A New Approach for Surface Intersection

Dinesh Manocha and **John Canny**
Computer Science Division
University of California
Berkeley, CA 94720

Abstract

Evaluating the intersection of two rational parametric surfaces is a recurring operation in solid modeling. However, surface intersection is not an easy problem and continues to be an active topic of research. The main reason lies in the fact that any good surface intersection technique has to balance three conflicting goals of accuracy, robustness and efficiency. In this paper, we formulate the problems of curve and surface intersections using algebraic sets in a higher dimensional space. Using results from Elimination theory, we project the algebraic set to a lower dimensional space. The projected set can be expressed as a matrix determinant. The matrix itself, rather than its symbolic determinant, is used as the representation for the algebraic set in the lower dimensional space. This is a much more compact and efficient representation. Given such a representation, we utilize properties of straight line programs and results from linear algebra for performing geometric operations on the intersection curve. Most of the operations involve evaluating numeric determinants and computing the rank, kernel and eigenvalues of matrices. The accuracy of such operations can be improved by pivoting or other numerical techniques. We use this representation for inversion operation, computing the intersection of curves and surfaces and tracing the intersection curve of two surfaces in lower dimension.

1 Introduction

Many current geometric and solid modeling systems use parametric curves and surfaces for designing geometric objects to satisfy various interpolatory, smoothness and aesthetic requirements. When it comes to dealing with intersections of these curves and surfaces, current algorithms fail to meet the minimum standards of robustness, accuracy and efficiency that these applications demand. Most of the difficulties arise due to the fact that any exact algorithm for these problems tends to be slow due to the large coefficient size of the exact result. The approximate algorithms are far from being robust and their accuracy varies with the surface degree, with the local surface geometry at the intersection curve, and the angle at which the surface intersect. Moreover, the numerical stability of these calculations is not completely understood, but appears to be poor for at least some of the necessary calculations. As a result, it is widely believed that any good surface intersection technique has to balance three conflicting goals of accuracy, robustness and efficiency [Ho89; PG86].

Earlier approaches to surface intersection used recursive subdivision techniques based on the paradigm of 'divide and conquer' [LR80; PG86]. The main idea is to subdivide the surfaces into small pieces until each piece satisfies some flatness criterion. However the main problem with the approach lies in constructing the topology of the resulting curve from the individual intersection pieces. The algorithm is not robust and may fail in the presence of simple singularities.

Another approach for evaluating surface intersections is that of *tracing the intersection curve* [BHLH88; BK90; PG86]. The main idea behind tracing techniques is to compute a starting point on each component and locate all the singular points. Given the starting points, these algorithms use marching methods to trace the intersection curve and in the process use robust methods to determine all the branches at singular points.

It is possible to represent the intersection curve as an algebraic set in the higher dimensional space spanned by the parameters of two surfaces. Given such a formulation, techniques like implicitization can be used to obtain a closed form and exact representation as an algebraic plane curve of the form $f(u,v) = 0$, where $f(u,v)$ is a bivariate polynomial [Fa86]. Such a representation is obtained by implicitizing one of the parametric surfaces and substituting the other parametrization into the implicit representation. When it comes to tracing, we may choose to trace the curve in the higher dimension or its projection in the lower dimension [Ho88]. However, [Ho88; Ho90] counsels against computing the projection in the lower dimensional space for the following reasons:

- Implicitizing a parametric surface entails substantial symbolic computation for degrees higher than cubic. Generally resultants are used for implicitization and it is believed that the use of resultants may introduce extraneous factors, which pose additional problems.

- Substitution of the parametric formulation into the implicit form, although conceptually simple, is numerically delicate and can lead to substantial errors. This is mainly due to catastrophic cancellation [PP88].

- By Bezout's theorem, the degree of the intersection curve is equal to the product of the degree of the intersecting surfaces. Thus, with the resulting high algebraic degrees numerical difficulties arise even when evaluating the resulting curve at some point.

As far as the problem of computing a point on each component is concerned, a recent approach of *loop detection* is presented in [SM88]. The basic idea is to subdivide the parametric surfaces, such that the intersection of subdivided surfaces (considered piecewise) has no closed loops and all starting points can be obtained by the intersection of boundary curves of one surface patch with the other surface. The algorithm in [SM88] does not account for singular points and has been found slow in practice. Thus, no good algorithms are available for finding a point on each component or the singular points on the intersection curve.

We use results from Elimination theory to represent the implicit equation of a rational parametric surface as a matrix determinant [MC90a]. The main idea involves using the matrix itself, rather than its symbolic determinant, for representing the implicit representation and the projection of intersection curve. To evaluate such a representation we use numeric substitution and Gauss elimination. The resulting algorithm is efficient and its numeric accuracy is improved by techniques like pivoting. Furthermore, we use properties of straight line programs to evaluate the partial derivatives of the function used for representing the intersection curve. As a result, we are able to reduce the problems of finding a starting point on each component and the singular points to curve-surface intersection and computing solutions of nonlinear equations.

We use our representation in coming up with efficient and robust algorithms for curve-surface intersection and computing the inverse image of a point on the parametric surface. The techniques involved are computing the eigenvalues and eigenvectors of a matrix, singular value decomposition and determinant computation. The numerical accuracy of such operations is well understood and efficient implementations are available as part of standard packages like LINPACK and EISPACK [BDMS79; GBDM77]. Moreover, we use this representation for tracing the intersection curve in lower dimension.

The rest of the paper is organized in the following manner. In Section 2 we present some background material on surface intersection, implicitizing parametric surfaces and how Elimination theory can be used for representing the implicit equation of a parametric surface as a matrix determinant. This formulation is used for representing the intersection curve and efficient and numerically stable algorithms are presented for evaluating the function and its partial derivatives in Section 3. In Section 4 we present two main applications of our representation: computing the inverse image of a point and curve-surface intersection. We also present results of our implementation of these applications. Finally in Section 5, we address the problem of tracing the intersection curve and reduce the problem of finding a point on each component and the singular points to equation solving. We consider the curve represented in higher dimensional space as well as its projection in the lower dimensional space. Various approaches for equation solving are compared and some open issues are presented.

2 Background

The problem of surface intersection involves computing an appropriate representation of the intersection curve and designing suitable algorithms for evaluation and performing geometric operations. The exact requirements on the representation and algorithms are application dependent. In general, any representation should provide functionality for the following operations:

- Evaluate and render the intersection curve (all components and branches).
- Decide whether a point (X, Y, Z) lies on the intersection curve.
- Sort the points lying on the intersection curve.
- Use it as a boundary edge for a trimmed surface.

Many a times the intersection curve consists of a single component without any singularities. However, even simple cases like intersection of two cylinders can give rise to singularity. In this case the intersection curve is an algebraic space curve of degree four. For tensor product bicubic Bézier patches the intersection curve is a space curve of degree 324 and it is simple to come up with cases where the intersection curve has more than one component (as shown in Fig. I) and thereby adding to the complexity of the intersection problem.

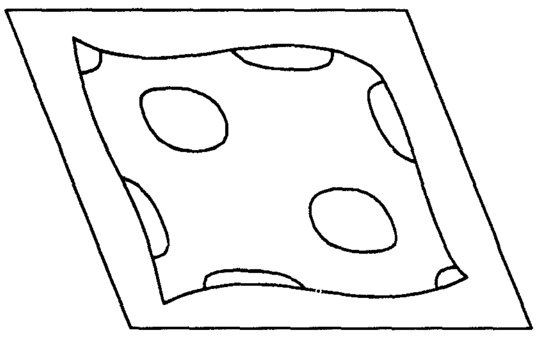

Fig. I
Intersection curve consisting of eight components

2.1 Implicitization

It is well known in algebraic geometry that if one of the surfaces is represented parametrically and the other one implicitly, an implicit representation of the intersection curve can be obtained by substituting the parametric formulation into the implicit representation. A Bèzier surface is a rational parametric surface. The fact that the set of rational parametric surfaces is a proper subset of the algebraic surfaces implies that every Bézier surface can be represented as an algebraic surface of the form $f(x, y, z, w) = 0$, where $f(x, y, z, w)$ is an irreducible homogeneous polynomial [SR85]. In other words, given two Bézier surfaces

$$\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t)) \tag{1}$$

and

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

Implicitize $\mathbf{F}(s,t)$ into an algebraic surface of the form $f(x, y, z, w) = 0$. Substitute the other parametrization to obtain an exact representation of the intersection curve as:

$$f(\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)) = 0,$$
$$0 \leq u \leq 1, \quad 0 \leq v \leq 1.$$

The problem of implicitization corresponds to formulating parametric equations like

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

and eliminating the variables s and t. Techniques for eliminating variables have been well known in classical algebraic geometry for more than a hundred years [Sa1885; Wd50]. As far as geometric and solid modeling are concerned, these techniques were resurrected by [SAG84]. In particular, [SAG84] used resultants to implicitize tensor product surfaces into their corresponding implicit representation. Some recent approaches to the problem of implicitization include the use of Gröbner bases [Bu85; Ho89]. However, most of the results were negative and in general, it is believed that any algorithm based on the implicitization approach can be inefficient and numerically unstable for surface intersection for reasons highlighted in the previous section and given in detail in [Ho88; Ho90].

The problem of implicitization has recently been analyzed in [Ch90; MC90a]. In particular, it has been shown that if a parametrization has no base points, the resultant of the parametric equations corresponds exactly to the implicit representation. The base points of a parametrization are defined as the common roots[1] of

$$X(s,t) = 0; \quad Y(s,t) = 0; \quad Z(s,t) = 0; \quad W(s,t) = 0.$$

For a random choice of coefficients a parametrization has no base points. The three parametric equations are of equal degree and [MC90a] use this property along with results from Elimination theory [Di08] to

[1] They also include the roots at infinity

show that the implicit equation can be represented as a matrix determinant. For a tensor product surface of the form $(s^m t^n)$ the order of the matrix is $2mn$ and for a triangular patch of degree n the matrix has order $2n^2 - n$ [MC90a]. Each entry of the matrix is a linear polynomial of the form

$$a_{ij}x + b_{ij}y + c_{ij}z + d_{ij}w,$$

where a_{ij}, b_{ij}, c_{ij} and d_{ij} are rational functions of the given parametrization. If a parametrization has simple base points, it is still possible to represent the implicit representation as a matrix determinant. An efficient algorithm based on Vandermonde interpolation has been presented in [MC90a] for computing the resultant of polynomial equations and thereby used for implicitization. In particular, it has been shown that it is possible to implicitize parametrizations like tensor product bicubic surfaces in less than two minutes on machines like the IBM RS/6000. The implementation has been restricted to exact arithmetic. Although this algorithm is fast for practical applications, it becomes unattractive due to issues of numeric stability in the context of floating point computations and the degree of the resulting polynomials obtained after substitution.

3 Representation of Intersection Curve

We make use of results of [Di08] and [MC90a] to represent the implicit equation as an unexpanded determinant and present algorithms to perform geometric operations on the representation. Given two Bézier surfaces, $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$ as in (1), we use the corresponding formulation of resultant (depending on the fact whether $\mathbf{F}(s,t)$ is a tensor product surface or a triangular patch) and represent the implicit equation as a matrix determinant. The main idea is to use the matrix itself, rather than its symbolic determinant, as a representation for the implicit form and subsequently for the intersection curve. Later on we will show that this is a much more compact, efficient and numerically stable representation.

Given $f(x,y,z,w)$, as a matrix determinant

$$f(x,y,z,w) =$$

$$\begin{vmatrix} f_{11}(x,y,z,w) & \dots & f_{1n}(x,y,z,w) \\ f_{21}(x,y,z,w) & \dots & f_{2n}(x,y,z,w) \\ \vdots & \vdots & \vdots \\ f_{n1}(x,y,z,w) & \dots & f_{nn}(x,y,z,w) \end{vmatrix}, \qquad (2)$$

where $f_{ij}(x,y,z,w)$ is a linear polynomial of the form $a_{ij}x + b_{ij}y + c_{ij}z + d_{ij}w$. The coefficients a_{ij}, b_{ij}, c_{ij} and d_{ij} can be represented as rational functions of the given parametrization. Furthermore, their computation is efficient and numerically stable.

The implicit representation of intersection curve is obtained by substituting the parametrization $\mathbf{G}(u,v)$ into $f(x,y,z,w)$. As a result, the intersection curve is represented as zero set of a determinant. The corresponding matrix is

$$M(u,v) = f(\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)) \quad (3)$$

$$= \begin{bmatrix} g_{11}(u,v) & \dots & g_{1n}(u,v) \\ g_{21}(u,v) & \dots & g_{2n}(u,v) \\ \vdots & \vdots & \vdots \\ g_{n1}(u,v) & \dots & g_{nn}(u,v)) \end{bmatrix},$$

$$u_1 \le u \le u_2, \quad v_1 \le v \le v_2,$$

where

$$g_{ij}(u,v) = f_{ij}(\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)).$$

In this case we substitute the parametrization into a linear polynomial of the form $f_{ij}(x,y,z,w)$. In practice, we represent each entry of $M(u,v)$ as 4-tuple of $(a_{ij}, b_{ij}, c_{ij}, d_{ij})$. As a result, there is no significant loss of information due to catastrophic cancellation, which is the case when the implicit representation corresponds to a polynomial of degree n [PP88]. The representation of the curve corresponding to Fig. I has been shown in Fig. II. In this case, the planar curve (which is birationally equivalent to the space curve) has eight components. In particular, the curve has two kind of components. Closed loops are shown as $C1$ and $C2$ and the open components (the ones that intersect with the boundary) are $O1, \dots, O6$.

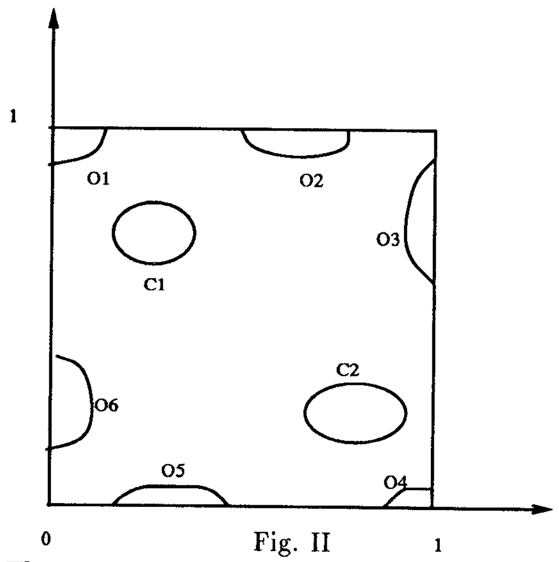

Fig. II
The representation of intersection curve in Fig. I

3.1 Matrix Operations on the Representation

We make use of the representation presented in the previous section to perform geometric operations. We use the notations $M(u,v)$ to denote the matrix used for representing the intersection curve and $D(u,v)$ to represent the polynomial corresponding to *Determinant*$(M(u,v))$. For geometric operations like tracing or marching through the intersection curve, we need to effectively evaluate expressions like $D(u_1,v_1)$, $D^u(u_1,v_1)$, $D^{uu}(u_1,v_1)$, where u_1 and v_1 are real numbers and D^u and D^{uu} represent the first and second partial derivatives with respect to u.

$D(u_1,v_1)$ can be efficiently and accurately evaluated in the following manner:

- Compute the entries of $M(u_1,v_1)$ by evaluating Bernstein polynomials $\overline{X}(u_1,v_1)$, $\overline{Y}(u_1,v_1)$, $\overline{Z}(u_1,v_1)$ and $\overline{W}(u_1,v_1)$ and taking their respective combinations. Techniques for efficient and accurate evaluation of Bernstein polynomials are well known [Fa86].
- Given $M(u_1,v_1)$, a matrix with numerical entries, use Gauss elimination to compute $D(u_1,v_1)$. Furthermore, use pivoting techniques to improve the numeric stability of the resulting computation [GV89; Wi63]. It is possible to compute the *condition number* of the matrix and come up with a tight bound on the numerical accuracy of the result. Furthermore, efficient and well tested software for such operations is available as part of LINPACK [BDMS79].

To compute the first and higher order partials, we make use of the fact that a determinant can be represented as a straight line program. In this case, we modify the matrix structure such that entry consists of a tuple $\mathbf{G}_{ij}(u_1,v_1) = (g_{ij}(u_1,v_1), g^u_{ij}(u_1,v_1))$, where $g^u_{ij}(u_1,v_1)$ represents the partial derivative of $g_{ij}(u,v)$ with respect to u and specializing $u = u_1$ and $v = v_1$. The resulting matrix structure is of the form

$$\overline{M}(u_1,v_1) =$$

$$\begin{bmatrix} \mathbf{G}_{11}(u_1,v_1) & \dots & \mathbf{G}_{1n}(u_1,v_1) \\ \vdots & \dots & \vdots \\ \mathbf{G}_{n1}(u_1,v_1) & \dots & \mathbf{G}_{nn}(u_1,v_1) \end{bmatrix}.$$

To compute $D(u_1,v_1)$ and $D^u(u_1,v_1)$ we perform Gauss elimination. We consider the matrix formed by first entry of each tuple (equivalent to $M(u_1,v_1)$) and proceed as if we are trying to compute its determinant. As a side effect we change the entry in the second tuple. Assume we are operating on the ith and kth rows of the matrix. A typical step of Gauss elimination is of the form

$$g_{kj} = g_{kj} - \frac{g_{ki}}{g_{ii}} g_{ij},$$

where g_{kj} represents the element in the kth row and jth column of the matrix. In the new formulation this step is replaced as

$$g_{kj} = g_{kj} - \frac{g_{ki}}{g_{ii}} g_{ij}$$

$$g^u_{kj} = g^u_{kj} - \frac{(g^u_{ki}g_{ij} + g_{ki}g^u_{ij})g_{ii} - (g_{ki}g_{ij})g^u_{ii}}{(g_{ii})^2}). \qquad (4)$$

We make a choice for the pivot element based on the first tuple (i.e. g_{ij} entry). After Gauss elimination is complete, we compute $D(u_1,v_1)$ and $D^u(u_1,v_1)$ in the following manner:

$$D(u_1,v_1) = \prod_{i=1}^{n} g_{ii}$$

$$D^u(u_1,v_1) = D(u_1,v_1) \sum_{i=1}^{n} \frac{g^u_{ii}}{g_{ii}}.$$

This procedure can be easily extended to compute the higher order partials. Furthermore, the analysis of Gauss elimination may be used for analyzing the numerical accuracy of partial derivatives computation. To insure the numeric stability of Gauss elimination, it is required that the intermediate coefficients being generated do not grow in magnitude. Since g_{ii} occurs in the denominator term in (4), the pivoting process chooses g_{ii} to be the element of maximum magnitude (with respect to the ith column or the rest of the matrix). The computation of g^u_{kj} involves division by $(g_{ii})^2$. At the moment we have partial results from this analysis and our implementation indicating that this method should be numerical stable.

4 Applications of the Representation

In this section we highlight two main applications of our representation. They are computing the inverse coordinates for a point (x_1, y_1, z_1) lying on the surface and the intersection of curves and surfaces. Both these operations are used in the algorithm used for tracing intersection curves in the next section.

4.1 Inversion Operation

Frequently we are given a point (x_1, y_1, z_1) and asked to determine if this point lies on the intersection curve of $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$. Previous approaches to this problem take the surface parametrization and formulate the equations

$$x_1 W(s,t) - X(s,t) = 0$$

$$y_1 W(s,t) - Y(s,t) = 0$$

$$z_1 W(s,t) - Z(s,t) = 0$$

and determine whether these equations have a common solution in the range $0 \leq s \leq 1$ and $0 \leq t \leq 1$ (assuming that we are considering the intersection of Bézier surfaces). The same procedure is repeated for the other surface. However, root finding can be slow in practice and in this section, we show how to effectively use our representation for solving the inversion problem.

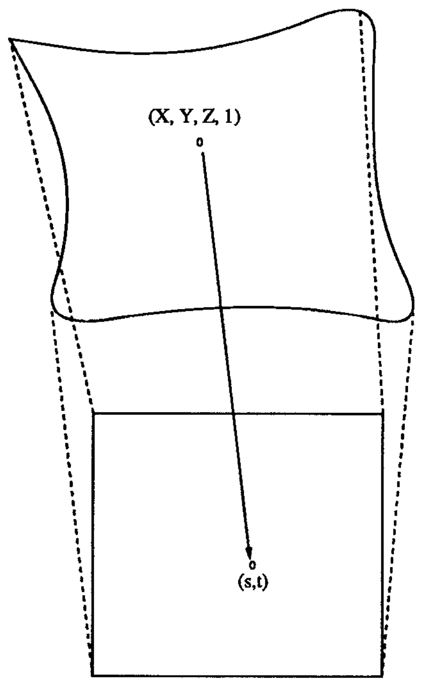

Fig. III
Inversion operation

The implicit representation of the surface is the determinant of $f(x,y,z,w)$, as formulated in (2). The point (x_1, y_1, z_1) lies on the surface if and only if the determinant of $(f(x_1, y_1, z_1, 1))$ obtained after substitution is zero. While using floating point arithmetic, it is also possible to obtain tight bound on the errors of computation [Wi63].

We used results from Elimination theory to represent the implicit equation as a matrix determinant [MC90a]. Assume that the given surfaces are tensor product surfaces of the form $(s^d t^d)$. Techniques from Elimination theory take the following parametric equations:

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

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

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

and reduce it to a problem in linear algebra of the form

$$MX = \mathbf{0},$$

where

$$M = \begin{bmatrix} f_{11}(x,y,z,w) & \dots & f_{1n}(x,y,z,w) \\ \vdots & \vdots & \vdots \\ f_{n1}(x,y,z,w) & \dots & f_{nn}(x,y,z,w) \end{bmatrix}$$

$$X = \begin{bmatrix} 1 \\ s \\ t \\ \vdots \\ s^{2d-1}t^{d-1} \end{bmatrix}, \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

and M is equivalent to $f(x,y,z,w)$ in (2) and $n = 2d^2$.

The fact that the determinant of $f(x_1, y_1, z_1, w_1)$ is zero (or very close to zero) implies that the numeric matrix is singular (or nearly singular). Let us assume that the given parametrization is faithful and (x_1, y_1, z_1, w_1) does not correspond to a singular point. Consequently it has a unique preimage and $f(x_1, y_1, z_1, w_1)$ is a matrix whose kernel has dimension one. We can use a technique like SVD (singular value decomposition) to accurately determine the vector in that kernel [GV89; Wi63]. Let that vector be $\mathbf{V} = (v_1 v_2 \dots v_{2d^2})^T$. As a result, the preimage of the point (x_1, y_1, z_1, w_1) can be obtained by solving the equation

$$k \begin{bmatrix} 1 \\ s \\ t \\ s^2 \\ \vdots \\ s^{2d-1}t^{d-1} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{2d^2} \end{bmatrix},$$

where k is a scalar. Good implementations of SVD are available as part of LINPACK [BDMS79]. Since we are only interested in verifying whether zero is an eigenvalue of the matrix and compute the corresponding eigenvector, we may use routines for condition estimator (SGECO from LINPACK) [BDMS79].

It is more efficient than computing the SVD of the matrix.

A similar procedure can be applied to the other parametrization. Thus, the problem of inversion has been reduced to the computation of determinants and the kernel of matrices.

4.2 Intersecting Curves and Surfaces

Given a Bézier curve

$$\mathbf{G}(u) = (X(u), Y(u), Z(u), W(u)) \quad 0 \le u \le 1$$

of degree d and a Bézier surface $\mathbf{F}(s,t)$. Let $f(x,y,z,w)$ be the matrix corresponding to its implicit representation. The problem of intersecting $\mathbf{G}(u)$ and $\mathbf{F}(s,t)$ can be reduced to solving three non-linear equations in three unknowns. However, our aim is to find all the roots in the domain of interest (as shown in Fig. IV). Algebraic techniques like resultants and Gröbner bases are too slow for practical usage and in the context of floating point computation, the accuracy of results using these methods is not completely understood. Newton's method is fast and numerically stable, however it does not guarantee all the roots in the given domain.

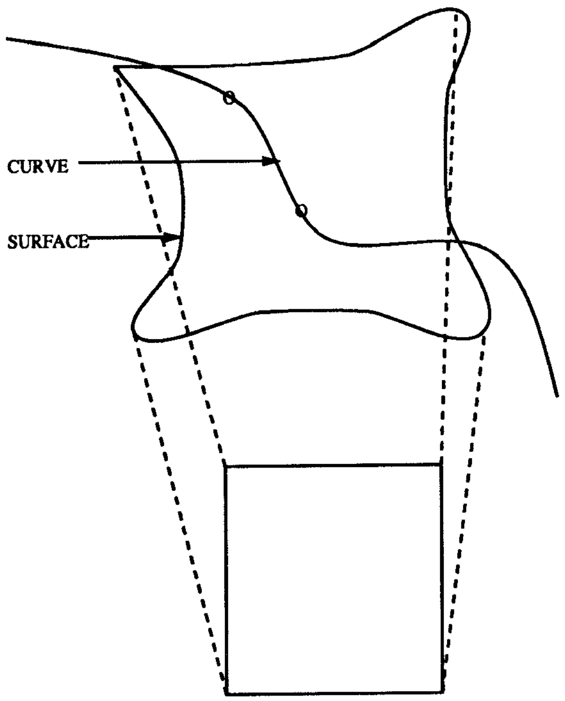

Fig. IV
Intersecting curves and surfaces

In this section we use our formulation of the implicit representation and reduce the problem of curve-surface intersection to an eigenvalue problem. Efficient algorithms for solving eigenvalue problems are well known in numerical analysis [GLR82; GV89]. Furthermore, good implementations of these algorithms are available as part of a package like EISPACK [GBDM77].

Given $f(x,y,z,w)$, we substitute the parametric equation of the curve to obtain a matrix of the form

$$M(u) = \begin{bmatrix} g_{11}(u) & \dots & g_{1n}(u) \\ \vdots & \dots & \vdots \\ g_{n1}(u) & \dots & g_{nn}(u) \end{bmatrix},$$

where

$$g_{ij}(u) = f_{ij}(X(u), Y(u), Z(u), W(u)).$$

The determinant of $M(u)$ is a univariate polynomial in u, and its roots correspond to the preimages of intersection points between the curve and the surface. Each entry of $M(u)$ is a univariate polynomial of degree d and let us represent it as matrix polynomial

$$M(u) = u^d M_d + u^{d-1} M_{d-1} + \ldots + u M_1 + M_0,$$

where M_i's are matrices of order n with numeric entries. Let us assume that M_d is a non-singular matrix. As a result the roots of the following equations are equivalent

$$\text{Det}(M(u)) = 0,$$

$$\text{Det}(M_d^{-1})\ \text{Det}(M(u)) = 0.$$

Let

$$\overline{M}(u) = u^d I_n + u^{d-1}\overline{M}_{d-1} + \ldots + u\overline{M}_1 + \overline{M}_0,$$

where

$$\overline{M}_i = M_d^{-1} M_i, \quad 0 \le i < d$$

and I_n is an $n \times n$ identity matrix. Given $\overline{M}(u)$, we use Theorem 1.1 [GLR82] to construct a matrix of the form

$$C = \begin{bmatrix} 0 & I_n & 0 & \dots & 0 \\ 0 & 0 & I_n & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & I_n \\ -\overline{M}_0 & -\overline{M}_1 & -\overline{M}_2 & \dots & -\overline{M}_{d-1} \end{bmatrix}, \quad (5)$$

such that the eigenvalues of C correspond exactly to the roots of $\text{Det}(\overline{M}(u)) = 0$. C is a numeric matrix of order dn. If M_d is a singular matrix, techniques to compute the roots of $\text{Det}(M(u)) = 0$ are given in [GLR82].

Most of the currently known algorithms find all the eigenvalues of the given matrix. In our applications, we are only interested in the eigenvalues lying in the domain, say $[u_1, u_2]$. For example, when we are dealing with Bézier curves and surfaces, the domain is $[0, 1]$.

4.2.1 Implementation

We used EISPACK routines for computing the eigenvalues of matrices. Many special purpose algorithms are available for computing the eigenvalues of matrices, which make use of the structure of the matrix. As far as matrix C in (5) is concerned, we treat it as a general unsymmetric matrix. We use the routine RG from EISPACK for computing the eigenvalues [GBDM77]. Given a general unsymmetric matrix, it makes use of balancing techniques, reduces it to upper Hessenberg form and uses the shifted QR algorithm on the resulting matrix to compute the eigenvalues [GV89]. The current implementation of these routines compute all the eigenvalues. The performance of eigenvalue computation routines for matrices of different order are given in Table I. The timings correspond to the implementation on an IBM RS/6000.

Order of Matrix	Time in seconds
15	$8631.839844 \times 10^{-6}$
20	$15717.63965 \times 10^{-6}$
25	$25753.00000 \times 10^{-6}$
30	$38763.23828 \times 10^{-6}$
35	$57124.16016 \times 10^{-6}$
40	$77398.03906 \times 10^{-6}$
45	$103343.5234 \times 10^{-6}$
50	$133956.2344 \times 10^{-6}$
55	$165395.0469 \times 10^{-6}$
60	$212041.2812 \times 10^{-6}$
65	$262103.1250 \times 10^{-6}$

Table I
The performance of eigenvalue computation routines

While considering the intersection of a cubic Bézier curve with a bicubic patch, the order of C in (5) is 54. However, the performance of the implementation presented above can be improved by making use of the structure of C to reduce it to an upper Hessenberg form. Typically, the intersection of a Bézier curve with a bicubic patch result in two or three intersections in the domain of interest and we are only in interested in those eigenvalues. The current implementation computes all the 54 eigenvalues and many of them are complex numbers, too. However, this algorithm guarantees all the intersection points and is therefore, robust.

This technique is also directly applicable for ray tracing parametric surfaces. Every ray is a parametric curve of degree one and as a result each entry of $M(u)$ is a linear polynomial. It is simple to find the intersections of the ray with the control polyhedra (of the parametric surface). Those bounds can be used to define the domain of the variable u. For ray tracing bicubic patches, the order of C is 18.

5 Surface Intersections

In the previous sections, we presented a representation of the projection of the intersection curve, as a matrix determinant, $M(u, v)$ in (3). The determinant is denoted as $D(u, v)$. In this section we consider the problem of intersecting two Bézier surfaces. To evaluate the intersection curve, we use the marching technique. Therefore, we need to determine a point on each component of the intersection curve and all the singular points. We consider the problem of tracing the intersection curve in lower as well as higher dimensional space.

Tracing in lower dimensions correspond to tracing an algebraic plane curve. The main advantage of this approach lies its abilities to deal with singularities. Simple singularities like cusps and loops can be easily characterized. Moreover, there is an elegant theory of *resolution of singularities* in algebraic geometry, which can be used for dealing with complicated singularities [Wa50]. An analogous process for surface intersection in higher dimensional space could be devised in principle, but it would be substantially more complex because it would map the intersecting surfaces simultaneously such that the singularity of the intersection would be resolved [Ho89].

It is widely believed that the lower dimensional approach has to cope up with a number of practical difficulties. Some of them were highlighted in Section 1 though a detailed analysis has been presented in [Ho88; Ho90]. However, we feel that our representation in terms of an unexpanded determinant and subsequent operations using techniques from linear algebra and matrix computations take care of the practical problems highlighted in [Ho88; Ho90]. As a result, it is worthwhile to reconsider the projected curve in the lower dimensional space as a practical method for evaluating surface intersections. In this section, we reduce the problem of finding a point on each component and singular points to curve-surface intersection and solving systems of non-linear equations. Therefore, we do not need techniques like *loop detection* for finding a point on each component.

5.1 Curve in Lower Dimension

In this case the curve has been represented as a matrix determinant, $D(u,v)$. The singular points of the projected curve correspond to the roots of the equations [Wa50]

$$\begin{aligned} D(u,v) &= 0 \\ D^u(u,v) &= 0 \\ D^v(u,v) &= 0 \end{aligned} \tag{6}$$

and as a result, the problem of computing singular points has been reduced to equation solving.

The intersection curve has two kinds of components:

Closed loops :- They are of the form $C1$ and $C2$ in Fig. II. In other words, the component is contained in the domain and does not intersect with any of the four lines, $u = 0$, $u = 1$, $v = 0$ and $v = 1$. In this case, at least one point on the component satisfies the equations

$$\begin{aligned} D(u,v) &= 0 \\ D^u(u,v) &= 0 \end{aligned}$$

and we can compute it by finding all the roots of these equations in the appropriate domain.

Open components :- All other components fall into this category ($O1, \ldots, O6$ in Fig. II). Points on such components can be determined by finding the intersections of the curves corresponding to $\mathbf{G}(0,v)$, $\mathbf{G}(1,v)$, $\mathbf{G}(u,0)$ and $\mathbf{G}(u,1)$ with the surface $\mathbf{F}(s,t)$. Techniques for curve-surface intersection were presented in the previous section.

Thus the problem of computing a point on each component and the singular points has been reduced to curve-surface intersection and finding all roots of

$$\begin{aligned} D(u,v) &= 0 \\ D^u(u,v) &= 0 \end{aligned} \tag{7}$$

$$0 \le u \le 1, \quad 0 \le v \le 1.$$

Given a root of the above equations, it can be substituted into the equation $D^v(u,v) = 0$ to check whether it corresponds to a singular point.

5.2 Curve in Higher Dimension

We consider the surfaces $\mathbf{F}(s,t)$ and $\mathbf{G}(u,v)$ as defined in (1). The curve is defined in the parameter space of both the surfaces as an algebraic set of the form

$$\begin{aligned} F_1(s,t,u,v) = X(s,t)\overline{W}(u,v) - \overline{X}(u,v)W(s,t) &= 0 \\ F_2(s,t,u,v) = Y(s,t)\overline{W}(u,v) - \overline{Y}(u,v)W(s,t) &= 0 \\ F_3(s,t,u,v) = Z(s,t)\overline{W}(u,v) - \overline{Z}(u,v)W(s,t) &= 0. \end{aligned} \tag{8}$$

$$0 \le s \le 1, \quad 0 \le t \le 1, \quad 0 \le u \le 1, \quad 0 \le v \le 1$$

The components of the curve can be classified into closed loops and open components. The points on the open components can be computed by reducing the problem to curve surface intersection. As far as the problem of finding points on closed loops and singular points is concerned, we use an approach similar to the one used for the curve in the lower dimensional space. Lets consider the algebraic set defined by (8) and the extrema in the s direction is obtained by considering the total derivatives of the three equations and substituting $ds = 0$. The resulting equation is formulated by considering

$$\begin{aligned} \frac{\partial F_1}{\partial t}dt + \frac{\partial F_1}{\partial u}du + \frac{\partial F_1}{\partial v}dv = 0, \\ \frac{\partial F_2}{\partial t}dt + \frac{\partial F_2}{\partial u}du + \frac{\partial F_2}{\partial v}dv = 0, \\ \frac{\partial F_3}{\partial t}dt + \frac{\partial F_3}{\partial u}du + \frac{\partial F_3}{\partial v}dv = 0, \end{aligned}$$

where

$$\frac{\partial F_i}{\partial x} = \frac{\partial F_i(s,t,u,v)}{\partial x}, \quad i = 1,2,3, \quad x \in \{t,u,v\}.$$

This is equivalent to

$$\begin{bmatrix} \frac{\partial F_1}{\partial t} & \frac{\partial F_1}{\partial u} & \frac{\partial F_1}{\partial v} \\ \frac{\partial F_2}{\partial t} & \frac{\partial F_2}{\partial u} & \frac{\partial F_2}{\partial v} \\ \frac{\partial F_3}{\partial t} & \frac{\partial F_3}{\partial u} & \frac{\partial F_3}{\partial v} \end{bmatrix} \begin{bmatrix} dt \\ du \\ dv \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Let J correspond to the matrix on the left hand side of the above equation and all extremal points of the curve in s direction satisfy the equation

$$Det(J) = 0.$$

All the singular points on the curve satisfy the above equation, too. Furthermore, they also satisfy the extremal equations corresponding to t, u and v direction. As a result at least one point on each closed loop and the singular points are contained in the roots of

$$\begin{aligned} F_1(s,t,u,v) &= 0 \\ F_2(s,t,u,v) &= 0 \\ F_3(s,t,u,v) &= 0 \\ Det(J) &= 0 \end{aligned} \tag{9}$$

$$0 \le s \le 1, \quad 0 \le t \le 1, \quad 0 \le u \le 1, \quad 0 \le v \le 1$$

These are four equations in four unknowns and hence have a finite number of solutions (in general).

5.3 Equation Solving

In the previous section we reduced the problem of computing singular points and a starting point on closed loops to finding roots of nonlinear equations. In particular, the equations for curves in lower and higher dimensional space are highlighted in (7) and (9), respectively. In general, the complexity of exact algorithms for finding solutions of nonlinear equations is a function of *Bezout number* of the given system. The *Bezout number* corresponds to the total number of solutions that the system has in the complex projective space (counted properly) [Sa1885; WM88]. Given a system of equations, techniques for computing the *Bezout number* are presented in [WM88]. The domain is not restricted to a subset of the real space (as in our case). Lets consider the case of intersecting two bicubic Bézier surface patches. The intersection curve is of degree 324 (in general). Lets analyze the problem of finding roots of the equations for this case. For the projected curve, the monomials of highest degree in the system of equations (7) are $u^{54}v^{54}$ and $u^{53}v^{54}$. As a result, its Bezout number is 5778. In other words, the system has 5778 non-trivial solutions in the complex projective plane. Similarly, the Bezout number of the system (9) is 5346.

The high Bezout number of these system make algebraic techniques like resultants and Gröbner bases impractical [Bu85; MC90b]. Furthermore, in the context of floating point computations, the numerical accuracy of the results obtained is poorly understood.

Another method for solving system of nonlinear polynomial equations is the *homotopy method* [Mo87]. It is also possible to use it on polynomials expressed as unexpanded determinants. In the homotopy method, we start with a known system of equations (whose solutions are known) and march along to compute the solutions of the given system. While marching the number of paths correspond to the Bezout number of the given system of equations. Therefore, this approach becomes unattractive due to the high Bezout numbers of the systems expressed in (7) and (9). Furthermore, the accuracy of the solutions for such high degree equations is not well understood.

In practice, we would expect that the system of equations, (7) and (9), to have very few solutions in the domain of interest. As a consequence, we propose to use Newton's method for equation solving. However, Newton's method can never guarantee all the solutions in the given domain. Two main techniques to improve its performance are the use of interval arithmetic and constrained optimization [PP88; Ho89]. [PP88] used this method on low degree curves and it seems to work well for most cases. We are currently working on an implementation of this method for curves expressed in higher and lower dimensional space.

6 Conclusion

In this paper, we have presented a novel representation for the surface intersection problem. In particular, the computation of the representation is efficient and numerically stable and it is being used to develop a robust strategy for evaluating the surface intersection. We used results from linear algebra and numerical analysis to come up with efficient and numerically accurate algorithms for curve-surface intersection and computing the inverse image of a point on the surface. In the process we made use of routines from LINPACK and EISPACK for their implementation.

As far as the problem of tracing the intersection curve is concerned, we laid stress on the computation of singular points and a start point on each component. We reduced the problem to equation solving and are currently working on applying Newton's method and studying its performance. Many issues concerning the marching method, like the choice of step size and robust evaluation of all branches and components are still open.

The techniques presented in this paper are also useful for ray tracing parametric surface and representing offsets and blends of curves and surfaces.

7 Acknowledgements

We are grateful to Prof. J. Demmel and Prof. W. Kahan for productive discussions. This research was supported in part by David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577).

8 References

[BDMS79] Bunch, J., Dongarra, J., Moler, C. and Stewart, G.W. (1979) "LINPACK user's guide", SIAM, Philadelphia.

[BHLH88] Bajaj, C.L., Hoffmann, C.M., Lynch, R.E. and Hopcroft, J.E.H. (1988) "Tracing surface intersections", *Computer Aided Geometric Design*, vol. 5, pp. 285–307.

[BK90] Barnhill, R. E. and Kersey, S. N. (1990) "A marching method for parametric surface/surface intersection", *Computer Aided Geometric Design*, vol. **7**, pp. 257–280.

[Bu85] Buchberger, B. (1987) "Gröbner bases: An algorithmic method in polynomial ideal theory", in *Multidimensional Systems Theory,* edited by N.K. Bose, pp. 184-232, D. Reidel Publishing Co.

[Ch90] Chionh, E. W. (1990) "Base points, resultants, and the implicit representation of rational surfaces", Ph.D. thesis, Department of Computer Science, University of Waterloo, Canada.

[Di08] Dixon, A.L. (1908) "The eliminant of three quantics in two independent variables", *Proceedings of London Mathematical Society,* vol. **6**, pp. 49-69, 473-492.

[Fa86] Farouki, R.T. (1986) "The characterization of parametric surface sections", *Computer Vision, Graphics and Image Processing,* vol. **33**, pp. 209–236.

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

[GBDM77] Garbow, B.S., Boyle, J.M., Dongarra, J. and Moler, C.B. (1977) "Matrix eigensystem routines – EISPACK guide extension", *Lecture Notes in Computer Science,* vol. **51**, Springer-Verlag, Berlin.

[GLR82] Gohberg, I., Lancaster, P. and Rodman, L. (1982) *Matrix polynomials,* Academic Press, New York.

[GV89] Golub, G.H. and Van Loan, C. F. (1989) *Matrix computations,* The John Hopkins Press, Baltimore, Maryland.

[Ho88] Hoffmann, C. (1988) "A dimensionality paradigm for surface interrogations", Tech. report CSD-TR-837, Department of Computer Science, Purdue University.

[Ho89] Hoffmann, C. (1989) *Geometric and solid modeling: An introduction,* Morgan Kaufmann Publishers Inc.

[Ho90] Hoffmann, C. (1990) "Algebraic and numeric techniques for offsets and blends", in *Computation of Curves and Surfaces,* eds. W. Dahmen et. al., pp. 499-529, Kluwer Academic Publishers.

[LR80] Lane, J.M. and Riesenfeld, R.F. (1980) "A theoretical development for the computer generation and display of piecewise polynomial surfaces", *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. **2**, no. **1**.

[MC27] Morley, F. and Coble, A.B. (1927) "New results in elimination", *American Journal of Mathematics,* vol. **49**, pp. 463-488.

[MC90a] Manocha, D. and Canny, J. (1990a) "Implicitizing rational parametric surfaces", to appear in *Proceedings of IV IMA Conference on Mathematics of Surfaces,* University of Bath, England. Also available as Tech. report UCB/CSD 90/592, Computer Science division, University of California, Berkeley.

[MC90b] Manocha, D. and Canny, J. (1990b) "Multipolynomial resultant algorithms", *Proceedings of International Symposium on Intelligent Robotics,* pp. 348-358, Bangalore, India.

[Mr87] Morgan, A.P. (1987) *Solving polynomial systems using continuation for scientific and engineering problems,* Prentice-Hall, Englewood Cliffs, New Jersey.

[PK90] Ponce, J. and Kriegman, D.J. (1990) "Computing exact aspect graphs of curved objects: parametric surfaces", Tech. report UIUCDCS-R-90-1579, Department of Computer Science, University of Illinois at Urbana-Champaign.

[PP88] Prakash, P.V. and Patrikalakis, N.M. (1988) "Surface-to-surface intersections for geometric modeling", Tech. report MITSG 88-8, MIT Sea Grant College Program, MIT.

[Pr86] Pratt, M. J. (1986) " Surface/surface intersection problems", in *The Mathematics of Surfaces,* ed. by J.A. Gregory, pp. 118–142, Claredon Press, Oxford.

[Sa1885] Salmon, G. (1885) *Lessons introductory to the modern higher algebra,* G.E. Stechert & Co., New York.

[SAG84] Sederberg, T.W., Anderson, D.C. and Goldman, R.N. (1984) "Implicit representation of parametric curves and surfaces", *Computer Vision, Graphics and Image Processing,* vol. **28**, pp. 72-84.

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

[SR85] Semple, J.G. and Roth, L. (1985) *Introduction to algebraic geometry,* Claredon Press, Oxford, Great Britian.

[Wa50] Walker, Robert J. (1950) *Algebraic curves,* Princeton University Press, New Jersey.

[Wd50] van der Waerden B. L. (1950) *Modern algebra,* (third edition) F. Ungar Publishing Co., New York.

[Wi63] Wilkinson, J.H. (1963) *Rounding errors in algebraic processes,* Prentice-Hall, Englewood Cliffs, NJ.

[WM88] Wampler, C.W. and Morgan, A.P. (1988) "Numerical continuation methods for solving polynomial pystems arising in kinematics", Tech. report GMR-6372, Mathematics Department, General Motors Research Laboratories, Warren, Michigan.