A NEW APPROACH FOR SURFACE INTERSECTION

DINESH MANOCHA

Computer Science Division,
University of California at Berkeley,
Berkeley, CA 94720, U.S.A.

and

JOHN F. CANNY

Computer Science Division,
University of California at Berkeley,
Berkeley, CA 94720, U.S.A.

Received 7 June 1991
Revised 6 September 1991

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 perform matrix operations for evaluation and use results from linear algebra for 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.

Keywords: Curve, Surface, Intersection, Solid Modeling, Resultants, matrix operations

* Supported by IBM Graduate Fellowship, David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577).
† Supported in part by David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577).
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.\(^1\)

Earlier approaches to surface intersection used recursive subdivision techniques based on the paradigm of ‘divide and conquer’.\(^2\)\(^,\)\(^7\) 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.\(^2\)\(^,\)\(^3\)\(^,\)\(^4\)\(^,\)\(^5\)\(^,\)\(^6\)\(^,\)\(^7\) Tracing techniques involve the computation of starting point on each component and locating 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.

As far as the problem of computing a point on each component is concerned, an algorithm based on loop detection is presented by Sederberg and Meyers.\(^8\) 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 resulting algorithm does not account for singular points and has been found slow in practice.\(^8\) Thus, no good algorithms are available for finding a point on each component or the singular points on the intersection curve.

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.\(^3\) 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.\(^10\) However, Hoffmann counsels against computing the projection in the lower dimensional space for the following reasons:\(^10\)\(^,\)\(^11\):
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.

- 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.

We use results from Elimination theory to represent the implicit equation of a rational parametric surface as a matrix determinant. 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 a variation of Gauss elimination to compute 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. 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 and present an algorithm for evaluating surface intersections.
2. Background

2.1. Surface Intersection

Computing the intersection of solids and surfaces is a fundamental operation in solid modeling systems. This operation arises in performing Boolean operations on B-rep solids. Most of the solid modeling systems were earlier limited to solids, whose boundaries were composed of linear or quadric surfaces. Currently these systems are extending their geometric coverage to use free-form surfaces (represented as rational parametric surfaces) for defining the solid boundaries. As a result, a fundamental problem in such systems is that of computing the intersection of rational parametric surfaces. The problem 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 dependent on the particular application of the solid modeling system. However, 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.

The intersection of parametric surfaces results in a high degree algebraic curve. The algebraic complexity makes it difficult to compute an exact representation as an algebraic set and therefore, most of the current techniques aim at an approximate representation as a piecewise linear curve (obtained by subdivision or tracing methods). However, this representation is not robust for answering many of the queries highlighted above.

The topology of the intersection curve can be very complicated, too. The intersection curve may have more than one component, singular points and thereby adding to the geometric complexity of the problem. 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. One such case involving the intersection curve of a plane and bicubic patch is shown in Fig. 1. The intersection curve has degree 18.

2.2. Rational Parametric Surfaces

Many current geometric modeling systems use rational parametric surfaces, which includes Bézier curves and surfaces and their equivalents (like B-spline patches). These surfaces are used for designing geometric objects to satisfy various interpolatory, smoothness and aesthetic requirements. Currently they are being used for defining the boundaries of B-rep solids in the solid modeling systems. Given two
B-spline surface patches, we use the knot insertion algorithms and thereby convert them into a series of Bézier patches.\cite{17}. As a result, the problem of surface intersection of rational parametric surfaces reduces to computing the intersection of a series of Bézier patches.

![Intersection curve consisting of eight components](image)

The modeling systems use two kind of parametric patches for defining the boundaries. They are tensor product Bézier patches and triangular Bézier patches.

2.2.1. Tensor Product Bézier Patches

Tensor product patches are represented in homogeneous coordinates as\cite{17}:

\[
\mathbf{F}(s, t) = (X(s, t), Y(s, t), Z(s, t), W(s, t)) = \left( \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{V}_{ij} B_{i,m}(s), B_{j,n}(t) \right),
\]

where \( \mathbf{V}_{ij} = (x_{ij}, y_{ij}, z_{ij}, w_{ij}) \) and

\[
B_{i,m}(s) = \binom{m}{i} s^i (1-s)^{m-i}, \quad B_{j,n}(t) = \binom{n}{j} t^j (1-t)^{n-j}
\]

are the Bernstein polynomials. The domain of the surface is restricted to \( s \in [0, 1], \ t \in [0, 1] \). For polynomial patches all \( w_{i,j} = 1 \).

Many of the algorithms presented in this paper assume polynomials represented in power basis as opposed to Bernstein basis. Converting from Bernstein to power basis involves a linear transformation expressed as multiplying by a matrix. However, this transformation can introduce numerical problems.\cite{18} To circumvent this problem we perform a reparametrization of the form

\[
\tau = g(s) = \frac{s}{1-s}, \quad \tau = g(t) = \frac{t}{1-t},
\]
and the resulting surface parametrization is of the form

\[ F(\xi, \zeta) = (X(\xi, \zeta), Y(\xi, \zeta), Z(\xi, \zeta), W(\xi, \zeta)) = (\sum_{i=0}^{m} \sum_{j=0}^{n} V_{ij} \left( \binom{m}{i} \right) (\xi)^i (\zeta)^j). \]

The domain of this surface parametrization is \( \xi \geq 0, \ zeta \geq 0. \) Moreover

\[ F(s, t) = F(g(s), g(t)). \]

The reparametrization, \( g(s) \), is also used on the Bézier curve in the curve surface intersection algorithm. Unfortunately, this reparametrization can cause overflow problems for \( s \approx 1 \), since it makes \( \xi \) very large. For example, when we are computing curve surface intersection and the intersection point is close to one. We introduce checks based on properties of matrices and change the algorithms accordingly. More details are given in Section 3 and 4.

2.2.2. Triangular Bézier Patches

An alternate representation for Bézier patches in terms of bivariate Bernstein basis is \(^{15}\):

\[ F(s, t) = (X(s, t), Y(s, t), Z(s, t), W(s, t)) = (\sum_{i=0}^{n} \sum_{j=0}^{n-i} V_{ij} \cdot B_{i,j}(s, t)), \]

where \( V_{ij} = (x_{ij}, y_{ij}, z_{ij}, w_{ij}) \) and

\[ B_{i,j}^n(s, t) = \binom{n}{i,j} s^i t^j (1 - s - t)^{n-i-j}, \quad \binom{n}{i,j} = \frac{n!}{i! j! (n - i - j)!}. \]

The domain of the surface is \( 0 \leq (s + t) \leq 1. \)

To obtain an equivalent surface in power basis we perform a reparametrization of the form

\[ \bar{\xi} = g(s, t) = \frac{s}{1 - s - t}, \quad \bar{zeta} = h(s, t) = \frac{t}{1 - s - t}, \]

and the resulting surface parametrization is of the form

\[ \bar{F}(\bar{\xi}, \bar{zeta}) = (\bar{X}(\bar{\xi}, \bar{zeta}), \bar{Y}(\bar{\xi}, \bar{zeta}), \bar{Z}(\bar{\xi}, \bar{zeta}), \bar{W}(\bar{\xi}, \bar{zeta})) = (\sum_{i=0}^{n} \sum_{j=0}^{n-i} V_{ij} \left( \binom{n}{i,j} \right) (\bar{\xi})^i (\bar{zeta})^j). \]

The domain of this surface parametrization is \( \bar{\xi} \geq 0, \ \bar{zeta} \geq 0 \) and

\[ \bar{F}(s, t) = \bar{F}(g(s), g(t)). \]

As for tensor product surfaces, this reparametrization can introduce overflow problems, whenever \( s + t \approx 1 \). We check for the condition number of the resulting matrices to detect these conditions.

6
2.3. 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 Bézier 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.\(^\text{20}\) In other words, given two Bézier surfaces

\[
\mathbf{F}(s, t) = (X(s, t), Y(s, t), Z(s, t), W(s, t))
\]

and

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

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:

\[
f(X(u, v), Y(u, v), Z(u, v), W(u, v)) = 0,
\]

\[
0 \leq u \leq 1, \quad 0 \leq v \leq 1.
\]

The problem of implicitization corresponds to formulating parametric equations like

\[
xW(s, t) - X(s, t) = 0
\]
\[
yW(s, t) - Y(s, t) = 0
\]
\[
zW(s, t) - Z(s, t) = 0
\]

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.\(^\text{21,22}\) As far as geometric and solid modeling are concerned, these techniques were resurrected by Sederberg.\(^\text{9,23}\) In particular, Sederberg 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.\(^\text{1,24}\) 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.\(^\text{10,11}\)

The problem of implicitization has been recently analyzed.\(^\text{14,25,26}\) 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\(^\text{6}\) of

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

\(^{6}\)They also include the roots at infinity
For a random choice of coefficients a parametrization has no base points. The three parametric equations are of equal degree and the implicit representation corresponds to the resultant of the parametric equations.\textsuperscript{13,14} Using Dixon’s formulation,\textsuperscript{27} the resultant corresponds to the determinant of a matrix. 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 \(2n^2 - n.\textsuperscript{14,27}\) Each entry of the matrix is a linear polynomial of the form\textsuperscript{3,30},

\[
a_{ij} x + b_{ij} y + c_{ij} z + d_{ij} w,
\]

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 used for computing the resultant of polynomial equations and thereby the implicit equation.\textsuperscript{14,38} 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

In this Section, we make use of Dixon’s resultant formulation\textsuperscript{27} to represent the implicit equation as an unexpanded determinant and present algorithms to perform geometric operations on the representation. 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 two Bézier surfaces, \(\textbf{F}(s, t)\) and \(\textbf{G}(u, v)\) as in (1), we use the corresponding formulation of resultant (depending on the fact whether \(\textbf{F}(s, t)\) is a tensor product surface or a triangular patch) and represent the implicit equation as a matrix determinant, \(f(x, y, z, w)\). The algorithm for computing resultant assumes that the polynomials are represented in power basis.\textsuperscript{27} We use the reparametrizations mentioned in Sections 2.2.1 and 2.2.2 to transform the Bézier surfaces into power basis and apply the resultant formulation on the resulting parametrizations. More details on the computation and implementation of the algorithm are given in.  \textsuperscript{30}

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

\[
f(x, y, z, w) = \begin{vmatrix}
 f_{11}(x, y, z, w) & \cdots & f_{1n}(x, y, z, w) \\
 f_{21}(x, y, z, w) & \cdots & f_{2n}(x, y, z, w) \\
 \vdots & \vdots & \vdots \\
 f_{n1}(x, y, z, w) & \cdots & f_{nn}(x, y, z, w)
\end{vmatrix}, \quad (3)
\]

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 algebraic plane curve, birational to the intersection curve, is obtained by substituting the parametrization \( G(u, v) \) into \( f(x, y, z, w) \). As a result, the intersection curve is represented as zero set of a determinant. The corresponding matrix is

\[
M(u, v) = f(X(u, v), Y(u, v), Z(u, v), W(u, v)) = \begin{bmatrix}
g_{11}(u, v) & \ldots & g_{1n}(u, v) \\
g_{21}(u, v) & \ldots & g_{2n}(u, v) \\
\vdots & \vdots & \vdots \\
g_{n1}(u, v) & \ldots & g_{nn}(u, v)
\end{bmatrix},
\]

where

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

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 \( n \). We use \( M(u, v) \) to denote the matrix used for representing the intersection curve and \( D(u, v) \) to represent the polynomial corresponding to \( \text{Determinant}(M(u, v)) \). The algebraic plane curve corresponds to \( D(u, v) = 0 \) lying in the region \( 0 \leq u \leq 1, \ 0 \leq v \leq 1 \) and this subset corresponds to the preimage of the intersection curve of Bézier patches. In Fig. 2 the intersection curve of two tensor product Bézier patches has been shown. The intersection curve, \( I \), is in \( (X, Y, Z) \) space and its preimages, \( P1 \) and \( P2 \), in the parametric domain of two surfaces, are trimmed algebraic plane curves. The curve \( P1 \), a subset of \( T(s, t) = 0 \), is obtained by implicitizing \( G(u, v) \) and substituting the parametrization, \( F(s, t) \), into the resulting matrix obtained after implicitization. There is a one-to-one correspondence between the points on \( I \), \( P1 \) and \( P2 \). \( G^{-1}(x, y, z) \) correspond to the preimage of the point \((x, y, z)\) lying on the Bézier surface. If the parametrization is faithful, it is possible to express \( G^{-1}(x, y, z) \) as a rational function. As a result,

\[
(s, t) = \left( \frac{P_1(u, v)}{Q(u, v)}, \frac{P_2(u, v)}{Q(u, v)} \right)
\]

and

\[
(u, v) = \left( \frac{S_1(s, t)}{T(s, t)}, \frac{S_2(s, t)}{T(s, t)} \right),
\]

where \( P_1 \), \( S_1 \), \( Q \) and \( T \) are polynomials. These relationships are defined only for points lying on \( P1 \) and \( P2 \). In Section 4 we use properties of the matrix formulation to compute \( G^{-1}(x_1, y_1, z_1) \). This relationship is used for trimming \( T(s, t) = 0 \) and \( D(u, v) = 0 \) to compute \( P1 \) and \( P2 \), respectively. \( P1 \) and \( P2 \) are referred to as the curves in lower dimension. The higher dimensional curve lies in the four dimensional
space spanned by the parameters, $s, t, u$ and $v$. It is represented as an algebraic set defined by the intersection of

$$
F_1(s, t, u, v) = X(s, t)W(u, v) - X(u, v)W(s, t) = 0
$$
$$
F_2(s, t, u, v) = Y(s, t)W(u, v) - Y(u, v)W(s, t) = 0
$$
$$
F_3(s, t, u, v) = Z(s, t)W(u, v) - Z(u, v)W(s, t) = 0
$$

in the domain

$$
0 \leq s \leq 1, \ 0 \leq t \leq 1, \ 0 \leq u \leq 1, \ 0 \leq v \leq 1
$$

The one-to-one correspondence is useful for performing operations like sorting points on the intersection curve. It is much simpler to sort points on the planar curve as opposed to curves in higher dimension. Furthermore, this correspondence is used for developing a robust strategy for tracing in Section V. The planar preimage of the intersection curve corresponding to Fig. 1 has been shown in Fig. 3. 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 of the surface) are $O1, \ldots, O6$.

![Diagram](image)

**Fig. 3.** The projection of intersection curve in Fig. 1

### 3.1. Matrix Operations on the Representation

We make use of the representation presented in the previous section to perform geometric operations. For tracing or marching through the intersection curve in lower dimension, we need to effectively evaluate expressions like $D(u_1, v_1)$, $Du(u_1, v_1)$, $DuDu(u_1, v_1)$, where $u_1$ and $v_1$ are real numbers and $D$ and $Du$ 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.

- **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.** 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.

To compute the first and higher order partials, we use a simple variation of Gauss elimination. The basic idea is to compute the partial derivative of each matrix entry at the beginning of computation and update the derivative information along with each step of Gauss elimination. In this case, we modify the matrix
structure such that entry consists of a tuple $G_{ij}(u_1, v_1) = (g_{ij}(u_1, v_1), g_{ij}^u(u_1, v_1))$, where $g_{ij}^u(u_1, v_1)$ 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

$$M(u_1, v_1) =$$

$$\begin{bmatrix}
  G_{11}(u_1, v_1) & \cdots & G_{1n}(u_1, v_1) \\
  \vdots & \ddots & \vdots \\
  G_{n1}(u_1, v_1) & \cdots & G_{nn}(u_1, v_1)
\end{bmatrix}.$$  

To compute $D(u_1, v_1)$ and $D^{\mu}(u_1, v_1)$ 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 $i$th and $k$th rows of the matrix. A typical step of Gauss elimination is of the form

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

where $g_{kj}$ represents the element in the $k$th row and $j$th column of the matrix. In the new formulation this step is replaced as

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

(6)

$$g_{kj}^{\mu} = g_{kj}^{\mu*} - \frac{(g_{ki}^{\mu*} g_{ij} + g_{ki} g_{ij}^{\mu*}) g_{ii} - (g_{ki} g_{ij}) g_{ii}^{\mu*}}{(g_{ii})^2}.$$  

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 $D^{\mu}(u_1, v_1)$ in the following manner:

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

$$D^{\mu}(u_1, v_1) = D(u_1, v_1) \sum_{i=1}^{n} g_{ii}^{\mu*}.$$  

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 numerical 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 (6), the pivoting process chooses $g_{ii}$ to be the element of maximum magnitude (with respect to the $i$th column or the rest of the matrix). The computation of $g_{ij}^{\mu}$ 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 \(F(s, t)\) and \(G(u, v)\). Previous approaches to this problem take the surface parametrization and formulate the equations

\[
\begin{align*}
  x_1 W(s, t) - X(s, t) &= 0 \\
  y_1 W(s, t) - Y(s, t) &= 0 \\
  z_1 W(s, t) - Z(s, t) &= 0
\end{align*}
\]

and determine whether these equations have a common solution in the domain used for defining the surfaces. For Bézier patches the domain is \(0 \leq s \leq 1\) and \(0 \leq t \leq 1\). 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.

The implicit representation of the surface is the determinant of \(f(x, y, z, w)\), as formulated in (3). 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.

We used resultants to represent the implicit equation as a matrix determinant. Assume that the given surfaces are tensor product surfaces of the form \((s^d t^d)\). Similar analysis is applicable if either of the surfaces is a triangular patch. Techniques from Elimination theory take the following parametric equations:

\[
\begin{align*}
   xW(s, t) - X(s, t) &= 0 \\
   yW(s, t) - Y(s, t) &= 0 \\
   zW(s, t) - Z(s, t) &= 0
\end{align*}
\]

and reduce it to a problem in linear algebra of the form\(^{32}\):

\[
MX = 0,
\]

where

\[
M = \begin{bmatrix}
   f_{11}(x, y, z, w) & \cdots & f_{1n}(x, y, z, w) \\
   \vdots & \ddots & \vdots \\
   f_{n1}(x, y, z, w) & \cdots & f_{nn}(x, y, z, w)
\end{bmatrix}
\]

\[
X = [(1 - s)^{2d-1} (1 - t)^{d-1}, (1 - s)^{2d-1} t (1 - t)^{d-2}, \ldots, (1 - s)^{2d-1} t^{d-1}, \ldots, \\
   s (1 - s)^{2d-2} (1 - t)^{d-1}, \ldots, s (1 - s)^{2d-2} t^{d-1}, \ldots, s^{2d-1} t^{d-1}]^T
\]

\[
0 = [0 \ 0 \ 0 \ \ldots \ 0]^T,
\]

and \(M\) is equivalent to \(f(x, y, z, w)\) in (3) and \(n = 2d^2\), for such tensor product surfaces.

The fact that the determinant of \(f(x_1, y_1, z_1, 1)\) is zero implies that the numeric matrix is singular. Let us initially consider the case when \((x_1, y_1, z_1, 1)\) has a unique preimage. From the properties of implicit representation it follows that \(f(x_1, y_1, z_1, w_1)\) is a matrix with a kernel of dimension one.\(^{14}\) We can use techniques like SVD (singular value decomposition) to accurately determine whether the matrix is singular or not. In case, it is singular the problem of computing the kernel reduces to finding the eigenvector corresponding to the zero eigenvalue.\(^{30,32}\) Let that vector be \(V = (v_1 \ v_2 \ \ldots \ v_n)^T\). As a result, the preimage of the point \((x_1, y_1, z_1, w_1)\) can be obtained by solving the equation

\[
[k [(1 - s)^{2d-1} (1 - t)^{d-1}, (1 - s)^{2d-1} t (1 - t)^{d-2}, \ldots, \\
   s (1 - s)^{2d-2} (1 - t)^{d-1}, \ldots, s (1 - s)^{2d-2} t^{d-1}, \ldots, s^{2d-1} t^{d-1}]^T = [v_1 \ v_2 \ \ldots \ v_n]^T,
\]

where \(k\) is a scalar. The point lies on the Bézier surface, if \(0 \leq s \leq 1, 0 \leq t \leq 1\).

Lets consider the case when \((x_1, y_1, z_1, 1)\) has more than one preimage. This hold when the given parametrization is unfaithful or the given point is a singular point on the surface. Such cases are rare in practice. From the properties of implicit representation and resultant formulation, it follows that the number of distinct
preimages correspond to the dimension of the kernel of \( f(x_1, y_1, z_1, 1) \). Let that dimension be \( k \) and the corresponding vectors be \( V_1, V_2, \ldots, V_k \). Furthermore

\[
V_i = [v_{i,1} \, v_{i,2} \, \ldots \, v_{i,n}]^T.
\]

Our aim is now to compute the \( k \) dimension in terms of the preimages. \( X_i \) is of the form

\[
X_i = [(1 - s_i)^{2d-1}(1 - t_i)^{d-1}, (1 - s_i)^{2d-1}t_i(1 - t_i)^{d-2}, \ldots, (1 - s_i)^{2d-1}t_i^{d-1}, \ldots, s_i(1 - s_i)^{2d-2}t_i^{d-1}, \ldots, s_i^{2d-1}t_i^{d-1}]^T
\]

We know that \( X_i \) belongs to the vector space spanned by \( V_1, V_2, \ldots, V_k \). Therefore,

\[
X_i = a_1V_1 + a_2V_2 + \ldots + a_kV_k. \tag{7}
\]

Our problem reduces to computing the \( a_i \)'s by making use of the properties among the elements of \( X_i \).

It follows from (7):

\[
x_{i,j} = a_1v_{1,j} + a_2v_{2,j} + \ldots + a_kv_{k,j}.
\]

Using the properties of Bernstein basis, we obtain:

\[
E1 = x_{i,1} + \left( \frac{d-1}{1} \right) x_{i,2} + \ldots + \left( \frac{d-1}{d-1} \right) x_{i,d} = (1 - s_i)^{2d-1}
\]

\[
E2 = x_{i,d+1} + \left( \frac{d-1}{1} \right) x_{i,d+2} + \ldots + \left( \frac{d-1}{d-1} \right) x_{i,2d} = s(1 - s_i)^{2d-2}
\]

\[
E2d^2 = x_{i,2d^2-d+1} + \left( \frac{d-1}{1} \right) x_{i,2d^2-d+2} + \ldots + \left( \frac{d-1}{d-1} \right) x_{i,2d^2} = s^{2d-1}
\]

The symbols \( E1, E2, \ldots, E2s^2 \) are used to represent the left hand sides of the equations. From these relations we derive the equations \((a_i)'s\ are \( the \ variables:\)

\[
E1 + \left( \frac{2d-1}{1} \right) E2 + \ldots + \left( \frac{2d-1}{2d-1} \right) E2d^2 = 1,
\]

\[
\frac{E1}{E2} = \frac{E2}{E3} = \ldots = \frac{E2d^2 - 1}{E2d^2}. \tag{8}
\]

All these equations are not independent and they have \( k \) distinct solutions. We can use techniques from Elimination theory to solve these equations. A detailed method is given in. \(^{28}\) It uses \( u \)-resultants, \(^{22}\) and reduces the problem to eigenvalue problems. For most cases \( k \) may be at most two or three and we can easily compute the solutions.

Good implementations of SVD are available as part of LINPACK. \(^{15}\) 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). 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

\[
\mathbf{G}(u) = (X(u), Y(u), Z(u), W(u)) \quad 0 \leq u \leq 1
\]

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. 5). 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.

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.\textsuperscript{33,31}
Furthermore, good implementations of these algorithms are available as part of a package like EISPACK.\textsuperscript{16} We use the reparametrization, $\overline{u} = \frac{u}{w}$, to obtain a formulation of the Bézier curve in power basis. Given $f(x, y, z, w)$, we substitute the parametric equation of the curve to obtain a matrix of the form

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

where

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

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

$$M(\overline{u}) = \overline{u}^d M_d + \overline{u}^{d-1} M_{d-1} + \ldots + \overline{u} M_1 + M_0, \quad (9)$$

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

$$\det(M(\overline{u})) = 0,$$

$$\det(M_d^{-1}) \det(M(\overline{u})) = 0.$$

Let

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

where

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

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

$$C = \begin{bmatrix} 0 & I_0 & 0 & \ldots & 0 \\ 0 & 0 & I_0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & I_0 \\ -\overline{M}_0 & -\overline{M}_1 & -\overline{M}_2 & \ldots & -\overline{M}_{d-1} \end{bmatrix}, \quad (10)$$

such that the eigenvalues of $C$ correspond exactly to the roots of $\det(\overline{M}(\overline{u})) = 0$. $C$ is a numeric matrix of order $dn$. If $M_d$ is a singular matrix, techniques to compute the roots of $\det(M(\overline{u})) = 0$ are given in.\textsuperscript{30} In particular, we reduce the problem to an eigenvalue problem (obtained after a transformation) or a generalized eigenvalue problem depending on the condition numbers of $M_d, M_{d-1}, M_{d-2}, \ldots, M_0$.\textsuperscript{30,33} It is also possible that $M_d$ has a high condition number and the computation of $M_d^{-1}$
may not be numerically stable. In that case, we treat $M_d$ as being close to a singular matrix.

Let's consider the case, when the intersection point $u \approx 1$. As a result, $\mathbf{u} \approx \infty$ and the computation of eigenvalues and eigenvectors of $C$ may be numerically inaccurate due to overflow errors. In this case, the determinant of $M(\mathbf{u})$, in (9), has a root $\approx \infty$. This is possible if and only if $M_d$ is close to being singular. As a result, our algorithm would treat this case separately and may reduce the problem of curve surface intersection to an eigenvalue problem (obtained after a transformation) or a generalized eigenvalue problem.

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 (and their reparametrizations), the domain is $[0, 1]$ or $[0, \infty]$.

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 (10) is concerned, we treat it as a general unsymmetric matrix. We use the routine RG from EISPACK for computing the eigenvalues.$^{16}$ 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.$^{31}$ 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.

<table>
<thead>
<tr>
<th>Order of Matrix</th>
<th>Time in milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>8.6</td>
</tr>
<tr>
<td>20</td>
<td>15.7</td>
</tr>
<tr>
<td>25</td>
<td>25.7</td>
</tr>
<tr>
<td>30</td>
<td>38.7</td>
</tr>
<tr>
<td>35</td>
<td>57.1</td>
</tr>
<tr>
<td>40</td>
<td>77.3</td>
</tr>
<tr>
<td>45</td>
<td>103.3</td>
</tr>
<tr>
<td>50</td>
<td>133.9</td>
</tr>
<tr>
<td>55</td>
<td>165.3</td>
</tr>
<tr>
<td>60</td>
<td>212.1</td>
</tr>
<tr>
<td>65</td>
<td>262.1</td>
</tr>
</tbody>
</table>

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 (10) 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 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. Moreover, the accuracy of the results is well understood. In particular, it is possible to compute the condition number of the eigenvalue which gives a lower bound on the number of correct digits in the computed eigenvalue.\footnote{In case, the condition number is low, we may use higher precision arithmetic.}

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. Evaluating 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 (4). 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. Furthermore, it is relatively simpler to determine the number of branches at a singular point. Moreover, there is an elegant theory of resolution of singularities in algebraic geometry, which can be used for dealing with complicated singularities.\footnote{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.}

It is widely believed that the lower dimensional approach has to cope up with a number of practical difficulties. 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.\footnote{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.}

We propose an algorithm which makes use of the curve expressed in higher as well as lower dimensional. We make use of the fact that there is one-to-one correspondence between the points on these curves. As a result both the curves contain the same number of components. We use this property to find a point
on each component of the higher dimensional curve. Given a start point, we use marching methods on the lower dimensional curve (expressed as an algebraic plane curve). While marching along the curve we make use of the properties of algebraic plane curves to evaluate all the branches.

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

\[
D(u, v) = 0 \\
D'(u, v) = 0 \\
D''(u, v) = 0
\]

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. 3. 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

\[
D(u, v) = 0 \\
D'(u, v) = 0
\]

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. 3. Points on such components can be determined by finding the intersections of the boundary curves corresponding to \( G(0, v), G(1, v), G(u, 0) \) and \( G(u, 1) \) with the surface \( F(s, t) \). Techniques for curve-surface intersection were presented in the previous section. Many of these intersections can be eliminated by using the bounding box test comparing the control polygon of the curves with the control polyhedra of the surface.

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

\[
D(u, v) = 0 \\
D'(u, v) = 0
\]

\( 0 \leq u \leq 1, \quad 0 \leq v \leq 1. \)

Given a root of the above equations, it can be substituted into the equation \( D''(u, v) = 0 \) to check whether it corresponds to a singular point.
The number of branches at a singular point, \((u_0, v_0)\), are determined by evaluating the second or higher order partial derivatives at that point (like \(D^{uu}(u_0, v_0)\), \(D^{uv}(u_0, v_0)\)). This corresponds to computing the number of places the curve has at that point. If a curve has more than one place, the local directions for tracing can be computed from these partial derivatives.

5.2. Curve in Higher Dimension

We consider the surfaces \(F(s,t)\) and \(G(u,v)\) as defined in (1). The curve is defined in the parameter space of both the surfaces as an algebraic set of the form, (5),

\[
F_1(s,t,u,v) = 0 \\
F_2(s,t,u,v) = 0 \\
F_3(s,t,u,v) = 0.
\]

\(0 \leq s \leq 1, \; 0 \leq t \leq 1, \; 0 \leq u \leq 1, \; 0 \leq v \leq 1\)

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. Let's consider the algebraic set defined by (5) 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

\[
\frac{\partial F_1}{\partial t} dt + \frac{\partial F_1}{\partial u} du + \frac{\partial F_1}{\partial v} dv = 0, \\
\frac{\partial F_2}{\partial t} dt + \frac{\partial F_2}{\partial u} du + \frac{\partial F_2}{\partial v} dv = 0, \\
\frac{\partial F_3}{\partial t} dt + \frac{\partial F_3}{\partial u} du + \frac{\partial F_3}{\partial v} dv = 0,
\]

where

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

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

$$F_1(s, t, u, v) = 0$$
$$F_2(s, t, u, v) = 0$$
$$F_3(s, t, u, v) = 0$$
$$\text{Det}(J) = 0$$

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

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 and curve surface intersections. In particular, the equations for curves in lower and higher dimensional space are highlighted in (12) and (13), 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). It highlights the algebraic complexity of the given problem. Given a system of equations, techniques for computing the Bezout number are presented in. The domain is not restricted to a subset of the real space (as in our case). Let's consider the intersection of two bicubic Bézier surface patches. The intersection curve is of degree 324 (in general). We 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 (12) 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 (13) is 5346.

The high Bezout number of these systems make algebraic techniques like resultants and Gröbner bases impractical. Furthermore, in the context of floating point computations, the numerical accuracy of the results obtained using Gröbner bases is poorly understood. In an algorithm for computing roots of non-linear equations is presented. It expresses the $u$-resultant of the given system of equations as a matrix determinant. The problem of computing roots is eventually reduced to an eigenvalue problem. For our application this algorithm can only be applied to the system of equations corresponding to the curve in higher dimension, (13). However, the Bezout number of this system imposes a lower bound on the order of the matrix, whose eigenvalues are to be computed. As a result, this approach becomes unattractive.
Another method for solving system of nonlinear polynomial equations is the homotopy method. 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 (12) and (13). 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, (12) and (13), 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 not guarantee all the solutions in the given domain. Two main techniques to improve its performance are the use of interval arithmetic and constrained optimization. This method has been used on algebraic plane curves and it seems to work well for most cases. It is possible to represent the equations in Bernstein basis and make use of the properties of the resulting control polyhedron for determining a sufficient criterion for the nonexistence of the solutions in a particular interval. In case, the criterion is not satisfied, we subdivide the resulting polytope (and thereby obtaining a closer approximation to the surface). This process of solving equations by Newton’s iteration, constrained optimization and subdividing the control polytope can be repeated until all the solutions are computed. This technique has been applied to computing intersections of two algebraic plane curves in a particular interval. We propose to use this algorithm on the system of equations corresponding to the curve in higher dimension, (13). The resulting tracing algorithm is:

- Compute a starting point on each component. The points on the open components are obtained by curve-surface intersection algorithms and the points on closed components are obtained by computing roots of (13).
- Given a start point, use marching method on the lower dimensional curve. A marching algorithm for plane curves has been highlighted in.
- As we are marching along a component, compute the first order partials. If the magnitude of the first order partials is lower than a certain threshold, we may be close to a singular point. Use Newton’s iteration to accurately compute the singular point and the higher order partials to determine the number of places the curve has at that point. We keep track of all the branches the curve has at that point and use the higher order partials for determining the local direction for each branch. This process is repeated until all the branches have been evaluated.

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 presented an algorithm making use of the properties of the curve expressed in higher and lower dimensional spaces.

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

**Acknowledgements**

We are grateful to Prof. J. Demmel and Prof. W. Kahan for productive discussions.


Cornell University, Department of Computer Science, 1987.


