

# Algorithm for algebraic curve intersection

T W Sederberg

---

*A robust algorithm is presented for computing all real points at which two planar algebraic curves intersect within a specified area. The classical theory of polar curves is reviewed and applied to the problem of computing the double points on an algebraic curve. Using polar curves, the intersection algorithm can quickly and robustly compute all real double points on an algebraic curve, as well as silhouette points. Polar curves are shown to have a strikingly simple expression in Bernstein polynomial form. Hessian curves are also discussed. The inflection points of an algebraic curve can be found by intersecting the curve with its Hessian.*

*computer-aided design, geometry, curve intersections, polar curves, Hessian curves*

---

Intersection algorithms are fundamental to computer-aided geometric design. Algorithms for intersecting planar *parametric* curves have been studied extensively, and include those based on implicitization<sup>1</sup>, convex hulls and subdivision<sup>2</sup> and interval analysis<sup>3</sup>.

Algorithms for computing the intersection points of *algebraic* curves (that is, curves defined by a polynomial implicit equation  $f(x, y) = 0$ ) have not been the focus of as much research, probably because algebraic curves are not as widely used as parametric curves. In the computer-aided geometric design literature, previous robust algorithms have employed resultants<sup>4,5,6</sup>. This approach reduces the problem of intersecting two algebraic curves of degree  $m$  and  $n$  respectively to the problem of finding the roots of a univariate polynomial of degree  $mn$ . If the two algebraic curves are given by the equations  $f_1(x, y) = 0$  and  $f_2(x, y) = 0$ , then the resultant of  $f_1$  and  $f_2$  with respect to  $y$  gives a univariate polynomial in  $x$  whose roots are the  $x$ -coordinates of the points of intersection. This method works very well for curves of low degree (and is the method of choice for low degrees), but in the author's experience it can be numerically unreliable for curves with degrees above five.

The algorithm presented in this paper works directly from the given curves, without resorting to high degree entities such as resultants. The numerical condition of intersecting algebraic curves is presented by Farouki and Rajan<sup>7</sup>. The algorithm in this paper has the flavour of an interval algorithm, taking advantage of the special

properties of bivariate polynomials in Bernstein form. The user defines a triangle within which the algorithm searches for all intersection points. If there is a single intersection point, the algorithm behaves roughly as follows. With each iteration, the algorithm determines how far it can safely shrink the triangle and still enclose the intersection point. In the neighbourhood of an intersection, each iteration reduces the area of the triangle by several orders of magnitude while guaranteeing that the intersection point lies within the triangle. If there is more than one intersection point, the area reduction slows, or even stops. In that event, the triangle is split four ways and each of the four subtriangles is processed.

This paper also addresses the problem of computing the real double points of an algebraic curve. One global solution to this problem is to compute the discriminant of the curve, which is the resultant of the curve and one of its partial derivatives. Again, floating-point solution limits the reliability of the discriminant method to curves of low degree. Bajaj and associates<sup>8</sup> discuss a local method for computing a double point (or points of higher multiplicity) once the neighbourhood of such a point is identified. The solution presented in this paper finds all double points within a triangular domain. It proceeds by computing the intersection of three *polar curves*, using a modification of the algebraic curve intersection algorithm.

The use of polar curves in computing silhouette points is also reviewed, as well as how to compute all points on an algebraic curve that have a specified tangent direction. The intersection algorithm can also be used to compute the inflection points of an algebraic curve. This is accomplished by intersecting the curve with its Hessian.

The algorithms in this paper require the algebraic curves to be expressed in Bernstein form. This is reviewed next. The following section presents the intersection algorithm. A discussion of polar curves in terms of Bernstein polynomials follows, and then Hessian curves and the computation of inflection points are dealt with.

## PIECEWISE ALGEBRAIC CURVES

Algebraic curves are most commonly expressed in standard power basis:

$$f(x, y) = \sum_{i+j \leq n} c_{ij} x^i y^j = 0$$

---

Department of Civil Engineering, Brigham Young University, Provo, UT 84602, USA

For computer-aided geometric design applications, there are advantages to expressing algebraic curves in the Bernstein polynomial basis. For one thing, floating-point computations are more stable<sup>7</sup>. The Bernstein basis also provides a meaningful relationship between the coefficient values and the shape of the curve<sup>5,9</sup>.

Algebraic curves expressed in Bernstein form have been termed *piecewise algebraic curves*<sup>5</sup>, because it is possible to piece them together with derivative continuity. Hereafter, this paper will refer to them as PACs. This section reviews the basics of PACs.

A general triangular Bezier surface patch of degree  $n$  is defined by the equation

$$\mathbf{P}(s, t, u) = \sum_{i+j+k=n} \mathbf{P}_{ijk} \binom{n}{ijk} s^i t^j u^k \quad (1)$$

A so-called nonparametric patch is one for which the projection of its control points onto the  $x$ - $y$  plane forms a triangular lattice. We denote by  $\mathbf{x}_{ijk}$  the  $x$  and  $y$  Cartesian coordinates of  $\mathbf{P}_{ijk}$ , and by  $c_{ijk}$  the  $z$  coordinate of  $\mathbf{P}_{ijk}$ . A triangular patch is then nonparametric if

$$\mathbf{x}_{ijk} = \frac{i\mathbf{x}_{n00} + j\mathbf{x}_{0n0} + k\mathbf{x}_{00n}}{n} \quad (2)$$

The orthographic view of the control points for a cubic nonparametric patch shown in Figure 1 illustrates the control point topology.

For a nonparametric patch, it can be shown from equations (1) and (2) that the Cartesian coordinates ( $x, y$ ) are linear functions of the barycentric coordinates ( $s, t, u$ ):

$$\mathbf{x} = (x(s, t, u), y(s, t, u)) = s\mathbf{x}_{n00} + t\mathbf{x}_{0n0} + u\mathbf{x}_{00n}$$

The patch equation can therefore be written  $z = f(x, y)$

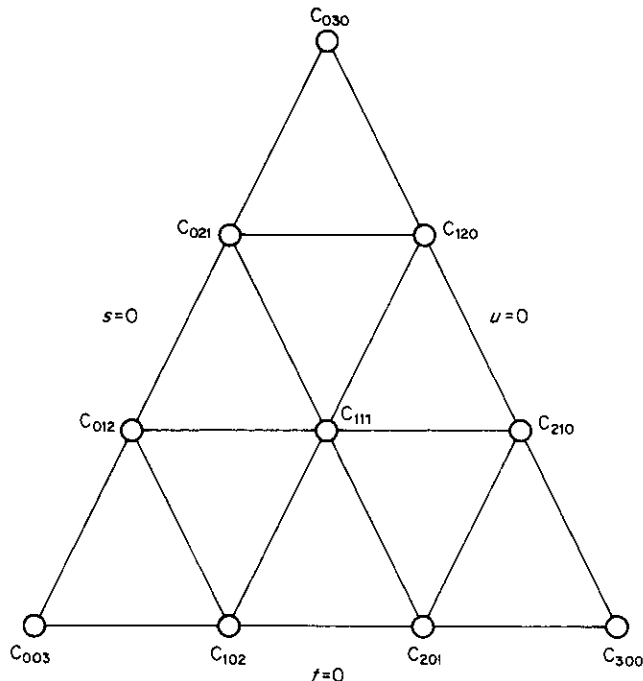


Figure 1. Cubic nonparametric patch

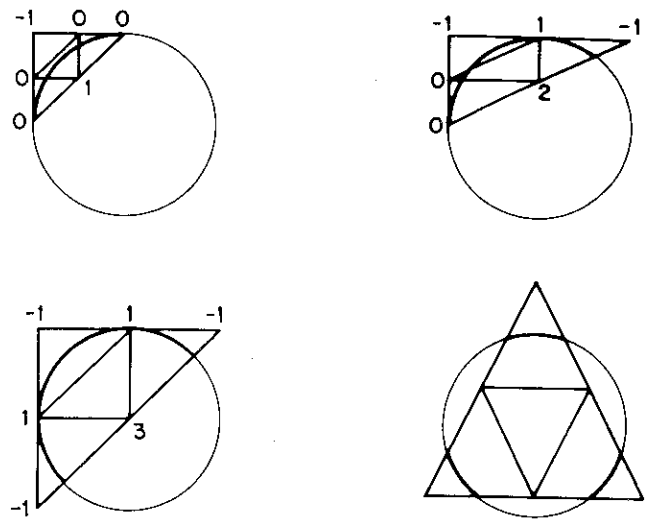


Figure 2. Circle as a PAC

$= f(x(s, t, u), y(s, t, u))$  where  $f(x, y)$  is a degree  $n$  polynomial function.

A PAC amounts to the intersection of a nonparametric triangular Bézier surface patch with the plane  $z = 0$ . If the nonparametric patch is given by  $z = f(x, y)$ , then its intersection with the plane  $z = 0$  produces the algebraic curve  $f(x(s, t, u), y(s, t, u)) = 0$ .

A PAC is defined within a triangle with vertices  $\mathbf{x}_{n00}$ ,  $\mathbf{x}_{0n0}$ ,  $\mathbf{x}_{00n}$ . This triangle can be thought of as a window through which a portion of the algebraic curve is viewed. The triangle vertices can be any three noncollinear points: for the present application, the triangle defines the search domain for curve intersections. For example, Figure 2 shows a circle defined as a PAC within four different triangles. Note that the PAC coefficients change as the triangle is altered. The new coefficients can be found using the de Casteljau algorithm. Note, too, that the Cartesian coordinates of the triangle vertices need to be specified in order to define a PAC totally. However, linear transformations (such as scaling, rotation and translation) do not change the PAC coefficients in the sense that any quarter circle can be defined using the PAC coefficients in the top-left of Figure 2 by simply transforming the Cartesian coordinates of the triangle vertices.

### Power-to-PAC conversion

Any algebraic curve given in power form  $f(x, y) = \sum_{i+j \leq m} a_{ij} x^i y^j = 0$  can be expressed as a PAC within a triangle whose vertices are  $(x_{00}, y_{00})$ ,  $(x_{01}, y_{01})$ ,  $(x_{10}, y_{10})$ . One solution to this problem is discussed by Waggenaspak and Anderson<sup>10</sup>. For completeness, an alternate solution is presented here using a recursive algorithm based on Euler's law for homogeneous polynomials. The following algorithm is not the most efficient for power-to-PAC conversion, but it is easily implemented and readily extends to deal with any polynomial basis. Furthermore, it is 'high-level' in that coefficients are not looked at explicitly, but rather higher level operations are examined such as polynomial multiplication.

First, the power equation must be put in homogeneous

form:

$$F(X, Y, W) = \sum_{i+j+k=m} a_{ij} X^i Y^j W^k$$

where  $x = X/W$  and  $y = Y/W$ . Note that the coefficients  $a_{ij}$  for the homogeneous polynomial are the same as for the nonhomogeneous polynomial, and that  $f(x, y) = F(X, Y, 1)$ . Euler's law states that for a degree  $m$  homogeneous polynomial,

$$F(X, Y, W) = \frac{X \times F_x + Y \times F_y + W \times F_w}{m}$$

where  $F_x$  indicates partial differentiation. Recall that for  $i + j + k = m$ ,

$$\frac{\partial^m F}{\partial X^i \partial Y^j \partial W^k} = i! j! k! a_{ij}$$

Then  $f(x, y) = 0$  can be expressed as a PAC  $g(s, t, u) = 0$  with vertices  $(x_{00}, y_{00})$ ,  $(x_{01}, y_{01})$ ,  $(x_{10}, y_{10})$  using the following recursive algorithm:

$$g(s, t, u) = \mathbf{C2B}(0, 0, 0)$$

where the function **C2B** returns a polynomial in Bernstein form in variables  $s, t, u$  and is defined:

**Function C2B**( $i, j, k$ )

**If**  $i + j + k = m$ , **then**

$$\mathbf{C2B} = i! j! k! a_{ij}$$

**else**

$$\mathbf{C2B} = \frac{\left[ \begin{array}{l} x(s, t, u) \times \mathbf{C2B}(i \times 1, j, k) + y(s, t, u) \\ \times \mathbf{C2B}(i, j + 1, k) + \mathbf{C2B}(i, j, k + 1) \end{array} \right]}{m - i - j - k}$$

**endif**

**end C2B**

In **C2B**,  $i, j, k$  indicate partial differentiation with respect to  $X, Y$  and  $W$  respectively. The partial differentiation is not explicitly performed at each level of recursion; it is simply noted that after  $m$  levels of differentiation have been performed, the answer is  $i! j! k! a_{ij}$ .

The  $\times$  operation indicates Bernstein polynomial multiplication. The functions  $x(s, t, u)$  and  $y(s, t, u)$  return the Cartesian coordinates  $x, y$  given the barycentric coordinates  $s, t, u$ :

$$x(s, t, u) = x_{10}s + x_{01}t + x_{00}u$$

$$y(s, t, u) = y_{10}s + y_{01}t + y_{00}u$$

## PAC INTERSECTION ALGORITHM

This section presents the algorithm for computing the points of intersection of two PACs defined over the same triangle. The algorithm iteratively shrinks the triangle until it converges to an intersection. This is clearly conveyed in the example illustrated in Figure 3 involving two quadratic PACs. Figure 3 shows the triangles generated by the first four iterations.

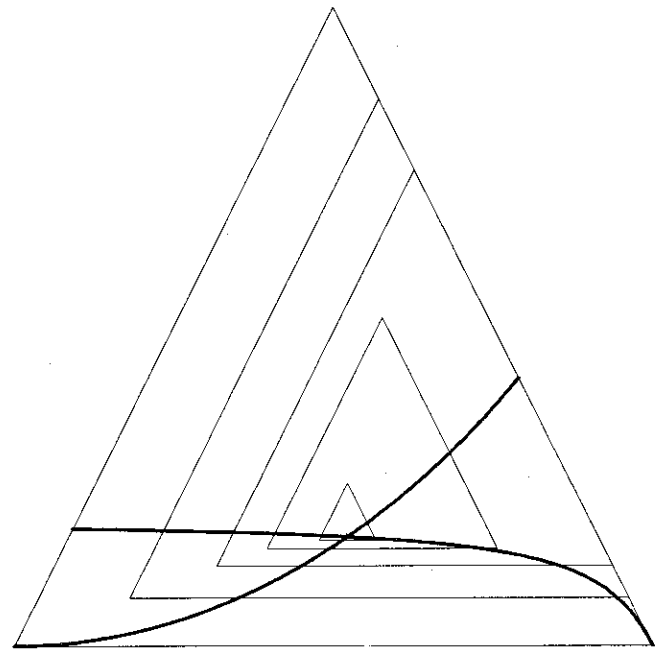


Figure 3. Intersecting two quadratic PACs

Table 1. Reduction of triangle areas for example shown in Figure 3

Iteration	Area	(Area) <sup>1/2</sup>
0	1.0	1.0
1	0.61	0.78
2	0.38	0.62
3	0.13	0.36
4	$7.9 \times 10^{-3}$	.089
5	$3.1 \times 10^{-5}$	.0056
6	$5.1 \times 10^{-10}$	$2.5 \times 10^{-5}$
7	$1.5 \times 10^{-19}$	$3.9 \times 10^{-10}$

subsequent iterations, the triangles are obscured by the curve pen width. The area of the triangles in this example reduces as shown in Table 1 (where the area of the initial triangle is defined to be 1). The square root of the area is proportional to the length of a triangle edge, so after seven iterations, the intersection point coordinates are known to ten places of accuracy.

An explanation is now given as to how to shrink the domain triangle *without excluding any intersection points*. Figure 4 shows the two quadratic PACs used in this example, along with their coefficient values. Denote the two curves by  $f_1(s, t, u) = 0$  and  $f_2(s, t, u) = 0$ . The pencil of these two PACs is the one parameter family of curves  $f_3(\alpha_1, \alpha_2, s, t, u) = \alpha_1 f_1(s, t, u) + \alpha_2 f_2(s, t, u) = 0$  where  $\alpha_1$  and  $\alpha_2$  are any two constants (not both zero).

Note that all curves in the pencil meet in the intersection points of  $f_1 = 0$  and  $f_2 = 0$ . Therefore, if a triangle edge can be moved inward such that some member of the pencil is not clipped in the process, then no intersection points will be clipped. Thus, a method is sought for determining a safe distance for moving each triangle edge inward such that in each case at

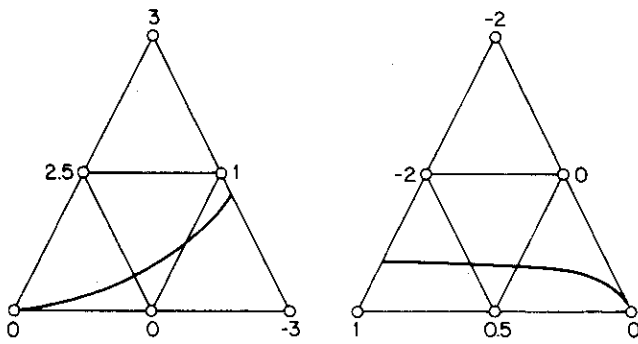


Figure 4. Two quadratic PACs

least one curve of the pencil remains unclipped. The following method of edge moving has the virtues of simplicity and robustness.

First a safe distance for translating the edge  $t=0$  (the bottom edge) is determined. There are many heuristics that could be used in identifying a pencil member that allows a healthy edge translation. A reasonable pencil member choice is the one for which  $c_{2,0,0} = c_{0,0,2} = 1$ . This choice works well because it is easy to solve for the values of  $\alpha_1$  and  $\alpha_2$  for which  $c_{2,0,0} = c_{0,0,2} = 1$ . Furthermore, in the limit as the triangle gets small, the surfaces behave like planes, so if we force  $c_{2,0,0} = c_{0,0,2} = 1$ , the chances are increased that the entire boundary curve  $t=0$  will lie above the  $z=0$  plane. In this example, the pencil member for which  $c_{2,0,0} = c_{0,0,2} = 1$  is given by  $\alpha_1 = -1/3$ ,  $\alpha_2 = 1$ . Figure 5 shows the resulting curve.

A safe distance for moving edge  $t=0$  can now be determined using the convex hull property of the triangular surface patch  $z = f_3(-1/3, 1, s, t, u)$ . The convex hull property states that the surface  $z = f_3(-1/3, 1, s, t, u)$  lies within the convex hull of the control points. Therefore, the curve  $f_3(-1/3, 1, s, t, u) = 0$  lies within the intersection of the plane  $z=0$  with the convex hull of the control points. Hence, the edge  $t=0$  can safely be translated parallel to itself to the limit of the convex hull.

Convex hulls of 3D control points are very costly to compute, especially for higher degree surfaces. The following is more efficient. View the surface  $z = f_3(-1/3, 1, s, t, u)$  orthographically along the line  $t=0$ . The plane  $z=0$  appears as a horizontal line as shown

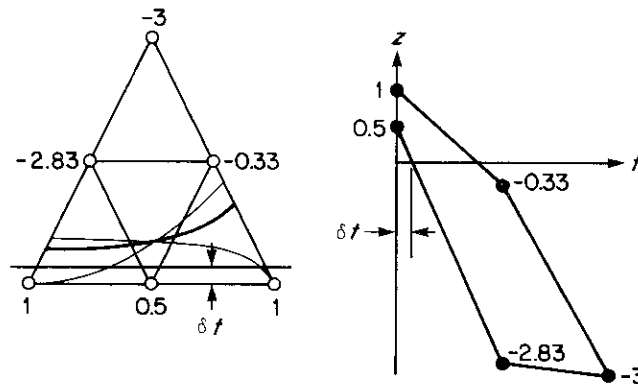


Figure 5. Moving the line  $t=0$  in pencil member  $f_3(-1/3, 1, s, t, u) = 0$

in Figure 5 and the projection of the surface control points lie in  $n+1$  columns. Clearly, the 2D convex hull of the projected control points contains the 2D projection of the convex hull of the 3D control points. Therefore, the edge  $t=0$  can safely be translated to the nearest point at which the convex hull intersects the line  $z=0$ .

The same process is used to determine a safe distance to move the other two edges. In every case, the pencil member is chosen that forces the two corner coefficients on the edge being translated to have a value of one. The translation of edge  $s=0$  is shown in Figure 6. Here, the pencil member with  $\alpha_1 = 1/3$  and  $\alpha_2 = 1$  is chosen. Finally, Figure 7 illustrates the attempted move of edge  $u=0$ . In this case, the convex hull straddles the  $z=0$  axis and the edge cannot be translated—even though the pencil member keeps its distance from  $u=0$ .

Once the translation distances have been computed, the PAC is subdivided by three applications of the de Casteljau algorithm and then undergoes another iteration of edge translation. Iteration halts when the triangle area reaches the prescribed tolerance.

It is acknowledged that the choice of pencil member (the one that forces the two corner coefficients on the edge being translated to have a value of one) does not necessarily provide the maximum translation. However, in the limit as the triangular surfaces approach planarity, it does produce a near-optimal translation. Even though a more thorough search could be performed to find the maximum safe translation, it is doubtful that it would improve the overall run time of the algorithm. It is further noted that the choice for the pencil member

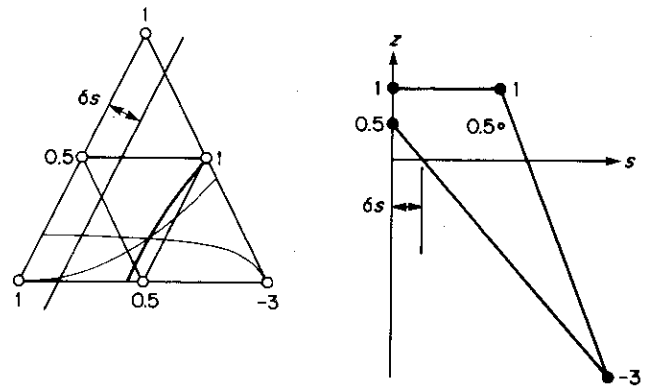


Figure 6. Moving the line  $s=0$  in pencil member  $f_3(1/3, 1, s, t, u) = 0$

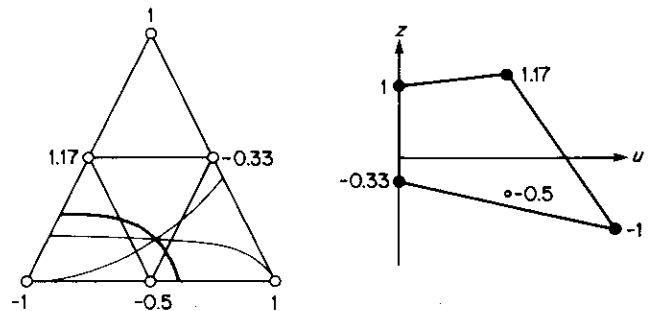


Figure 7. Line  $u=0$  cannot be moved in pencil member  $f_3(-1/3, -1, s, t, u) = 0$

may not even exist. In this rare event, that edge is simply not translated.

In actual implementation, it has been found to be more efficient to translate each edge independently, rather than to translate all three edges simultaneously. The reason for this is that translation of one edge usually increases the distance that the other two edges can be translated.

### Four-way split

If the shrinking moves too slowly (or not at all), more than one intersection may exist. In this case, the triangular domain is split and the shrink procedure is applied to the divided triangles. A four-way split is used to maintain similar triangles. However, a two-way split (splitting the longest triangle leg) is likewise a reasonable implementation choice.

In the neighbourhood of a simple intersection, each iteration results in an area reduction of several orders of magnitude. As a rule of thumb, a four-way split is applied if an iteration fails to result in an area reduction of 30%. Even if there is only one intersection (or none at all), this heuristic speeds up the algorithm.

### Degree elevation

The intersection algorithm requires that the degrees of the two curves be the same. If the degrees are different, one of the curves can be degree-elevated. A PAC of degree  $n$  can be represented as a PAC of degree  $n + 1$  by multiplying it by  $s + t + u$ . Since  $s + t + u = 1$ , this clearly does not alter the curve at finite points. (Actually, degree elevation has the effect of creating a reducible curve consisting of the original curve and the line at infinity.)

If the original PAC is degree  $n$  and has coefficients  $c_{ijk}$ , then the coefficients  $c'_{ijk}$  of the degree-elevated PAC are

$$c'_{ijk} = \frac{ic_{i-1,j,k} + jc_{i,j-1,k} + kc_{i,j,k-1}}{n+1}$$

Figure 8 shows a degree-two PAC and its degree-elevated version.

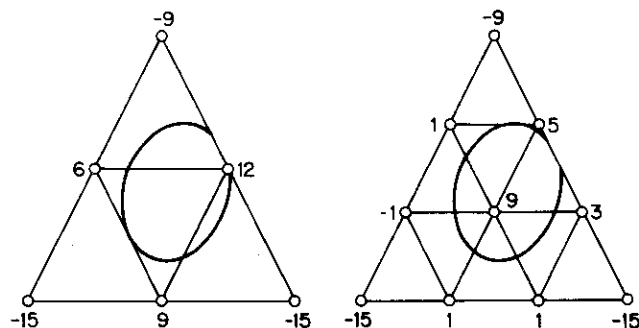


Figure 8. Cubic PAC as degree-elevated quadratic PAC

### Tangency

This paper does not address the question of how many times two curves meet at an intersection point. If the

intersection point is a simple point on each curve and if the curves are not tangent at that point, then the curves have single contact at that point. Various degrees of tangency result in multiple contact. Investigations continue as to how to approximate the degree of contact of PACs defined in floating-point arithmetic.

### POLAR CURVES

A thorough discussion of polar curves is presented by Salmon<sup>11</sup>. To define a polar curve, two things are needed: an algebraic curve  $F = 0$  and a point  $\mathbf{X}$ . The algebraic curve is typically defined in homogeneous power basis form:

$$F(x, y, w) = \sum_{i+j+k=n} c_{ijk} x^i y^j w^k = 0$$

The polar curve of  $F$  with respect to  $\mathbf{X} = (x', y', w')$  is denoted  $\Delta F(\mathbf{X})$ , and

$$\Delta F(\mathbf{X}) = x'F_x + y'F_y + w'F_w = 0$$

Polar curves have several practical uses. For one thing, the polar curve with respect to any point passes through all multiple points on  $F$ . Thus, double points of  $F$  can be located by applying the intersection algorithm to two polar curves of  $F$ . Also, the polar curve passes through all silhouette points of  $F$  as viewed from  $\mathbf{X}$ . A silhouette point is a point on  $F$  whose tangent line passes through  $\mathbf{X}$ .

The polar curves with respect to the domain triangle corners of a PAC happen to be trivial to compute. The polar of a degree- $n$  PAC with respect to the point  $s = 1$  is a PAC of degree  $n - 1$  over the same triangle whose coefficients are identical to those of the original PAC with the row of coefficients along the line  $s = 0$  deleted.

Figure 9 shows a cubic PAC with a double point,

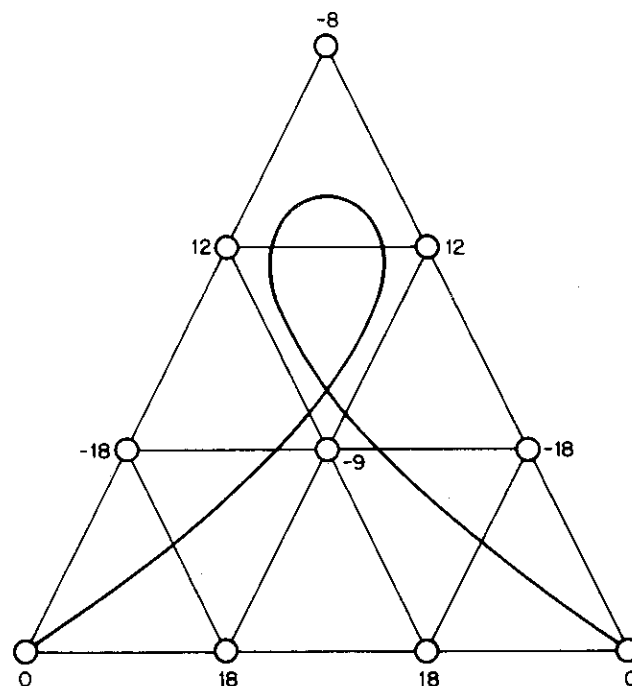


Figure 9. Cubic PAC with double point

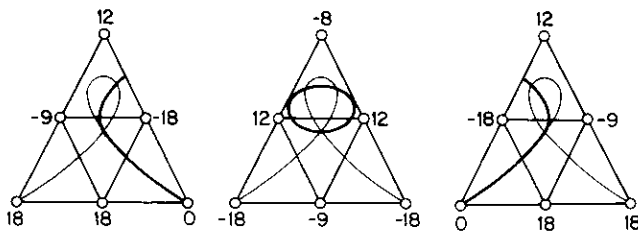


Figure 10. Polars of the PAC in Figure 9 with respect to  $s = 1$  (left),  $t = 1$  (middle) and  $u = 1$  (right)

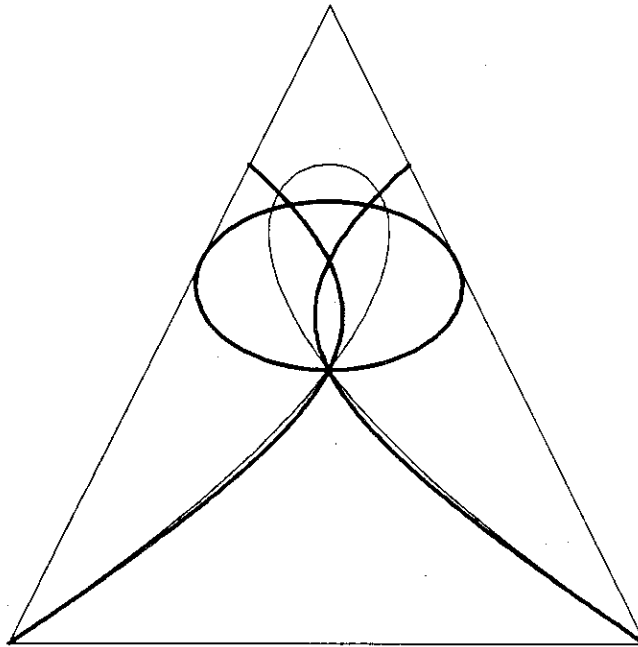


Figure 11. PAC with three polars

and Figures 10 show the polars of that PAC with respect to the three triangle corners. In each case, the original PAC is shown in fine pen width. Figure 11 shows the original PAC with all three polars.

### Double-point computation

Any two polar curves can intersect at points that are not double points on the original curve, but the three polar curves can only intersect at double points. Thus, the proposed algorithm for computing double points of a PAC is a slight variation of the PAC intersection algorithm. The approach taken is to intersect the polars with respect to the three triangle corners simultaneously. The edge translation distance is taken to be the maximum of the distance determined for each of the three pairs of polars.

Figure 12 shows the succession of triangles isolating the double point of the PAC in Figure 9. If the area of the original triangle is 1, the areas of the succeeding triangles are shown in Table 2. Note that progress was deemed too slow after the first iteration, and a four-way split was executed as the second iteration.

### Silhouette points

As mentioned, the polar curve of PAC  $F$  with respect to point  $\mathbf{X}$  passes through all silhouette points of  $F$  as

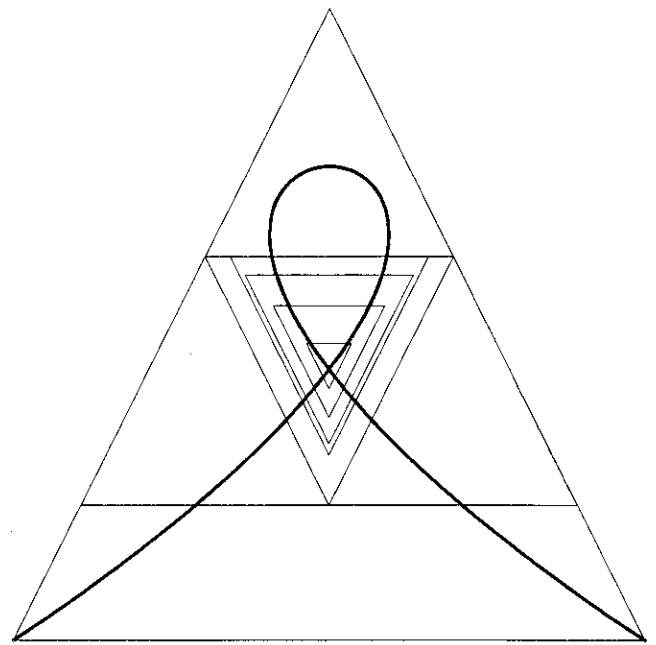


Figure 12. Double-point computation

Table 2. Areas of succeeding triangles for example shown in Figure 12

Iteration	Area	(Area) <sup>1/2</sup>
0	1.0	1.0
1	0.61	0.78
2	0.15	0.62
3	0.098	0.36
4	$7.1 \times 10^{-2}$	0.27
5	$3.1 \times 10^{-2}$	0.18
6	$5.1 \times 10^{-3}$	$7.1 \times 10^{-2}$
7	$1.2 \times 10^{-4}$	$1.1 \times 10^{-2}$
8	$5.8 \times 10^{-8}$	$2.4 \times 10^{-4}$
9	$1.4 \times 10^{-14}$	$1.2 \times 10^{-7}$

viewed from  $\mathbf{X}$ . A silhouette point is a point for which the line tangent to the PAC at that point passes through  $\mathbf{X}$ . Silhouette points can be computed by applying the intersection algorithm to  $F$  and  $\Delta F(\mathbf{X})$ . Since the degree of the polar curve is one less than the degree of  $F$ , the polar curve must be degree-elevated.

Figure 13 shows a cubic PAC and its polar with respect to the point  $t = 1$ . The polar is drawn in finer pen width. Also shown are the lines from point  $t = 1$  which are tangent to the PAC. Those points of tangency are the silhouette points. The detection of silhouette points can be used in computer graphics display of algebraic surfaces. The algebraic surfaces in Sederberg<sup>12</sup> were rendered using a scanline algorithm that computes silhouette points.

### Polars with respect to general points

The polar of a PAC

$$F = \sum_{i+j+k=n} c_{ijk} \binom{n}{ijk} s^i t^j u^k = 0$$

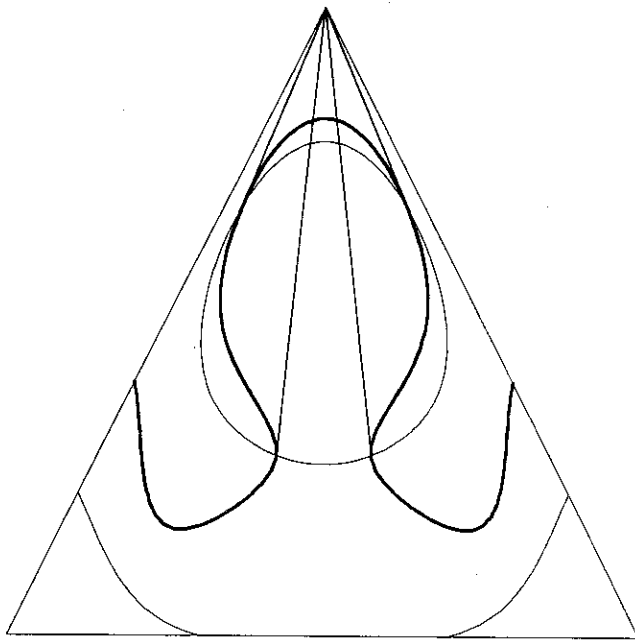


Figure 13. Silhouette-point computation

with respect to an arbitrary point  $\mathbf{P} = (s', t', u')$  is:

$$\Delta F(\mathbf{P}) = \sum_{i+j+k=n-1} (s'c_{i-1,k} + t'c_{i,j-1,k} + u'c_{i,j,k-1}) \times \binom{n-1}{ijk} s'^i t'^j u'^k = 0$$

The coefficients of  $\Delta F(\mathbf{P})$  are seen to be the coefficients arrived at by applying the first iteration of the de Casteljau algorithm, subdividing the PAC at  $\mathbf{P}$ .

Of special interest are polars with respect to points at infinity, which amount to directional derivatives of the PAC<sup>13</sup>. A point at infinity  $\mathbf{P}_\infty$  can be defined in terms of a direction: it is arrived at by travelling an infinite distance in the specified direction. Let  $(s_1, t_1, u_1)$  be a finite distance from  $(s_0, t_0, u_0)$  in the direction of  $\mathbf{P}_\infty$ . Then the polar of a PAC with respect  $\mathbf{P}_\infty$  is

$$\Delta F(\mathbf{P}) = \sum_{i+j+k=n-1} ((s_1 - s_0)c_{i-1,k} + (t_1 - t_0)c_{i,j-1,k} + (u_1 - u_0)c_{i,j,k-1}) \binom{n-1}{ijk} s'^i t'^j u'^k = 0$$

Polars with respect to points at infinity are useful for computing horizontal and vertical tangents on a PAC, or tangent points with respect to any other direction. Figure 14 shows a PAC and its polar with respect to the point  $x = \infty$ . The two curves meet at points on the PAC of horizontal tangency.

### HESSIAN CURVES

Another useful curve from classical algebraic geometry is the Hessian. A Hessian curve is computed from a given curve. If the given curve is degree  $n$ , then its Hessian curve is degree  $3(n-2)$ . An algebraic curve is intersected by its Hessian curve at its inflection points

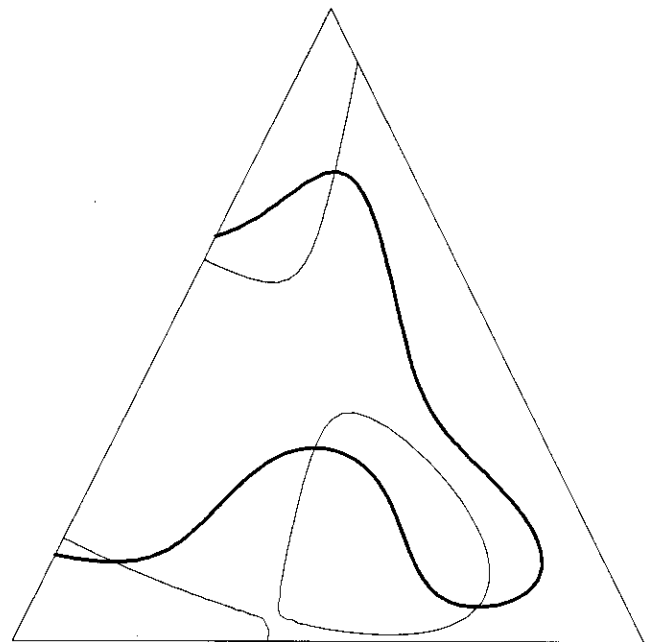


Figure 14. Horizontal tangency computation

and multiple points. Thus, the Hessian curve provides a means for computing the inflection points of an algebraic curve.

Salmon gives the equation of the Hessian of  $F(x, y, w) = 0$  as:

$$abc + 2fgh - a^2 - bg^2 - ch^2 = 0$$

where  $a = F_{xx}$ ,  $b = F_{yy}$ ,  $c = F_{ww}$ ,  $f = F_{yz}$ ,  $g = F_{xw}$ , and  $h = F_{xy}$ . To express the Hessian as a PAC, assuming the given curve  $F$  is a degree- $n$  PAC, note that  $a, b, c, f, g, h$  are themselves PACs of degree  $n-2$ . If the coefficients of  $F$  are  $F_{ijk}$ , then the coefficients of  $a, b, c, f, g, h$  are simply:

$$\begin{aligned} a_{ijk} &= F_{i+2,j,k} & b_{ijk} &= F_{i,j+2,k} & c_{ijk} &= F_{i,j,k+2} \\ f_{ijk} &= F_{i,j+1,k+1} & g_{ijk} &= F_{i+1,j,k+1} & h_{ijk} &= F_{i+1,j-1,k} \end{aligned}$$

Figure 15 shows a quartic PAC and its Hessian, which is a degree-six PAC. Note that the Hessian meets the PAC at the PAC's inflection points. The PAC intersection algorithm can thus compute those inflection points, though it must first degree-elevate the PAC to degree six.

### CONCLUSIONS

The algorithm presented herein for computing the intersection points of two planar algebraic curves is robust. For curves of degree less than five, intersection methods based on resultants work well and are much faster. The algorithm presented in this paper is less susceptible to floating-point error than the resultant method and should be reliable for curves up to degree 25, although the speed of the algorithm degrades significantly as the degree of the curves increases.

Since it is possible to express any parametric curve as a PAC (through implicitization), one may wonder

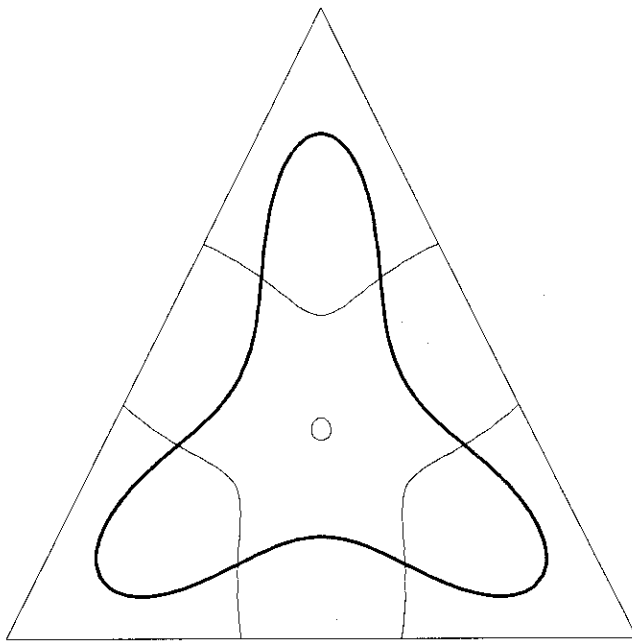


Figure 15. Degree-four PAC and its Hessian

if this PAC intersection algorithm might work for intersecting two parametric curves. The answer is that the algorithm does work, but it is much more efficient to simply use conventional algorithms for parametric curves.

An extension to this algorithm might be made to intersect two algebraic space curves, or three algebraic surfaces. An algebraic space curve is represented as the intersection of two algebraic surfaces, which in turn can be expressed using the 3D version of the PAC: the piecewise algebraic surface<sup>12</sup>. In this case, the algorithm would translate faces of a tetrahedron, rather than edges of a triangle.

## ACKNOWLEDGEMENTS

Alan Zundel created the figures. Malcolm Sabin and two referees deserve thanks for several valuable comments.

## REFERENCES

- 1 Sederberg, T W and Parry, S R 'Comparison of three curve intersection algorithms' *Comput.-Aided Des.* Vol 18 No 1 (1986) pp 58–63
- 2 Lane, J M and Riesenfeld, R F 'A theoretical development for the computer generation and display of piecewise polynomial surfaces' *IEEE Trans. RAMI* Vol 2 (1980) pp 35–46
- 3 Koparkar, P A and Mudur, S P 'A new class of algorithms for the processing of parametric curves' *Comput.-Aided Des.* Vol 15 No 1 (1983) pp 41–45
- 4 Kajiya, J T 'Ray tracing parametric patches' *Comput. Graph.* Vol 16 (1982) pp 245–254
- 5 Sederberg, T W 'Piecewise algebraic curves' *Comput. Aided Geom. Des.* Vol 1 (1984) pp 241–255
- 6 Goldman, R N and Sederberg, T W 'Some applications of resultants to problems in computation geometry' *Visual Computer* Vol 1 (1985) pp 101–107
- 7 Farouki, R T and Rajan, V T 'On the numerical condition of algebraic curves and surfaces. 1. Implicit equations' *IBM Research Report RC 13263* (1987)
- 8 Bajaj, C L, Hoffmann, C M, Hopcroft, J E and Lynch, R E 'Tracing surface intersections' *CSD-TR-728* Computer Science Department, Purdue University, West Lafayette, IN, USA (1987)
- 9 Geisow, A 'Surface interrogations' *PhD Thesis* University of East Anglia, Norwich, UK (1983)
- 10 Waggenspack, W N Jr and Anderson, C D 'Converting standard bivariate polynomials to Bernstein form over arbitrary triangular regions' *Comput.-Aided Des.* Vol 18 No 10 (1986) pp 529–532
- 11 Salmon, G *Higher Plane Curves* Stechert, New York, USA (1934)
- 12 Sederberg, T W 'Piecewise algebraic surface patches' *Comput. Aided Geom. Des.* Vol 2 (1985) pp 53–59
- 13 Farin, G 'Triangular Bernstein-Bezier patches' *CAGD Report 11-1-85* Department of Mathematics, University of Utah, Salt Lake City, UT, USA (1985)