

Symbolic-Numeric Methods of Loop Detection for Curve and Surface Interrogations*

Shankar Krishnan

Dinesh Manocha

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

Abstract: We present efficient algorithms based on a combination of numeric and symbolic techniques for detecting closed components (loops) of one-dimensional algebraic sets in a subset of the real domain. This paper describes two techniques for loop detection. Given a description of a one-dimensional algebraic set, the first algorithm computes its projection using resultants. The resulting plane curve is represented as a singular set of a matrix polynomial as opposed to roots of a bivariate polynomial. Given this matrix formulation, we make use of algorithms from numerical linear algebra to compute start points on all the loop components. This algorithm has been implemented in floating-point arithmetic and we highlight its performance in the context of computing intersections and *silhouettes* of high-degree rational algebraic surfaces. The second method is based on a symbolic approach to detecting critical points of a plane vector field. This method, is however, restricted to finding loops while computing planar sections of surfaces. Unlike some other numerical loop detection algorithms, both our methods can be applied even when the algebraic curve contain(s) singularities.

*Supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract DAAH04-96-1-0257, NSF Grant CCR-9319957, NSF Grant CCR-9625217, ONR Contract N00014-94-1-0738, ARPA Contract DABT63-93-C-0048 and NSF/ARPA Center for Computer Graphics and Scientific Visualization

1 Introduction

The problem of evaluating one-dimensional algebraic sets is fundamental in numeric and symbolic computing. It can be simply stated as finding roots of n affine algebraic equations in $(n+1)$ unknowns. Algebraic sets are widely used for representing objects and constraints in computer graphics, geometric modeling, robotics, computer vision and molecular modeling. Many of the fundamental problems like surface-surface intersection, offsets of curves and surfaces, slicing operations on surface models, Voronoi sets generated by curves and surfaces in geometric modeling [Hof89], kinematic analysis of a redundant robot [Cra89], robot motion planning [Can88], object recognition in computer vision [PK92] and conformation space of molecular chains [CH] correspond to evaluating one-dimensional algebraic sets. In most cases we are interested in evaluating all the components in the subset of the real domain.

The resulting algebraic curves typically are of high degree with multiple components (see Fig. 1). 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 the two surfaces in Figure 2 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 high-degree curves continues to be a major challenge. In this paper, we present two techniques for efficient and accurate evaluation of loops using a combination of symbolic and numeric methods.

Main Results: We present two algorithms for detecting loops of one-dimensional algebraic sets in a subset of the real domain. Given a curve, we compute a birationally equivalent algebraic plane curve using resultants in the first method. The plane curve is represented as the singular set of a matrix polynomial (or a ratio of matrix polynomials). Given the matrix representation, we make use of algorithms based on eigenvalues and complex tracing to compute a start point on each loop component of the curve in the given domain. Unlike some of the previous numerical loop detection algorithms, our method can be applied even when the curve contains singularities. The overall algorithm has been implemented in finite precision arithmetic and works well for well-conditioned problems. Its main advantages are *efficiency* and *accuracy*. The resulting algorithm is iterative and its performance is a function of the degree of the algebraic curve, the number of components in the given domain and the accuracy desired. In practice, we have been able to evaluate all the loops of curves of degree

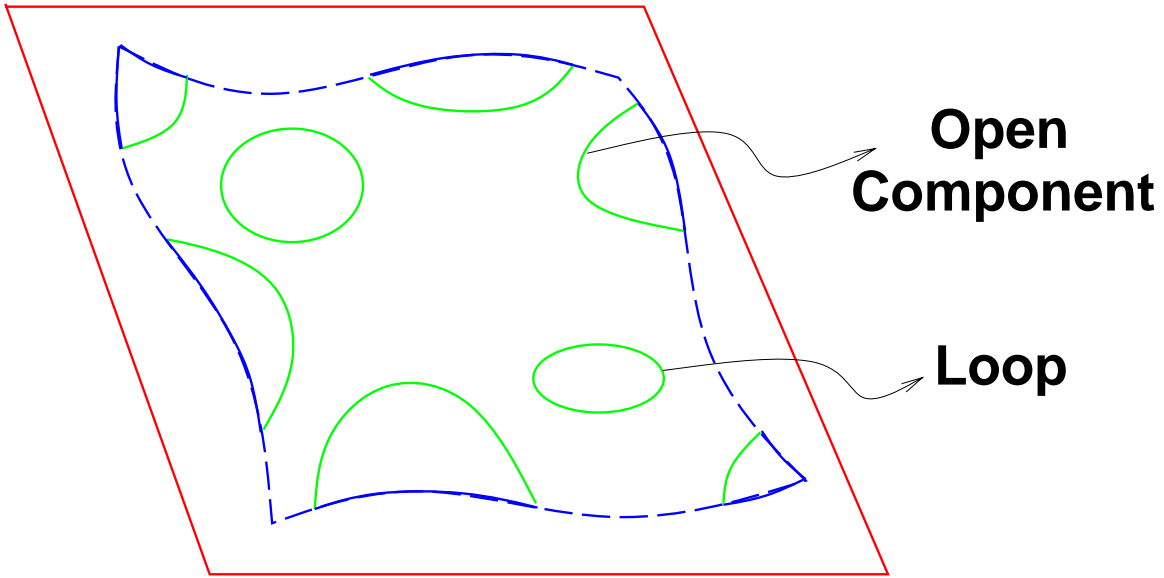


Figure 1: Multiple components of algebraic curves

as high as 324 in a few seconds on an SGI with a 200 MHz clock and R4400 CPU. We discuss the algorithm's performance evaluating the intersection of two algebraic surfaces, boundary computation and evaluating the *silhouettes* of a parametric surface patch from a given viewpoint.

The second algorithm presents a technique to robustly determine all the loops of the intersection curve arising out of planar sections of surface models. In particular, we count all the critical points of a planar vector field which corresponds to presence of loops. The algorithm employs the use of *multivariate Sturm sequences* [Mil92] in determining the critical points. We have implemented this algorithm in exact rational arithmetic, although, modifying it to double-precision arithmetic is fairly straightforward.

Prior Work: There is a considerable amount of work in classic and modern literature related to evaluation of algebraic curves. Every algebraic space curve is birationally equivalent to an algebraic plane curve and the latter can be computed using Gröbner bases [Buc89] and resultants. Given an algebraic plane curve, techniques for desingularization based on quadratic transformations are given in [Wal50, Abh90, AB88]. However, the resulting algorithm can be exponential in the degree of the curve. Algorithms based on Collins' *cylindrical algebraic decomposition* (CAD), [Col75, ACM84], have been used for evaluating all components of algebraic curves [Arn83, SS83]. However, its worst case complexity is doubly exponential in the number of variables. For plane curves, improved polynomial

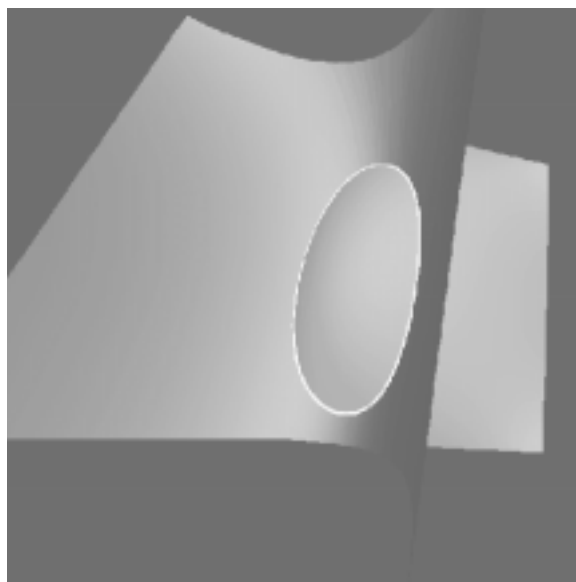


Figure 2: Two surfaces intersecting in a loop

time algorithms based on CAD have been presented in [AF88, AM88]. Other algorithms include those based on Whitney's stratified sets and gap theorems [Can88]. In practice all these algorithms are efficient for low degree curves only. Numerical and finite precision algorithms based on interval arithmetic [Moo79] and homotopy methods [GZ79, Mor92] have been used for evaluating algebraic sets. While the former are slow in practice, the latter have been restricted to zero-dimensional algebraic sets and suffer from problems like component jumping. There is a considerable amount of emphasis in the modeling literature to evaluate surface intersections and offset curves [Hof89, Hof90, SN91, MC91, Hoh91] and in vision literature to compute aspect graphs [PK92]. This includes algorithms for computing all components including the closed loops. However, these algorithms are somewhat restrictive and cannot be used for evaluating general algebraic curves. The combination of resultant formulations and matrix computations have been used for evaluating zero-dimensional algebraic sets in [Laz83, AS86, Man92, Man94].

The problem of evaluating all the loops of an algebraic curve numerically has been extensively studied in the modeling literature and a number of techniques based on subdivision methods, marching methods and lattice evaluations [Hof89, RR92] have been developed. 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, KM96, KM97]. 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, Kim90, 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. 2 takes 8 levels of subdivision. Furthermore, these algorithms may not work

well if the intersection curve is self-intersecting.

Techniques based on finding critical points of plane vector fields inside the domain of the surfaces have been proposed by [Che89, KPP90, KPW90, ML95]. Cheng [Che89] defined a plane vector field as the gradient of an oriented distance function of one surface from the other. The critical points are found by following special integral curves that connect all the critical points. [KPP90, KPW90, ML95] use rotational indices of (planar and three-dimensional) vector fields to determine presence of critical points. However, these methods rely on some form of subdivision in the domain, and hence, cannot robustly guarantee detection of all the critical points.

The algorithms based on Gauss maps and vector fields 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 first loop detection algorithm. The implementation of this algorithm and its performance on various applications are highlighted in section 4. Section 5 presents the formulation of loop detection in surface slicing problem as critical points of a vector field. The details of our algorithm along with a brief introduction to multivariate Sturm sequences is presented in section 6. We demonstrate this algorithm on a few examples in section 7 and we conclude in section 8.

2 Algebraic Formulation of the curve

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

$$\begin{aligned}
 F_1(u_1, u_2, \dots, u_{n-1}, u, v) &= 0 \\
 F_2(u_1, u_2, \dots, u_{n-1}, u, v) &= 0 \\
 &\vdots \\
 F_n(u_1, u_2, \dots, u_{n-1}, u, v) &= 0.
 \end{aligned} \tag{1}$$

Moreover, we are only interested in evaluating all the components of the curve inside the region $D = [U_{11}, U_{12}] \times [U_{21}, U_{22}] \times \dots \times [U_{n+1,1}, U_{n+1,2}] \in \mathcal{R}^{n+1}$. Formally, the functions $F_i, i = 1, 2, \dots, n$, 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 $\mathbf{G}(u, v) = (\bar{X}(u, v), \bar{Y}(u, v), \bar{Z}(u, v), \bar{W}(u, v))$ 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:

$$\begin{aligned}
 F_1(s, t, u, v) &= X(s, t)\bar{W}(u, v) - \bar{X}(u, v)W(s, t) = 0 \\
 F_2(s, t, u, v) &= Y(s, t)\bar{W}(u, v) - \bar{Y}(u, v)W(s, t) = 0 \\
 F_3(s, t, u, v) &= Z(s, t)\bar{W}(u, v) - \bar{Z}(u, v)W(s, t) = 0,
 \end{aligned} \tag{2}$$

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. We eliminate $n - 1$ variables from equation (1) using *multipolynomial resultant* algorithms. Almost all the projections are one-to-one and result in a birationally equivalent plane curve. In our case, we perform a generic linear transformation and eliminate u_1, \dots, u_{n-1} from the resulting set. The resultant can be expressed in terms of matrices and determinants. In particular, single determinant formulations are known for values of $n = 2, 3, 4, 5, 6$ [Dix08, Jou91, MC27, SZ94]. We use

$\mathbf{M}(u, v)$ to represent the resulting matrix polynomial. The most general formulation of the resultant expresses it as a ratio of two determinants [Mac02]. Let us denote the top and bottom matrices as $\mathbf{P}(u, v)$ and $\mathbf{Q}(u, v)$ respectively. A similar formulation for $\mathbf{P}(u, v)$ for sparse polynomial systems has been highlighted in [CE93]. In our case, we will make use of the matrix formulation and represent it as an *unevaluated determinant*. It is possible that both $\mathbf{P}(u, v)$ and $\mathbf{Q}(u, v)$ are singular, while the resultant is non-zero. In such cases we use the leading non-vanishing minor of $\mathbf{P}(u, v)$ and represent it as $\mathbf{M}(u, v)$. It contains the resultant and an extraneous factor. The algorithm evaluates the resulting algebraic set and substitutes the values back into the original equations to discard the solutions corresponding to the extraneous factor. The degree of the algebraic curve, N , is given by the Bezout or Bernstein bound of the given system of equations.

The plane curve birationally equivalent to the algebraic curve corresponds to the singular set of $\mathbf{M}(u, v)$. For the general Macaulay's formulation the plane curve corresponds to the difference of singular set of $\mathbf{P}(u, v)$ and singular set of $\mathbf{Q}(u, v)$ taking into account the multiplicities of individual factors. For the rest of the paper, we perform a number of numerical computations like determinants, eigenvalues, singular values of $\mathbf{M}(u_i, v_i)$ and similar analysis is applicable in the general case based on the ratio of $\mathbf{P}(u, v)$ and $\mathbf{Q}(u, v)$. Given a point on the plane curve, (u_0, v_0) , the corresponding point on the space curve, $(w_{1_0}, w_{2_0}, \dots, w_{n-1_0})$, is computed using the *kernel* of $\mathbf{M}(u_0, v_0)$ [Man92]. The algorithms in the rest of the paper is described for applications involving surfaces (intersection curves and silhouettes), but these techniques are general.

The plane curves with one-to-one correspondence with the intersection curve in space are shown in Fig. 3. 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 parametric surfaces. However, our method can be easily applied to 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 *Bézier patch*. A Bézier patch is an often used parametric surface in many solid modeling

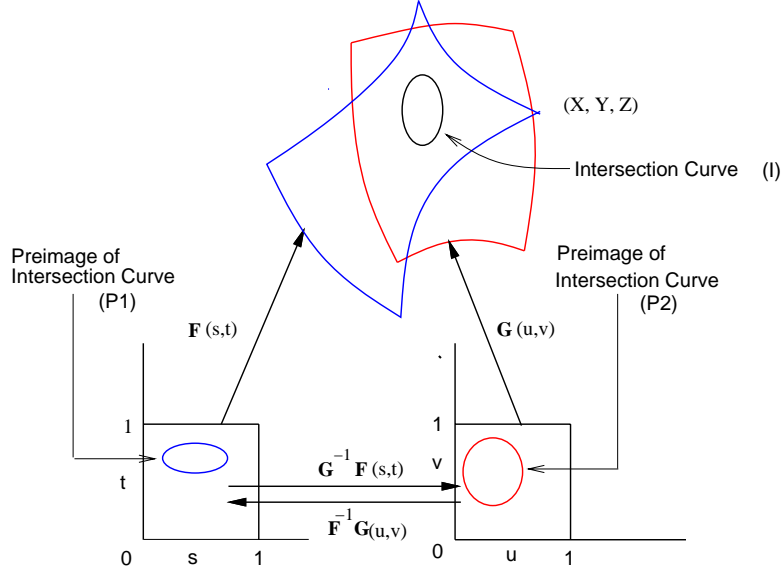


Figure 3: Intersection curve and its planar preimages

and computer graphics applications, and is of the form

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

where $V_{ij} = (x_{ij}, y_{ij}, z_{ij}, w_{ij})$ are the *control point coordinates* and $B_{i,m}(s) = \binom{m}{i} s^i (1-s)^{m-i}$ is the Bernstein polynomial. The popularity of these surfaces is due to the fact that their shape is determined by its control points.

Given two Bézier surfaces, $\mathbf{F}(s, t) = (X(s, t), Y(s, t), Z(s, t), W(s, t))$ and $\mathbf{G}(u, v) = (\bar{X}(u, v), \bar{Y}(u, v), \bar{Z}(u, v), \bar{W}(u, v))$ 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

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

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

$$\begin{aligned} xW(s, t) - X(s, t) &= 0 \\ yW(s, t) - Y(s, t) &= 0 \\ zW(s, t) - Z(s, t) &= 0 \end{aligned} \tag{3}$$

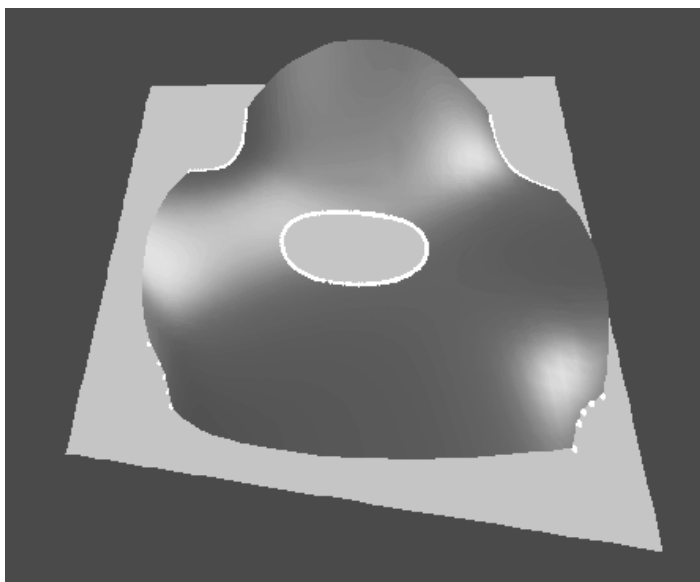


Figure 4: A pair of intersecting surfaces

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 $\mathbf{G}(u, v)$.

2.2 Parameterizations with base points

The base points of a parameterization are the common solutions of the equations:

$$\begin{aligned} X(s, t) = 0, & \quad Y(s, t) = 0, \\ Z(s, t) = 0, & \quad W(s, t) = 0 \end{aligned} \tag{4}$$

The base points also include common solutions at infinity. In general, any faithful parameterization

of a rational surface whose algebraic degree is not a perfect square has base points. Consider a base point $\mathbf{p} = (s', t')$. By definition,

$$X(s', t') = Y(s', t') = Z(s', t') = W(s', t') = 0$$

It is therefore obvious that (s', t') is a non-trivial solution to equation (3). Further, this is a solution irrespective of the values of x, y or z . Therefore, the resultant of these set of equations is identically zero.

Checking for base points: The most obvious way to identify if the given parameterization contains base points is to compute all the common solutions of equation (4). The basic idea is to find all the solutions to two of the equations (say, $X(s, t) = Y(s, t) = 0$). Each element of the solution set (assuming a finite set) is tested for satisfiability by substituting in the other two equations to recover the base points. However, this method cannot detect parameterizations which are very close to having base points (we call it the *near base point case*). Further, recovering the implicit form of the surface is not possible from this method.

It was mentioned earlier that in the presence of base points, the resultant of equations (3) is identically zero (independent of the values of x, y or z). Therefore, the matrix $\overline{\mathbf{M}}(x, y, z, w)$ (whose determinant gives the resultant) is always singular (rank deficient). Singular Value Decomposition (SVD) is a popular method to find the rank of a matrix. We substitute random values for x, y and z (making sure they do not lie on the original surface) in the matrix and perform SVD. If it contains zero singular values, it implies that the given parameterization contains base points. This method can also identify parameterizations with *near base points*. In such cases, some singular values of the matrix $\overline{\mathbf{M}}(x, y, z, w)$ are very close to zero. We treat near base point cases as if they contain base points (by zeroing the corresponding singular values).

Computing the implicit form: The resultant (determinant of $\overline{\mathbf{M}}(x, y, z, w)$) provides the implicit representation of the surface if its parameterization does not contain base points. However, in their presence, it is not possible. However, it was shown in [MC91] that the rank submatrix (maximal non-vanishing minor) contains the implicit form as a factor in such cases. Therefore, in order to obtain the implicit representation of the surface, we have to find the rank submatrix. This can be achieved by performing Gaussian elimination on the original matrix. Substitution of the parameterization of $\mathbf{G}(u, v)$ into this minor gives us planar projection of the intersection curve. It must be observed that the rank submatrix could contain extraneous factors (other than the implicit form) and must be eliminated by testing the solutions obtained with the original set of surface equations.

2.3 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 $\mathbf{F}(u, v)$.

2.4 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.

$$\begin{aligned} \mathbf{F}(u, v) &= \langle X(u, v), Y(u, v), Z(u, v), W(u, v) \rangle \\ \phi_1(u, v) &= \frac{X(u, v)}{W(u, v)}, \quad \phi_2(u, v) = \frac{Y(u, v)}{W(u, v)}, \quad \phi_3(u, v) = \frac{Z(u, v)}{W(u, v)} \end{aligned}$$

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

$$N_z = \begin{vmatrix} \phi_{1u} & \phi_{1v} \\ \phi_{2u} & \phi_{2v} \end{vmatrix} \quad (5)$$

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

$$\begin{aligned} \phi_{1u} &= \frac{(W X_u - W_u X)}{W^2} & \phi_{1v} &= \frac{(W X_v - W_v X)}{W^2} \\ \phi_{2u} &= \frac{(W Y_u - W_u Y)}{W^2} & \phi_{2v} &= \frac{(W Y_v - W_v Y)}{W^2} \end{aligned}$$

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

$$N_z = \begin{vmatrix} (W X_u - W_u X) & (W X_v - W_v X) \\ (W Y_u - W_u Y) & (W Y_v - W_v Y) \end{vmatrix} = 0 \quad (6)$$

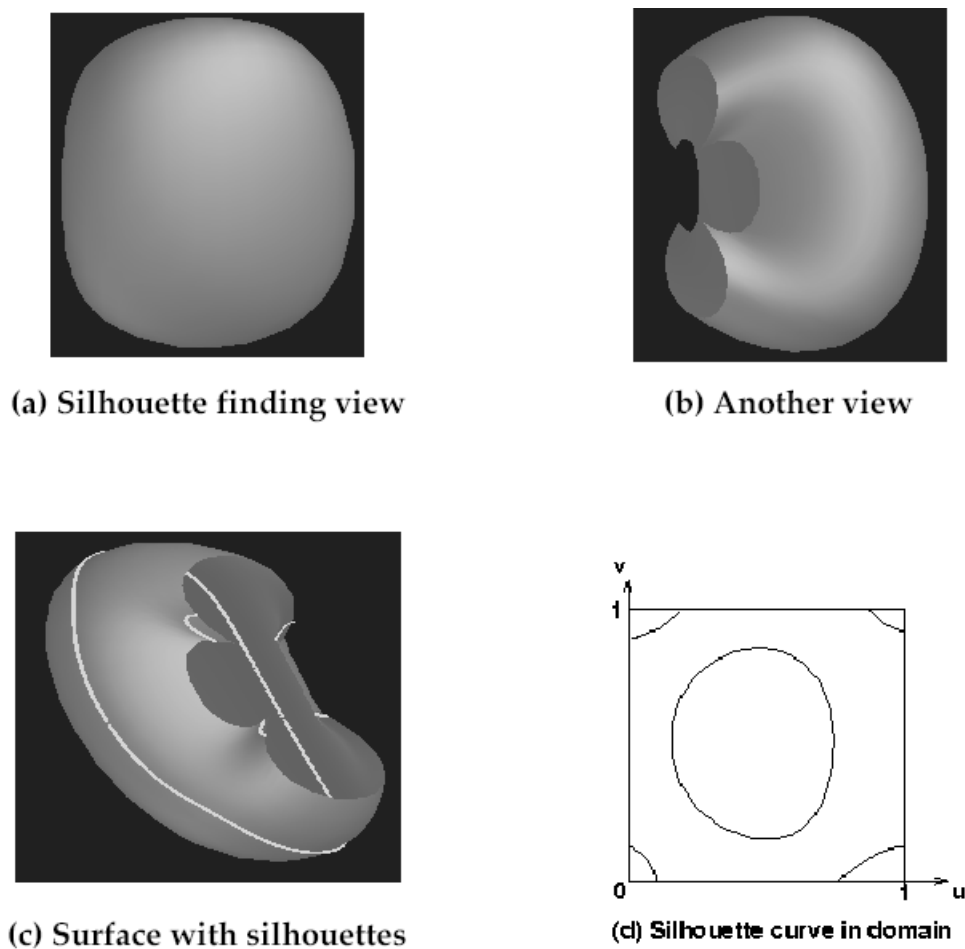


Figure 5: Loop as part of a silhouette curve

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

$$\mathbf{M}(u, v) = \begin{pmatrix} X(u, v) & Y(u, v) & W(u, v) \\ X_u(u, v) & Y_u(u, v) & W_u(u, v) \\ X_v(u, v) & Y_v(u, v) & W_v(u, v) \end{pmatrix} = 0 \quad (7)$$

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

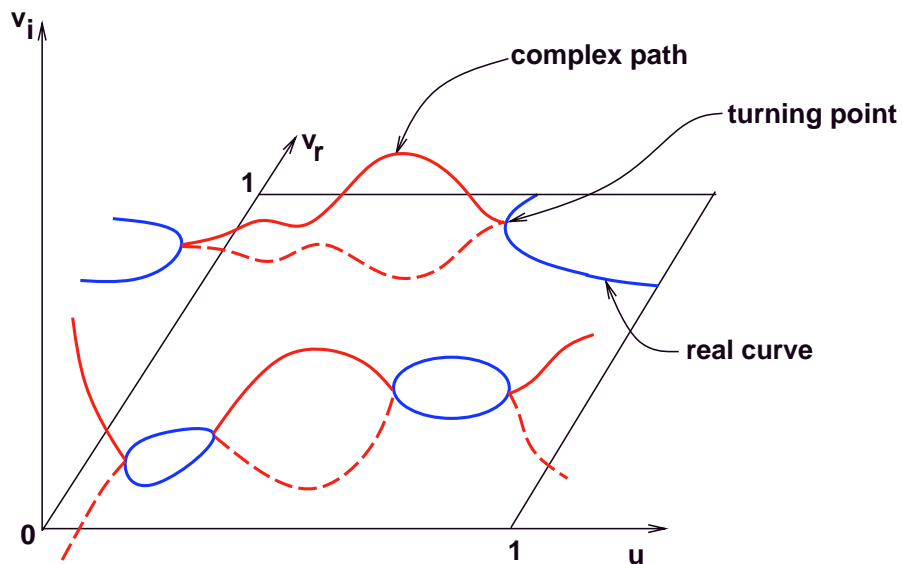


Figure 6: Algebraic curve continuous in complex projective plane

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 (see Fig. 6). 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

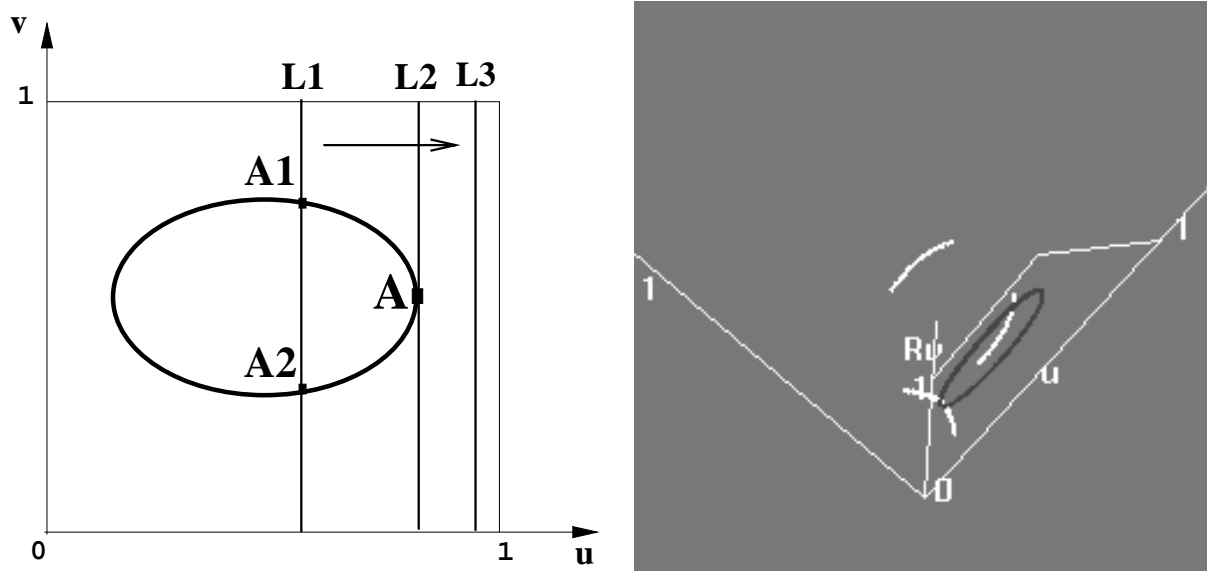


Figure 7: Characterization of loops based on complex tracing

($u = u_1 - \delta$, 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.*

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.2. 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 the Bezout bound in complex space. Fig.7 (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.7 (right). The third axis corresponds to the imaginary components of v . It represents a continuous component of the intersection curve. The white curve is the intersection curve in the complex space and the dark 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

$$\mathbf{M}(v) = v^d M_d + v^{d-1} M_{d-1} + \dots + v M_1 + M_0 = 0. \tag{8}$$

where M_i 's are numeric matrices (of order $2mn$ for the intersection curve and 3 for the silhouette curve) and d is the maximum degree of v in $\mathbf{G}(u, v)$. The solution of this matrix equation can be

reduced to the eigenvalues of an associated companion matrix of order $2mnd$.

$$C = \begin{bmatrix} 0 & I_n & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & I_n \\ -\overline{\mathbf{M}}_0 & -\overline{\mathbf{M}}_1 & -\overline{\mathbf{M}}_2 & \dots & -\overline{\mathbf{M}}_{d-1} \end{bmatrix} \quad (9)$$

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 [Man94]. 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. (9)). 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

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

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 v_2 , 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 v_2 (in this case, $-v_2$), and \mathbf{P}_i 's are $n \times n$ matrices which are functions of \mathbf{M}_i 's and v_2 . There is no extra cost involved in setting up the matrix \mathbf{B} (it can be directly obtained from equation (8)). 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. Performing LQ factorization is about twice as costly as LU decomposition. The conditioning of the LU decomposition can be estimated by performing LU decomposition on \mathbf{R}_m (a much smaller matrix) and comparing the diagonal elements of \mathbf{L}_m and \mathbf{U}_m (*i.e.*, it is well-conditioned if the diagonal elements are of the same order of magnitude). LQ factorization can be performed if LU decomposition thus performed is ill-conditioned.

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.4). 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

$$\det(\mathbf{M}_r(u, v_r) + i\mathbf{M}_i(u, v_r)) = 0 \quad (10)$$

It is easy to show that the solutions (u, v_r) satisfying equation (10) also satisfy the solution of $\det \mathbf{P}(u, v_r) = 0$, where

$$\mathbf{P}(u, v_r) = \begin{bmatrix} \mathbf{M}_r(u, v_r) & -\mathbf{M}_i(u, v_r) \\ \mathbf{M}_i(u, v_r) & \mathbf{M}_r(u, v_r) \end{bmatrix} \quad (11)$$

As before, the solutions to (11) 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.(9). 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.

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

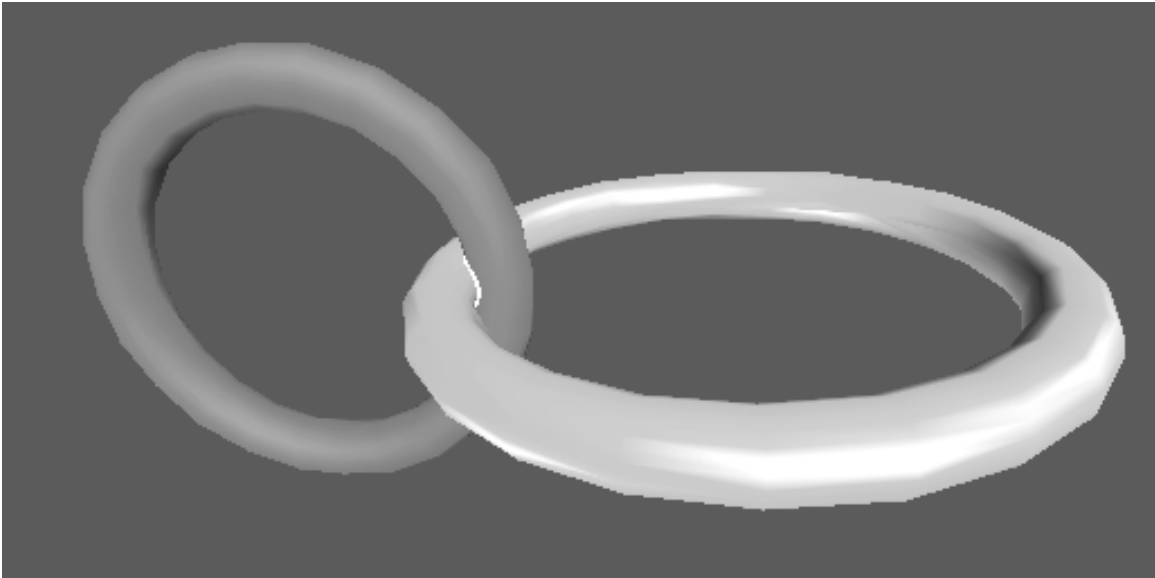


Figure 8: Two tori intersecting in a small loop

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. 2 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. 2. 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. 2 and Fig. 8, our method performs better than Hohmeyer’s algorithm. His method, however, performed better when applied to the surfaces in Fig. 4.

The surface intersection algorithm is part of a solid modeling system which computes the B-rep (boundary representation) of CSG solids. Fig. 9 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

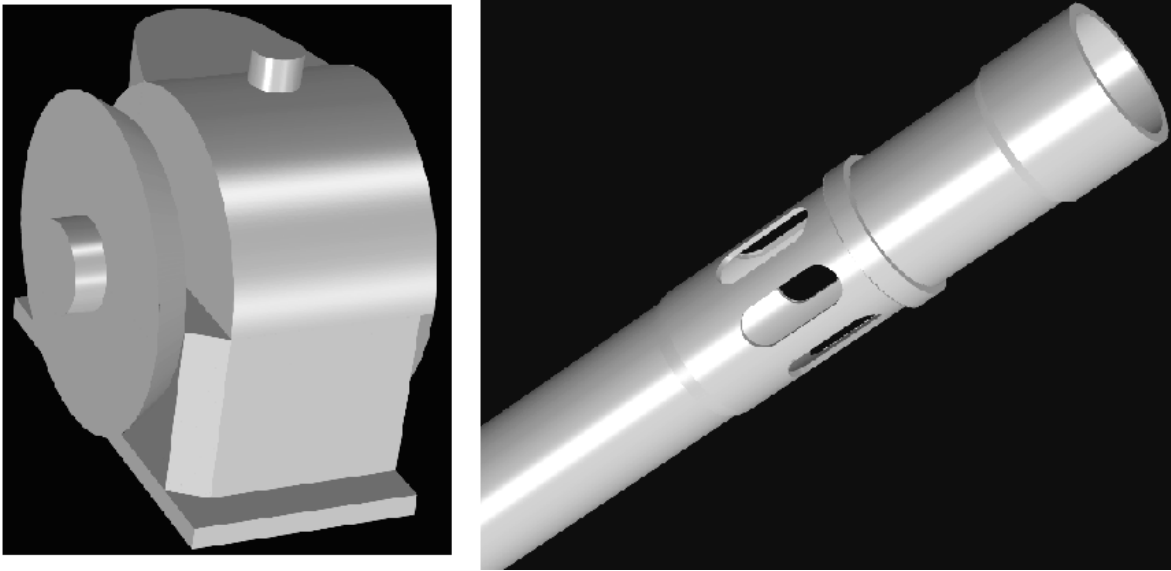


Figure 9: Loop detection used in B-rep computation

compute the silhouettes of surfaces. Fig. 5 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. 5(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. 5(d).

5 Loop Detection in Surface Sectioning

In this section, we describe an algorithm to perform loop detection on intersection curves obtained by taking planar sections of surface models. This operation is widely used in rapid prototyping to obtain

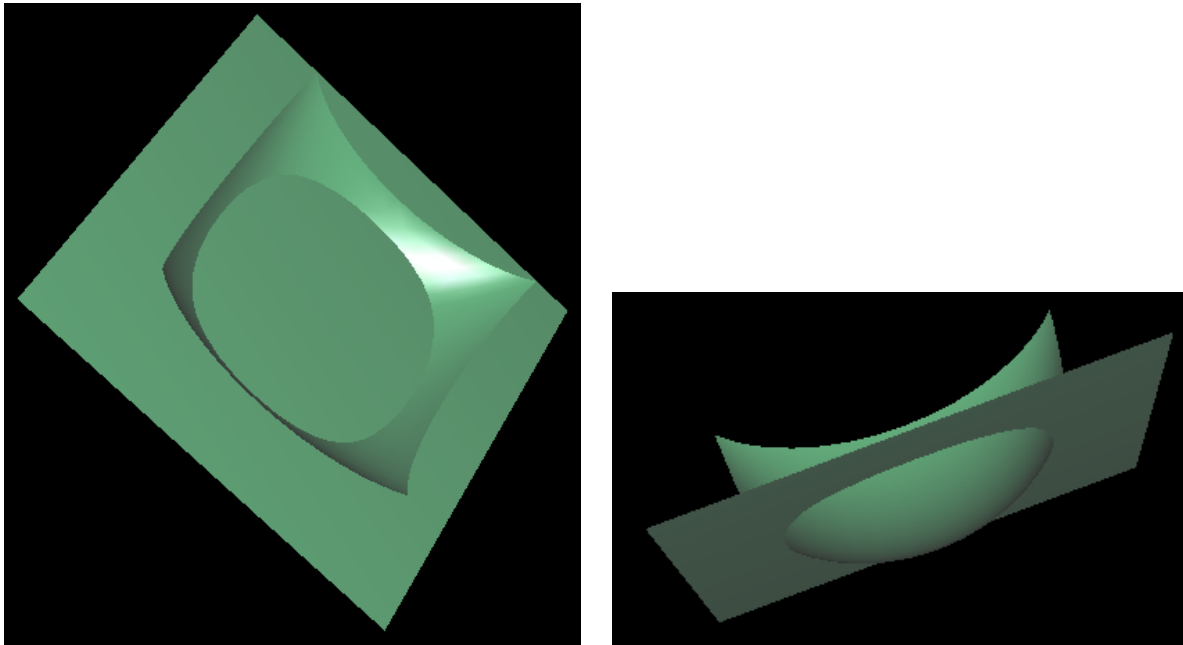


Figure 10: Intersection of a plane with a biquadratic surface

cross-sectional information of such models. A major concern in such applications is the correctness of the resulting curve *i.e.*, its topological type and detection of all components.

Fig. 10 shows a simple example of a biquadratic surface intersected by a plane. In this case, the intersection curve has a single loop component. We shall now formulate the loop detection problem as critical points of a plane vector field. The vector field is obtained as the gradient of a distance function introduced by [Che89]. We shall adopt the same notation in this paper.

5.1 Intersection formulation using distance function

The intersection set between a pair of parametric surfaces can be formulated as a minimization problem in which the distance between two variable points on the two surfaces becomes zero. Basically, the intersection set can be expressed as the sequence of points in the two surfaces with zero distance between them.

The oriented distance function ϕ between a surface $\mathbf{Q}(s, t)$ and a point moving on another surface

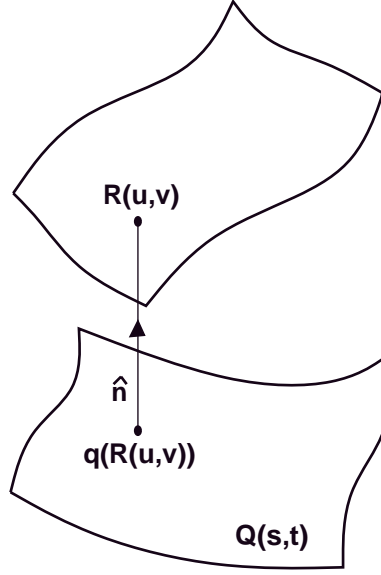


Figure 11: Distance function between two surfaces

$\mathbf{R}(u, v)$ is defined on the (u, v) parameter space as follows:

$$\phi(u, v) = \hat{\mathbf{n}}[\mathbf{q}(\mathbf{R}(u, v))] \bullet [\mathbf{R}(u, v) - \mathbf{q}(\mathbf{R}(u, v))] \quad (12)$$

where $\mathbf{q}(\mathbf{R}(u, v))$ is a point on the surface $\mathbf{Q}(s, t)$ which is nearest to the point $\mathbf{R}(u, v)$, and $\hat{\mathbf{n}}$ is the unit normal vector on $\mathbf{Q}(s, t)$ at the point $\mathbf{q}(\mathbf{R}(u, v))$ (see Fig. 11). \bullet is the dot product operator for vectors. In our case, however, one of the surfaces is a plane, and the normal is constant at all points (say $\hat{\mathbf{n}}$). Therefore the distance function becomes

$$\phi(u, v) = \hat{\mathbf{n}} \bullet [\mathbf{R}(u, v) - \mathbf{q}(\mathbf{R}(u, v))] \quad (13)$$

Assuming that ϕ is a well-defined distance function, the intersection set is the zero set of ϕ . There are cases when ϕ is not well-defined (when there are more than one closest point to $\mathbf{R}(u, v)$, or when the line joining the two points $\mathbf{R}(u, v)$ and $\mathbf{q}(\mathbf{R}(u, v))$ is not collinear to $\hat{\mathbf{n}}$ because of patch boundaries). However, these special cases stay away from loops in the intersection curve. Therefore, we can assume for the purposes of this paper that ϕ is well-defined.

5.2 Collinear normal points and Distance function

Sederberg [SM88, THS89] was the first to recognize the importance of collinear normals in detecting existence of closed loops in intersection problems. It is easy to see that collinear normal points between two surfaces are critical points of the distance function ϕ . This is because collinear normal points are extremal distance point pairs and the gradient vector of the distance function is zero. Therefore, if we have a method to find the number of collinear normal points within a particular domain of interest, we can use a simple subdivision scheme to compute these points to arbitrary precision. Most existing methods use the idea of rotational index of the vector field inside a closed curve [KPP90, KPW90]. However, this test is inconclusive because if a particular region contains two critical points of opposite rotational index, then we obtain net rotational index of zero. Recently [ML95] extended this to a three-dimensional vector field such that rotational index of this field decides conclusively the number of critical points (provided they are *non-degenerate*). However, their method is susceptible to failure if the sampling grid in the domain contains contours of zero Jacobians completely inside them. Further, the use of local minimization methods and Newton type marching methods to locate all the critical points error-prone.

The following theorem formulates the gradient of the distance function. The critical points of this vector field provides the set of collinear normal points.

Theorem 1 *Given the oriented distance function ϕ as in eq. (13), the gradient is given by*

$$\phi_u(u, v) = \hat{\mathbf{n}} \bullet \mathbf{R}_u(u, v) \phi_v(u, v) = \hat{\mathbf{n}} \bullet \mathbf{R}_v(u, v) \quad (14)$$

Proof: The distance function $\phi(u, v)$ is given by

$$\phi(u, v) = \hat{\mathbf{n}} \bullet [\mathbf{R}(u, v) - \mathbf{q}(\mathbf{R}(u, v))]$$

Taking partial derivative with respect to u , we get

$$\begin{aligned} \phi_u(u, v) &= \hat{\mathbf{n}} \bullet [\mathbf{R}_u(u, v) - \mathbf{q}_u(\mathbf{R}(u, v))] + \hat{\mathbf{n}}_u \bullet [\mathbf{R}(u, v) - \mathbf{q}(\mathbf{R}(u, v))] \\ &= \hat{\mathbf{n}} \bullet \mathbf{R}_u(u, v) \end{aligned}$$

This is because $\hat{\mathbf{n}}_u$ is zero because normal does not change for a planar patch, and so is $\hat{\mathbf{n}} \bullet \mathbf{q}_u(\mathbf{R}(u, v))$ since \mathbf{q}_u lies in the tangent plane of $\mathbf{Q}(s, t)$ and $\hat{\mathbf{n}}$ is the normal to it.

The result for partial w.r.t v can be proved similarly.

□

6 Loop Detection Algorithm

In this section, we describe our loop detection algorithm based on finding all the critical points of a two-dimensional vector field. Sturm sequences were introduced by Hermite (1853) in order to count the number of real roots of a univariate polynomial inside a given interval. Sturm sequences are generated by performing gcd computation using Euclid's algorithm on the given polynomial and its negative derivative. The number of real roots is computed by looking at the number of sign changes of this sequence at the endpoints of the interval. Extending this idea to multivariate polynomial systems (that yield zero-dimensional solution sets) has been the focus of research for quite some time. Milne [Mil92] introduced the *volume function* which essentially achieved the extension to multivariate polynomial systems.

6.1 Multivariate Sturm sequences

Here, we describe briefly the algorithm proposed by Milne [Mil92] to compute the number of common real solutions of n polynomials in n variables inside an n -dimensional rectangle. This algorithm is an extension of the univariate case which constructs a polynomial sequence, and measures sign variations of this sequence at the endpoints of the interval. We restrict ourselves to the case when $n = 2$.

Given two polynomials, $f_1(s, t)$ and $f_2(s, t)$, we construct the *volume function*, $V(u, s, t)$, as follows:

$$V(u, s, t) = \frac{Res_{a_2}(Res_{a_1}(f_1(a_1, a_2), f_3), Res_{a_1}(f_2(a_1, a_2), f_3))}{u^{deg(f_1(s,0))deg(f_2(s,0))}},$$

where $f_3(u, s, t, a_1, a_2) = u + (s - a_1)(t - a_2)$, Res_x refers to the resultant of two polynomials after eliminating x , and deg refers to the degree of the polynomial. We use the Sylvester resultant [Sal85] to eliminate one variable from two polynomials.

Sylvester's method [Sal85] can be used to express the resultant of two polynomials of degree m and n respectively as a determinant of a matrix with $(m + n)$ rows and columns. For the polynomials,

$$f^n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (15)$$

and

$$g^m(x) = b_m x^m + b_{m-1} x^{m-1} + \dots + b_1 x + b_0 \quad (16)$$

where $n \geq m$, the Sylvester's resultant is

$$\begin{vmatrix}
a_n & a_{n-1} & \dots & & a_0 & 0 & \dots & 0 \\
0 & a_n & a_{n-1} & \dots & a_0 & 0 & \dots & 0 \\
0 & \dots & 0 & a_n & a_{n-1} & & \dots & a_0 \\
0 & \dots & & 0 & b_m & b_{m-1} & \dots & b_0 \\
0 & \dots & 0 & b_m & b_{m-1} & \dots & b_0 & 0 \\
b_m & b_{m-1} & \dots & b_0 & 0 & \dots & & 0
\end{vmatrix} \tag{17}$$

We use an algorithm based on multivariate interpolation [MC93] to compute the resultant of a set of polynomials efficiently. The main bottleneck in most resultant algorithms is the symbolic expansion of determinants. Most of the computer algebra systems use symbolic algorithms like polynomial manipulations for resultants, which are very expensive. Further, the magnitude of intermediate expressions grows quickly, and the memory requirements are high. The algorithm in [MC93] performs all computations over finite fields, and uses a probabilistic algorithm based on the Chinese Remainder Theorem to recover actual coefficients.

A practical implementation of the Sylvester resultant introduces extraneous factors in the resultant that must be removed. For the special bilinear form of f_3 , we can show that the extraneous factor introduced by Sylvester resultant is u^{pq} , where p and q are the maximum degrees of s in f_1 and f_2 respectively.

Given a square-free polynomial $p(x)$ we can construct a Sturm sequence of polynomials $S_i = -\text{remainder}(S_{i-2}(x), S_{i-1}(x))$, where $S_1(x) = p(x)$ and $S_2(x) = p'(x)$. Treating the volume function V as a univariate polynomial in u , we construct its Sturm sequence $S_i(u, s, t)$. The Sturm sequence is specialized at $u = 0$ to give a sequence of bivariate polynomials $M(s, t)$.

Definition 3 Given a sequence of polynomials $M(s, t)$ of length n , the \mathbf{V} operator at (a_1, a_2) ($\mathbf{V}(M(a_1, a_2))$) gives the number of sign changes between consecutive terms of the sequence evaluated at (a_1, a_2) . Correspondingly, the \mathbf{P} operator is defined as $\mathbf{P}(M(a_1, a_2)) = n - 1 - \mathbf{V}(M(a_1, a_2))$.

Given the bivariate sequence $M(s, t)$ and a rational axis aligned rectangle $\mathcal{R} = [a_1, b_1] \times [a_2, b_2]$, the number of real roots of f_1 and f_2 inside \mathcal{R} is given by

$$\frac{\mathbf{P}(M(b_1, b_2)) + \mathbf{P}(M(a_1, a_2)) - \mathbf{P}(M(b_1, a_2)) - \mathbf{P}(M(a_1, b_2))}{2}.$$

The justification for various steps and extension to arbitrary dimensions can be found in [Mil92].

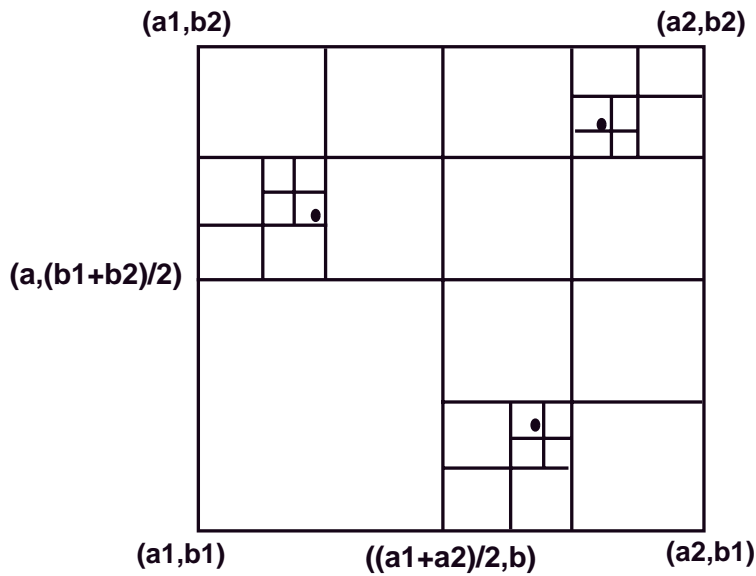


Figure 12: Linear convergence of roots

6.2 Converging to the critical points

In order to find all the collinear normal points between the two surfaces, we have to converge to each critical point within a given tolerance. Once that is done, subdividing the domain of the surface at these points ensures that there are no loops within each subdomain. We could then use any marching method to trace the intersection curves.

The algorithm to converge to each critical point within a tolerance is fairly simple. Given an initial domain, we compute the number of common roots within it. If it is zero, we stop. Otherwise, the domain is divided into four parts (by simple bisection), and the computation for number of solutions is performed again. It should be noted that the most expensive step of computing the Sturm sequence is performed only once. Substitution at the various endpoints of the interval is done at each step of the recursion. Once the interval size is within the tolerance, we stop and declare that as a root. It is easy to see that the convergence of this method is linear. Figure 12 shows the sequence of subdivisions for a particular case with three real roots.

7 Implementation and Demonstration on Examples

In this section, we illustrate the running of our algorithm on a few examples. The implementation of our algorithm was carried out in exact rational arithmetic using LiDIA (a rational number library). By using exact arithmetic, we can assure that the algorithm gives robust results, however, for the sake of efficiency it is better to implement the algorithm in finite precision.

Example 1: The first example is that of a biquadratic (degree 2×2) Bézier patch sectioned by a plane parallel to the xy -plane (see Fig. 10). The parametric form of the biquadratic patch is given below.

$$\begin{aligned} X(s, t) &= -2 + 4s + 2t - 4st - 2t^2 + 4s^2t \\ Y(s, t) &= -2 + 2s - 2s^2 + 4t - 4st + 4s^2t \\ Z(s, t) &= 3 - 2s + 2s^2 - 4t - 16st + 16s^2t + 4t^2 + 14st^2 - 14s^2t^2 \end{aligned}$$

Using the above patch equations and computing their partial derivatives, we obtain the following planar vector field.

$$\begin{aligned} f_1(s, t) &= -2 + 4s - 16t + 32st + 14t^2 - 28s^2t \\ f_2(s, t) &= -4 - 16s + 16s^2 + 8t + 28st - 28s^2t \end{aligned}$$

After adding the third polynomial, $f_3(s, t) = st + u - tx_1 - sx_2 + x_1x_2$ into the given system of equations and computing successive Sylvester resultants, we obtain the volume function for this bivariate case.

$$\begin{aligned} V(u, x_1, x_2) &= -104 - 3702u + 28244u^2 + 80362u^3 - 333592u^4 - 236670u^5 - 528x_1 + 184ux_1 - \\ & 30728u^2x_1 + 530012u^3x_1 + 667184u^4x_1 + 920x_1^2 + 36202ux_1^2 - 77280u^2x_1^2 - \\ & 530012u^3x_1^2 + 3680x_1^3 - 72772ux_1^3 + 51520u^2x_1^3 - 6440x_1^4 + 36386ux_1^4 + \\ & 2576x_1^5 - 1469x_2 - 21374ux_2 - 130525u^2x_2 + 358064u^3x_2 + 591675u^4x_2 - \\ & 7458x_1x_2 - 28704ux_1x_2 - 533968u^2x_1x_2 - 2668736u^3x_1x_2 - 1183350u^4x_1x_2 + \\ & 12995x_1^2x_2 + 131744ux_1^2x_2 + 2385054u^2x_1^2x_2 + 2668736u^3x_1^2x_2 + 51980x_1^3x_2 - \\ & 206080ux_1^3x_2 - 1590036u^2x_1^3x_2 - 90965x_1^4x_2 + 103040ux_1^4x_2 + 36386x_1^5x_2 - \\ & 2080x_2^2 + 84920ux_2^2 + 166152u^2x_2^2 - 338100u^3x_2^2 - 10560x_1x_2^2 - 218546ux_1x_2^2 + \\ & 1669248u^2x_1x_2^2 + 2366700u^3x_1x_2^2 + 18400x_1^2x_2^2 - 1371490ux_1^2x_2^2 - 6004656u^2x_1^2x_2^2 - \\ & 2366700u^3x_1^2x_2^2 + 73600x_1^3x_2^2 + 3180072ux_1^3x_2^2 + 4003104u^2x_1^3x_2^2 - 128800x_1^4x_2^2 - \end{aligned}$$

$$\begin{aligned}
& 1590036 u x^4 x^2 + 51520 x^5 x^2 + 21398 x^2^3 - 113134 u x^2^3 - 84525 u^2 x^2^3 + \\
& 108636 x^1 x^2^3 + 404432 u x^1 x^2^3 - 1014300 u^2 x^1 x^2^3 - 189290 x^1^2 x^2^3 + 2264304 u x^1^2 x^2^3 + \\
& 3550050 u^2 x^1^2 x^2^3 - 757160 x^1^3 x^2^3 - 5337472 u x^1^3 x^2^3 - 2366700 u^2 x^1^3 x^2^3 + 1325030 x^1^4 x^2^3 + \\
& 2668736 u x^1^4 x^2^3 - 530012 x^1^5 x^2^3 - 26936 x^2^4 + 48510 u x^2^4 - 136752 x^1 x^2^4 - \\
& 169050 u x^1 x^2^4 + 238280 x^1^2 x^2^4 - 1014300 u x^1^2 x^2^4 + 953120 x^1^3 x^2^4 + 2366700 u x^1^3 x^2^4 - \\
& 1667960 x^1^4 x^2^4 - 1183350 u x^1^4 x^2^4 + 667184 x^1^5 x^2^4 + 9555 x^2^5 + 48510 x^1 x^2^5 - \\
& 84525 x^1^2 x^2^5 - 338100 x^1^3 x^2^5 + 591675 x^1^4 x^2^5 - 236670 x^1^5 x^2^5
\end{aligned}$$

We computed the Sturm sequence of this volume function, and isolated the roots of the original equation in the interval $[0, 1] \times [0, 1]$ to within a precision of $\frac{1}{100}$. There was a single root as expected. The domain value of the collinear normal was

$$(s, t) = \left(\left[\frac{39038}{78125}, \frac{7894}{15625} \right], \left[\frac{8326}{15625}, \frac{42278}{78125} \right] \right)$$

If we subdivide the original patch at this point, we are guaranteed to have no loop in the resulting intersection.

Example 2: This is a slightly complicated example with multiple collinear normals (see Fig. 13). Here a bicubic (degree 3×3) parametric surface is cut by a plane parallel to the xy -plane. The coordinate equations of the surface are

$$\begin{aligned}
X(s, t) &= -2 + 3s + s^3 \\
Y(s, t) &= -2 + 3t + t^3 \\
Z(s, t) &= 2 + 3s - 6s^2 + 3s^3 - 72st + 189s^2t - 117s^3t + 171s^2t^2 - \\
&\quad 486s^2t^2 + 315s^3t^2 - 99st^3 + 294s^2t^3 - 195s^3t^3
\end{aligned}$$

Corresponding to these equations, the vector field is

$$\begin{aligned}
f_1(s, t) &= 1 - 4s + 3s^2 - 24t + 126st - 117s^2t + 57t^2 - 324s^2t^2 + 315s^2t^2 - 33t^3 + 196st^3 - 195s^2t^3 \\
f_2(s, t) &= -8s + 21s^2 - 13s^3 + 38st - 108s^2t + 70s^3t - 33st^2 + 98s^2t^2 - 65s^3t^2
\end{aligned}$$

The volume function for this vector field corresponds to a polynomial which is of degree 13 in u, x^1 and x^2 (the number of terms in this polynomial is too large to list it here). Computation the Sturm sequence and isolation within $\frac{1}{100}$ of precision yielded four roots. They are

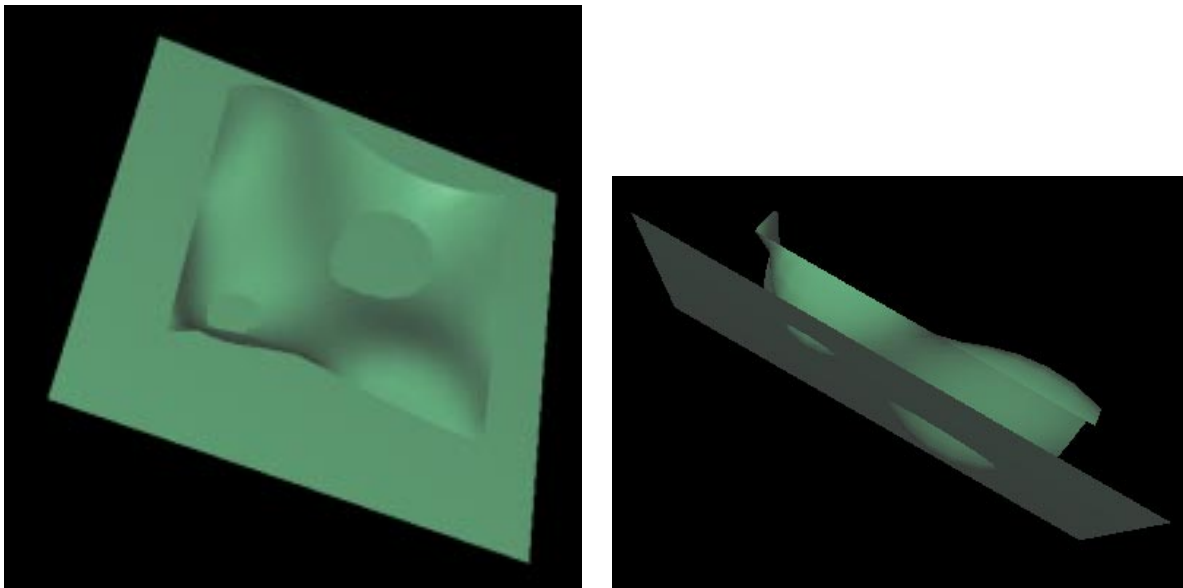


Figure 13: Planar section of a bicubic surface

$$(s, t) = \left(\left[\frac{2499}{15625}, \frac{528}{3125} \right], \left[\frac{71874}{78125}, \frac{14484}{15625} \right] \right), \left(\left[\frac{3187}{15625}, \frac{663}{3125} \right], \left[\frac{2453}{15625}, \frac{2547}{15625} \right] \right), \\ \left(\left[\frac{1962}{3125}, \frac{9861}{15625} \right], \left[\frac{55937}{78125}, \frac{11281}{15625} \right] \right), \left(\left[\frac{499}{625}, \frac{2518}{3125} \right], \left[\frac{2078}{15625}, \frac{2204}{15625} \right] \right)$$

Even though there are only two loops in the cross section, it is clear that there are other points where the normals of the two surfaces actually match.

8 Conclusion

We have presented two algorithms based on a combination of numeric and symbolic methods for loop detection of algebraic curves arising from surface interrogations. The first method 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. The second algorithm provides a robust method for detecting all the collinear normals of a surface when intersected with a plane. By subdividing the surface accordingly, all the components of the intersection curve can be found by solving for zero-dimensional algebraic sets. The algorithm has currently been

implemented in exact rational arithmetic and applied to fairly low degree surfaces. If efficiency and generality (application to higher degree surfaces) are the main emphasis of the application, the same algorithm can be implemented in double-precision arithmetic.

References

- [AB88] S.S. Abhyankar and C. Bajaj. Computations with algebraic curves. In *Lecture Notes in Computer Science*, volume 358, pages 279–284. Springer Verlag, 1988.
- [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.
- [Abh90] S. S. Abhyankar. *Algebraic Geometry for Scientists and Engineers*. American Mathematical Society, Providence, R. I., 1990.
- [ACM84] D. S. Arnon, G.E. Collins, and S. McCallum. Cylindrical algebraic decomposition. *SIAM J. on Computing*, 13:878–889, 1984.
- [AF88] S. Arnborg and H. Feng. Algebraic decomposition of regular curves. *Journal of Symbolic Computation*, 5:131–140, 1988.
- [AM88] D. Arnon and S. McCallum. A polynomial time algorithm for the topological type of a real algebraic curve. *Journal of Symbolic Computation*, 5:213–236, 1988.
- [Arn83] D. S. Arnon. Topologically reliable display of algebraic curves. *Computer Graphics*, 17:219–227, 1983.
- [AS86] W. Auzinger and H.J. Stetter. An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations. In *International Series of Numerical Mathematics*, volume 86, pages 11–30, 1986.
- [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.

- [Buc89] B. Buchberger. Applications of groebner bases in non-linear computational geometry. In D. Kapur and J. Mundy, editors, *Geometric Reasoning*, pages 415–447. MIT Press, 1989.
- [Can88] J.F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988.
- [CE93] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Proceedings of AAEECC*, pages 89–104. Springer-Verlag, 1993.
- [CH] G.W. Crippen and T.F. Havel. *Distance geometry and molecular conformation*. Research Studies Press, New York.
- [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.
- [Col75] G.E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lecture Notes in Computer Science*, number 33, Springer-Verlag, 1975.
- [Cra89] J.J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Publishing Company, 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 freeform 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.
- [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.
- [GZ79] C.B. Garcia and W.I. Zangwill. Finding all solutions to polynomial systems and other systems of equations. *Math. Prog.*, 16:159–176, 1979.
- [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.
- [Jou91] Jean-Pierre Jouanolou. *Le Formalisme du Résultant*, volume 90 of *Advances in Mathematics*. 1991.
- [Kim90] Deok-Soo Kim. *Cones on Bezier Curves and Surfaces*. PhD thesis, University of Michigan, Ann Arbor, 1990.
- [KM96] S. Krishnan and D. Manocha. Algebraic loop detection and evaluation algorithms for curve and surface interrogations. In *Proceedings of Graphics Interface*, pages 87–94, Toronto, Canada, 1996.
- [KM97] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on the lower dimensional formulation. *ACM Transactions on Graphics*, 16(1):74–106, 1997.
- [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.
- [Laz83] D. Lazard. Groebner bases, gaussian elimination and resolution of systems of algebraic equations. In *EUROCAL'83*. European Computer Algebra Conference, Springer-Verlag, 1983.

- [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.
- [Mac02] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, 1(33):3–27, May 1902.
- [Man92] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics*. PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.
- [Man94] 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.
- [MC27] F. Morley and A.B. Coble. New results in elimination. *American Journal of Mathematics*, 49:463–488, 1927.
- [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.
- [MC93] D. Manocha and J.F. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15(2):99–122, 1993.
- [Mil92] P. S. Milne. On the solutions of a set of polynomial equations. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102, 1992.
- [ML95] Y. Ma and R. C. Luo. Topological method for loop detection of surface intersection problems. *Computer-Aided Design*, 27(11):811–820, 1995.
- [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.
- [PK92] J. Ponce and D.J. Kriegman. Elimination theory and computer vision: Recognition and positioning of curved 3d objects from range, intensity, or contours. In *Symbolic and Numerical Computation for Artificial Intelligence*, pages 123–146, 1992.
- [RR92] A.A.G. Requicha and J.R. Rossignac. Solid modeling and beyond. *IEEE Computer Graphics and Applications*, pages 31–44, September 1992.
- [Sal85] G. Salmon. *Lessons Introductory to the Modern Higher Algebra*. G.E. Stechert & Co., New York, 1885.
- [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 Computational 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.
- [SN91] T.W. Sederberg and T. Nishita. Geometric hermite approximation of surface patch intersection curves. *Computer Aided Geometric Design*, 8:97–114, 1991.
- [SS83] J. T. Schwartz and M. Sharir. On the piano movers problem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351, 1983.
- [SZ94] B. Sturmfels and A. Zelevinsky. Multigraded resultants of sylvester type. *Journal of Algebra*, 1994. To appear.
- [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.

- [Wal50] R.J. Walker. *Algebraic Curves*. Princeton University Press, New Jersey, 1950.
- [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.