Techniques for Conic Splines

Vaughan Pratt
Sun Microsystems Inc.

Abstract

A number of techniques are presented for making conic splines more effective for 2D computer graphics. We give a brief account of the theory of conic splines oriented to computer graphics. We make Pitteway's algorithm exact, and repair an "aliasing" problem that has plagued the algorithm since its introduction in 1967. The curvature-matching problem for conics is solved by way of a simple formula for curvature at an endpoint which permits curvature to be matched exactly at non-inflectior points and more closely than was previously realized possible at points of inflection. A formula for minimum-curvature-variation of conic splines is given. These techniques provide additional support for Pavlidis' position [6] that conics can often be very effective as splines.

This work was motivated by, and provides much of the foundation for, an implementation of conic splines at Sun Microsystems as part of Sun's Pixrect graphics package, the lowest layer of Sun's graphics support.

1. Introduction

1.1. Rationale for Conic Splines

It may be of interest to understand how we arrived at both the techniques and the conclusions of this paper. We set out a year ago to develop an outline font system. We considered many families of curves [5,11], but rejected most of these as either computationally impractical or unsuited for splines. It appeared that the most suitable curve technology from a user standpoint was cubic splines. We soon found however that incremental algorithms for plotting (e.g. Pitteway's algorithm for conics [7]) were significantly faster than the recursive subdivision algorithms for parametric curves (e.g. the various algorithms discussed by Catmull [2]). The former had excellent velocity control (constant relation between number of pixels plotted and amount of work done), no context switching, and a trivial stopping condition. The latter had simple implementations, both in hardware and high level programming languages, along with good ways to organize the arithmetic. However they had poor velocity control (even with an optimal stopping condition for the recursion one can only come to within at best a factor of two of the ideal velocity), expensive context switching (for every single step there is a context switch entailing much pushing and popping of data on a stack, aggravated by trying to program cleanly by using recursion), and an awkward stopping condition (there is a complex tradeoff between cost of stopping condition and quality of velocity control).

It was natural therefore to want to combine the user convenience of splines with the performance of incremental methods. Since splines were defined for parametric curves whereas incremental plotting was for algebraic curves it was clear that some combination of these classes was called for.

As degree increases the complexity of both exact implicitization and of exact and antialiased Bresenham-Pitteway style tracking increases very rapidly. We feel particularly intimidated by the prospect of duplicating the performance-oriented techniques of this paper for rational or even polynomial cubics.) Hence one does not casually increase degree from two to three without good reason.

At higher degrees higher orders of continuity become possible. Two important properties of polynomial cubics are the possibility of C_2-continuity between adjoining cubics (as easily achieved using B-splines), and zero curvature at inflection points. C_3-continuity is however only possible between adjoining cubics trivially, namely when the two cubics are adjoining segments of a common cubic.

We developed an easy method of achieving C_2-continuity between conics. We also worked out how to approximate very closely zero curvature at inflection points. Hence for degree of continuity cubics and conics appear to have essentially the same capabilities.

The one thing cubics have that conics do not is an additional degree of freedom. Six degrees of freedom are encompassed in the basic endpoint-and-tangent conditions. For conics the seventh degree of freedom is sharpness or bulge. Cubics have bulge along with an additional degree of freedom that might be called skew, allowing their interior to be pushed in a direction parallel to the line between the endpoints, one way to get inflection points in cubics. (The other is that cubics can bulge further than conics, producing first inflection points, then a cusp, then a loop.) It appeared to us that on those occasions calling for this control one could get its effect quite satisfactorily via a single subdivision, so its loss was not an unmitigated disaster. More pragmatically, we doubted whether many users would be comfortable visualizing skew as a concept independent of the endpoint tangents; even bulge is somewhat obscure in that regard.

These considerations made rational quadratics look reasonable similar to polynomial cubics in practice. The very much greater complexity of dealing with implicit cubics for tracking led us to adopt conics as a useful curve form for high-performance graphics. We are not alone in advocating conics as a useful curve form; Pavlidis [6] is also an enthusiastic proponent of the virtues of conics.

1.2. Conics and Cubics in Graphics Standards

Recent efforts to integrate our conic package into Sun's standard graphics packages have led us to ponder the proper way to introduce curves into graphics standards. We feel sufficiently strongly about our conclusions as to warrant including a brief summary of them in this forum.

Curves are not provided at all in such graphics standards as CORE or GKS, other than as unspecified nonstandard extensions. CGI (formerly VDI) has circles and ellipses, but no splines of any type (polynomial or rational) or degree (beyond 1). Furthermore those circles and ellipses can be used as region boundaries only in the most parochial ways, namely as pies (radius-arc-radius) and chords (chord-arc), rather than the far more general and useful approach of generalizing polygons to piecewise-elliptical region boundaries. Not one of these standards has useful compound region boundaries for curves of degree higher than one.

Without some nonlinear spline capability all of these standards are very difficult to use for curve-oriented work of the kind arising in say outline font design.

The reasons given above for conic splines, together with the observation that CGI has conics (ellipses) but not cubics, suggests that it is not unreasonable to begin the introduction of splines to such standards with conic splines. We therefore advocate extending the existing line capabilities of graphics standards to include conic splines. This would constitute a simply described yet very useful extension to those standards.

Given the reluctance of some of the graphics standardizing bodies to introduce even quadratic curves into graphics standards, it would seem appropriate to increase curve degree in graphics reasonably cautiously. To go from linear to cubic curves in one step is akin to going from crawling to running without learning to walk.

Before turning our attention to the theory of conic splines let us mention some of the more notable works on conics relevant to computer graphics. For general information on conics an excellent text is Salmon's classic [13]. A scholarly account of both the subject and its many contributors is given by Coolidge [4]. A comprehensive though at times dense text on projective geometry is Todd [10]; it is the source of most of our ideas about conics. General implicitization methods, based on classical techniques, for both polynomial and rational curves can be found in Sederberg's thesis [8]; our methods below are more narrowly focused on conics per se, with an emphasis on performance. An early but excellent work on coping with rational quadratic and cubic curves in the context of computational geometry is Forrest's thesis [3].

2. Principles of Conic Splines

A conic spline is a conic defined by three points *A,B,C* in the view plane. It passes through **A** and **C** and has tangents *AB* and *CB* respectively at those points. The family of conics satisfying these conditions, which we call the *ABC* family, is illustrated in Figure 1.

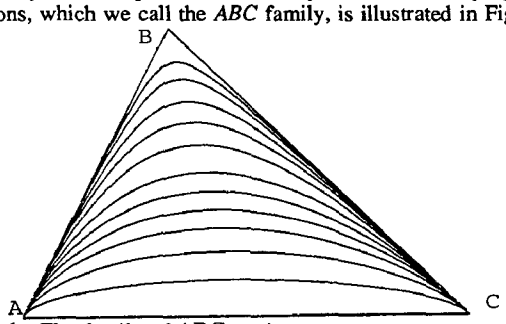

Figure 1. The family of ABC conics

This family may be indexed in a variety of ways. Some authors use the scalar ratio *WD/WB* where **W** is the midpoint of *AC* and **D** is the intersection of the conic with *WB*. Another method is simply to provide **D** directly, or for that matter to give any point on the conic besides **A** and **C**. We have found the scalar *S*, for *sharpness*, to be useful. *S* measures the departure of the conic from a parabola. For $S = 1$ we have the parabola itself, $S < 1$ (the flatter curves) yields ellipses, $S > 1$ (the sharper curves) hyperbolas. Pavlidis [6] uses a scalar *K*, representable in terms of our *S* as $K=1/4S^2$. Another parameter we have found useful is what we call *variation*, which measures the departure of the conic from the ellipse of minimum eccentricity; variation is directly proportional to sharpness, but with the constant of proportionality a function of the shape of triangle. Some authors have proposed to use eccentricity itself as the index; unfortunately this index is ambiguous since eccentricity attains a minimum for some positive *S*.

The most useful property of conic splines for applying Pitteway's algorithm is its implicit or *algebraic* form. If we take the vertices of the triangle as our basis, with corresponding coordinate variables a,b,c, then the equation is (as in (3.13) of [6])

$$b^2 - 4S^2ac = 0 \qquad (2.1)$$

This is actually the equation not just of the conic but of the whole surface consisting of the lines of sight from the observer **O** (obligingly seated at the origin) to points on the conic. This surface is a cone.

Since **ABC** will not in general be the coordinate system in which the rendering is to be done, a change of basis from **ABC** to **XYZ** is needed. This is accomplished by making the following substitutions for *a,b,c* (cf. (3.10) of [6]).

$$a = \begin{vmatrix} x & x_B & x_C \\ y & y_B & y_C \\ z & z_B & z_C \end{vmatrix} \quad b = \begin{vmatrix} x_A & x & x_C \\ y_A & y & y_C \\ z_A & z & z_C \end{vmatrix} \quad c = \begin{vmatrix} x_A & x_B & x \\ y_A & y_B & y \\ z_A & z_B & z \end{vmatrix} \qquad (2.2)$$

Each of these substitutions is a linear expression in *x,y,z*, whence making those substitutions in equation (2.1) turns it into a quadratic equation in x,y,z, which we express as

$$\alpha x^2 + 2\beta xy + \gamma y^2 + 2\delta xz + 2\varepsilon yz + \zeta z^2 = 0. \qquad (2.3)$$

Furthermore if the **XYZ** coordinates of *A,B,C* are integers and S^2 is a rational then the coefficients in (2.3) will all be integers, with the coefficients of the "mixed" terms *xy,yz,zx* being even, whence the 2's. Since *A,B,C* all lie on the view plane $z = 1$, we have $z_A = z_B = z_C = 1$.

Finally we intersect the cone with the view plane $z = 1$ simply by setting z to 1 in the equation for the cone to yield

$$\alpha x^2 + 2\beta xy + \gamma y^2 + 2\delta x + 2\varepsilon y + \zeta = 0. \qquad (2.4)$$

This could have been done earlier in (2.2) just by having the whole bottom row be all 1's; leaving z in until the end shows that the 1's arise through intersecting the view plane $z = 1$ with the cone.

The practically-minded reader may now skip to the next section. The purpose of the remainder of this section is to provide additional insight into conic splines.

First let us derive (2.1) directly from the definition of a conic as the intersection of a cone with a plane. Since this is a 3D construction we do this in a 3D vector space with origin the point **O**. The view plane resides in this space and contains **A,B,C** which therefore uniquely determine the view plane (unless **A,B,C** are collinear), and may also be taken as a basis for the space (unless **O,A,B,C** are coplanar).

So let $\mathbf{U} = (\mathbf{A}-\mathbf{C})/2$, $\mathbf{V} = S\mathbf{B}$, $\mathbf{W} = (\mathbf{A}+\mathbf{C})/2$ (which may be seen to form a basis provided **ABC** form a basis and *S* is nonzero) with associated coordinates *u,v,w* and associated substitutions $u = a-c$, $v = b/S$, and $w = a+c$. The surface $u^2+v^2 = w^2$ is evidently a circular cone (with respect to the **UVW** basis) containing **A** and **C** (the

points (1,0,1) and (−1,0,1)) and having *OAV* and *OCV*, and hence *OAB* and *OCB*, as tangent planes at **A** and **C** respectively. We then derive (2.1) by making the above substitutions for *u,v,w* in this equation. This is nicely illustrated by Figure 2, which also shows two other important points **D** and **E** at **W**±**V** (note that **A** and **C** are at **W**±**U**).

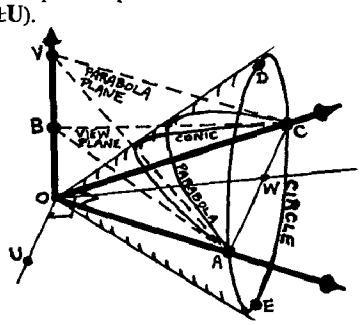

Figure 2. The *AVC* cone, *AVC* parabola, and *ABC* conic

Substitution (2.2) may be derived in terms of *areal coordinates*. A closely related notion is that of *barycentric coordinates* in which a point **P** may be specified on a plane by giving its coordinates as a weighted sum $\mathbf{P} = a\mathbf{A}+b\mathbf{B}+c\mathbf{C}$ of three points *A,B,C* in the plane with $a+b+c = 1$ (normalized weights). The concept of areal coordinates is the idea that the coordinates *a,b,c* may be specified as the areas of the triangles *PBC*, *PCA*, and *PAB* respectively, or to fit (2.2), twice these areas. Areal coordinates are related to barycentric coordinates by a common ratio amounting to twice the area of the ABC triangle, given by

$$\begin{vmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{vmatrix} \qquad (2.5)$$

Substituting **P** (denoted by omitting the subscript) for each of **A**, **B**, or **C** in this determinant, yields the determinant for the corresponding coordinate *a*, *b*, or *c*. The resulting weights are not normalized. They could be normalized by dividing by twice the area of **ABC**, but since (2.1) is homogeneous this is unnecessary. Most importantly, in the unnormalized form integer **XYZ** coordinates of **ABC** yield integer coefficients in (2.3).

When z's are used in place of 1's these areal coordinates become volumetric coordinates for a point moving in all 3 dimensions, the volumes being those of tetrahedra having O as the additional vertex (besides the three appearing in the determinant).

(Our original SIGGRAPH submission derived the equivalent of (2.2) by dropping the common denominator (twice the area of ABC) from Cramer's rule as used to obtain the transformation from **XYZ** coordinates to **ABC** coordinates as the inverse of the **ABC**-to-**XYZ** tranformation, which is in effect what one has when given the **XYZ** coordinates of **A**,**B**,**C** arranged as a matrix. A pointer from T. Pavlidis, one of the referees, to the determinant form of his equation 3.10 of [6] converted us from Cramer's rule to the formally equivalent but more appealing account of this in terms of areal coordinates.)

Parametric Representations

A parametric representation of a curve is an expression for a point in terms of an independent parameter t. We easily obtain a parametric representation of a conic from a parametric representation of a point (x,y,z) on the cone, via the standard projection $(x,y,z) \rightarrow (x/z,y/z)$. This is fortunate since parametric representations of the cone are simpler than of the conic, due to the projection's being omitted. This constitutes a major motivation for applying projective geometry and homogeneous coordinates to conics.

There are several worthwhile parametric representations of the cone. They are all most easily introduced via **UVW** coordinates. First, *u* and *v* themselves immediately provide a fine pair of independent parameters, making $w = \pm\sqrt{u^2+v^2}$ the one dependent variable. The cone is then the parametrically defined surface $u\mathbf{U}+v\mathbf{V}+\sqrt{u^2+v^2}\mathbf{W}$. The points **A**,**C**,**D**,**E** on the cone are given by the respective parameter values (1,0), (−1,0), (0,1), and (0,−1). This parametrization is readily transformed into other coordinate systems such as **ABC** and **XYZ** by making the appropriate substitutions for **UVW** in $u\mathbf{U}+v\mathbf{V}+\sqrt{u^2+v^2}\mathbf{W}$, e.g. $\mathbf{U} = (\mathbf{A}-\mathbf{C})/2$, $\mathbf{V} = S\mathbf{B}$, $\mathbf{W} = (\mathbf{A}+\mathbf{C})/2$ to get to **ABC** coordinates.

Another parametrization makes *u* and *v* dependent on polar coordinates *r* and θ as $u = r\cos(\theta)$, $v = r\sin(\theta)$, yielding the form $r\cos(\theta)\mathbf{U}+r\sin(\theta)\mathbf{V}+r\mathbf{W}$ for the cone, with **A**,**C**,**D**,**E** all at $r = 1$ and having θ respectively 0, π, π/2, and 3π/2. The substitutions for **UVW** to transform this parametrization into other coordinates are as before. This parametrization is useful in relating conic splines to uniform motion around the circle from which the conic spline has been projected. It replaces the square root in the uv parametrization by trig functions, if that is an advantage.

The parametrization that is most popular for splines is the *st* parametrization, obtainable via $s+it = \sqrt{u+iv}$. Squaring both sides yields $u = s^2-t^2$, $v = 2st$, so $w = s^2+t^2$. While this parametric form of the conic is not particularly attractive in **UVW** coordinates, transforming it to **ABC** coordinates yields the very appealing $s^2\mathbf{A}+2st\mathbf{S}\mathbf{B}+t^2\mathbf{C}$, that is, $a = s^2$, $b = 2Sst$, $c = t^2$. Observe that $a+b/S+c = (s+t)^2$. The *AVC* plane is $a+b/S+c = 1$, whence it may also be specified as $s+t = \pm 1$. If we therefore substitute $1-t$ for s in the cone we obtain $(1-t)^2\mathbf{A}+2(1-t)t\mathbf{S}\mathbf{B}+t^2\mathbf{C}$, immediately recognizable as the Bezier parabola with vertices **A**,*S***B**,**C**. Other constant values of $s+t$ yield planes parallel to this, with $s+t = 0$ yielding the plane tangent to the cone at **E**.

2.1. Tangent Planes

One application of the *st* parametric form is to tangent planes to the cone. The partial derivatives of the parametric form of the cone $s^2\mathbf{A}+2st\mathbf{S}\mathbf{B}+t^2\mathbf{C}$ with respect to *s* and *t* are $2s\mathbf{A}+2St\mathbf{B}$ and $2Ss\mathbf{B}+2t\mathbf{C}$ respectively. The plane tangent to the cone at (s,t) is then the plane containing the three points **O**, $s\mathbf{A}+St\mathbf{B}$ and $Ss\mathbf{B}+t\mathbf{C}$, the 2's being of no consequence.

Tangent planes to the cone provide a straightforward way of finding tangents to the conic. Suppose we wish to locate the two points on a conic having a given slope. Let **P** be any point other than **O** on a line through **O**, parallel to the view surface, and having the given slope. Then the intersection of the view surface, the cone, and one of the two planes tangent to the cone and containing **P**, will be one of the two points where the conic has that slope.

If **A**,**B**,**C**,**P** are all represented in **XYZ** coordinates, we may express the coplanarity of the three points $s\mathbf{A}+St\mathbf{B}$, $Ss\mathbf{B}+t\mathbf{C}$, and **P** as the vanishing of the determinant whose columns are the **XYZ** coordinates of those three points.

$$\begin{vmatrix} x_P & sx_A+tSx_B & sSx_B+tx_C \\ y_P & sy_A+tSy_B & sSy_B+ty_C \\ z_P & sz_A+tSz_B & sSz_B+tz_C \end{vmatrix} \qquad (2.6)$$

The determinant reduces to a homogeneous quadratic in *s* and *t*, whose roots constitute the two lines of tangency in the cone. The two points of tangency on the conic may then be obtained by setting $s+t = 1$.

For the points of horizontal tangency, $\mathbf{P} = \mathbf{X} = (1,0,0)$ in **XYZ** coordinates. In this case the determinant reduces to $S(y_A-y_B)s^2+(y_A-y_C)st+S(y_B-y_C)t^2$. For vertical tangents use **x**

coordinates in place of y; for any other angle take the appropriate linear combination of these two forms for horizontal and vertical tangents.

2.2. Conic Determined by Additional Points

A common problem is to find the sharpness of the *ABC* conic containing an additional point **F**, with **A,B,C,F** all lying in the view plane. The solution is to substitute the **XY** coordinates of **F** for x and y in (2.4) and solve the resulting linear equation in S.

A related problem is to find an implicit form for the conic containing five distinct points **A,B,C,D,E**. Again there is an immediate solution:

$$\begin{vmatrix} x^2 & xy & y^2 & x & y & 1 \\ x_A^2 & x_Ay_A & y_A^2 & x_A & y_A & 1 \\ x_B^2 & x_By_B & y_B^2 & x_B & y_B & 1 \\ x_C^2 & x_Cy_C & y_C^2 & x_C & y_C & 1 \\ x_D^2 & x_Dy_D & y_D^2 & x_D & y_D & 1 \\ x_E^2 & x_Ey_E & y_E^2 & x_E & y_E & 1 \end{vmatrix} = 0 \qquad (2.7)$$

The determinant is a quadratic in x and y, so the equation is that of a conic. When $(x,y) = \mathbf{A}$ the first two rows are identical, so the determinant vanishes, so **A** is a solution. Similarly **B,C,D,E** are solutions. Hence the conic defined by this equation contains those five points.

When the five points are all lattice points the resulting equation will have integer coefficients, which may be computed exactly using only addition and multiplication. Hence we have a method of computing exactly an implicit form of the desired conic, as well as a short proof that any conic containing five distinct lattice points has an implicit form with integer coefficients.

This method is simple to implement, but is not particularly fast, since it involves computing six 5×5 determinants, a size better handled by triangulation than by brute force. Furthermore the magnitudes of the coefficients of x^2, xy, and y^2 grow as the sixth power of the magnitudes of the coordinates, those of x and y the seventh power, and the constant the eighth power. Hence to capitalize on the full precision of the method requires multiple precision arithmetic for all but the smallest conics. In practice 32-bit floating point, i.e. shedding precision at the low end, should provide a satisfactorily accurate approximation to the desired conic.

2.3. Conversion to Normal Form

It will be noticed that having **A** and **C** on the plane plays no significant role. Any parabola projects to a conic whether or not its endpoints coincide with those of the conic. Having **A** and **C** on the plane merely constitutes a normal form representation (though one that simplifies some calculations later).

On occasion we encounter points not in normal form, most often as the result of subdividing a conic by subdividing its underlying parabola. We then need to restore them to normal form

To put three arbitrary points **A,B,C** into normal form, first rescale the space by replacing all three points **A,B,C** by $p\mathbf{A}, p\mathbf{B}, p\mathbf{C}$ where $p = 1/\sqrt{z_Az_C}$, to drive z_Az_C to 1. Then multiply **A** by z_C and divide **C** by z_C to yield the desired normal form. It is readily checked that neither operation on these points alters the surface defined by $b^2-4ac = 0$ in the corresponding bases.

Note that the only impact of these two operations on the **ABC** axes has been to scale them; their points of intersection with z=1 remain unchanged.

We may now easily apply this to the problem of subdividing the conic. Subdivide the corresponding Bezier parabola $(1-t)^2\mathbf{A}+2(1-t)tS\mathbf{B}+t^2\mathbf{C}$ in the usual way. This yields two parabolas. These may each be projected back to two halves of our conic by the above conversion to normal form.

3. Curvature

We distinguish two curvature issues, external and internal. External curvature deals with the curvature at endpoints. It affects curvature matching between curves. Internal curvature deals with curvature along a single conic. It affects the amount by which curvature varies around the conic.

3.1. External (Endpoint) Curvature

It is often important to be able to match up curve segments not only in slope but in curvature. For cubics and up, B-splines make this very easy - curvature matching is obtained automatically as a byproduct of the representation. This method can be seen to be generalizable to parametric rational curves if we regard both the curves and their control points as the perspective projection of polynomial B-splines.

This method does not start working until degree three since parabolas (polynomial quadratics) cannot be curvature-matched, in the following sense. Given a sequence of n+1 points with slopes specified at those points, there does not in general exist a corresponding sequence of n parabolas (parametric quadratics) connecting those points and having the specified slopes there, such that curvature varies continuously around the curve (C_2–continuity). In fact these constraints uniquely determine those parabolas, making this limitation obvious.

It follows that the approach of projecting a C_2–continuous polynomial curve to yield a C_2–continuous rational curve fails for quadratics because of the unavailability of appropriate polynomial curves to project. Hence piecewise-conic curves cannot be made C_2–continuous by this method.

Nevertheless C_2–continuity of piecewise-conic curves, with specified location and slope of junctions, can always be achieved by adjusting only sharpness. As the sharpness S of a conic goes from zero to infinity the curvature at both endpoints can be seen to go from infinity to zero. For n conics connecting n+1 given point-slope pairs, there remain n degrees of freedom, namely the sharpnesses of those conics. We may adjust in turn the sharpness of each conic but the first to match the curvature of its predecessor. This consumes n-1 degrees of freedom to achieve C_2–continuity, leaving the sharpness of the first curve (now linked to the n-1 other sharpnesses) as the remaining degree of freedom for the whole curve.

For this approach it suffices to have a formula for curvature at endpoints. Writing Δ for the area of the triangle *ABC* and a,b,c for the lengths of the sides BC,CA,AB respectively, the curvature K at **A** measured in **XY** coordinates is given by the formula

$$K = \frac{\Delta}{S^2c^3} \qquad (3.1)$$

We may derive this as follows. Rotate and translate the **XY** coordinate system (preserving curvature) to put the origin at **A** and the **X** axis on *AB*. Now take the parametric form of the conic, which will be two rational quadratics in t, one for each of x and y, namely

$$x = \frac{2(1-t)tSx_B+t^2x_C}{1+2(S-1)(1-t)t}, \quad y = \frac{t^2y_C}{1+2(S-1)(1-t)t} \qquad (3.2)$$

Differentiate these expressions with respect to t and evaluate the resulting derivatives at $t = 0$ to obtain $\dot{x} = 2Sx_B$, $\dot{y} = 0$, and $\ddot{y} = 2y_C$. The usual formula $\dot{x}^2 = \ddot{y}r$ relating tangential velocity $\dot{x}$,

radial acceleration $\ddot{y}$, and radius r when $\dot{y} = 0$ leads us to $K = y_C/2S^2x_B^2$ where K is curvature, the reciprocal of the radius r. Finally we use $\Delta = x_B y_C/2$ to obtain (3.1).

Besides its application to exact matching, the formula yields much insight into nonmatching, at opposite ends of the same conic as well as at junctions.

Let us consider first opposite ends of the same conic. We shall show how to build a "curvature transformer."

By symmetry the curvature at C must be Δ/S^2a^3. Hence the ratio of the curvature at A to that at C must be the cube of a/c, a quantity independent of everything but the lengths of *AB* and *BC*. Thus to build a "curvature transformer" consisting of an *ABC* conic to connect two points A and C at which the curvatures are in the ratio of 8 to 1 in a C_2–continuous way, with the smaller curvature at A, a necessary and sufficient condition (not including other requirements such as C_1–continuity) is that *AB* be twice as long as *BC*.

Now let us consider junctions, and address the problem of C_2–continuity at inflection points. Points of zero curvature are nonexistent in conics other than lines. Hence piecewise-conic curves cannot be C_2–continuous at inflection points. This is a major reason for preferring cubics to conics.

The above formula for curvature however gives us some idea of how closely zero curvature at a point of inflection can be *approximated.* Whether conic or cubic, *a major use of splines is to approximate curves*, to within acceptable visual or numerical limits. Hence we are not breaking any rules when we ask whether a good approximation is possible.

Whatever the prevailing curvature is at some distance removed from a point of inflection, a curvature transformer can be used to connect points on either side of an inflection in such a way that the curvature at the inflection is very small by comparison to the prevailing curvature. This can be done with a quite reasonable choice of ratio of sides; by the time the ratio reaches 3 to 1 the curvature at the inflection will be 1/27 the prevailing value.

This argument, together with its application to examples we have seen with outline fonts defined by conics, has convinced us that this ability to approximate a zero-curvature point of inflection so closely disposes almost entirely of the objection that conics lack such points. In support of this we ask the reader to inspect the following characters from a conic representation of Times Roman (Figure 3) to decide whether the points of inflection exhibit any objectionable or even detectable curvature discontinuity. The attached spline skeletons show the locations of the ABC control points (though not the sharpnesses).

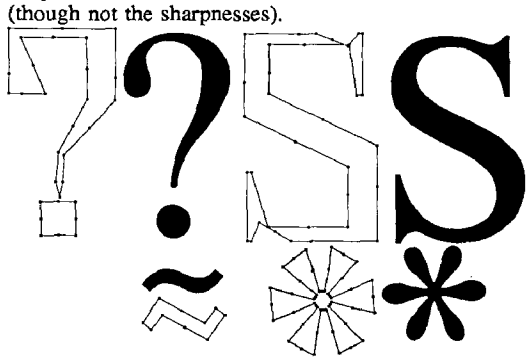

Figure 3. Conic-spline Characters With Points of Inflection

3.2. Internal (Variational) Curvature

We are interested in this section in the smoothness of curves. A plausible measure of smoothness is degree of variation of curvature around the curve. Extremely sharp or flat conics are obviously unreasonable candidates for "reasonably smooth" curves. At the elliptical extreme (low sharpness) the total curvature of the curve (the angle between its endpoint tangents) is concentrated at the endpoints, whereas at the hyperbolic extreme it is concentrated in the middle. Such extremes are far from "reasonably smooth."

One might settle for $S = 1$, a parabola, as a somewhat arbitrary but convenient intermediate value. Unfortunately, for curves with total curvature approaching 180 degrees (angle *ABC* approaching zero), a parabola suffers from the same problem as an extreme hyperbola, concentrating the curvature in the middle. Furthermore, when $AB = BC$ (isosceles triangle) there is an obvious candidate for the conic of least curvature variation, namely the circle, which has no curvature variation and is easy to represent, manipulate, plot, and understand, and, only partly for these reasons, is in much demand.

This is not to say that minimum variation is always what is wanted. Font designers often want superellipses. If we measure superellipticity in terms of the sharpness of a conic approximating a quadrant of a letter's outline, a commonly preferred superellipticity is one lying somewhere between a circle and a parabola.

An obvious measure of curvature variation is ellipse eccentricity. One problem with this measure is that it attains infinity at the parabola; this puts a discontinuity in the middle of a range that goes from ellipses to hyperbolas. Another problem is that it is not the easiest measure to calculate with.

The measure we introduce here relates the actual sharpness to the sharpness that yields the ellipse of minimum eccentricity. If as before we let a,b,c be the lengths of sides BC,CA,CB respectively, this minimum-eccentricity sharpness is given by

$$S = \frac{b}{\sqrt{2(a^2+c^2)}} \qquad (3.3)$$

This formula was arrived at after much conversation with Macsyma, and we have been unable to construct a summary of the rationale behind it.

An equivalent criterion for minimum eccentricity told to us by Lyle Ramshaw is when the line from the center of the ellipse to a corner of the rectangle bounding the ellipse and aligned with the major and minor axes also passes through the B vertex of the triangle. We have not yet verified the equivalence of Ramshaw's nonmetrical criterion with our metrical one. However we have found the metrical one easy to apply.

For a right triangle, meaning here that *ABC* is a right angle, $a^2+c^2 = b^2$ by Pythagoras, so $S = 0.7071...$ for the ellipse of minimum eccentricity. From Ramshaw's criterion, this ellipse will be one quadrant of an ellipse whose major and minor axes are parallel to *AB* and *BC* (not necessarily respectively).

For an isosceles triangle, meaning here that $a = c$, there exists an *ABC* conic which is a circular arc from A to C. Since its eccentricity is 0, it follows that the formula for minimum eccentricity must yield the sharpness for a circle (which it does). In this case the formula simplifies to $b/2a$, recognizable as the cosine of the angle *BCA* (and of *BAC*).

We make use of this formula for minimum curvature variation to provide an alternative measure of sharpness that we call variation. It is the ratio of the actual sharpness to the sharpness of minimum curvature variation. When the variation is 1 the curvature variation is at a minimum over all conics in this triangle. Larger variation means a sharper curve, smaller means flatter. Larger variations are useful for drawing superelliptical figures, assuming that they are partitioned into four quadrants at their four extrema.

A necessary and sufficient condition for a conic to be an arc of a circle is for its triangle to be isosceles and its variation to be 1.

As one application for this concept, a simplified conic spline

package could restrict itself to conic arcs of variation 1. Each arc would be specified by three points. A connected sequence of arcs would require only two points for each new arc. This package would allow any circular arc to be drawn, corresponding to isosceles triangles, and offer a limited variety of elliptical arcs as well. Our experience with fonts however indicates that variation 1 is much too restrictive for that application.

It should be noted that, even for conic splines that are segments of ellipses, the eccentricity of the ellipse does not uniquely determine sharpness, contrary to what has sometimes been stated in the literature. As sharpness increases from 0, eccentricity starts at infinity, decreases until the ellipse of minimum eccentricity is reached, after which the eccentricity increases again, returning to infinity when the sharpness reaches unity. Hence any given eccentricity is associated with two possible sharpnesses. In addition, an eccentricity below the minimum is meaningless. This makes eccentricity an unworkable substitute for sharpness. Variation on the other hand is linearly related to sharpness for any given triangle and hence does not have these problems.

4. Rendering

The problem is to visually approximate a two-dimensional *figure* drawn in the real (Euclidean) plane with an *image* consisting of a uniform and independent shading of each of the squares or *pixels* of a grid. In this paper we shall treat only *zero-aperture* imaging, in which the image is chosen to (best) match the figure at pixel centers. Zero-aperture is more readily implemented and performs better than larger apertures since it does not involve any area measurement, but is inferior at rendering fine detail, where a unit aperture (each image pixel matches the average of the corresponding figure pixel) yields somewhat better results and dithering with an aperture covering several pixels can produce a significantly better image. One may of course have one's cake and eat it too by constructing a sequence of images with increasing aperture sizes, so that a crude image appears immediately using the techniques of this paper and then as you watch is more carefully refined using other techniques.

We shall furthermore treat only the case of black and white figures and images; in any such case where the image shades include all the figure shades, the figure-image match at pixel centers is exact.

We assume an XY coordinate system with one lattice (integer) point per pixel. (These are *device coordinates*; pixels need not actually be upright, square, rectangular, or even congruent in physical coordinates.) We shall refer to increasing x as right and increasing y as up. The exact location of the lattice points inside the pixels may vary in half-pixel steps for each dimension with each conic arc; this turns out to allow a representable conic to be translated by half-pixel steps in the plane without making a mess of the associated arithmetic.

We treat only figures having region boundaries that, in device coordinates, are piecewise conic. These will also be conics in physical coordinates if device and physical coordinates are projectively related.

There are three stages in the way we discretize a conic segment: subdivision, alignment, and tracking. *Subdivision* partitions the conic into smaller conics of a size manageable by our algorithm, the limiting size being in proportion to the number of bits in an integer. We have already described how to subdivide conics in the section on principles of conic splines. *Alignment* adjusts the points **A,B,C** of each conic arc so that **A–B** and **B–C** have integer x and y components, yielding a *representable* conic, which may thereafter be translated in the plane by half-pixel increments. Tracking is the Pitteway process.

4.1. Alignment

Our exact version of Pitteway's algorithm depends on having the control points A,B,C separated by integer distances in each dimension and the square of the sharpness being a rational. We call conics satisfying these conditions (including the size limitations) *representable*. Alignment makes small adjustments to a conic to make it representable.

A set of n reals may be adjusted to make all their differences integral with an adjustment to each of at most $(1-1/n)/2$. (Reduce the n reals modulo 1 to form a ring, i.e. arrange the fractional parts of the n reals in a circle, find the biggest gap, of size at least $1/n$, and move all points away from that gap by at most $(1-1/n)/2$ to coincide.) Hence three reals a,b,c may be adjusted to make their three differences all integer by adjusting each by at most 1/3. For this case this can be accomplished by moving only two of the three by at most 1/3 each, so in the worst case the average motion of the three points becomes 2/9. It is incorrect however to conclude that the expected value of the average motion is exactly 1/9, though it should be somewhere in the neighborhood of 1/8.

Full quadrants provide a common special case admitting a better treatment than the above. This is when one end is horizontal and the other vertical. Adjust each endpoint along the normal at that endpoint to the nearest half-pixel. Then adjust each endpoint along the tangent at that endpoint to make **A–C** integer in x and y. Then **B** may be chosen to make the tangents exactly horizontal and vertical; it may be seen that B's adjustment tracks only the first adjustment (along the normals), which is the smaller adjustment. This method results in an expected movement of any point of the curve of about 1/8 of a pixel.

Better effects are obtained at extrema when the extremum is near a pixel boundary. This avoids both flat spots and pimples, one of which arises when the extremum occurs near a pixel center. To arrange this requires a coarser adjustment to the conic. In our experience it has usually proved more esthetically satisfying to settle for the larger distortion than to have flat spots or pimples.

4.2. Sharpness

The problem is to find the closest rational approximation to S^2 with given bounds on the numerator and denominator. We use what amounts to binary search in the Farey sequence. The Farey sequence is an ordered list of reduced fractions, with bounded numerator and denominator. It may be generated by starting with just two fractions, 0/1 and 1/0, being the reduced forms of respectively zero and infinity. The element intermediate between any two elements a/b and c/d is defined to be $(a+c)/(b+d)$. Thus the first rational between 0/1 and 1/0 is 1/1. Next we get 1/2 between 0/1 and 1/1, and 2/1 between 1/1 and 1/0, then 1/3 between 0/1 and 1/2, and so on. It can be shown that only reduced fractions can be produced in this way; for example 2/6 will never appear. (Hint: for any adjacent pair a/b, c/d of fractions, $ad-bc = 1$.)

To search the interval bounded by a/b and c/d, compare $(a+b)/(c+d)$ with the number being approximated to see on which side of $(a+b)/(c+d)$ to continue the search. Note that it is only necessary to have around at any one time the two bounds on the interval and the new "midpoint," that is, the Farey sequence can be computed "on the fly."

By careful arrangement one may avoid all arithmetic save additions and comparisons in computing a rational approximation with this method. (Hint: maintain a copy of the three active denominators, each scaled by the number to be approximated, and compare it with the corresponding numerator.)

Gaps between Farey fractions vary considerably. The gap between fractions with denominators b and d is $1/bd$. In practice the worst gap is at $S^2 = 1/2$, where the nearest rationals are 12/25 and 13/25, leading to a worst-case error of 1% in S^2 and hence a 0.5% error in S.

Errors in approximating sharpness have little effect on the endpoints of the curve. They leave the position and tangent of each endpoint unchanged and affect only the curvature. In the interior they of course flatten or bulge the curve a little, without however upsetting the overall smoothness of the curve. Thus sharpness errors tend to be more tolerable than an error in **B**, which in turn is usually more tolerable than errors in **A** and **C**.

The tradeoff between bounds on numerator and denominator and bounds on size of conic were made with the above considerations in mind. These tradeoffs can be changed as required.

4.3. Bresenham-Pitteway Tracking

The tracking considered in this section is at the heart of the line and conic algorithms described by Bresenham [1] and Pitteway [7] respectively.

We take a *quadrant* of a curve to mean an open (i.e. not containing its endpoints) C_1-continuous curve segment whose slope is nowhere either horizontal or vertical. We let *Bresenham-Pitteway tracking* denote the process of following a quadrant of an implicitly defined curve $F(x,y) = 0$ by sampling F at pixel centers in the neighborhood of the curve to determine on which side of the curve each pixel center falls. One tracks quadrants partly so that "side" is well-defined, partly to simplify the tracking process.

The Bresenham-Pitteway process is the following trivial procedure, parametrized by the direction of tracking, expressed as a pair of procedures, one called *hor* - either *left*() or *right*() - for moving in the horizontal direction, the other *vert* - either *up*() or *down*() - for vertical motion. (In practice *track* should be expanded as a macro, with all four combinations of these two procedures instantiated as inline code.) These two procedures are responsible for all consequences of motion, both moving the drawing device and updating the x,y coordinates and associated values. The predicate *side*(x,y,Q) tests which side of Q the point (x,y) is on, returning true if we are on the side where *hor*() brings us closer to Q.

```
track(proc hor, proc vert):
    while not done() do
        if side(x,y,Q) then hor() else vert()
```

Even before we fill in the details of *done*(), *side*(), *hor*(), and *vert*(), this simple procedure raises a couple of not so simple problems. First, although the set of horizontal and vertical moves it makes is closely related to the pixels required for zero-aperture rendering, the relationship needs to be tightened up further to capture those pixels exactly. As partial evidence for this it should be noted that the procedure visits a different set of pixels depending on which direction it traverses the curve. Second, getting the exact starting and stopping conditions turns out to be a most exasperating exercise. We have not to date found a simpler treatment of these problems than the following.

The exact pixels we want as the final product of tracking a quadrant Q may be thought about as follows. We want those pixels either immediately to the left of, or to the right of, or above, or below Q, which we may denote L(Q), R(Q), A(Q), and B(Q) respectively. For example when we track the left boundary Q of a horizontally scanned region we want R(Q), which consists of the leftmost pixels of the scanned region. Similarly when we track the lower boundary of a vertically scanned region we want A(Q).

A good way to understand such a sequence of pixels is in terms of dual pixels. A *dual pixel* is a unit square whose corners are pixel centers (and hence vice versa - the corners of a pixel are dual pixel centers). Equivalently a dual pixel is the translation of a pixel by one half in each dimension. Associated with each dual pixel d is its North-West pixel NW(d), and similarly for NE(d), SW(d), and SE(d). The inverse association, from pixels to dual pixels, uses the same names, so that SE(NW(d)) = d.

(A detail unimportant in understanding the general idea, which should therefore be skipped on a first reading, is the treatment of ties, pixel centers actually on Q. Suppose ties are resolved by declaring them to be below and to the left of the curve. Corresponding to this, dual pixels are taken to include their left and bottom edges and bottom left corner, but not their top left or bottom right or top right corners or top or right edges. In general each dual pixel includes just that corner which lies in the same direction from the dual pixel as that used to resolve ties.)

The significance of dual pixels is that they provide a more straightforward characterization of the discrete essence of Q than do pixels. We define the *discretization* $\delta(Q)$ of quadrant Q to be the set of dual pixels intersecting Q. The sense in which $\delta(Q)$ captures Q is given by the following theorem.

Theorem.
$L(Q) = NW(\delta(Q)) \cap SW(\delta(Q))$, $R(Q) = NE(\delta(Q)) \cap SE(\delta(Q))$,
$A(Q) = NW(\delta(Q)) \cap NE(\delta(Q))$, $B(Q) = SW(\delta(Q)) \cap SE(\delta(Q))$.

Proof. By symmetry it suffices to consider L(Q). Since Q is open, if Q intersects an edge of a dual pixel it must also extend beyond it. Now $L(Q) \subseteq NW(\delta(Q)) \cap SW(\delta(Q))$ because Q must intersect the right-going edge coming from each pixel in L(Q), whence Q must intersect the dual pixels above and below that edge. Conversely, $L(Q) \supseteq NW(\delta(Q)) \cap SW(\delta(Q))$ because given two adjacent vertically aligned dual pixels, Q must cut their common edge, whence the left end of that edge must be a pixel center immediately to the left of Q. □

The significance of this theorem is that it demonstrates that $\delta(Q)$ contains all the information needed for whichever set of pixels, e.g. L(Q), we need for a particular tracking applications. Furthermore, given all four of these sets of pixels one may infer the position of any pixel center relative to Q, though not its exact distance from Q. Hence any position-based tracking method such as zero-aperture only needs to know $\delta(Q)$.

Bresenham-Pitteway tracking cannot be either a direct enumeration of the desired pixels (e.g. L(Q)) or of $\delta(Q)$. Instead tracking follows a greedy tour of Q, defined as follows.

A *tour* is a sequence of pixels connected by length-1 rook moves, all horizontal moves being in the same direction and similarly for vertical moves. We say that a pixel is *n-valent* in Q when it intersects n elements of $\delta(Q)$. (We omit 'in Q' when it is provided by context.) A *tour of* Q is a maximal tour having all pixels of valency 2 or 3. (Quadrants have no 4-valent pixels.)

Remark. All tours of Q have the same length, namely $|\delta(Q)|-1$.

Remark. All tours of Q intersect Q equally often, namely one more than the number of trivalent pixels. (Exception: when $|\delta(Q)| = 1$ there are no intersections.)

A *greedy* tour of Q is one whose intersections with Q are as far as possible towards one end. There are therefore two greedy tours of Q, one for each end, which are the same tour if and only if $\delta(Q)$ is a pure diagonal (one having at most two divalent corners).

The two key properties of greedy tours are that they are what Bresenham-Pitteway tracking yields, and the desired pixels (e.g. L(Q)) can be very easily extracted from them. The former property is best verified by the reader by inspection. L(Q) or R(Q) may be enumerated in order by taking those pixels the tracking algorithm is at just before making each vertical move, plus or minus one in the horizontal direction depending on the direction of travel and whether L or R is needed. Dually the pixels of A(Q) or B(Q) may be enumerated just before each horizontal move, plus or minus one vertically. Since the tracking process is all incremental

the plus-or-minus-one arithmetic need be performed only once, at the beginning.

This perspective on tours now makes it easy to deal with the endpoints. As may be easily verified by inspection, the first pixel enumerated by the immediately above procedure should be discarded, and an additional pixel at the other end should be obtained by allowing the procedure to continue for one more pixel output.

4.4. The Numerical Component

The procedures *up*(), *down*(), *left*(), and *right*(), and the predicate *side*(x,y,Q), are implemented as the numerical component of Pitteway's conic-tracking algorithm. The predicate tests the sign of $F(x,y)$ at the current location (x,y), and also the sign of a partial derivative of $F(x,y)$ when necessary. The four motion procedures keep the value of $F(x,y)$ up to date, which they do incrementally by also keeping up to date the values at the current location of the partial differences

$$F_x(x,y) = F(x+1,y)-F(x,y) \quad F_y(x,y) = F(x,y+1)-F(x,y)$$
$$F_{xx}(x,y) = F_x(x+1,y)-F_x(x,y) \quad F_{yy}(x,y) = F_y(x,y+1)-F_y(x,y)$$
$$F_{xy}(x,y) = F_x(x,y+1)-F_x(x,y)$$

For each move up, down, left, or right, the updating procedure may be expressed as follows. (We use the convenient += and −= notation of C; x += y denotes the operation of adding y to x, and similarly for −=.)

up(): $F \mathrel{+}= F_y;\quad F_x \mathrel{+}= F_{xy};\quad F_y \mathrel{+}= F_{yy};$

down(): $F_y \mathrel{-}= F_{yy};\quad F_x \mathrel{-}= F_{xy};\quad F \mathrel{-}= F_y;$

right(): $F \mathrel{+}= F_x;\quad F_x \mathrel{+}= F_{xx};\quad F_y \mathrel{+}= F_{xy};$

left(): $F_y \mathrel{-}= F_{xy};\quad F_x \mathrel{-}= F_{xx};\quad F \mathrel{-}= F_x;$

The initial values of F and the partials at (x,y) (chosen to lie at a distance of less than one pixel from the curve), may be computed from the following formulas, which are obtainable directly from the definitions. Note that these are partial differences, not partial derivatives, whence the extra α in F_x and γ in F_y.

F	$\alpha x^2 + 2\beta xy + \gamma y^2 + \delta x + \varepsilon y + \zeta$
F_x	$2\alpha x + \alpha + 2\beta y + \delta$
F_y	$2\gamma y + \gamma + 2\beta x + \varepsilon$
F_{xx}	2α
F_{xy}	2β
F_{yy}	2γ

As shown earlier, when the control points are lattice points the Greek letters are all integers. To take best advantage of this the coordinate system may be chosen separately for each conic to make the control points lattice points. The sampled points will then in general not be lattice points. However if they are half lattice points ($2x$ and $2y$ are integers) then all the partial derivatives will be integers, a property we need for exact tracking. It does not matter that F is not an integer since we shall only make comparisons with F of the form $F<0$ and $F \geq 0$, so taking F to be the floor of its true value will not affect the outcome of such tests.

Overflow may occur during the calculation of F. However the final value of F should be close to 0 since the initial pixel is adjacent to the curve. To be exact, its magnitude cannot exceed that of F_x or F_y, which must be arranged not to overflow, both for this reason and to make reliable increments. Hence all values lost by overflow will cancel, whence overflow may be ignored.

Stopping Condition

For the stopping condition for tracking, represented by the procedure *done*(), it suffices to test either the y or x coordinate, depending on whether we are scanning horizontally (L(Q) or R(Q) pixels) or vertically (A(Q) or B(Q) pixels) respectively. The test is against a limit that is precomputed by inspection of the endpoints. This test may be reduced to a single decrement-and-branch-on-zero instruction by performing it as a part of the pixel output process.

Aliasing

Pitteway's algorithm is a sampling process; it stays on the curve by sampling lattice points in the neighborhood of the curve, never sampling more than a pixel away. As in any sampling process, the signal may contain frequencies high enough to confuse the sampling. In the case of a conic this can happen when rendering a thin ellipse, where the opposite sides of the ellipse are less than a pixel apart. In this case it is possible to cross both sides in one step and miss the region in between. When this happens Pitteway's algorithm goes in a vertical or horizontal line searching for the boundary until the stopping condition is met or overflow happens.

The starting position and stopping condition define a band parallel to the scanning direction outside which nearby edges do not matter. If two edges of the conic occur inside this band, they will be separated by the line consisting of the zeros of the partial derivative of F with respect to the direction of scanning, i.e. the line $F_x' = 0$ for horizontal scanning and $F_y' = 0$ for vertical scanning. Hence whenever there is doubt about whether two edges have been crossed at once, the doubt may be resolved by inspecting the partial derivative. This doubt arises just when we are on either side of both edges. This adds a test (of the sign of the partial derivative) to the code for one of the outcomes of the *side*() predicate but not the other. By symmetry the *side*() predicate should hold half the time on average, diluting the expected cost of the test by a factor of two.

This method may be considered a form of antialiasing, in that it takes advantage of the derivative not containing any high frequencies, unlike F itself.

There is no need to keep a separate copy of F_x' up to date since $F_x' = F_x - F_{xx}/2$, reducing the test $F_x' < 0$ to the comparison $F_x < F_{xx}/2$. $F_{xx}/2$ should be kept in a register (a machine-dependent performance consideration), but unlike F_x' it needs no updating. Since the test of the derivative is only done half the time, this is better than having to update a separate copy of the derivative itself at every step.

The appropriate sign of F_x', corresponding to which edge we are tracking, is determined at the outset by evaluating F_x' at each of A and C; since not both A and C can be at horizontal extrema, at least one of them will yield a nonzero value for F_x', which then yields the desired sign. An easy test for whether F_x' is zero at A is whether $y_A = y_B$. With this test it suffices to evaluate F_x' at only one of A or C.

It is important to realize that the correctness of this method depends on the stopping condition for the basic loop, which determines the band within which the tracking takes place. The claim is that the algorithm can rely on the signs of F and F_x' (if scanning horizontally) to infer its current position relative to the curve *provided* the current position has not gone past the y limit. Consider a very thin horizontal ellipse, for which $F_x = 0$ is a vertical line through its center. One can cross the whole of this ellipse in one step moving vertically without detecting a sign change in either F or F_x'. However one cannot do so without also going past the y limit.

4.5. Strokes

A commonly occurring problematical region type is the *stroke*. This is a region having two essentially parallel (whether or not curved) boundaries only a few pixels apart, with single pixel separation being the most critical. The problem is that if the two boundaries are rendered independently the sampling artifacts in each boundary may beat with each other to create even worse

artifacts.

A somewhat simple-minded solution that is independent of curve family is to partition strokes into *shallow* and *steep* components, throughout which the slope (of the medial axis) stays within 45 degrees of horizontal or vertical respectively. Then for each such component adjust matching pairs of shallow or steep boundaries to make them respectively vertical or horizontal integer-distance translations of each other. This eliminates all beating between the two boundaries. Steep (shallow) strokes then have the same number of pixels in every row (column).

This adjustment will leave a gap at shallow-steep junctions, due in part to the ends of the adjoining strokes being at right angles and in part to the ends being moved to the appropriate halfpoints. The gap may be closed by connecting corresponding boundaries (outer to outer, inner to inner) with straight lines, providing tangent continuity thought not curvature continuity. Also thickness is no longer uniform, decreasing at diagonal points to 70% of the thickness at extrema. As thickness increases the boundary interference problem becomes less objectionable while the gap closure and the thickness variation become more noticeable. Hence at some thickness, in the vicinity of 3 to 5 pixels, this treatment of strokes does more harm than good.

Better effects than are possible with this simple-minded method may be had using the polygonal pens described in J. Hobby's thesis [12]. His treatment is equivalent to the above at a thickness of one pixel, but breaks thicker strokes up into smaller pieces each with its own treatment, with the two boundaries still being the same within each piece but with the corresponding points between those boundaries no longer always being parallel to an axis but rather being approximately normal to the curve. This work is remarkable for its extensive and effective use of elementary number theory.

5. Precision and Performance Considerations

The coefficients α through γ of the implicit equation of the conic range from quadratic to quartic polynomials in the **XYZ** coordinates of A,B,C. The coefficients α, β, and γ are quadratic, δ and ϵ are cubic, and ζ is quartic. To fit the resulting large numbers into a 32-bit word, it is necessary to hold down the size of the coordinates of A,B,C.

The origin is translated to the center of the rectangular hull of the triangle to minimize the magnitudes of these coordinates.

For 32-bit arithmetic we limit this magnitude to 100, whence the triangle must fit in a square of 200 pixels on a side; larger triangles must have their conics subdivided. A proportionately larger triangle is possible with 64-bit arithmetic.

This size is achievable if the numerator of the square of the sharpness is limited to 15 and the denominator to 25. For the values of sharpness so representable, the arithmetic is exact. More precision in sharpness may be had by reducing the limit on the size of the triangle.

On the 68010 the inner loop of the exact Pitteway process, without the modification for aliasing, is a constant 7 instructions: three adds, compare, conditional branch, render, and decrement-and-branch (all arithmetic being exact, using 32-bit integers). With the antialiasing modification an additional comparison instruction and associated branch instruction are executed every second time around the loop on average, increasing the number to an average of 8 instructions per step around the curve. The result is that the running time for the curve-drawing phase of our algorithm is on the order of 10L microseconds where L is the length of the curve in pixels in the L_1 or Manhattan metric (number of length-1 rook moves).

6. Prototyping, Packaging and Integration

A system based on the above algorithms was built at Sun three times. The first implementation, between February and May of 1984, was done concurrently with working out the necessary theory. It performed well but suffered from a lack of perspective on our part at the time as to what functions were needed and how the interfaces should be structured. As a prototype it was invaluable for testing the principles. A second implementation was carried out in June and July to improve both the interfaces and some of the algorithms. This could be called the packaging phase. The system was then used extensively for a number of months as a component of an outline font design system, incorporating a spline interpolating package for digitized point data, and an outline font editor. A third implementation, still under way, constitutes the integration phase, to allow the system to fit in smoothly with Sun's version of the graphics universe: CORE and GKS on top of CGI on top of Pixrect (Sun's internal bitmap standard).

Acknowledgments

Martin Brooks, Craig Taylor, Mike Shantz, Jerry Evans, Leo Guibas, Lyle Ramshaw, John Hobby, Don Knuth, Chuck Bigelow, and Kris Holmes formed a patient and sympathetic audience for many discussions of this material. Martin Brooks provided the first of the three implementations.

7. Bibliography

[1] Bresenham, J.E. Algorithm for computer control of a digital plotter, IBM Systems Journal, Vol. 4, p.25, 1965

[2] Catmull, E., Computer Display of Curved Surfaces, Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structure, p.11, May 1975.

[3] Forrest, A.R., Curves and Surfaces for Computer-Aided Design, Ph.D. Thesis, Mathematical Laboratory and Engineering Dept., University of Cambridge, July 1968.

[4] Coolidge, J.L, **A History of the Conic Sections and Quadric Surfaces**, Oxford University Press, 1945.

[5] Lockwood, E.H., **A Book of Curves**, Cambridge University Press, Cambridge, 1961.

[6] Pavlidis, T., Curve Fitting with Conic Splines, ACM Trans.on Graphics, 2, 1, 1-31, January 1983.

[7] Pitteway, M.L.V., Algorithm for drawing ellipses or hyperbolae with a digital plotter, Computer J., B10P, p282-289, 1967.

[8] Sederberg, T.W., **Implicit and Parametric Curves and Surfaces for Computer Aided Geometric Design,** Ph.D. Thesis, School of Mech. Eng., Purdue U., August 1983.

[9] Tiller, W., Rational B-splines for Curve and Surface Representation, IEEE CG&A, 61-69, September 1983.

[10] Todd, J.A., **Projective and Analytical Geometry,** Pitman, London, 1947.

[11] Yates, R.C, **Curves and Their Properties,** Classics in Mathematics Education Series, National Council of Teachers of Mathematics, 245pp., 1974.

[12] Hobby, J.D., **Digitization of Brush Trajectories**, Ph.D. thesis, Stanford University, 1985.

[13] Salmon, G., **A Treatise on Conic Sections,** Longmans, Green, & Co., 6th edition, London, 1879. Reprinted by Dover Publications Inc, NY.