Second International Conference on Curves and Surfaces
Chamonix-Mont-Blanc, France, June 10-16, 1993

Evaluating Surface Intersections in Lower Dimensions

Dinesh Manocha, Amitabh Varshney, and Hans Weber

Abstract. We highlight a new algorithm for evaluating the surface intersection curve using a matrix formulation. The projection of the intersection curve is represented as the singular set of a bivariate matrix polynomial. The resulting algorithm for evaluating the intersection curve is based on matrix computations like eigendecomposition and singular value decomposition. Furthermore, at each stage of the algorithm we make use of inverse power iterations to march back to the curve. We also describe the performance of the resulting robust and accurate approach.

§1. Introduction

Evaluating the intersection of parametric and algebraic surfaces is a recurring operation in geometric and solid modeling. Its applications include boundary evaluations, simulation of manufacturing processes, contouring of scattered data, and finite element mesh generations. Surface intersections have been extensively studied in the literature, and the main approaches may be classified as analytic, lattice, marching, or subdivision methods. The analytic approach is considered slow in practice due to the algebraic complexity of the intersection curve, and approaches based on subdivision, lattice, and marching methods may not be robust. Furthermore, their accuracy varies with the surface degree, the local surface geometry at the intersection curve, and the angles at which the surfaces intersect. As a result, it is believed that any surface intersection algorithm has to balance three conflicting goals of accuracy, robustness, and efficiency [7,13].

Earlier approaches to surface intersection used the subdivision properties of NURBS surfaces based on a "divide and conquer" paradigm [13]. However, this approach may be slow and is not guaranteed to be robust in terms of finding all components of the intersection curve. In the last decade "marching methods" have received a lot of attention for the evaluation of intersection curves [1,2,3,9,13]. Tracing techniques involve the computation of a 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. In particular, the tracing algorithms use the algebraic formulation of the intersection problem (e.g. three algebraic equations in four unknowns for NURBS surfaces) and find successive points on the curve with first order approximations and refinement using Newton-Raphson's method. However, it is not clear what a good step size should be at each stage. A small step size makes the overall approach slow, and a large step size may result in convergence problems. It is possible to compute a higher order local approximation at each point on the curve [1], however this involves a great deal of symbolic computation and may not be efficient for high degree intersection curves. Therefore, implementing a robust tracer based on Newton-Raphson's method can be fairly non-trivial and in many cases may not work at all [5]. Other approaches consist of approximating the intersection curve using lattice methods or geometric Hermite approximation [15].

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 based on elimination theory can be used to project the intersection curve to a plane curve [10,11,14]. However, the degree of the resulting curve is fairly high (e.g. 108 for the intersection of two bicubic patches) and computation and evaluation based on such a representation can involve efficiency and accuracy problems [8]. However, it is possible to represent the plane curve as a matrix determinant and use matrix computations for evaluation. In this paper, we present a robust algorithm for tracing curves based on a matrix representation. In particular, we represent the intersection curve as the singular set of a bivariate matrix polynomial and use algorithms based on eigendecomposition and singular value decomposition to trace the resulting curve. This involves the use of inverse power iterations for eigenvalue computation and step sizes based on the local geometry of the curve. The main advantage of this approach is in its robustness. Tracing in lower dimension reduces the *geometric complexity* of the curve. Moreover the convergence of power iterations is well understood, and this representation is used in computing the appropriate step sizes. Furthermore, the matrix representation is used for computing the singular points on the intersection curve.

The rest of the paper is organized in the following manner. In §2 we present some background material on surface intersection and review the matrix determinant representation of intersection curves. In §3, we characterize the intersection curve in terms of singular sets of matrix polynomials and characterize the accuracy of various matrix operations performed on these polynomials. §4 describes our tracing algorithm, including choice of step size and the use of power iterations to march along the curve. We also describe how the structure of the matrix formulation can be exploited when performing the inverse power iterations. Finally, in §5 we discuss our implementation and its application to surface intersection and the computation of superbolas

for toolpath generation.

§2. Background

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.

The topology of the intersection curve can be very complicated. The intersection curve may have more than one component and may have singular points, thereby adding to the *geometric complexity* of the problem. Simple cases like intersection of two cylinders can give rise to a 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 in the (x, y, z) space and degree 108 in the (u, v) domain, and it is simple to come up with cases where the intersection curve has more than one component.

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. Since a Bézier surface is a rational parametric surface, and we wish to compute the intersection of two such surfaces, we need to implicitize the parametric representation of one of the surfaces. It has been shown that if a parameterization has no base points, the resultant of the parametric equations corresponds exactly to the implicit representation. Using Dixon's formulation[4], the resultant corresponds to the determinant of a matrix.

Thus, given two Bézier surfaces $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$ and $\mathbf{G}(u,v) = (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v))$, we can implicitize $\mathbf{F}(s,t)$ into an algebraic surface of the form $f(x,y,z,w) = 0$, where

$$f(x,y,z,w) = \det \begin{pmatrix} f_{11}(x,y,z,w) & \dots & f_{1n}(x,y,z,w) \\ f_{21}(x,y,z,w) & \dots & f_{2n}(x,y,z,w) \\ \vdots & \ddots & \vdots \\ f_{n1}(x,y,z,w) & \dots & f_{nn}(x,y,z,w) \end{pmatrix}.$$

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

$$f(\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)) = \begin{pmatrix} g_{11}(u,v) & \dots & g_{1n}(u,v) \\ g_{21}(u,v) & \dots & g_{2n}(u,v) \\ \vdots & \ddots & \vdots \\ g_{n1}(u,v) & \dots & g_{nn}(u,v)) \end{pmatrix},$$

where

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

Let $D(u,v)$ be the polynomial corresponding to $\det(M(u,v))$. Therefore, the algebraic plane curve corresponds to $D(u,v) = 0$.

§3. Properties of the Matrix Representation

Let the degree of a tensor product Bézier patch $\mathbf{F}(s,t)$ be m_f in t and n_f in s. Similarly, let the degree of a tensor product Bézier patch $\mathbf{G}(u,v)$ be m_g in u and n_g in v. The order of the matrix arising from the implicitization of $\mathbf{F}(s,t)$ by using Dixon's Resultant will be $2m_f n_f$. After substituting the parameterization of $\mathbf{G}(u,v)$ into the implicit representation of $\mathbf{F}(s,t)$, we get a bivariate matrix polynomial $M(u,v)$. Given $v = v_i$, we can write $M(u,v_i)$ as a univariate matrix polynomial of degree m_g in u:

$$M(u,v_i) = M_a u^a + M_{a-1} u^{a-1}(1-u) + \ldots + M_0(1-u)^a$$

where $a = m_g$, and each of the matrices M_j have order $b = 2m_f n_f$. After dividing this formulation by $(1-u)^a$ and re-parameterizing by $w = \frac{u}{1-u}$, we get a matrix polynomial of the form

$$M'(w) = M_a w^a + M_{a-1} w^{a-1} + \ldots + M_0.$$

When M_a is a non-singular and well conditioned matrix, the roots of the matrix polynomial $M'(w)$ are given by the eigenvalues of the companion matrix [11]:

$$C(w) = \begin{pmatrix} \mathbf{0_b} & \mathbf{I_b} & \mathbf{0_b} & \ldots & \mathbf{0_b} \\ \mathbf{0_b} & \mathbf{0_b} & \mathbf{I_b} & \ldots & \mathbf{0_b} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0_b} & \mathbf{0_b} & \mathbf{0_b} & \ldots & \mathbf{I_b} \\ -\overline{M}_0 & -\overline{M}_1 & -\overline{M}_2 & \ldots & -\overline{M}_{a-1} \end{pmatrix},$$

where $\mathbf{0_b}$ and $\mathbf{I_b}$ are null and identity matrices respectively, each of order $b = 2m_f n_f$, and $\overline{M}_i = M_a^{-1} M_i, 0 \leq i < a$. When M_a is singular or close to being singular, the roots of the matrix polynomial $M'(w)$ are given by the eigenvalues of the generalized eigensystem [11].

We utilize algorithms which rely on computing eigenvalues and eigenvectors of matrices because eigenvalue and eigenvector computations are backwards stable and thus provide improved accuracy. To find all of the roots of the matrix polynomial we can perform a full eigendecomposition on the companion matrix, $C(w)$, using the QR algorithm [6]. More importantly, if we are interested in only one of the roots of the matrix polynomial $M'(w)$, and if we are given a value λ' close to that root, we can perform inverse power iterations to quickly and robustly converge to the actual root λ [6]. [11] also points out that the corresponding eigenvector is of the form

$$[\ \mathbf{V}, \lambda\mathbf{V}, \lambda^2\mathbf{V}, \ldots, \lambda^{m_g-1}\mathbf{V}\ ],$$

where

$$\mathbf{V} = [\ 1, s, s^2, \ldots, s^{m_f-1}, t, st, \ldots, s^{m_f-1}t^{2n_f-1}\ ].$$

Finally, using a method described in [11], we can compute the partial derivatives of $M(u,v)$ at a given point (u_i, v_i). This computation proceeds by a modified Gaussian elimination method, and the accuracy of the partial derivative calculation is approximately the same as that of the standard Gaussian elimination process.

For further details on the properties of this matrix representation, the interested reader can refer to [11] as a starting point.

§4. Tracing Algorithm

Our aim has been to develop a tracing algorithm that satisfies to a reasonable extent the conflicting goals of accuracy, efficiency, and robustness. To this end, we have designed an algorithm based on a robust formulation which relies on accurate matrix operations and takes advantage of the special structure of the problem and the matrices involved for efficiency.

The algorithm proceeds as follows:

(1) Given two tensor product Bézier surfaces $\mathbf{F}$ and $\mathbf{G}$, we first generate the matrix representation $M(u,v)$ as described in §2.

(2) Given a starting point (u_i, v_i) on the intersection curve, we find the tangent $(\delta u_i, \delta v_i)$, in the uv space, to the curve at (u_i, v_i).

(3) Given a step size $\sigma(u_i, v_i)$, and assuming $\delta v_i = \max(\delta u_i, \delta v_i)$ [1], we move to (u_i, v_{i+1}), where $v_{i+1} = v_i + \sigma(u_i, v_i)\delta v_i$.

(4) To compute u_{i+1}, we form the companion matrix $C(w)$ (where $w = \frac{u}{1-u}$, as in §3) of the matrix polynomial $M(u, v_{i+1})$, and then perform inverse power iterations on $C(w)$ to find the nearest eigenvalue λ_i to u_i. As discussed in §2, $D(\lambda_i, v_{i+1}) = 0$. Thus, $u_{i+1} = \lambda_i$ and we return to (2), with (u_{i+1}, v_{i+1}) as our starting point.

To find the tangent $(\delta u_i, \delta v_i)$ at (u_i, v_i), we compute $\delta u_i = D^u(u_i, v_i)$ and $\delta v_i = D^v(u_i, v_i)$ as mentioned in §3. The step size $\sigma(u_i, v_i)$ can be adaptively based upon the higher order derivatives of the intersection curve in the uv-space. Higher order derivatives such as D^{uu}, D^{vv}, etc., can be computed in an analogous fashion to the computation of D^u and D^v. Such derivatives can be used to formulate higher order approximants for use as $(\delta u_i, \delta v_i)$. The decision on the number of higher order derivatives to be computed is a trade-off between robustness and efficiency.

Inverse power iterations normally require that we perform a full LU decomposition of the matrix involved, which in this case is the companion matrix of order ab, where $a = m_g$ and $b = 2m_f n_f$. LU decomposition for such a matrix would normally require $\mathcal{O}(a^3b^3)$ operations. However, by taking advantage of the sparse structure of the matrix, we can reduce this to $\mathcal{O}(ab^3)$ operations. If $\mathbf{F}$ and $\mathbf{G}$ are tensor product bicubic patches, this provides a speedup by a factor of about 9.

[1] The case where $\delta u_i = \max(\delta u_i, \delta v_i)$ proceeds similarly.

When computing the intersection of two parametric surfaces like $\mathbf{F}$ and $\mathbf{G}$, we are often only interested in the region where $0 \leq u, v, s, t \leq 1$. Since we are tracing in the uv-space, it is obvious when we have left the region of interest for the patch $\mathbf{G}(u, v)$. To find out when whether we still lie in the region $0 \leq s, t \leq 1$, we can use the special structure of the eigenvector corresponding to the eigenvalue λ_i (as noted in §3) to derive (s_{i+1}, t_{i+1}).

The tracing algorithm also provides a simple method by which we can compute the starting points for the open components of the intersection curve. If the patch boundaries are $0 \leq u, v, s, t \leq 1$, then we can find the starting points along the boundary $v = 0$ simply by forming the companion matrix corresponding to $M(u, 0)$ and performing a full eigendecomposition. The real eigenvalues in the interval $[0, 1]$ correspond to starting points for tracing the open components of the intersection curve which intersect $v = 0$. This can be repeated for $u, s, t = 0$ and $u, v, s, t = 1$. The closed loops can be found by tracing paths in the complex space [12].

§5. Implementation and Results

We have implemented the tracing algorithm described in §4 in C on a variety of platforms using double precision calls to the Fortran libraries LAPACK and BLAS. A simple window interface has been set up in which we can observe the tracing progressing in the uv, st, and xyz spaces at the same time. We take two tensor product Bézier patches, calculate Dixon's resultant to implicitize one, and then substitute the other into the resultant to generate $M(u, v)$. We currently only trace the open components for which we obtain starting points by the process described at the end of §4. Our code uses a fixed step size σ and the first order partials of $D(u, v)$ to form the local approximant. We also take full advantage of the sparse structure of the companion matrix when performing the inverse power iterations. To get a rough feeling for the performance of the algorithm, we ran it on an SGI Onyx[1] and came up with the following average timing figures[2] for tracing the intersection curve of two bicubic tensor product Bézier patches:

Implicitization of one surface:	0.089 secs
Full eigendecomposition to find starting points on one edge:	0.149 secs
Tracing one point of a path using inverse power iterations:	0.072 secs

The tracing algorithm is really just a method for tracing a one dimensional algebraic set which is represented in our matrix formulation, so for a sample test of the robustness we set up the matrix so it would trace the degree 20 superbola described in a toolpath generation example in [5]. Field and Field found that conventional tracing and marching techniques failed to

[1] Our Onyx was configured with the default of two processors, but we did not parallelize the algorithm. Also, during our test runs the load of the networked machine (excluding our process) was close to zero.

[2] Timing figures were obtained using the 'gettimeofday' call.

properly follow the curve, but when using the matrix formulation our tracing algorithm encountered no problems in generating the superbola.

While implementing this tracing algorithm, we came upon a number of ideas which could be used to improve and augment the tracer.

First, there is an obvious coarse grain parallel structure to the algorithm, and this could easily be exploited on a MIMD or SIMD architecture. Given n starting points, we could distribute one starting point and copies of $M(u,v)$, $F(s,t)$, and $G(u,v)$ to each of a set of n processors, and they could trace out a set of components in parallel.

One of the major reasons for tracing in the lower dimensional space is that we have reduced the *geometric complexity* of the problem. Performing any kind of tracing method in higher dimensional spaces is conceptually much more difficult than performing a similar algorithm in the plane. Furthermore, there has been a fair amount of work done on the resolution of singularities in a plane curve [7], and we can reason much more easily about singularities on a plane than about singularities in higher dimensional spaces. Based on the matrix formulation, we can detect singularities on paths as we trace out the curve and use branching computations to resolve them.

Other Applications and Future Work: The algorithm presented above is applicable to all geometric applications related to the computation of one dimensional algebraic sets. These include silhouette computations, offsets and blends, Voronoi curves, etc. This algorithm, combined with robust techniques for finding loops and all branches around a singular point, is part of a system being developed for performing CSG operations on spline surfaces.

References

1. Bajaj, C. L., C. M. Hoffmann, J. E. H. Hopcroft, and R. E. Lynch, Tracing surface intersections, Computer Aided Geometric Design **5** (1988), 285–307.
2. Barnhill, R., G. Farin, M. Jordan, and B. Piper, Surface/surface intersection, Computer Aided Geometric Design **4** (1987), 3–16.
3. Barnhill, R. E., and S. N. Kersey, A marching method for parametric surface/surface intersection, Computer Aided Geometric Design **7** (1990), 257–280.
4. Dixon, A. L., The eliminant of three quantics in two independent variables, Proceedings of the London Mathematical Society **6** (1908), 49–69, 209–236.
5. Field, D. A., and R. Field, A new family of curves for industrial applications, Technical Report GMR-7571, General Motors Research Laboratories, 1992.
6. Golub, G. H., and C. F. Van Loan, *Matrix Computations*, John Hopkins Press, Baltimore, 1989.
7. Hoffman, C. M., *Geometric and Solid Modeling*, Morgan Kaufmann, San Mateo, California, 1989.

8. Hoffman, C. M., A dimensionality paradigm for surface interrogations, Computer Aided Geometric Design **7** (1990), 517–532.
9. Kriezis, G. A., P. V. Prakash, and N. M. Patrikalakis, Method for intersecting algebraic surfaces with rational polynomial patches, Computer-Aided Design **22**(10) (1990), 645–654.
10. Manocha, D., Algebraic and numeric techniques for modeling and robotics, dissertation, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1992.
11. Manocha, D., and J. F. Canny, A new approach for surface intersection, International Journal of Computational Geometry and Applications **1**(4) (1991), 491–516.
12. Manocha, D. and S. Krishnan, Robust and Efficient Surface Intersections, Unpublished Manuscript.
13. Pratt, M. J., Surface/surface intersection problems, in *The Mathematics of Surfaces II*, J.A. Gregory (ed.), Claredon Press, Oxford, 1986, pages 117–142.
14. Sederberg, T. W., Implicit and parametric curves and surfaces, dissertation, Purdue University, 1983.
15. Sederberg, T. W., and T. Nishita, Geometric hermite approximation of surface patch intersection curves, Computer Aided Geometric Design **8** (1991), 97–114.

Acknowledgments. Dinesh Manocha is supported in part by a Junior Faculty Award and ARPA Contract #DAEA 18-90-C-0044. Amitabh Varshney is supported under the NIH NCRR Grant #5-P41-RR02170. Hans Weber is supported under ARPA Contract #DAEA 18-90-C-0044.

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