Algebraic Pruning: A Fast Technique for Curve and Surface Intersection *

Dinesh Manocha
manocha@cs.unc.edu

Shankar Krishnan
krishnas@cs.unc.edu

Department of Computer Science,
University of North Carolina,
Chapel Hill, NC 27599-3175

Abstract: Computing the intersection of parametric and algebraic curves and surfaces is a fundamental problem in computer graphics and geometric modeling. This problem has been extensively studied in the literature and different techniques based on subdivision, interval analysis and algebraic formulation are known. For low degree curves and surfaces algebraic methods are considered to be the fastest, whereas techniques based on subdivision and Bézier clipping perform better for higher degree intersections. In this paper, we introduce a new technique of algebraic pruning based on the algebraic approaches and eigenvalue formulation of the problem. The resulting algorithm corresponds to only computing the selected eigenvalues in the domain of intersection. This is based on matrix formulation of the intersection problem, power iterations and geometric properties of Bézier curves and surfaces. The algorithm prunes the domain and converges to the solutions rapidly. It has been applied to intersection of parametric and algebraic curves, ray tracing and curve-surface intersections. The resulting algorithm compares favorably with earlier methods in terms of performance and accuracy.
Keywords: Intersection, curves, surfaces, ray-tracing, resultants, eigendecomposition, solid modeling

1 Introduction

The problems of computing the intersection of curves and surfaces are fundamental in computer graphics and geometric modeling. Common applications include computation of boundary representation from a CSG model, ray-tracing surfaces, hidden-curve removal and visibility applications [Hof89, EC90, NSK90, SP86]. In this paper, we consider only finite dimensional intersections like curve-curve intersection, curve-surface intersection and ray-surface intersection.

*Supported in part by Sloan Foundation, university research council grant, NSF grant CCR-9319957, ONR contract N00014-94-1-0738, ARPA contract DABT63-93-C-0048, NSF/ARPA Science and Technology Center for Computer Graphics & Scientific Visualization and NSF Prime contract No. 8920219

There are three major approaches for computing these intersections based on subdivision, interval arithmetic and algebraic methods. Subdivision based approach is based on the geometric properties of curve and surface representations [LR80]. Given two Bézier curves, the intersection algorithm proceeds by comparing the convex hulls of their control polytopes. If they do not overlap, the curves or surfaces do not intersect. Otherwise the curves are subdivided and the resulting convex hulls are checked for intersection. At each iteration the algorithm rejects regions of the curve that do not contain any intersection point. Eventually, the curve segments are approximated by straight lines up to certain tolerance, their intersection point is accepted as the intersection of two curves. Simple subdivision algorithm has linear convergence in the domain. Its convergence is being improved by Bézier clipping in [SWZ89, Sed89, NSK90]. In particular, Bézier clipping uses the convex hull property in a powerful way, by determining parameter ranges which are guaranteed to not include points of intersection. The interval arithmetic approach is similar to subdivision [KM83]. The curve is divided into intervals using vertical and horizontal tangents and thereby computing rectangular bounding boxes. The subdivision amounts to evaluating the coordinate of the midpoint of the interval and defining the resulting rectangles.

Algebraic methods formulate the intersection problem in terms of solutions of a system of algebraic equations. Given the equations, the variables are eliminated using techniques from elimination theory [Sal85] and the problem is reduced to finding roots of a univariate polynomial. This approach was applied to ray-tracing by Kajiya [Kaj82] and to curve intersections by Sederberg [Sed83]. For lower degree curve intersection (up to degree three or four), implicitization approach results in the fastest algorithms. However, the problem of finding roots of higher degree polynomials can be numerically unstable [Wil59]. Therefore, the overall algorithm for intersection may not be accurate. Moreover, the symbolic expansion of determinants to compute resultants can be computationally expensive as well [Hof90]. To circumvent these problems, [Man92, MD92] have proposed methods combining elimination theory with matrix computations. The resulting problem is reduced to computing the eigenvalues of a matrix as opposed to roots of a polynomial. Eigenvalue algorithms like the QR algorithm [GL89] are backward stable and as a result the intersections can be computed accurately and efficiently for high degree curves and surfaces. However, the QR algorithm computes all the eigenvalues of the matrix. For intersection problems we are only interested in the solutions that lie in the domain of the curves and surfaces. For example, the intersection of a cubic curve and bicubic patch results in a 54×54 matrix and typically there are three or four solutions in the domain of interest.

In this paper, we introduce the technique of *algebraic pruning*. This is based on eigenvalue formulation of the intersection problems. In particular, we only compute the solutions in the domain of interest by making use of properties of Bézier curves and surfaces, structure of the matrix and inverse power iteration. The overall algorithm converges to intersections in the domain and at the same time prunes out portions of the domain containing no intersections. Its convergence and performance is a function of number of intersections in the domain. In the local neighborhood we obtain almost *cubic convergence*. Although our formulation is algebraic, the overall algorithm behaves in a manner similar to Bézier clipping. It works very well on intersection problems consisting of a few intersections only and in practice compares favorably with other approaches.

The rest of the paper is organized in the following manner. We review the intersection problem and eigenvalue formulation in Section 2 and the matrix computations used in the algorithm in Section 3. In Section 4, we present the pruning approach and how it is combined with the matrix structure and properties of Bézier curves and surfaces to formulate the algorithm. Finally, we discuss its implementation and performance in Section 5.

2 Curve and Surface Intersections

In this paper, we only consider the intersections of rational parametric and algebraic curves and surfaces. These include Bézier curves and surfaces, NURBS, quadric patches etc. A rational Bézier curve is represented as [Far90]:

$$\mathbf{P}(t) = (X(t), Y(t)) = \frac{\Sigma_{i=0}^{n} w_i \mathbf{P}_i B_{i,n}(t)}{\Sigma_{i=0}^{n} w_i B_{i,n}(t)}, \qquad 0 \leq t \leq 1$$

where $\mathbf{P}_i = (X_i, Y_i)$ are the coordinates of a control point, w_i is the weight of the control point and $B_{i,n}(t)$ corresponds to the Bernstein polynomial

$$B_{i,n}(t) = \begin{pmatrix} n \\ i \end{pmatrix} (1-t)^{n-i} t^i.$$

Other rational formulations like B-splines can be converted into a series of rational Bézier curves by knot insertion algorithms [Far90]. Algebraic plane curves (of degree n) are generally expressed in standard power basis:

$$F(x, y) = \Sigma_{i+j \leq n} c_{ij} x^i y^j = 0.$$

They can also be represented in Bernstein basis. The problem of intersection corresponds to computing the common points on such curves in a particular domain.

The parametric surfaces may correspond to tensor product Bézier patches or triangular Bézier patches. Tensor product patches are represented in homogeneous coordinates as [Far90]:

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

where $\mathbf{V}_{ij} = (x_{ij}, y_{ij}, z_{ij}, w_{ij})$. The domain of the surface is restricted to $s \in [0,1], \;\; t \in [0,1]$. The triangular Bézier patches are represented as [Far90]:

$$\mathbf{F}(s, t) = (X(s,t), Y(s,t), Z(s,t), W(s,t)) = (\sum_{i=0}^{n} \sum_{j=0}^{n-i} \mathbf{V}_{i,j} B_{i,j}^{n}(s,t)),$$

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

$$B_{i,j}^{n}(s,t) = \begin{pmatrix} n \\ i, j \end{pmatrix} s^i t^j (1-s-t)^{n-i-j}, \quad \begin{pmatrix} n \\ i, j \end{pmatrix} = \frac{n!}{i! j! (n-i-j)!}.$$

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

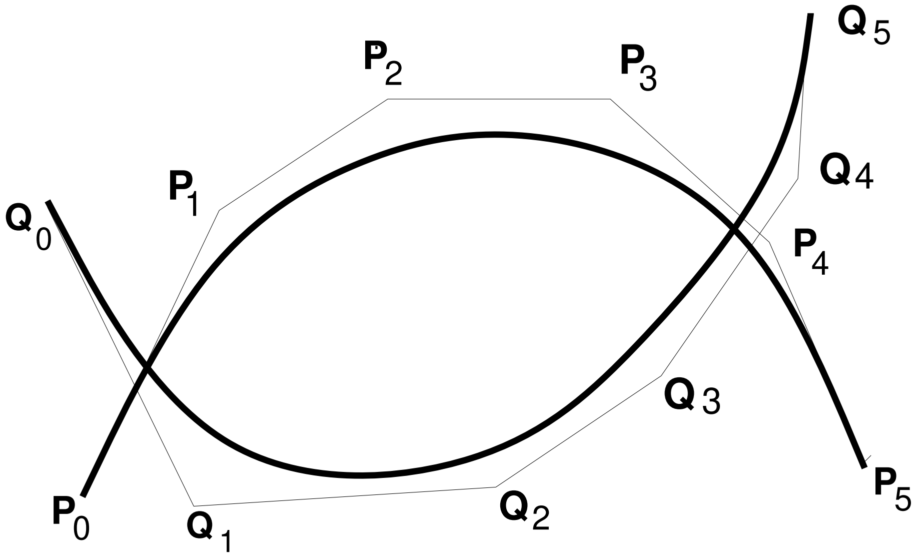

Figure 1: Intersection of Bézier curves

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. However, this transformation can introduce numerical problems [FR87]. To circumvent this problem we perform a reparametrization of the form

$$\overline{s} = g(s) = \frac{s}{(1-s)}, \quad \overline{t} = g(t) = \frac{t}{(1-t)},$$

for tensor product surfaces. In the resulting formulation we substitute $\overline{s} = \frac{s}{1-s}, \quad \overline{t} = \frac{t}{1-t}$ and the resulting parametrizations are in power basis in terms of $\overline{s}$ and $\overline{t}$. The domain of the surfaces is suitably transformed. For triangular patches the reparametrization is of the form $\overline{s} = g(s,t) = \frac{s}{1-s-t}, \quad \overline{t} = h(s,t) = \frac{t}{1-s-t}$.

2.1 Intersection Problems and Algebraic Formulation

The problem of intersection can always be reduced to solving a system of algebraic equations. For example, given the homogeneous representation of two rational Bézier curves, $\mathbf{P}(s) = (X(s), Y(s), W(s)))$ and $\mathbf{Q}(t) = (\overline{X}(t), \overline{Y}(t), \overline{W}(t)))$, the problem of intersection corresponds to computing all the common solutions of

$$\begin{aligned} X(s)\overline{W}(t) - \overline{X}(t)W(s) &= 0 \\ Y(s)\overline{W}(t) - \overline{Y}(t)W(s) &= 0 \end{aligned} \tag{1}$$

in the domain $(s,t) \in [0,1] \times [0,1]$.

Given a Bézier surface $\mathbf{F}(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t))$, its intersections with a ray represented as the intersection of two planes

$$a_1 X + b_1 Y + c_1 Z + d_1 = 0$$

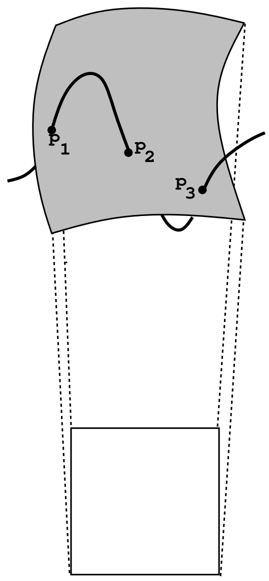

Figure 2: Intersection of a Bézier curve and surface

and

$$a_2X + b_2Y + c_2Z + d_2 = 0$$

can be reduced to the solutions of

$$\begin{aligned} a_1X(s,t) + b_1Y(s,t) + c_1Z(s,t) + d_1W(s,t) &= 0 \\ a_2X(s,t) + b_2Y(s,t) + c_2Z(s,t) + d_2W(s,t) &= 0 \end{aligned} \tag{2}$$

in the domain $(s,t) \in [0,1] \times [0,1]$.

Given a Bézier space curve, $\mathbf{P}(u) = (\overline{X}(u), \overline{Y}(u), \overline{Z}(u), \overline{W}(u))$, its intersections with the Bézier surface can be formulated as all solutions of

$$\begin{aligned} X(s,t)\overline{W}(u) - \overline{X}(u)W(s,t) &= 0 \\ Y(s,t)\overline{W}(u) - \overline{Y}(u)W(s,t) &= 0 \\ Z(s,t)\overline{W}(u) - \overline{Z}(u)W(s,t) &= 0 \end{aligned} \tag{3}$$

in the domain $(s,t,u) \in [0,1] \times [0,1] \times [0,1]$.

2.2 Reduction to Eigenvalue Formulation

Given a system of equations, we eliminate variables using resultants. In intersection problems, we obtain systems consisting of two or three algebraic equations. For two equations corresponding to curve-curve intersection and ray-tracing we use Bezout's formulation of Cayley resultant or Sylvester resultant [Sal85]. For curve-surface intersections, we use Dixon's formulation [Dix08]. In either case the resultant can be expressed as a matrix determinant and the entries of the matrix are univariate polynomials. Instead of symbolically expanding the determinant, we reduce the problem to an eigenvalue formulation [Man92]. In particular, the resultant corresponds to a matrix polynomial of the form:

$$\mathbf{M}(s) = \mathbf{M}_n s^n + \mathbf{M}_{n-1} s^{n-1}(1-s) + \mathbf{M}_{n-2} s^{n-2}(1-s)^2 + \ldots + \mathbf{M}_0 (1-s)^n,$$

where $\mathbf{M}_i$ is an $m \times m$ matrix with numeric entries. m and n are function of the degree of the curves or surfaces. On dividing the matrix polynomial by $(1-s)^n$ and substituting $u = \frac{s}{1-s}$, we obtain a matrix polynomial of the form:

$$\mathbf{L}(u) = \mathbf{M}_n u^n + \mathbf{M}_{n-1} u^{n-1} + \ldots + \mathbf{M}_0. \tag{4}$$

The intersection algorithm corresponds to computing the roots of $Determinant(\mathbf{L}(u)) = 0$. They are computed by reducing to an eigenvalue problem in the following manner [Man92]:

Theorem 2.1 *Given the matrix polynomial,* $\mathbf{L}(u)$ *the roots of the polynomial corresponding to its determinant are the eigenvalues of the generalized system* $\mathbf{C}_1 u + \mathbf{C}_2$*, where*

$$\mathbf{C}_1 = \begin{bmatrix} \mathbf{I}_m & \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m & \mathbf{0} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{I}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{0} & \mathbf{M}_n \end{bmatrix} \quad \mathbf{C}_2 = \begin{bmatrix} \mathbf{0} & -\mathbf{I}_m & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I}_m & \ldots & \mathbf{0} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ldots & -\mathbf{I}_m \\ \mathbf{M}_0 & \mathbf{M}_1 & \mathbf{M}_2 & \ldots & \mathbf{M}_{n-1} \end{bmatrix}, \tag{5}$$

where $\mathbf{0}$ *and* $\mathbf{I}_m$ *are* $m \times m$ *null and identity matrices, respectively.*

Based on the problem formulation and properties of resultants, it follows that the eigenvalues of $C_1 u + C_2$ correspond to one of the unknowns in (1), (2) or (3). The other variables can be recovered from the corresponding eigenvectors. For most intersection applications, we are interested in computing the eigenvalues in a finite subset of the real domain. For example, for Bézier curves and surfaces, the domain is $s \in [0, 1]$. However, the variable u in $\mathbf{L}(u)$ takes values in the interval $[0, \infty]$. To avoid this problem, we back-substitute $u = \frac{s}{1-s}$ and transform the matrix pencil $C_1 u + C_2$ to $(\mathbf{C}_1 - \mathbf{C}_2)s + \mathbf{C}_2$. For the rest of the paper, we assume that this transformation has been performed and shall concentrate in computing all the eigenvalues of a matrix pencil in a finite domain.

3 Matrix Computations

Given an $n \times n$ matrix $\mathbf{A}$, its eigenvalues and eigenvectors are the solutions to the equation

$$\mathbf{A}\mathbf{x} = s\mathbf{x},$$

where s is the eigenvalue and $\mathbf{x} \neq \mathbf{0}$ is the eigenvector. The eigenvalues of a matrix are the roots of its characteristic polynomial, corresponding to Determinant$(\mathbf{A} - s\mathbf{I})$. Given $n \times n$ matrices, $\mathbf{A}$ and $\mathbf{B}$, the generalized eigenvalue problem corresponds to solving

$$\mathbf{A}\mathbf{x} = s\mathbf{B}\mathbf{x}.$$

We represent this problem as eigenvalues of $\mathbf{A} - s\mathbf{B}$. The vectors $\mathbf{x} \neq \mathbf{0}$ correspond to the eigenvectors of this equation. If $\mathbf{B}$ is non–singular and its condition number is low, the problem can be reduced to an eigenvalue problem by multiplying both sides of the equation by $\mathbf{B}^{-1}$ and thereby obtaining:

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = s\mathbf{x}.$$

However, $\mathbf{B}$ may have a high condition number and such a reduction may be numerically unstable. Standard algorithms for computing eigenvalues like the QR algorithm for the standard eigenvalue problem and QZ algorithm for the generalized eigenvalue problem are based on orthogonal similarity transformations [GL89]. However, they compute all the eigenvalues and it is difficult to restrict them to eigenvalues in the given domain. But in our applications we are only interested in finding intersections that lie inside the given domain.

3.1 Power Iterations

Power iterations are a fundamental technique to compute the eigenvalues and eigenvectors of a matrix. Given a diagonalizable matrix, $\mathbf{A}$, such that $\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = Diag.(\lambda_1, \lambda_2, \ldots, \lambda_n)$ with $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ and $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$. Given a unit vector $\mathbf{q}_0$, the *power method* produces a sequence of vectors $\mathbf{q}_k$ as follows:

$$\textbf{for } k = 1, 2, \ldots$$

$$\mathbf{z}_k = \mathbf{A}\mathbf{q}_{k-1}$$

$$\mathbf{q}_k = \mathbf{z}_k / \parallel \mathbf{z}_k \parallel_\infty$$

$$s_k = \mathbf{q}_k^T A \mathbf{q}_k$$

$$\textbf{end}$$

where $\parallel \mathbf{z}_k \parallel_\infty$ refers to the element of maximum magnitude in the vector $\mathbf{z}_k$. It is well known that in the power method, s_k converges to λ_1 and $\mathbf{q}_k$ converges of $\mathbf{x}_1$. Moreover the convergence is a function of the ratio $|\lambda_1|/|\lambda_2|$. λ_1 is the dominant eigenvalue of $\mathbf{A}$. Power method is described in detail in [GL89, Wil65]. In case, $\mathbf{A}$ is not diagonalizable or the corresponding eigenvector is ill-conditioned, the accuracy and convergence of the method is discussed in [Wil65] as well.

In our applications, we use power iterations to compute the smallest eigenvalues of matrix pencils of the form, $\mathbf{C}_1 s' + \mathbf{C}_2$. The smallest eigenvalue of $\mathbf{C}_1 s' + \mathbf{C}_2$ corresponds to the largest eigenvalues of $(\mathbf{C}_1 s' + \mathbf{C}_2)^{-1}$. As opposed to computing the inverse explicitly, the resulting algorithm of inverse power iteration is (given $\mathbf{q}_0$):

$$\textbf{for } k = 1, 2, \ldots$$

$$Solve\ (\mathbf{C}_1 s' + \mathbf{C}_2)\mathbf{z}_k = \mathbf{C}_1 \mathbf{q}_{k-1}$$

$$\mathbf{q}_k = \mathbf{z}_k / \parallel \mathbf{z}_k \parallel_\infty$$

$$s_k = -(\mathbf{q}_k^T \mathbf{C_2} \mathbf{q}_k)/(\mathbf{q}_k^T \mathbf{C_1} \mathbf{q}_k)$$

end

where $\parallel \mathbf{z}_k \parallel_\infty$ refers to the element of maximum magnitude of $\mathbf{z}_k$. To efficiently solve the system, we compute the LU decomposition of $\mathbf{C}_1 s' + \mathbf{C}_2$ using Gaussian elimination. Furthermore, the vector $\mathbf{q}_0$ is chosen randomly. Given the RHS vector, $\mathbf{C}_1 \mathbf{q}_{k-1}$, the resulting linear system can be solved in $O(n^2)$ steps by solving lower triangular and upper triangular systems. The convergence of the inverse power iteration is a function of the starting guess s' and the distance of eigenvalues from s'. In particular, let the eigenvalues of $\mathbf{C}_1 s' + \mathbf{C}_2$ be $\lambda_1, \lambda_2, \ldots, \lambda_n$ in an order such that:

$$|s' - \lambda_1| \leq |s' - \lambda_2| \leq, \ldots, \leq |s' - \lambda_n|.$$

The convergence is a function of $|s' - \lambda_1|/|s' - \lambda_2|$. If two of the eigenvalues, λ_1 and λ_2, are almost at the same distance from s', the overall convergence can be fairly slow. In such cases, the convergence can be further improved using the following procedure (given $\mathbf{q_0}$ and $\mathbf{u}_0$):

for $k = 1, 2, \ldots$

Solve $(\mathbf{C}_1 s' + \mathbf{C}_2)\mathbf{z}_k = \mathbf{C}_1 \mathbf{q}_{k-1}$

Solve $(\mathbf{C}_1 s' + \mathbf{C}_2)^T \mathbf{v}_k = \mathbf{C}_1 \mathbf{u}_{k-1}$

$$\mathbf{q}_k = \mathbf{z}_k / \parallel \mathbf{z}_k \parallel_\infty$$

$$\mathbf{u}_k = \mathbf{v}_k / \parallel \mathbf{v}_k \parallel_\infty$$

$$s_k = -(\mathbf{u}_k^T \mathbf{C_2} \mathbf{q}_k)/(\mathbf{u}_k^T \mathbf{C_1} \mathbf{q}_k)$$

end

This process is locally cubically convergent [Wil65]. Many other techniques for improving the accuracy and convergence of the algorithm in the presence of higher multiplicity eigenvalues, closely spaced eigenvalues are presented in [Wil65].

4 Algebraic Pruning

We have shown earlier how the intersection problem can be reduced to finding eigenvalues of the matrix pencil, $\mathbf{C}_1 s + \mathbf{C}_2$. To start with we use linear programming [Sei90] to check if the control

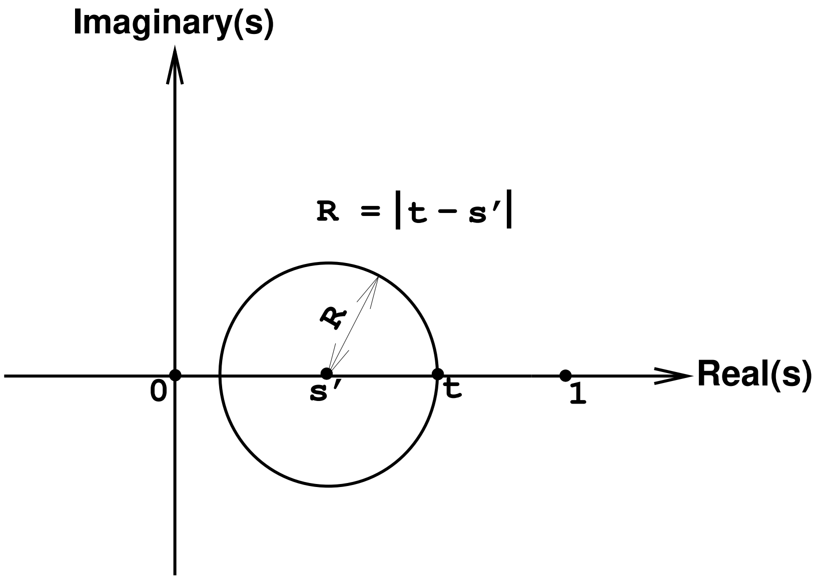

Figure 3: Domain Pruning based on Inverse Iteration

polytopes of the pairs of curves (or a curve and a surface) have a separating line (or plane) between them. If they do, then the given pair does not intersect. Otherwise, there is probably an eigenvalue of the pencil $\mathbf{C}_1 s + \mathbf{C}_2$ close to the domain $[0, 1]$. This includes complex eigenvalues as well.

The main idea behind our algorithm is to use inverse power iterations to find some eigenvalues in the domain, and at the same time prune out portions of the domain not containing any solution. In particular, we start with a guess $s^{'} \approx 0.5$, which corresponds to the midpoint of the domain. Using inverse power iteration, we find the eigenvalue closest to $s^{'}$. Let that eigenvalue be t. It is possible that t is a complex number and in that case we compute the complex conjugate pair of eigenvalues. Assuming that we chose random start vectors, $\mathbf{q}_0$ and $\mathbf{u}_0$, t is an eigenvalue of $\mathbf{C}_1 s + \mathbf{C}_2$ which is closest to $s^{'}$. As a result, there are no other eigenvalues of the pencil in the circle centered at $s = s^{'}$ and has radius $R = |t - s^{'}|$ (as shown in Fig. 3).

Based on the convergence of power iteration we make the following conclusions:

- If $t \in [0, 1]$, t corresponds to an intersection point. The rest of the unknowns are computed from the corresponding eigenvector.
- There are no other intersections in the real domain, $(s^{'} - R, s^{'} + R)$.
- The technique is recursively applied to find all the intersections in the following domains:
  - $[0, s^{'} - R]$, if $(s^{'} - R) \geq 0$.
  - $[s^{'} + R, 1]$, if $(s^{'} + R) \leq 1$.

Therefore, based on power iterations we are able to compute an intersection and prune the domain. The algorithm is applied recursively to each domain after pruning. Typically there are very few intersections in the domain. In those cases, the algorithm converges to the intersection fast and we need to apply this technique only a few times.

4.1 Computation of Multiple Solutions

In the previous section, we highlighted the technique of algebraic pruning based on inverse power iteration. Many a times the power iteration may not converge to any real solution t or the convergence may be slow. The former can be due to the fact that the closest eigenvalue corresponds to a pair of complex conjugate eigenvalues. The latter is due to the fact that there are two or more real eigenvalues which have roughly the same distance from $s^{'}$. In this section, we modify the inverse power iteration to compute more than one intersection point at the same time. This will include complex conjugate pairs as well. We highlight the technique to compute two solutions at the same time and it can be easily extended to find more than two solutions.

Given the approximation, $s = s^{'}$, let the two closest eigenvalues be t_1 and t_2. Let the corresponding eigenvectors be $\mathbf{x}_1$ and $\mathbf{x}_2$ and $\mathbf{A} = (\mathbf{C}_1 s^{'} + \mathbf{C}_2)^{-1}$. Based on the formulation, $\frac{1}{t_1}$ and $\frac{1}{t_2}$ are the largest eigenvalues of $\mathbf{A}$. As a result, if we start with a random unit vector $\mathbf{u}_0$ and solve for p and q such that

$$\lim_{s \to \infty} (\mathbf{A}^{s+2} - p\mathbf{A}^{s+1} - q\mathbf{A}^{s})\mathbf{u}_0 = 0,$$

then $\frac{1}{t_1}$ and $\frac{1}{t_2}$ are the solutions to the equation $\lambda^2 - p\lambda - q = 0$. The two closest eigenvalues to the given guess are real or complex depends on the sign of $(p^2 + 4q)$. The algorithm is iterative and at each stage computes p and q, until they converge. We compute these multiple eigenvalues in the following manner. Let $\mathbf{u}_0$ be a random start vector.

for $k = 1, 2, \ldots$

Solve $(\mathbf{C}_1 s^{'} + \mathbf{C}_2)\mathbf{v}_k = \mathbf{C}_1 \mathbf{u}_{k-1}$

$s_k = \| \mathbf{v}_k \|_\infty$

$\mathbf{u}_k = \mathbf{v}_k / s_k$

Solve for p_k *and* q_k *from*

$$s_k s_{k-1} \begin{pmatrix} \mathbf{u}_k^T \mathbf{u}_{k-1} \\ \mathbf{u}_{k-2}^T \mathbf{u}_k \end{pmatrix} = \begin{pmatrix} \mathbf{u}_{k-1}^T \mathbf{u}_{k-1} & \mathbf{u}_{k-1}^T \mathbf{u}_{k-2} \\ \mathbf{u}_{k-2}^T \mathbf{u}_{k-1} & \mathbf{u}_{k-2}^T \mathbf{u}_{k-2} \end{pmatrix} \begin{pmatrix} p_k s_{k-1} \\ q_k \end{pmatrix}$$

end

After p_k and q_k converge, we compute the closest eigenvalues in the following manner. Let $D = p^2 + 4 * q$. If $D > 0.0$ the two closest eigenvalues are

$$t_1 = s^{'} - 2.0/(p + \sqrt{D}),$$

$$t_2 = s^{'} - 2.0/(p - \sqrt{D}).$$

Furthermore, the radius R for pruning corresponds to the minimum of $|t_1 - s^{'}|$ and $|t_2 - s^{'}|$.

In case the closest pair of eigenvalues is a complex conjugate pair, it is computed as:

$$Real(t) = s^{'} - \frac{2p}{p^2 - D},$$

$$Imag(t) = \frac{2\sqrt{-D}}{p^2 - D}.$$

and the radius R for pruning is given as

$$R = \sqrt{(Real(t) - s')^2 + Imag(t)^2)}.$$

4.2 Use of Matrix Structure

In the power iterations, the two main operations highlighted are the LU decomposition of the matrix $\mathbf{C}_1 s^{'} + \mathbf{C}_2$ and solving the resulting upper triangular systems. The matrix pencil defined in Theorem 2.1 corresponds to a pencil of order $N = m * n$. In particular, it has a block companion structure being linearized from a $m \times m$ matrix polynomial of degree n. The LU decomposition is computed using Gaussian elimination and typically it takes about $\frac{1}{3}N^3$ operations (without pivoting). Solving each triangular system costs about $\frac{1}{2}N^2$ operations. As a result, the inverse power iteration takes about $\frac{1}{3}N^3 + \frac{k}{2}N^2$ operations, where k is the number of iterations.

It turns out that the structure of the matrices can be used to reduce the number of operations for LU decomposition as well as solving triangular systems. This is based on the following properties. Given $s^{'}$, let $\mathbf{A} = \mathbf{C}_1 s^{'} + \mathbf{C}_2$. It can be shown that $\mathbf{A}$ is a matrix of the form:

$$\mathbf{A} = \begin{pmatrix} \alpha_1 \mathbf{I}_m & \alpha_2 \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \alpha_1 \mathbf{I}_m & \alpha_2 \mathbf{I}_m & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \alpha_1 \mathbf{I}_m & \alpha_2 \mathbf{I}_m \\ \mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3 & \dots & \mathbf{P}_n \end{pmatrix}$$

where α_1 and α_2 are functions of $s^{'}$. $\mathbf{P}_i$'s are $m \times m$ matrices, which are functions of $\mathbf{M}_i$'s and $s^{'}$. The LU decomposition of $\mathbf{A}$ has a structure of the form:

$$\mathbf{A} = \begin{pmatrix} \alpha_1 \mathbf{I}_m & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \alpha_1 \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \alpha_1 \mathbf{I}_m & \mathbf{0} \\ \mathbf{R}_1 & \mathbf{R}_2 & \dots & \mathbf{R}_{n-1} & \mathbf{L}_n \end{pmatrix} \begin{pmatrix} \mathbf{I}_m & \frac{\alpha_2}{\alpha_1} \mathbf{I}_m & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_m & \frac{\alpha_2}{\alpha_1} \mathbf{I}_m & \dots & \mathbf{0} \\ \vdots & \vdots & \dots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I}_m & \frac{\alpha_2}{\alpha_1} \mathbf{I}_m \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{U}_n \end{pmatrix},$$

where

$$\mathbf{R}_1 = \mathbf{P}_1$$

$$\mathbf{R}_2 = \mathbf{P}_2 - \frac{\alpha_2}{\alpha_1} \mathbf{R}_1$$

$$\mathbf{R}_3 = \mathbf{P}_3 - \frac{\alpha_2}{\alpha_1} \mathbf{R}_2$$

$$\vdots$$

$$\mathbf{R}_{n-1} = \mathbf{P}_{n-1} - \frac{\alpha_2}{\alpha_1}\mathbf{R}_{n-2}$$

$$\mathbf{R}_n = \mathbf{P}_n - \frac{\alpha_2}{\alpha_1}\mathbf{R}_{n-1}.$$

Moreover, $\mathbf{L}_n$ and $\mathbf{U}_n$ correspond to the LU decomposition of $\mathbf{R_n}$.

This formulation is constructive and based on it, LU decomposition of $\mathbf{A}$ takes $\frac{1}{3}m^3 + (n-1)m^2$ operations. Furthermore, given the LU decomposition, solving for the lower triangular system takes $(n - \frac{1}{2})m^2$ operations and solving for upper triangular system takes $\frac{1}{2}m^2 + (n-1)m$ operations. As a result, at each iteration the total number of operations for solving the linear system corresponding to inverse power iteration are $nm^2 + (n-1)m$.

4.3 Algorithm for Intersection

The algorithm for intersection computation combines algebraic pruning with properties of curves and surfaces. In particular, we assume that each curve and surface is associated with a corresponding control polytope. For Bézier curves and surfaces such control polytopes are formed by their control points. Similar geometric representations are known for algebraic curves and surfaces as well [Sed89]. The simplest algorithm for intersection is based on the version of algebraic pruning highlighted earlier in Section 4. In particular, we start with the middle point of the domain, find the closest eigenvalue using power iterations and prune the domain. The algorithm can be recursively applied to each domain obtained after pruning.

Although this algorithm works well, its performance can be improved tremendously using properties of curves and surfaces and behavior of inverse iterations. The convergence of power iterations is a function of the number of eigenvalues close to the guess s'. In case, there are a number of intersections, the ratio $|s' - \lambda_1|/|s' - \lambda_2|$ may be relatively small and the overall convergence may therefore be slow. On the other hand, the convergence is faster if there are very few intersections in the domain. In the applications involving curve and surface intersections, we make use of the geometric properties of the control polytopes in the following manner:

- Compute the number of intersections between the control polytopes of the curves or curve-surface pair. This number is used as a good guess to the actual number of intersections.
- Subdivide the curves and surfaces such that each pair obtained after subdivision consists of at most one or two intersections between the control polytopes.

In case the actual number of intersections between the original pair of control polytopes is high, we use the earlier algorithm based on QR iterations for computing all the eigenvalues [MD92]. The technique of algebraic pruning works best when there are relatively few intersections. In particular, let k correspond to the number of intersections between the control polytopes. We use the algorithm based on algebraic pruning if $k < 2\sqrt{N}$, where N refers to the order of matrix pencil highlighted in Theorem 2.1. Eventually, we apply the algebraic pruning to find eigenvalues in each domain, such that the associated control polytopes have only one or two intersections. Although we subdivide the curve and obtain new control points, the implicit representation and the matrix pencil $\mathbf{C}_1 s + \mathbf{C}_2$ remain the same and we associate each subdivided curve with a different domain. It is still possible that after subdivision the algorithm does not converge to the given solution fast enough. In that case, it could be because complex eigenvalues are close to the chosen approximation ($s^{'}$).

The inverse power iteration is used until the algorithm converges to a specified eigenvalue. In particular, we compute the eigenvalues s_k based on inverse power iteration until the successive values of s_i differ by less than a tolerance, TOL. In applications involving computation of multiple solutions, we compute p_k and q_k, until they differ between successive iterations by at most TOL. The accuracy and performance of the overall method is a function of TOL. Depending on the number of iterations it takes to converge, we use the following values of TOL and use a one-pass or two-pass approach. Let $[a, b]$ be the interval in which all eigenvalues need to be computed. $\mathbf{u}_k$ refers to the eigenvector computed at each iteration.

- Use power iterations to compute the closest eigenvalue of $\mathbf{A} = \mathbf{C}_1 s^{'} + \mathbf{C}_2$. Initially use $TOL = 0.01/(b-a)$. After three iterations if

  $$|s_3 - s_2| < TOL, \qquad |\mathbf{u}_3[2]/\mathbf{u}_3[1] - \mathbf{u}_2[2]/\mathbf{u}_2[1]| < TOL$$

  we continue using power iterations and modify $TOL = (1.0e^{-6})/(b-a)$.

-
  - In case the power iterations does not converge to two digits of relative accuracy in the first three iterations, TOL remains the same and we compute p_k and q_k using the algorithm for multiple solutions. Let the power iteration converge to a real solution s_k or a pair of real solutions, t_{1k} and t_{2k}. Each of them is accurate up to two digits.
  - For each eigenvalue t computed with two digits of relative accuracy, we set $s^{'} = t$ and perform some power iterations on $\mathbf{A} = \mathbf{C}_1 t + \mathbf{C}_2$ using $TOL = (1.0e^{-6})/(b-a)$.

The tolerances highlighted above can be user driven and the number of iterations used to switch the tolerance can be varied by the user as well. The entries $\mathbf{u}_k[2]$ and $\mathbf{u}_k[1]$ of the eigenvector are used to compute the other variable in the curve-curve intersection problem.

4.4 Illustration

In this section, we demonstrate our algorithm on two examples:

- intersection of two planar Bézier curves each of degree four, and
- intersection of a cubic Bézier curve with a bicubic tensor-product Bézier patch.

4.4.1 Intersection of two planar curves

The control points of the curves in homogeneous coordinates are defined as:

$$\mathbf{P}(s) = ((-2.5, 0.3, 1.0), (4.0, 6.1, 1.0), (-1.0, -3.3, 1.0), (5.0, 1.4, 1.0), (7.2, 3.3, 1.0))$$

and

$$\mathbf{Q}(t) = ((-2.0, 2.3, 1.0), (3.0, 2.1, 1.0), (-4.0, -2.3, 1.0), (2.0, 1.3, 1.0), (3.0, 3.3, 1.0)).$$

The curves have been shown in Fig. 4. We implicitize the first curve and obtain its implicit representation as a 4×4 matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{pmatrix} 23.2x - 26.0y + 65.8w & -21.6x - 9.0y - 51.3w & 4.4x - 30.0y + 20.0w & 3.0x - 9.7y + 10.41w \\ -21.6x - 9.0y - 51.3w & -221.2x + 90.0y + 190.4z & -72.2x - 25.7y + 408.81z & -11.2x - 12.8y + 122.88w \\ 4.4x - 30.0y + 20.0w & -72.2x - 25.7y + 408.81w & 101.6x - 156.8y - 239.52w & 39.6x - 49.2y - 122.76w \\ 3.0x - 9.7y + 10.41w & -11.2x - 12.8y + 122.88w & 39.6x - 49.2y - 122.76w & 7.6x - 8.8y - 25.68w \end{pmatrix}.$$

We substitute the second parameterization and obtain a matrix polynomial. Using Theorem 2.1 it can be reduced to finding eigenvalues of the matrix pencil $\mathbf{C}_1 t + \mathbf{C}_2$, where

$$\mathbf{C}_1 = \begin{pmatrix}
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 40.40 & 28.80 & 57.80 & 17.90 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 28.80 & -839.80 & -494.10 & -115.84 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 57.80 & -494.10 & 803.36 & 315.12 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 17.90 & -115.84 & 315.12 & 61.12 \\
1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -323.20 & 540.00 & 119.20 & 3.84 \\
0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 540.00 & 1136.80 & -552.96 & -249.60 \\
0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 119.20 & -552.96 & 1056.00 & 429.12 \\
0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 3.84 & -249.60 & 429.12 & 85.44 \\
0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -196.80 & -334.80 & -428.40 & -124.32 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -334.80 & -5209.20 & -4540.32 & -1182.72 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -428.40 & -4540.32 & 1711.68 & 1008.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & -124.32 & -1182.72 & 1008.00 & 215.04 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & -313.60 & 424.80 & 40.80 & -15.20 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 424.80 & 540.00 & -924.00 & -335.36 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 & 40.80 & -924.00 & 960.64 & 430.08 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & -15.20 & -335.36 & 430.08 & 87.68
\end{pmatrix},$$

$$\mathbf{C}_2 = \begin{pmatrix}
-1.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -40.40 & -28.80 & -57.80 & -17.90 \\
0.0 & -1.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -28.80 & 839.80 & 494.10 & 115.84 \\
0.0 & 0.0 & -1.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -57.80 & 494.10 & -803.36 & -315.12 \\
0.0 & 0.0 & 0.0 & -1.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -17.90 & 115.84 & -315.12 & -61.12 \\
-1.0 & 0.0 & 0.0 & 0.0 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 323.20 & -540.00 & -119.20 & -3.84 \\
0.0 & -1.0 & 0.0 & 0.0 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -540.00 & -1136.80 & 552.96 & 249.60 \\
0.0 & 0.0 & -1.0 & 0.0 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -119.20 & 552.96 & -1056.00 & -429.12 \\
0.0 & 0.0 & 0.0 & -1.0 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & -3.84 & 249.60 & -429.12 & -85.44 \\
0.0 & 0.0 & 0.0 & 0.0 & -1.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 196.80 & 334.80 & 428.40 & 124.32 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 334.80 & 5209.20 & 4540.32 & 1182.72 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 428.40 & 4540.32 & -1711.68 & -1008.00 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & -1.00 & 124.32 & 1182.72 & -1008.00 & -215.04 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 264.00 & -279.00 & 25.00 & 27.80 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & -279.00 & -363.80 & 816.60 & 288.32 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 25.00 & 816.60 & -508.48 & -263.76 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 27.80 & 288.32 & -263.76 & -55.76
\end{pmatrix}.$$

We apply the pruning algorithm on this example. The behavior of the algorithm has been shown in Table 1. In the table, t refers to the eigenvalue converged based on the power iterations and $s = \mathbf{u}[1]/\mathbf{u}[0]$ refers to the corresponding point on the other curve, $\mathbf{P}(s)$, based on the eigenvector. At each instance, we first compare the control polytopes of the curves for intersection. The simplest algorithm involves use of bounding box tests followed by testing the convex hulls of their control polytopes for overlap. This is reduced to a linear programming problem, whose complexity is linear in the number of constraints. In this case, each control points contributes a constraint. Good randomized algorithms for linear programming are described in [Sei90] and it works very well in practice.

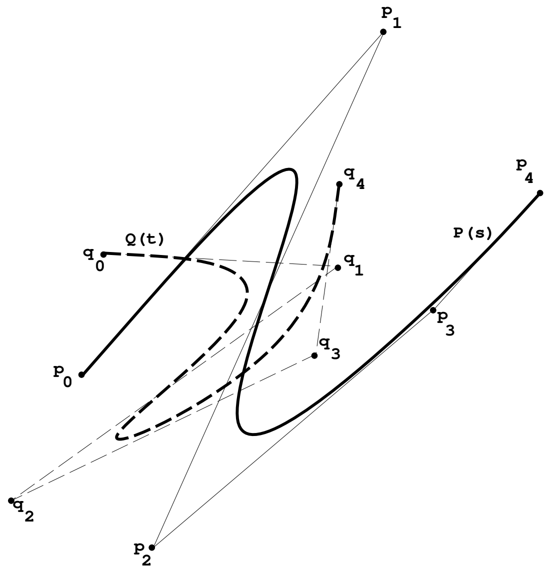

Figure 4: Intersection of Fourth Order Bézier Curves

Interval	t'	t	$s = \mathbf{u}[1]/\mathbf{u}[0]$	No. of Iterations
[0.0,1.0]	0.4	0.1297	0.1070	7
[0.0,0.5]	0.2	0.1298	0.1074	4
[.27,0.5]	0.362	0.1298	0.1066	5
[0.0,0.1298]	0.0528	0.1298	0.1073	3
[0.5,1.0]	0.7	0.7954	0.354	3
[0.7954,1.0]	0.8784	0.7951	0.351	3
[0.5,0.604]	0.541	0.7951	0.353	3

Table 1: Algebraic pruning on curves shown in Fig. 4

4.4.2 Curve-Surface Intersection

The control points of the curve in homogeneous coordinates is given by:

$\mathbf{P}(s) = ((85.0, 0.0, -150.0, 1.6), (104.0, -50.0, -50.0, 1.6), (90.0, 50.0, 50.0, 1.4), (120.0, 0.0, 150.0, 1.3))$.

The control points of the surface is given by the matrix:

$$\mathbf{Q}(u,v) = \begin{pmatrix} (160.0,-10.0,-140.0,1.8) & (140.0,-45.0,-70.0,1.4) & (135.0,64.0,40.0,1.7) & (148.0,10.0,120.0,1.2) \\ (104.0,20.0,-120.0,1.3) & (95.0,55.0,-40.0,1.2) & (109.0,-54.0,80.0,1.1) & (110.0,6.0,180.0,1.9) \\ (42.0,-159.0,-120.0,1.0) & (65.0,45.0,-60.0,1.8) & (50.0,-28.0,70.0,1.7) & (55.0,160.0,130.0,1.5) \\ (-3.0,154.0,-110.0,1.3) & (10.0,-35.0,-50.0,1.4) & (-9.0,40.0,55.0,1.6) & (9.0,-139.0,170.0,1.6) \end{pmatrix}$$

The implicit representation of the surface $\mathbf{Q}(u,v)$ is an 18×18 matrix and is denoted by $\mathbf{M}$. $\mathbf{M}$ is shown in Appendix A. After substituting the parameterization of $\mathbf{P}(s)$ (a cubic curve), we obtain

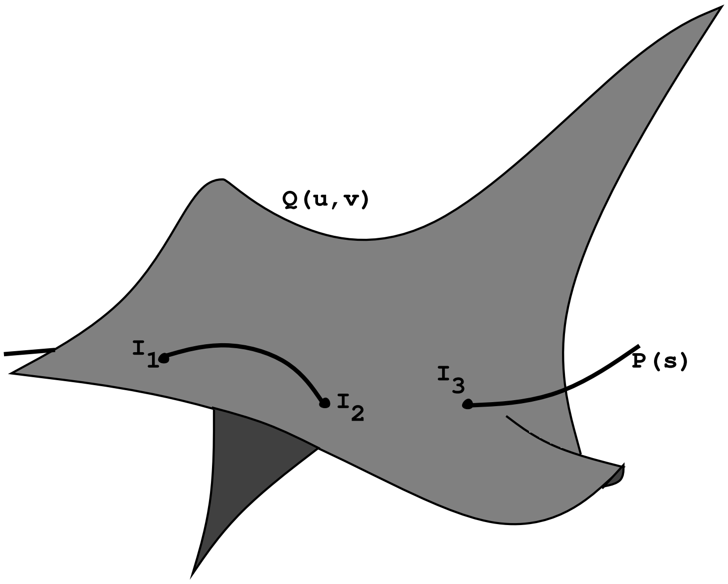

Figure 5: Intersection of a cubic Bézier Curve and a bicubic patch

a matrix pencil of order 54. The results of the pruning method are shown in Table. 2. s refers to the converged eigenvalue and u and v are obtained from the corresponding eigenvector. $\mathbf{Q}(u, v)$ thus obtained is the intersection point on the patch.

5 Performance and Comparison

We have implemented the algorithm using LAPACK routines [ABB$^+$92]. The overall algorithm has been applied to many cases of parametric curve intersection, algebraic curve intersection, intersection of a ray with a parametric surface and curve-surface intersections. In each case, the problem is reduced to an eigenvalue problem and we compute the eigenvalues in a domain. We have performed some preliminary comparisons with the algorithm based on QR iterations in [MD92] and an implementation of implicitization based algorithm described in [SP86] and Bézier clipping described in [SWZ89]. Over the course of implementation, we realized that the actual timings obtained can be greatly improved by careful programming and making use of the structure of the problem. For example, the technique applied to intersection of parametric curves results in a matrix polynomial with symmetric matrices. This structure can be exploited in solving linear equations arising in inverse power iterations and giving us almost a speed up of two over the implementation not making use of symmetric structure of the matrices. As a result, if we implement algorithms as part of a generic package or specialize to particular cases (like intersection of cubic Bézier curves), we can see a considerable difference in their running time. A similar analysis holds for the implementation of implicitization based algorithm described in [SP86]. Thus, it is rather difficult to perform an exact comparison between two algorithms and in this section we will analyze them more in terms of convergence per iteration and the total number of iterations required on various examples.

One of the main advantages of using inverse power iterations is that whenever it converges the

Interval	s'	s	$u = \mathbf{u}[1]/\mathbf{u}[0]$	$v = \mathbf{u}[3]/\mathbf{u}[0]$	No. of Iterations
[0.0,1.0]	0.5	0.5976	0.3367	0.5738	9
[0.0,0.3923]	0.1961	0.2021	-0.7150	0.4619	5
[0.0,0.1802]	0.0901	0.1210	2.3133	0.4864	7
[0.1310,0.1802]	0.1556	0.1663	0.4871	0.1582	3
[0.2121,0.3923]	0.3022	0.2900	0.1880	-1.0303	4
[0.2121,0.2800]	0.2461	0.2394	-0.5077	-0.3593	3
[0.6076,1.0]	0.8038	0.7973	0.2332	0.8176	4
[0.8204,1.0]	0.9102	$0.9177 \pm i0.0317$	–	–	5

Table 2: Algebraic pruning on the curve and surface shown in Fig.5

closest eigenvalue is obtained. This is the basis of algebraic pruning. It is, however, possible that inverse iterations do not converge for particular choices of initial eigenvalue and eigenvector. As discussed earlier, it can occur when the initial guess is such that the closest eigenvalues are a pair of reals or complex conjugates. Another possibility for non-convergence is the initial choice of the eigenvector, u_0. Let λ_i be the closest eigenvalue to s' with corresponding eigenvector x_i. If u_0 has a very small component of x_i then rounding errors may prevent these components from being enriched and the algorithm may not converge. However, according to [Wil65], this possibility is extremely rare if u_0 is chosen at random.

If a given pair of curves or surface has very few intersections, the technique based on algebraic pruning gives almost an order of magnitude improvement over the algorithm presented in [MD92]. This is mainly due to the fact that we are only computing the relevant solutions in the domain of interest as opposed to computing all the solutions. For example, on a DEC 5000/25, it takes about 8.5 milliseconds to compute all the intersections of the example highlighted in Section 4.4 using algebraic pruning. On the other hand, application of QR algorithm takes about 78.2 milliseconds on the same matrix to compute all the eigenvalues and the eigenvectors corresponding to eigenvalues in the domain. In the case of a cubic curve and bicubic surface intersections, we obtain a 54×54 matrix. For cases, involving two or three intersections the algebraic pruning performed better by more than an order of magnitude as compared to the QR algorithm. The QR algorithm has a comparable performance if the number of intersections in the domain is at least equal to $2\sqrt{N}$.

Compared to the implicitization based approach highlighted in [Sed83], algebraic pruning has almost the same order of magnitude (on low degree curves of degree three or four). The implicitization based approach involves expansion of the symbolic determinant and computing all the roots in the domain of interest of the resulting polynomial. The latter can be computed efficiently using the Bernstein representation of the resulting polynomial. On the other hand algebraic pruning reduces it to an eigenvalue problem and does not involve symbolic expansion. After running it on few examples, we have observed that the algorithm to find all the roots in the given domain of a polynomial [SP86] is slightly faster than computation of eigenvalues in the given domain. For most applications on degree three and degree four curves, both algorithms take about 5 to 7 milliseconds on the DEC 5000/25. However, on degree five curves consisting of at most two or three intersections, algebraic pruning performed better by a couple of milliseconds. We would again like to highlight the fact that these are the performance figures corresponding to our implementation and an earlier implementation of implicitization based intersection algorithm [SP86]. Other implementations may result in a different set of timings.

Finally, we compared our algorithm to Bézier clipping [SWZ89]. This comparison has been performed only for curve intersections. The actual performance of the algorithm is actually a function of the geometry of the curves and the number of intersections in the given domain. On low degree curves of degree five or six, both algorithms take about 8 to 16 milliseconds on the DEC 5000/25. The algebraic pruning is typically faster in cases consisting of one or two intersections, whereas Bézier clipping is faster by a few milliseconds in the other cases. This is consistent with the fact that the convergence of algebraic pruning is a function of the proximity to other intersections. At a conceptual level, it appears that the convergence of algebraic pruning is slightly better than that of Bézier clipping for cases consisting of a few intersections. On the other hand, the number of operations at each iteration of Bézier clipping is less than that of algebraic pruning. Subdividing a Bézier curve takes $O(n^2)$ operations whereas each each iteration of algebraic pruning takes $O(nm^2)$ operations. However, typically we are dealing with low values of m and n and one needs to take care of the constants in front of these asymptotic bounds.

In applications like ray-tracing, algebraic pruning can be very well combined with ray to ray coherence. For example, the intersection of the previous ray with the surface can be used as a starting guess for the next ray-surface intersection. Given a good starting guess, inverse power iterations converges in a very few iterations only. As a result, the inverse iteration combines very well with spatial and temporal coherence.

6 Acknowledgements

The authors are grateful to Tom Sederberg for an earlier implementation of intersection algorithms presented in [SP86, SWZ89].

References

[ABB+92] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, and D. Sorensen. *LAPACK User's Guide, Release 1.0.* SIAM, Philadelphia, 1992.

[Dix08] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[EC90] G. Elber and E. Cohen. Hidden curve removal for free form surfaces. *Computer Graphics*, 24(4):95–104, 1990.

[Far90] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide.* Academic Press Inc., 1990.

[FR87] R.T. Farouki and V.T. Rajan. On the numerical condition of polynomials in bernstein form. *Computer Aided Geometric Design*, 4:191–216, 1987.

[GL89] G.H. Golub and C.F. Van Loan. *Matrix Computations.* John Hopkins Press, Baltimore, 1989.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, California, 1989.

[Hof90] C.M. Hoffmann. A dimensionality paradigm for surface interrogations. *Computer Aided Geometric Design*, 7:517–532, 1990.

[Kaj82] J. Kajiya. Ray tracing parametric patches. *Computer Graphics*, 16(3):245–254, 1982.

[KM83] P.A. Koparkar and S. P. Mudur. A new class of algorithms for the processing of parametric curves. *Computer-Aided Design*, 15(1):41–45, 1983.

[LR80] J.M. Lane and R.F. Riesenfeld. A theoretical development for the computer generation and display of piecewise polynomial surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(1):150–159, 1980.

[Man92] D. Manocha. *Algebraic and Numeric Techniques for Modeling and Robotics.* PhD thesis, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, May 1992.

[MD92] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves. In *Graphics Interface '92*, pages 232–241, 1992. Revised version to appear in *ACM Transactions on Graphics.*

[NSK90] T. Nishita, T.W. Sederberg, and M. Kakimoto. Ray tracing trimmed rational surface patches. *Computer Graphics*, 24(4):337–345, 1990.

[Sal85] G. Salmon. *Lessons Introductory to the Modern Higher Algebra.* G.E. Stechert & Co., New York, 1885.

[Sed83] T.W. Sederberg. *Implicit and Parametric Curves and Surfaces.* PhD thesis, Purdue University, 1983.

[Sed89] T.W. Sederberg. Algorithms for algebraic curve intersection. *Computer-Aided Design*, 21(9):547–555, 1989.

[Sei90] R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pages 211–215, Berkeley, California, 1990.

[SP86] T.W. Sederberg and S.R. Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, 1986.

[SWZ89] T.W. Sederberg, S. White, and A. Zundel. Fat arcs: A bounding region with cubic convergence. *Computer Aided Geometric Design*, 6:205–218, 1989.

[Wil59] J.H. Wilkinson. The evaluation of the zeros of ill–conditioned polynomials. parts i and ii. *Numer. Math.*, 1:150–166 and 167–180, 1959.

[Wil65] J.H. Wilkinson. *The algebraic eigenvalue problem.* Oxford University Press, Oxford, 1965.

Appendix A

$$\mathbf{M} = \begin{pmatrix}
0.2x + 0.1y + 0.0z - 0.1w & -3.8x + 0.6y + 2.1z + 2.8w & 1.0x + 0.7y + 0.3z - 0.3w & 9.6x + 3.1y - 1.0z - 5.1w & -4.0x + 11.8y + 1.1z + 2.8w & 5.3x + 6.1y + 1.3z - 1.9w & \cdots \\
3.0x + 1.1y - 0.2z - 1.6w & -11.7x + 4.4y - 3.0z + 4.6w & 4.7x + 3.4y - 1.0z - 2.7w & 27.7x + 8.0y - 1.1z - 13.9w & -8.8x + 35.3y - 21.4z + 0.4w & 9.0x + 17.8y - 8.4z - 6.6w & \cdots \\
1.0x + 0.3y - 0.2z - 0.6w & -5.3x + 1.4y + 0.9z + 3.1w & 1.9x + 1.4y - 0.5z - 1.1w & 11.5x + 3.6y - 2.5z - 6.6w & -2.4x + 15.1y - 1.6z + 1.8w & 3.7x + 7.4y - 1.7z - 2.0w & \cdots \\
0.2x + 1.8y + 1.5z + 0.6w & -20.6x + 6.8y + 1.3z + 10.8w & 7.7x + 5.6y - 1.5z - 4.3w & 23.8x + 21.0y - 1.7z - 11.7w & -54.6x + 68.6y - 2.4z + 24.8w & 17.0x + 34.4y - 1.5z - 8.7w & \cdots \\
-6.3x + 6.9y - 2.6z + 2.2w & -72.5x + 22.5y + 3.2z + 35.0w & 22.5x + 17.4y + 0.5z - 10.9w & 78.9x + 61.6y - 19.4z - 41.2w & -101.7x + 192.8y + 3.3z + 51.6w & 32.8x + 91.9y + 9.5z - 15.1w & \cdots \\
-0.7x + 3.8y + 1.6z + 1.2w & -37.5x + 13.0y - 2.0z + 17.3w & 12.6x + 9.6y + 0.5z - 5.9w & 50.8x + 29.6y + 7.2z - 20.2w & -37.8x + 99.3y - 3.3z + 20.9w & 16.7x + 47.1y + 2.2z - 7.6w & \cdots \\
-2.6x + 9.3y - 3.8z - 0.1w & -44.1x + 26.1y - 32.3z + 5.3w & 24.1x + 16.0y - 10.7z - 15.2w & 12.2x + 58.5y - 2.9z - 0.5w & -154.0x + 163.3y - 102.7z + 35.5w & 30.8x + 78.9y - 42.8z - 25.3w & \cdots \\
-28.7x + 20.7y + 9.4z + 18.9w & -134.5x + 58.5y - 0.8z + 61.2w & 63.5x + 38.4y + 2.7z - 23.2w & 82.1x + 155.0y + 60.4z - 1.8w & -248.5x + 409.6y + 93.3z + 135.5w & 84.4x + 190.9y + 35.5z - 22.1w & \cdots \\
-17.9x + 12.5y - 5.3z + 7.0w & -70.6x + 32.9y - 16.4z + 22.8w & 32.1x + 20.4y - 2.6z - 14.2w & 31.0x + 82.6y - 10.7z - 8.6w & -139.1x + 208.9y - 51.0z + 36.7w & 25.0x + 95.5y - 15.5z - 20.4w & \cdots \\
-9.8x + 16.3y - 26.7z - 6.2w & -47.1x + 37.4y - 54.7z - 9.1w & 38.6x + 19.5y - 12.1z - 18.0w & -8.2x + 80.5y - 89.9z - 15.7w & -144.4x + 183.2y - 192.6z - 8.4w & 51.5x + 85.7y - 64.6z - 31.9w & \cdots \\
-31.7x + 32.9y - 18.3z + 8.8w & -102.9x + 76.9y - 40.4z + 20.9w & 97.5x + 41.7y - 2.8z - 27.5w & 44.4x + 182.3y + 0.8z + 6.5w & -229.0x + 413.7y - 31.9z + 73.4w & 150.1x + 194.9y - 8.8z - 37.1w & \cdots \\
-5.6x + 15.0y - 7.8z - 0.2w & -30.9x + 34.8y - 18.6z + 2.5w & 52.0x + 19.1y - 8.1z - 15.7w & 20.7x + 83.9y - 35.4z - 14.1w & -111.7x + 189.8y - 78.3z + 3.7w & 74.3x + 90.6y - 37.2z - 28.8w & \cdots \\
1.6x + 9.0y - 3.7z - 2.1w & 0.1x + 19.1y - 3.9z - 3.9w & 38.2x + 9.8y - 5.6z - 8.6w & 13.6x + 42.5y - 13.1z - 6.9w & -28.9x + 86.3y - 12.3z - 4.0w & 51.4x + 41.9y - 14.6z - 10.2w & \cdots \\
3.5x + 19.8y - 3.5z - 2.6w & 0.6x + 41.6y - 4.7z - 6.3w & 82.8x + 19.7y + 0.8z - 11.5w & 25.0x + 94.7y - 7.0z - 4.9w & -80.8x + 195.8y - 18.2z - 0.3w & 130.7x + 99.0y + 7.2z - 24.2w & \cdots \\
-1.6x + 8.2y - 3.6z - 0.5w & -6.6x + 17.7y - 5.3z - 1.4w & 31.3x + 8.8y - 1.5z - 6.1w & 9.5x + 42.5y - 9.9z - 3.3w & -38.7x + 88.9y - 15.5z + 0.7w & 54.8x + 44.0y - 9.5z - 14.0w & \cdots \\
1.5x + 1.3y + 3.6z + 0.8w & 4.0x + 2.7y + 7.1z + 1.6w & 9.4x + 1.3y + 1.9z + 0.2w & 6.6x + 7.1y + 12.5z + 1.3w & 2.8x + 13.7y + 26.1z + 3.3w & 13.4x + 6.8y + 15.6z + 3.0w & \cdots \\
4.9x + 3.8y + 3.1z - 1.0w & 9.8x + 7.3y + 6.0z - 1.9w & 21.0x + 3.0y + 2.9z - 0.3w & 10.6x + 17.8y + 5.3z - 2.9w & -0.4x + 35.1y + 10.3z - 6.3w & 41.0x + 19.0y + 13.4z - 4.1w & \cdots \\
2.6x + 2.0y + 2.1z - 0.3w & 5.3x + 3.9y + 4.0z - 0.6w & 11.6x + 1.7y + 1.3z - 0.3w & 5.8x + 9.5y + 7.2z + 0.1w & 0.5x + 18.8y + 14.8z + 0.3w & 22.7x + 10.2y + 8.9z - 1.0w & \cdots
\end{pmatrix}$$

$$\begin{pmatrix}
\cdots & 17.5x + 5.2y + 0.5z - 8.3w & 10.3x + 34.6y - 2.1z - 4.9w & 1.7x + 15.1y - 2.5z - 1.4w & -18.1x - 0.9y - 0.3z + 8.7w & -14.0x + 36.5y - 13.4z + 2.3w & -2.5x + 17.7y - 4.1z + 0.0w & \cdots \\
\cdots & 13.5x + 5.2y - 1.4z - 8.0w & 3.1x + 83.0y - 48.4z - 8.7w & 0.0x + 37.1y - 24.4z - 3.6w & -62.8x - 5.5y - 8.1z + 29.8w & -31.0x + 78.4y - 58.4z + 11.9w & -11.1x + 38.1y - 25.0z + 3.8w & \cdots \\
\cdots & 8.3x + 4.3y - 2.7z - 4.4w & 5.3x + 38.1y - 4.8z - 2.7w & 2.3x + 16.3y - 3.6z - 0.3w & -14.0x + 3.9y - 0.4z + 9.7w & -5.9x + 37.2y - 3.5z + 4.3w & -3.1x + 17.9y - 0.1z + 1.4w & \cdots \\
\cdots & 55.2x + 49.9y + 0.5z - 24.8w & -3.4x + 202.6y - 1.5z - 0.7w & -2.4x + 84.6y + 1.2z + 0.3w & -47.8x + 40.1y - 9.3z + 19.0w & 13.8x + 237.2y + 6.3z - 9.8w & 2.2x + 104.2y + 3.7z - 1.7w & \cdots \\
\cdots & 125.1x + 118.7y - 31.3z - 66.1w & 44.7x + 499.0y + 19.4z - 14.9w & 13.9x + 207.3y + 19.5z - 1.5w & -57.5x + 91.7y - 38.4z + 27.0w & 106.8x + 523.4y + 28.0z - 39.1w & 21.2x + 230.4y + 20.2z - 5.0w & \cdots \\
\cdots & 43.5x + 45.0y + 9.8z - 22.8w & 8.5x + 234.0y + 5.7z - 6.1w & 2.9x + 99.5y + 1.5z - 1.7w & -69.8x + 22.6y - 6.6z + 26.7w & -3.3x + 224.3y - 5.1z - 11.0w & -7.7x + 102.8y - 2.0z - 1.3w & \cdots \\
\cdots & 17.0x + 130.6y + 8.1z + 6.3w & -90.4x + 418.7y - 116.0z + 25.9w & -26.3x + 174.3y - 52.7z + 4.3w & -133.6x + 129.2y - 10.2z + 43.3w & 43.1x + 487.0y - 38.0z - 11.1w & -7.4x + 206.0y - 33.4z + 4.4w & \cdots \\
\cdots & 159.7x + 340.2y + 123.5z - 46.9w & -121.2x + 1017.8y + 318.6z + 66.1w & -42.2x + 421.3y + 115.5z + 16.7w & -136.3x + 318.0y + 95.9z + 27.4w & -14.9x + 1124.9y + 339.5z - 100.0w & -33.5x + 482.8y + 125.7z - 29.3w & \cdots \\
\cdots & 40.3x + 167.2y - 6.3z - 12.9w & -96.0x + 489.9y - 96.4z + 9.0w & -36.1x + 200.9y - 33.0z - 4.7w & -121.2x + 146.4y - 16.5z + 46.3w & 68.5x + 519.9y - 108.0z - 47.8w & 7.6x + 215.7y - 59.1z - 10.0w & \cdots \\
\cdots & 25.7x + 154.5y - 121.2z - 7.7w & 12.3x + 399.5y - 310.0z - 17.6w & 8.6x + 171.6y - 116.0z - 4.4w & -46.0x + 153.0y - 99.1z + 13.5w & 258.3x + 439.7y - 211.7z - 11.7w & 45.9x + 188.2y - 93.1z + 14.9w & \cdots \\
\cdots & 126.3x + 384.1y + 67.0z - 15.8w & -134.9x + 966.5y + 74.8z + 64.6w & -28.4x + 418.8y + 48.5z + 22.7w & -120.9x + 390.6y + 43.6z - 0.7w & 136.6x + 1146.1y + 203.1z - 100.8w & 21.9x + 502.5y + 84.2z - 24.2w & \cdots \\
\cdots & 16.3x + 172.5y - 91.0z - 22.4w & -109.0x + 432.3y - 195.1z - 14.8w & -31.2x + 189.8y - 82.7z - 4.1w & -65.5x + 165.0y - 128.9z + 8.4w & 179.9x + 479.5y - 279.5z - 65.4w & 33.4x + 216.0y - 105.3z - 12.6w & \cdots \\
\cdots & 8.2x + 77.8y - 20.0z - 2.6w & -40.7x + 173.7y - 34.5z + 1.7w & -17.4x + 85.4y - 38.4z - 3.6w & -20.1x + 78.3y - 21.3z + 1.9w & 33.7x + 189.6y - 34.5z + 7.4w & -30.6x + 90.9y - 22.8z + 8.0w & \cdots \\
\cdots & 29.3x + 191.2y - 4.1z - 3.9w & -112.1x + 434.7y - 40.0z + 13.1w & -38.6x + 211.0y - 23.2z + 1.5w & -38.1x + 203.9y - 11.0z - 1.8w & 124.6x + 531.3y + 3.4z - 18.9w & 12.9x + 246.4y + 10.7z - 6.4w & \cdots \\
\cdots & 17.7x + 89.5y - 15.5z - 6.1w & -44.1x + 202.0y - 28.2z + 4.2w & -10.4x + 93.8y - 28.0z - 0.4w & -20.4x + 93.8y - 21.3z + 0.4w & 48.6x + 240.5y - 19.5z - 9.4w & -5.4x + 111.1y - 26.1z - 4.0w & \cdots \\
\cdots & 4.5x + 14.5y + 19.6z - 0.3w & -10.7x + 30.3y + 42.0z - 0.3w & 1.8x + 17.4y + 21.9z - 1.1w & 1.1x + 16.0y + 16.5z - 2.7w & -7.0x + 35.7y + 37.6z - 5.9w & 2.1x + 19.8y + 21.6z - 2.8w & \cdots \\
\cdots & 4.7x + 35.9y + 3.1z - 2.8w & -22.1x + 76.9y + 3.6z - 6.3w & 1.2x + 43.3y - 2.1z - 4.2w & -6.1x + 38.4y - 0.9z + 0.7w & 17.4x + 91.2y + 2.3z + 1.4w & -6.2x + 47.5y + 2.7z + 1.5w & \cdots \\
\cdots & 5.7x + 19.7y + 14.7z + 0.7w & -9.3x + 41.4y + 30.9z + 2.1w & -2.7x + 22.5y + 13.4z + 3.0w & -1.1x + 21.2y + 17.7z - 0.2w & 3.4x + 48.4y + 40.8z - 0.9w & 0.6x + 26.6y + 24.5z - 1.7w & \cdots
\end{pmatrix}$$

$$\begin{pmatrix}
\cdots & -17.1x + 4.7y - 0.9z + 8.2w & 1.8x + 19.7y - 2.0z - 1.1w & -6.1x + 9.6y - 2.7z + 2.1w & -0.3x + 4.4y - 0.3z + 0.2w & 5.3x + 4.5y - 1.4z - 3.1w & -1.8x + 2.3y + 0.2z + 1.0w \\
\cdots & -40.2x + 9.4y - 8.1z + 21.1w & 4.1x + 40.2y - 13.7z + 0.6w & -13.8x + 19.5y - 12.2z + 7.3w & -0.5x + 8.6y - 2.2z + 1.2w & 10.7x + 8.9y - 4.1z - 5.5w & -3.6x + 4.4y - 0.6z + 2.6w \\
\cdots & -10.2x + 9.8y + 3.8z + 4.4w & 0.2x + 22.2y + 7.0z - 5.1w & -4.2x + 10.8y + 2.1z + 1.2w & -0.6x + 5.1y + 2.3z - 1.1w & 6.1x + 5.1y - 1.5z - 3.5w & -2.1x + 2.8y + 2.3z + 0.1w \\
\cdots & -33.8x + 33.3y + 0.1z + 14.6w & 46.0x + 113.0y - 2.8z - 25.6w & -11.3x + 55.8y + 2.6z + 4.8w & 4.5x + 14.7y - 0.8z - 3.3w & 36.0x + 18.3y + 2.1z - 17.1w & -12.1x + 9.7y - 0.5z + 5.3w \\
\cdots & -44.5x + 70.5y + 2.3z + 19.1w & 125.3x + 231.2y + 6.8z - 64.6w & -19.6x + 113.7y + 7.6z + 10.5w & 8.7x + 28.2y + 2.1z - 8.4w & 69.8x + 35.5y + 9.4z - 35.6w & -23.9x + 18.6y - 1.1z + 10.4w \\
\cdots & -41.6x + 19.7y - 9.4z + 17.1w & 39.9x + 91.9y - 7.0z - 23.2w & -18.7x + 47.1y - 3.4z + 7.0w & 6.4x + 8.6y - 9.1z + 0.3w & 32.2x + 13.0y + 2.1z - 12.9w & -10.4x + 7.0y - 2.8z + 5.3w \\
\cdots & -59.2x + 76.4y - 8.7z + 21.6w & 88.1x + 225.8y - 39.3z - 38.4w & -34.2x + 107.0y - 18.8z + 15.0w & 10.3x + 21.6y - 8.9z - 3.1w & 59.3x + 31.5y - 3.9z - 20.5w & -35.9x + 16.8y - 9.0z + 14.6w \\
\cdots & -91.2x + 185.9y + 59.5z + 6.4w & 173.8x + 504.2y + 63.0z - 138.1w & -102.6x + 241.1y + 43.1z + 4.1w & 16.4x + 52.4y + 2.7z - 15.2w & 121.3x + 71.4y + 6.2z - 54.0w & -72.9x + 38.0y - 4.6z + 23.4w \\
\cdots & -90.9x + 79.0y - 3.4z + 27.4w & 139.3x + 201.4y - 119.8z - 21.8w & -40.7x + 91.0y - 59.6z + 27.5w & 3.0x + 17.0y - 13.5z + 4.7w & 26.4x + 23.0y - 7.1z - 9.6w & -27.8x + 10.0y - 24.3z + 20.0w \\
\cdots & -16.1x + 78.6y - 38.2z + 9.7w & 142.3x + 208.5y - 94.5z - 13.0w & -29.8x + 95.0y - 58.3z + 26.4w & 7.3x + 17.4y - 7.9z - 1.4w & 37.3x + 28.8y - 16.3z - 4.0w & -50.8x + 15.4y - 13.8z + 17.2w \\
\cdots & -49.2x + 207.8y + 8.7z - 12.3w & 232.4x + 545.7y - 9.4z - 120.5w & -110.0x + 262.9y + 12.4z + 12.0w & 26.4x + 45.6y - 15.1z - 11.8w & 131.8x + 75.0y - 22.2z - 31.8w & -113.3x + 42.8y - 8.3z + 29.1w \\
\cdots & -28.8x + 83.6y - 88.2z + 14.2w & 261.1x + 196.9y - 241.1z + 0.3w & -36.3x + 97.5y - 97.9z + 41.0w & 13.4x + 15.4y - 29.9z + 6.7w & 61.6x + 22.9y - 62.2z + 12.4w & -36.1x + 14.2y - 26.5z + 23.6w \\
\cdots & -14.6x + 39.0y - 11.8z + 4.1w & 31.5x + 96.4y - 24.7z + 7.2w & -56.3x + 44.9y - 14.1z + 7.9w & -0.3x + 7.8y - 2.7z + 1.4w & -1.1x + 14.8y - 7.9z + 4.6w & -35.6x + 7.9y - 2.0z + 6.1w \\
\cdots & -33.1x + 107.7y - 5.9z + 2.5w & 84.7x + 279.4y + 6.4z - 14.0w & -117.1x + 124.4y - 31.0z + 15.1w & -2.4x + 22.7y - 0.2z + 0.7w & -9.0x + 43.7y - 3.5z + 5.0w & -103.6x + 22.2y - 5.1z + 17.0w \\
\cdots & -16.5x + 50.3y - 13.4z + 2.9w & 36.8x + 116.2y - 26.5z + 8.4w & -73.4x + 54.6y - 26.6z + 17.5w & -0.2x + 10.0y - 3.5z + 1.7w & -0.4x + 18.8y - 12.5z + 6.9w & -45.1x + 10.5y + 1.0z + 5.9w \\
\cdots & -3.2x + 8.4y + 8.1z - 1.8w & -3.5x + 20.0y + 19.0z - 4.3w & -12.7x + 9.5y + 8.5z - 2.4w & -1.3x + 1.7y + 1.9z - 0.3w & -5.4x + 3.7y + 4.2z - 0.5w & -9.5x + 1.8y + 2.3z - 0.2w \\
\cdots & -11.3x + 20.1y + 1.9z + 2.7w & 4.0x + 49.3y + 5.7z + 5.7w & -37.2x + 22.3y - 2.2z + 2.9w & -3.3x + 4.2y + 1.8z + 1.1w & -14.3x + 8.8y + 3.7z + 2.5w & -24.5x + 4.1y + 1.5z + 1.8w \\
\cdots & -2.9x + 11.7y + 13.4z - 2.2w & -11.8x + 25.4y + 29.2z - 4.6w & -16.5x + 12.9y + 14.0z - 1.5w & -1.7x + 2.4y + 4.1z - 1.2w & -7.0x + 5.3y + 9.3z - 2.7w & -12.5x + 2.6y + 6.0z - 1.9w
\end{pmatrix}$$