

Algebraic Loop Detection and Evaluation Algorithms 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

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.

Keywords: Surface Intersection, Silhouette Curve, Loop Detection, Gauss Map, Complex Projective Space

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 curves*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, Kim90, KPP90, KPW90, ZS93] and offset curves and surfaces [Hof90]. 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

*Supported in part by a Alfred P. Sloan Foundation Fellowship, ARO Contract P-34982-MA, 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

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



Figure 1: Two surfaces intersecting in a loop

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 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.

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 pose the problem as roots of two algebraic equations

in two unknowns [Buc85, Man94b, Mor92, Moo79]. However, the complexity of the resulting algebraic systems is quadratic in the degree of the curve, and are thus, not practical for high degree curves.

The subdivision based algorithms subdivide the domain up to a user-specified tolerance and evaluate the curves accordingly [Gei83]. 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 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]. In these class of methods, identifying a point on every loop is significantly harder than that on open components. Therefore, a 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. Since then, a number of techniques based on the same idea [THS89, SM88, KPP90, SM88, ZS93, Hoh91] have been developed to improve the efficiency of loop detection. 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 are bounded. Furthermore, these algorithms may not work well if the intersection curve is self-intersecting. While these methods can be extended to other surface interrogations (like silhouette computation), they tend to become inefficient and inaccurate.

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 al-

gorithm and its performance on various applications are highlighted in section 4, and we conclude in section 5.

Algebraic Formulation of the curve

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

$$\begin{aligned} F_1(u_1, u_2, \dots, u_n) &= 0 \\ F_2(u_1, u_2, \dots, u_n) &= 0 \\ &\vdots \\ F_{n-1}(u_1, u_2, \dots, u_n) &= 0. \end{aligned}$$

Moreover, we are only interested in evaluating all the components of the curve inside the region $D = [U_{11}, U_{12}] \times [U_{21}, U_{22}] \times \dots \times [U_{n1}, U_{n2}] \in \mathcal{R}^n$. Formally, the functions F_i , $i = 1, 2, \dots, n - 1$, are the components of a vector function $F : D \rightarrow \mathcal{R}^{n-1}$, $D \subset \mathcal{R}^n$. 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:

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

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.

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. Essentially, our evaluation algorithm is now restricted to the plane curve. Another advantage of this approach is that applications (like solid models), 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.

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 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

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

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. 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

$$\begin{aligned} x W(s, t) - X(s, t) &= 0, \\ y W(s, t) - Y(s, t) &= 0, \\ z W(s, t) - Z(s, t) &= 0 \end{aligned} \quad (2)$$

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 $\bar{\mathbf{M}}(x, y, z, w)$. 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 $\bar{\mathbf{M}}(x, y, z, w)$ 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

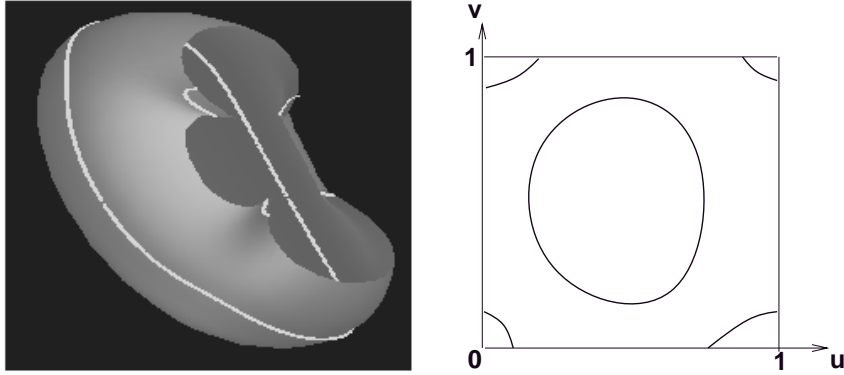


Figure 2: Loop as part of a silhouette curve

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

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

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.

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

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

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

$$N_z = \begin{vmatrix} \phi_{1_u} & \phi_{1_v} \\ \phi_{2_u} & \phi_{2_v} \end{vmatrix} \quad (3)$$

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

$$N_z = \begin{vmatrix} (WX_u - W_uX) & (WX_v - W_vX) \\ (WY_u - W_uY) & (WY_v - W_vY) \end{vmatrix} = 0 \quad (4)$$

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

$$\mathbf{M}(u, v) = \begin{pmatrix} X(u, v) & Y(u, v) & W(u, v) \\ X_u(u, v) & Y_u(u, v) & W_u(u, v) \\ X_v(u, v) & Y_v(u, v) & W_v(u, v) \end{pmatrix} = 0 \quad (5)$$

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

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

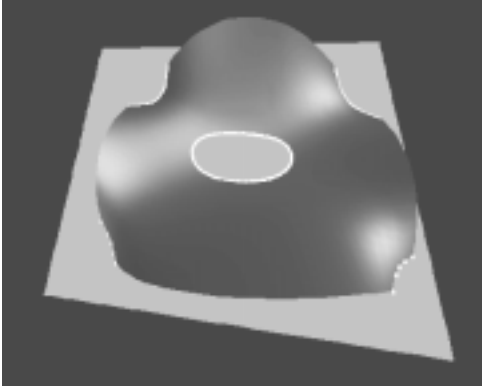


Figure 3: A pair of intersecting surfaces

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 (right) u -turning point* if the curve goes into the complex domain in the left (right) neighborhood of u_1 .

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. A detailed proof can be found in [KM95]. \square

The domain of the intersection curve in the complex space is shown in fig.4 (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

$$\mathbf{M}(v) = v^d M_d + v^{d-1} M_{d-1} + \dots + v M_1 + M_0 = 0. \quad (6)$$

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 $2mnd$.

$$C = \begin{bmatrix} 0 & I_n & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & I_n \\ -\overline{\mathbf{M}}_0 & -\overline{\mathbf{M}}_1 & -\overline{\mathbf{M}}_2 & \dots & -\overline{\mathbf{M}}_{d-1} \end{bmatrix} \quad (7)$$

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 C from $\mathbf{M}(u_2, v)$ (see eq. (7)). This reduces the problem to finding the eigenvalue of C closest to v_2 (or

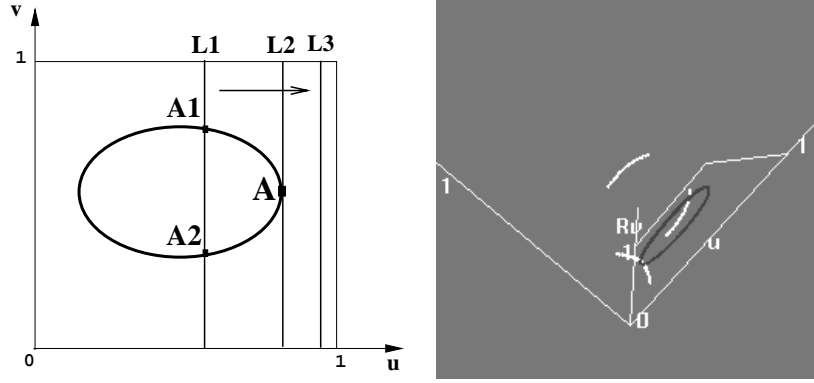


Figure 4: Characterization of loops based on complex tracing

smallest eigenvalue of $C - v_2\mathbf{I}$. The smallest eigenvalue of $C - v_2\mathbf{I}$ corresponds to the largest eigenvalue of $(C - v_2\mathbf{I})^{-1}$. 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

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

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

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

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 (6)). The LU decomposition of \mathbf{B} has the form:

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

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.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

$$\det(\mathbf{M}_{\mathbf{r}}(u, v_r) + i\mathbf{M}_{\mathbf{i}}(u, v_r)) = 0 \quad (8)$$

It is easy to show that the solutions (u, v_r) satisfying equation (8) also satisfy the solution of $\det \mathbf{P}(u, v_r) = 0$, where

$$\mathbf{P}(u, v_r) = \begin{bmatrix} \mathbf{M}_{\mathbf{r}}(u, v_r) & -\mathbf{M}_{\mathbf{i}}(u, v_r) \\ \mathbf{M}_{\mathbf{i}}(u, v_r) & \mathbf{M}_{\mathbf{r}}(u, v_r) \end{bmatrix} \quad (9)$$

As before, the solutions to (9) 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.(7). 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.

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 the efficiency of linear system ($Ax = b$) 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.

Application to surface intersection: 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.

Hybrid approach: A comparison of our method with Gauss map based approaches suggests that the latter performs better when the patches are relatively flat and do not intersect in loops. However, our method is faster when the patches have high curvature and intersect in small loops or singularities. We, therefore, 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.

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. 2(a)), 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. 2(b).

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.
- [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.
- [Kim90] Deok-Soo Kim. *Cones on Bezier Curves and Surfaces*. PhD thesis, University of Michigan, Ann Arbor, 1990.
- [KM95] S. Krishnan and D. Manocha. Algebraic loop detection and evaluation algorithms for curve and surface interrogations. Technical Report TR95-038, Department of Computer Science, University of North Carolina, 1995.
- [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.
- [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.
- [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.
- [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.
- [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.
- [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.
- [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.