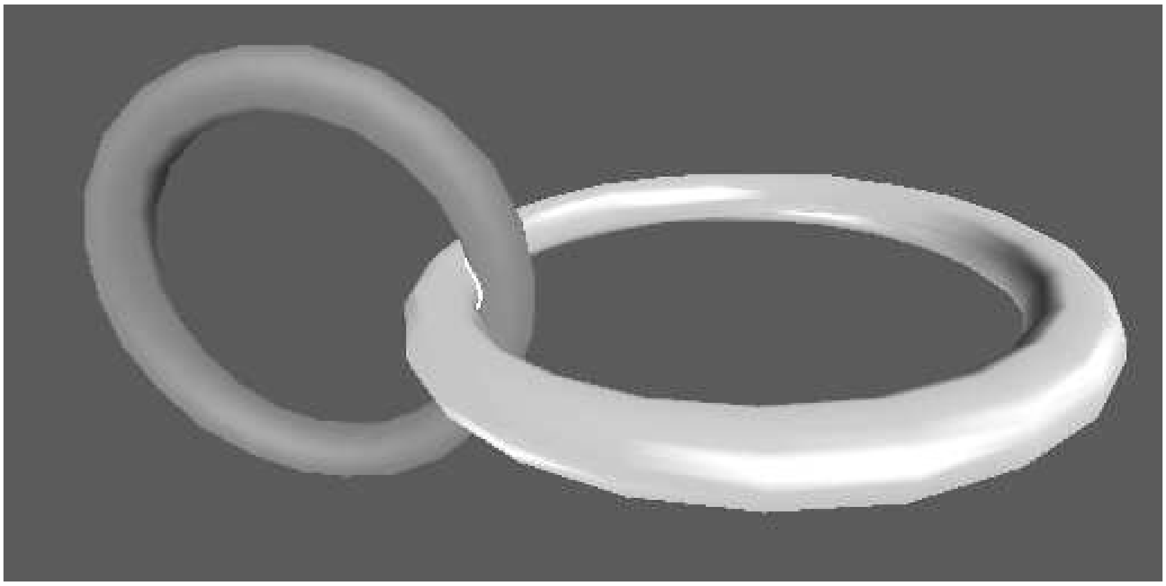


Figure 6: Two tori intersecting in a small loop

4 Implementation, Performance and Applications

The loop detection algorithm has been implemented and its performance was measured on a number of models. The algorithm uses existing EISPACK [GBDM77] and LAPACK [ABB+92] routines for some of the matrix computations. At each stage of the algorithm, we can compute bounds on the accuracy of the results obtained based on the accuracy, condition numbers and convergence of numerical methods used like eigenvalue computation, power iterations and Gaussian elimination. We report the results of our implementation on an SGI Onyx workstation with 128MB of main memory and a specFP rating of 97.1.

Tracing in the complex space is a guided form of search for all the turning points of the loops. In a purely algebraic form, all the turning points of a curve $f(u,v) = 0$ can be posed as the common solutions of $f(u,v) = f_u(u,v) = 0$. Using the Bezout bound, the number of possible turning points is *quadratic* in the degree of the curve. However, the maximum number of complex paths that need to be traced in our loop detection algorithm is *linearly* related to the degree of the curve. The performance of the tracing algorithm is directly dependent on how efficiency of linear systems ($Ax = b$) solvers. While methods like LU and LQ decomposition take $O(n^3)$ operations, our use of the special structure of the matrix has almost quadratic complexity. Our implementation of the algorithm consists of two major modules - the boundary computation part and the complex tracing part. The boundary computation module computes starting points on all the complex paths using eigensolvers. For our implementation, we used an ϵ (see section 3) value of 0.01. The complex tracing step is done using inverse power iterations. The total time taken to trace one such path across the domain is about 20-50 milliseconds.

4.1 Application to surface intersection and boundary computation

Our loop detection algorithm is part of a complete surface intersection algorithm. This, in turn, has been applied to a number of intersecting surfaces and has worked well consistently. Our algorithm evaluated the intersection curve of the surfaces in Fig. 1 in about 4 seconds. A total of 54 complex paths were traced which consumed about 70% of the time.

For efficiency considerations, it may not be necessary to trace all the complex paths. Typically, very few complex paths meet the real plane inside the domain of the patch. It is very difficult to give exact algorithms to prune out paths that cannot touch the real plane because of the high degree nature of the curve. However, through repeated application of our algorithm we found that paths that start very high in the complex axis rarely hit the real plane. This strategy could be used to speed up the tracing step depending on the robustness requirements of the application.

In order to compare our algebraic method with the Gauss map based approaches to loop detection, we implemented Hohmeyer's algorithm (in the context of surface intersection) using pseudo-normal patches [Hoh91]. His algorithm performed slightly slower than our algorithm on the example in Fig. 1. Eight levels of subdivision were performed, and most of the time was consumed in the repeated computation of the Gauss map and application of linear programming. We observed that his algorithm works very well when the patches are relatively flat and do not intersect in loops. However, these methods perform a number of subdivisions (to achieve the no loop criterion) when the patches have high curvature and intersect in small loops or singularities.

Hybrid approach: We suggest the following hybrid approach when dealing with intersection curves. Initially, we test for the possible absence of loops using the Gauss map approach. In the event that Gauss maps are not separated, we apply our algorithm to identify turning points on loops. This method has been applied to compute intersections of high degree surfaces. On an average, our algorithm takes less than one second to compute one patch-pair intersection. For the intersecting surfaces in Fig. 1 and Fig. 6, our method performs better than Hohmeyer's algorithm. His method, however, performed better when applied to the surfaces in Fig. 3.

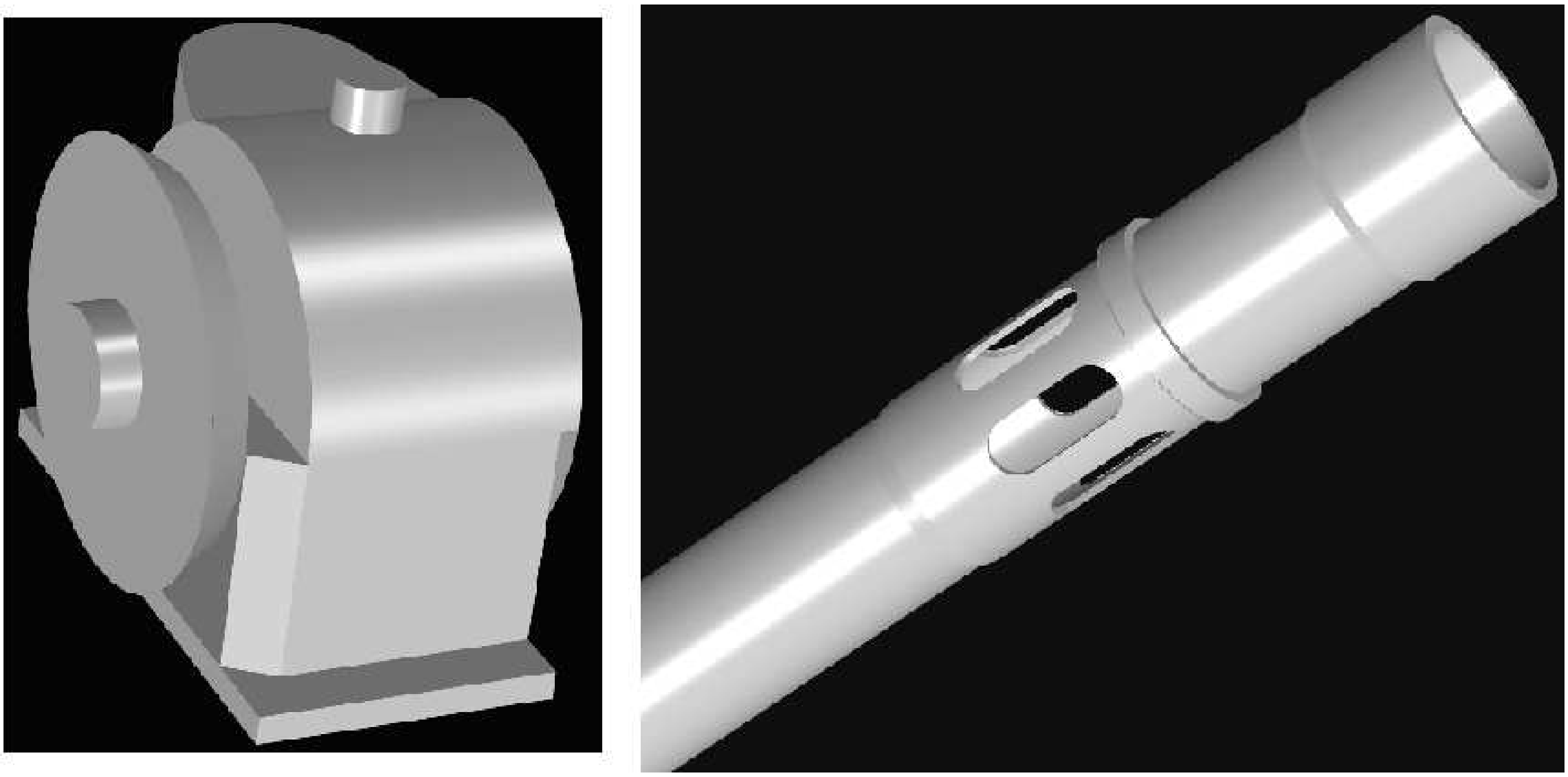


Figure 7: Loop detection used in B-rep computation

The surface intersection algorithm is part of a solid modeling system which computes the B-rep (boundary representation) of CSG solids. Fig. 7 shows two of the solids generated by the system. The solid on the left is composed of 69 Bézier patches using a 10 level CSG tree. Our system computed the entire B-rep in 66 secs. The B-rep of the solid on the right (consisting of 116 Bézier patches using a 5 level CSG tree) was computed in 41 secs. We have also applied the loop detection algorithm to compute the silhouettes of surfaces. Fig. 4 shows a patch that has a loop as part of its silhouette.

4.2 Silhouette Computation

For other curves like silhouettes, the Gauss map approach is not very practical. In order to apply their loop detection criteria on a bicubic patch (like that in Fig. 4(b)), one would have to perform repeated subdivisions on rational patches of degree 27×27. This makes the algorithm very slow because each subdivision step takes cubic time (in terms of the degree). We were able to determine all the components of the silhouette for the same surface using our algorithm in about 2 seconds. Performing boundary computations to determine all the starting points roughly takes 40% of this time. The rest of the time is spent in curve tracing. For this particular example, a total of *two complex paths* and *five real components* were traced along the entire domain. The real components of the silhouette curve in the domain are shown in Fig. 4(d).

5 Conclusion

We have presented a general algorithm based on numeric and symbolic methods for loop detection of algebraic curves. It performs boundary intersections followed by complex tracing to locate the turning points on every loop. The resulting algorithm is based on numerical matrix computations and algebraic characterization of the curve. It has been used to accurately compute intersection and silhouettes of high degree surfaces.

References

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0.* SIAM, Philadelphia, 1992.

[ACM84] D. S. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition. *SIAM J. on Computing*, 13:878–889, 1984.

[BFJP87] R. Barnhill, G. Farin, M. Jordan, and B. Piper. Surface/surface intersection. *Computer Aided Geometric Design*, 4(3):3–16, 1987.

[BHHL88] C.L. Bajaj, C.M. Hoffmann, J.E.H. Hopcroft, and R.E. Lynch. Tracing surface intersections. *Computer Aided Geometric Design*, 5:285–307, 1988.

[Buc85] B. Buchberger. Groebner bases: An algorithmic method in ideal theory. In N.K. Bose, editor, *Multidimensional Systems Theory*, pages 184–232. D. Reidel Publishing Co., 1985.

[Che89] K.P Cheng. Using plane vector fields to obtain all the intersection curves of two general surfaces. In *Theory and Practice of Geometric Modeling*, pages 187–204, 1989.

[Dix08] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[EC94] G. Elber and E. Cohen. Exact computation of gauss maps and visibility sets for freefrom surfaces. Technical report CIS #9414, Computer Science Department, Technion, 1994.

[Far93] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1993.

[FF92] D.A. Field and R. Field. A new family of curves for industrial applications. Technical report GMR-7571, General Motors Research Laboratories, 1992.

[FN90] R.T. Farouki and C.A. Neff. Analytic properties of plane offset curves. *Computer Aided Geometric Design*, 7:83–99, 1990.

[FS90] R.T. Farouki and T. Sakkalis. Singular points of algebraic curves. *Journal of Symbolic Computation*, 9(4):457–483, 1990.

[GBDM77] B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51. Springer-Verlag, Berlin, 1977.

[Gei83] A. Geisow. *Surface Interrogations.* PhD thesis, School of Computing Studies and Accountancy, University of East Anglia, 1983.

[GL89] G.H. Golub and C.F. Van Loan. *Matrix Computations.* John Hopkins Press, Baltimore, 1989.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, California, 1989.

[Hof90] C.M. Hoffmann. A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7:517–532, 1990.

[Hoh91] M.E. Hohmeyer. A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications*, 1(4):473–490, 1991. Special issue on Solid Modeling.

[KPP90] G.A. Kriezis, P.V. Prakash, and N.M. Patrikalakis. Method for intersecting algebraic surfaces with rational polynomial patches. *Computer-Aided Design*, 22(10):645–654, 1990.

[KPW90] G.A. Kriezis, N.M. Patrikalakis, and F.E. Wolter. Topological and differential equation methods for surface intersections. *Computer-Aided Design*, 24(1):41–55, 1990.

[LR80] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):150–159, 1980.

[Man94a] D. Manocha. Computing selected solutions of polynomial equations. In *Proceedings of International Symposium on Symbolic and Algebraic Computation*, pages 1–8, Oxford, England, 1994. ACM Press.

[Man94b] D. Manocha. Solving systems of polynomial equations. *IEEE Computer Graphics and Applications*, pages 46–55, March 1994. Special Issue on Solid Modeling.

[MC91] D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991. Special issue on Solid Modeling.

[Moo79] R.E. Moore. *Methods and applications of interval analysis.* SIAM studies in applied mathematics. Siam, 1979.

[Mor92] A. P. Morgan. Polynomial continuation and its relationship to the symbolic reduction of polynomial systems. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 23–45, 1992.

[MP93] T. Maekawa and N. M. Patrikalakis. Computation of singularities and intersections of offsets of planar curves. *Computer Aided Geometric Design*, 10, 1993.

[RR92] A.A.G. Requicha and J.R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, pages 31–44, September 1992.

[Sed83] T.W. Sederberg. *Implicit and Parametric Curves and Surfaces.* PhD thesis, Purdue University, 1983.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 211–215, 1990.

[SKW85] P. Sinha, E. Klassen, and K.K. Wang. Exploiting topological and geometric properties for selective subdivision. In *ACM Symposium on Computationl Geometry*, pages 39–45, 1985.

[SM88] T.W. Sederberg and R.J. Meyers. Loop detection in surface patch intersections. *Computer Aided Geometric Design*, 5:161–171, 1988.

[THS89] Sederberg T.W, Christiansen H.N, and Katz S. An improved test for closed loops in surface intersections. *Computer-Aided Design*, 21(8):505–508, 1989.

[ZS93] A. Zundel and T. Sederberg. Using pyramidal surfaces to detect and isolate surface/surface intersections. In *SIAM Conference on Geometric Design*, Tempe, AZ, 1993.