Curve intersection using Bézier clipping

T W Sederberg and T Nishita*

A technique referred to as Bézier clipping is presented. This technique forms the basis of an algorithm for computing the points at which two curves intersect, and also an algorithm for robustly and quickly computing points of tangency between two curves. Bézier clipping behaves like an intelligent interval Newton method, in which geometric insight is used to identify regions of the parameter domain which exclude the solution set. Implementation tests suggest that the curve intersection algorithm is marginally slower than an algorithm based on implicitization (though faster than other algorithms) for curves of degree four and less, and is faster than the implicitization algorithm for higher degrees.

Bézier clipping, curve intersection, tangency, focus, polynomial, collinear normal algorithm

This paper presents algorithms to solve the problems of curve/curve intersection and of locating points of tangency between two planar Bézier curves, based on a new technique which will be referred to as *Bézier clipping*. The basic strategy is to use the convex hull property of Bézier curves to identify regions of the curves which do not include part of the solution. By iteratively clipping away such regions, the solution is converged to at a quadratic rate and with a guarantee of robustness.

Several papers have addressed the problem of planar Bézier curve/curve intersection. Predominant approaches are the convex hull/de Casteljau subdivisional algorithm[1], the interval subdivision method adapted by Koparkar and Mudur[2], and implicitization[3]. Implementations of those algorithms have suggested that implicitization is easily the fastest of those algorithms for curves of degree less than five[3]. For higher degrees, the interval algorithm is generally fastest.

An algorithm for computing points of tangency between two parametric curves has recently been proposed[4], based on vector fields.

In the next section the curve intersection algorithm based on Bézier clipping is discussed; then Bézier clipping is applied to the problem of computing points of tangency; this is followed by some timing comparisons; the final section is devoted to some concluding observations.

Engineering Computer Graphics Lab, Brigham Young University, Provo, UT 84602, USA
*Department of Electrical Engineering, Fukuyama University, Fukuyama, Japan
Paper received: 24 January 1990. Revised: 6 April 1990

CURVE/CURVE INTERSECTION

Fat lines

Define a *fat line* as the region between two parallel lines. The curve intersection algorithm described here begins by computing a fat line which bounds one of the two Bézier curves. Similar bounds have been suggested in References 5, 6 and 7.

Denote by $\bar{L}$ the line $\mathbf{P}_0 - \mathbf{P}_n$. A fat line is chosen parallel to $\bar{L}$, as shown in Figure 1. If $\bar{L}$ is defined in its normalized implicit equation

$$ax + by + c = 0 \quad (a^2 + b^2 = 1) \tag{1}$$

then, the distance $d(x, y)$ from any point (x, y) to $\bar{L}$ is

$$d(x, y) = ax + by + c \tag{2}$$

Denote by $d_i = d(x_i, y_i)$ the signed distance from control point $\mathbf{P}_i = (x_i, y_i)$ to $\bar{L}$. By the convex hull property, a fat line bounding a given rational Bézier curve with non-negative weights can be defined as the fat line parallel to $\bar{L}$ which most tightly encloses the Bézier control points:

$$\{(x, y) \mid d_{\min} \leqslant d(x, y) \leqslant d_{\max}\} \tag{3}$$

where

$$d_{\min} = \min\{d_0, \ldots, d_n\}$$
$$d_{\max} = \max\{d_0, \ldots, d_n\} \tag{4}$$

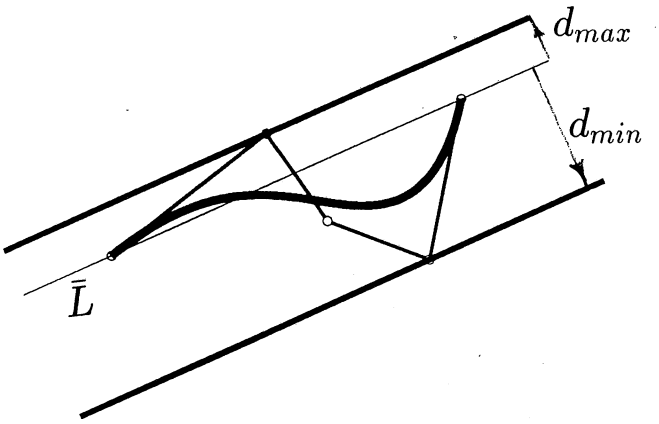


Figure 1. Fat line bounding a quartic curve

0010–4485/90/090538–12

Forrest, A R and Pankhurst, A F as *Numerical Control – Mathematics and Applications* (with Appendices by **Forrest, A R**) Wiley (1972))

10 **Forrest, A R** 'Coons' surfaces and multivariable functional interpolation' *Document No 38* Cambridge University CAD Group (July 1970)

11 **Davis, P J** *Interpolation and Approximation* Ginn-Blaisdell (1963)

12 **Gordon, W J** *Private communication* (May 1971)

13 **Coons, S A** 'Surfaces for computer-aided design of space forms' *MIT MAC-TR-41* (1967)

14 **Lee, T N P** 'Three-dimensional curves and surfaces for rapid computer display' *PhD thesis* Harvard University (April 1969)

15 **Forrest, A R** 'The twisted cubic curve' *Document No 50* Cambridge University CAD Group (November 1970)

18 **Talbot, J E** 'Experiments towards interactive graphical design of motor bodies *Document No 56* Cambridge University CAD Group (April 1971)

19 **Pankhurst, R J** 'Notes on experiments with command languages for graphic interaction' *Document No 37* Cambridge University CAD Group (February 1970)

20 **Armit, A P** 'Systems for interactive design of three-dimensional shapes' *PhD thesis* Cambridge University CAD Group (November 1970)

21 **Sabin, M A** *A 16-Point Bicubic Formulation Suitable for Multipatch Surfaces.* British Aircraft Corporation, Weybridge, UK VTO/MS/155 (March 1969)

22 **Forrest, A R** 'Curves and surfaces for computer-aided design' *PhD thesis* Cambridge University CAD Group (July 1968)

References not in original version

3 **Riesenfeld, R F** 'Applications of B-spline approximation to geometric problems of CAD' *PhD thesis* Syracuse University (published as University of Utah, Computer Science, *UTEC-CSc-73-126*) (March 1973)

4 **De Boor, C** 'On calculating with B-splines' *J. Approx. Theory* Vol 6 No 1 (July 1972)

5 **Cox, M G** 'The numerical evaluation of B-splines' *DNAC 4* National Physical Laboratory (August 1971)

16 **Forrest, A R** 'The twisted cubic curve: a computer-aided geometric design approach' *Comput.-Aided Des.* Vol 12 No 4 (July 1980)

17 **Roberts, L G** *Homogeneous Matrix Representation and the Manipulation of n-Dimensional Constructs* MIT Lincoln Laboratories (May 1965)

Other Cambridge CAD Group Documents relating to Bézier curves and surfaces

2 **Forrest, A R** 'Interpolation and approximation by Bézier polynomials' *Document No 45* Cambridge University CAD Group (October 1970)

Forrest, A R 'A re-examination of the Renault technique for curves and surfaces' *Document No 24* Cambridge University CAD Group (1969)

Forrest, A R 'Notes on topics in curve and surface fitting' *Document No 36* Cambridge University CAD Group (February 1970)

Forrest, A R 'Shape classification of the non-rational twisted cubic curve in terms of Bézier polygons' *Document No 52* Cambridge University CAD Group (December 1970)

These values for d_{min} and d_{max} are conservative. For polynomial Bézier curves (all weights = 1) of degree two and three (the most common cases), values of d_{min} and d_{max} can readily be found for which the fat line bounds the curve tightly.

Quadratic case

If $d(t)$ is the distance from any point on the curve $\mathbf{P}(t)$ to $\bar{L}$, then, for polynomial quadratic Bézier curves (see Figure 2)

$$d(t) = 2t(1-t)d_1 \tag{5}$$

from which the tight bounds are

$$d_{min} = \min\left\{0, \frac{d_1}{2}\right\}$$
$$d_{max} = \max\left\{0, \frac{d_1}{2}\right\} \tag{6}$$

Cubic case

For a cubic curve, the tightest possible fat line parallel to $\bar{L}$ can be computed in closed form as follows. In this case,

$$d(t) = 3t(1-t)[(1-t)d_1 + td_2] \tag{7}$$

The function $d(t)$ has an extremum where $d'(t) = 0$. If $d_1 d_2 > 0$, there is one extremum at

$$t_1 = \frac{d_1}{2d_1 - d_2 + \sqrt{d_1^2 - d_1 d_2 + d_2^2}} \tag{8}$$

and

$$d_{min} = \min\{0, d(t_1)\}$$
$$d_{max} = \max\{0, d(t_1)\} \tag{9}$$

If $d_1 d_2 \leqslant 0$, there are two extrema, at

$$t_1 = \frac{2d_1 - d_2 + \sqrt{d_1^2 + d_2^2 - d_1 d_2}}{3(d_1 - d_2)}$$
$$t_2 = \frac{2d_1 - d_2 - \sqrt{d_1^2 + d_2^2 - d_1 d_2}}{3(d_1 - d_2)} \tag{10}$$

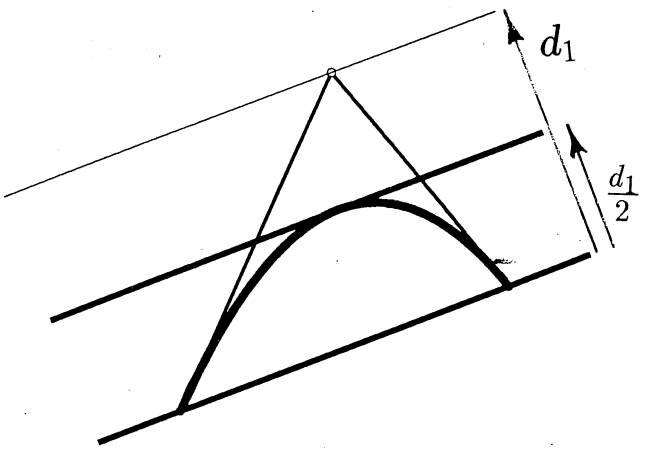


Figure 2. Fat line for a polynomial quadratic curve

and

$$d_{min} = \min\{d(t_1), d(t_2)\}$$
$$d_{max} = \max\{d(t_1), d(t_2)\} \tag{11}$$

In the experience of the authors, the expense of computing this tightest possible fat line is not justified by improved overall execution speed. It is better, in the case of polynomial cubic curves, to use the following values of d_{min} and d_{max}. From equation (7), if $d_1 d_2 > 0$,

$$\min\{0, d_1, d_2\}\, 3t(1-t) \leqslant d(t) \leqslant \max\{0, d_1, d_2\}\, 3t(1-t) \tag{12}$$

Thus if $d_1 d_2 > 0$, use

$$d_{min} = \frac{3}{4}\min\{0, d_1, d_2\}$$
$$d_{max} = \frac{3}{4}\max\{0, d_1, d_2\} \tag{13}$$

From equation (7), if $d_1 \leqslant 0$ and $d_2 \geqslant 0$, then

$$3t(1-t)^2 d_1 \leqslant d(t) \leqslant 3t^2(1-t)d_2 \tag{14}$$

Thus, if $d_1 d_2 \leqslant 0$, use

$$d_{min} = \frac{4}{9}\min\{0, d_1, d_2\}$$
$$d_{max} = \frac{4}{9}\max\{0, d_1, d_2\} \tag{15}$$

These fat lines are illustrated in Figure 3.

Bézier clipping

Figure 4 shows two polynomial cubic Bézier curves $\mathbf{P}(t)$ and $\mathbf{Q}(u)$, and a fat line $\mathbf{L}$ which bounds $\mathbf{Q}(u)$. This subsection is concerned with how to identify intervals

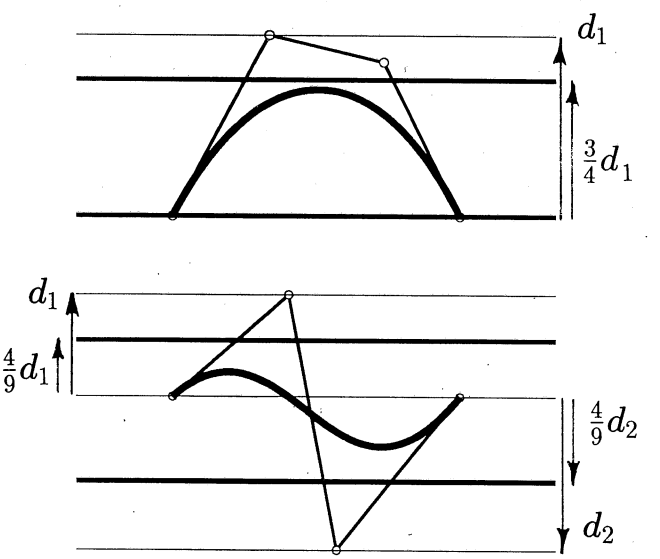


Figure 3. Fat lines for polynomial cubic curves

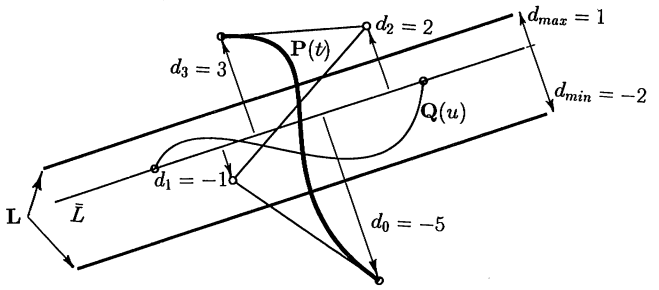


Figure 4. Bézier curve/fat line intersection

of t for which $\mathbf{P}(t)$ lies outside of $\mathbf{L}$, and hence for which $\mathbf{P}(t)$ does not intersect $\mathbf{Q}(u)$.

$\mathbf{P}$ is defined by its parametric equation

$$\mathbf{P}(t) = \sum_{i=0}^{n} \mathbf{P}_i B_i^n(t) \tag{16}$$

where $\mathbf{P}_i = (x_i, y_i)$ are the Bézier control points, and $B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$ denote the Bernstein basis functions. If the line $\bar{L}$ through $\mathbf{P}_0 - \mathbf{P}_n$ is defined by

$$ax + by + c = 0 \quad (a^2 + b^2 = 1), \tag{17}$$

then the distance $d(t)$ from any point $\mathbf{P}(t)$ to $\bar{L}$ can be found by substituting equation (16) into equation (17):

$$d(t) = \sum_{i=0}^{n} d_i B_i^n(t), \quad d_i = ax_i + by_i + c \tag{18}$$

Note that $d(t) = 0$ for all values of t at which $\mathbf{P}$ intersects $\bar{L}$. Also, d_i is the distance from $\mathbf{P}_i$ to $\bar{L}$ (as shown in Figure 4).

The function $d(t)$ is a polynomial in Bernstein form, and can be represented as a so-called 'non-parametric' Bézier curve[8] as follows:

$$\mathbf{D}(t) = (t, d(t)) = \sum_{i=0}^{n} \mathbf{D}_i B_i^n(t) \tag{19}$$

The Bézier control points $\mathbf{D}_i = (t_i, d_i)$ are evenly spaced in t ($t_i = i/n$). Since $\sum_{i=0}^{n}(i/n)B_i^n(t) \equiv t[(1-t)+t]^n \equiv t$, the horizontal coordinate of any point $\mathbf{D}(t)$ is in fact equal to the parameter value t. Figure 5 shows the curve $\mathbf{D}(t)$ which corresponds to Figure 4.

Values of t for which $\mathbf{P}(t)$ lies outside of $\mathbf{L}$ correspond to values of t for which $\mathbf{D}(t)$ lies above $d = d_{max}$ or below $d = d_{min}$. Parameter ranges of t can be identified for which $\mathbf{P}(t)$ is guaranteed to lie outside of $\mathbf{L}$ by identifying ranges of t for which the *convex hull* of $\mathbf{D}(t)$ lies above $d = d_{max}$ or below $d = d_{min}$. In this example, it is certain that $\mathbf{P}(t)$ lies outside of $\mathbf{L}$ for parameter values $t < 0.25$ and for $t > 0.75$.

Bézier clipping is completed by subdividing $\mathbf{P}$ twice using the de Casteljau algorithm[8], such that portions of $\mathbf{P}$ over parameter values $t < 0.25$ and $t > 0.75$ are removed.

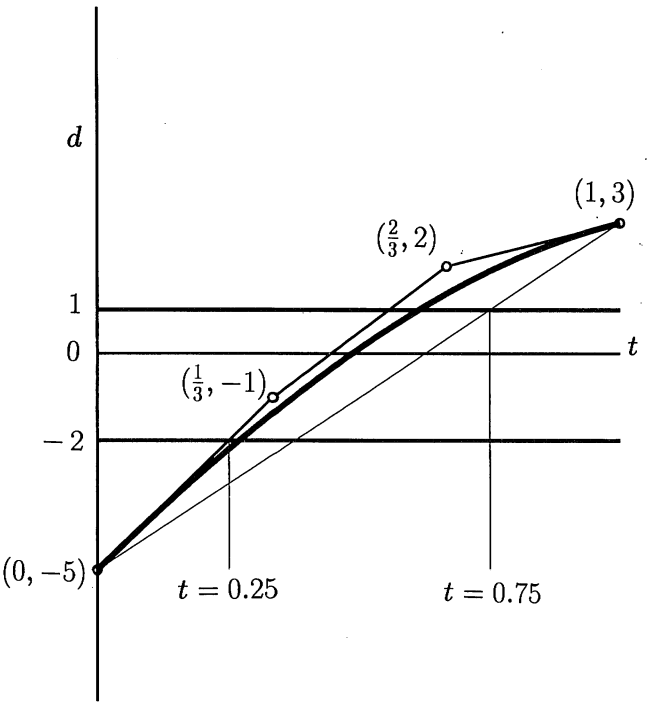


Figure 5. Non-parametric Bézier curve

Iterating

The notion of Bézier clipping has just been discussed in the context of curve intersection: regions of one curve which are guaranteed to not intersect a second curve can be identified and subdivided away. The Bézier clipping curve intersection algorithm proceeds by iteratively applying the Bézier clipping procedure.

Figure 6 shows curves $\mathbf{P}(t)$ and $\mathbf{Q}(u)$ from Figure 4 after the first Bézier clipping step in which regions $t < 0.25$ and $t > 0.75$ have been clipped away from $\mathbf{P}(t)$. The clipped portions of $\mathbf{P}(t)$ are shown in the fine pen width, and a fat line is shown which bounds $\mathbf{P}(t)$, $0.25 \leqslant t \leqslant 0.75$. The next step in the curve intersection algorithm is to perform a Bézier clip of $\mathbf{Q}(u)$, clipping away regions of $\mathbf{Q}(u)$ which are guaranteed to lie outside the fat line bounding $\mathbf{P}(t)$. Proceeding as before, a non-parametric Bézier curve is defined which expresses the distance from $\bar{L}$ in Figure 6 to the curve $\mathbf{Q}(u)$ (see Figure 7). From Figure 6, it is concluded that it is safe to clip off regions of $\mathbf{Q}(u)$ for which $u < 0.42$ and $u > 0.63$.

Next, $\mathbf{P}(t)$ is again clipped against $\mathbf{Q}(u)$, and so on. After three Bézier clips on each curve, the intersection is computed to within six digits of accuracy (see Table 1).

Clipping to other fat lines

The fat line defined above provides a nearly optimal Bézier clip in many cases. However, it is clear that *any* pair of parallel lines which bound the curve can serve as a fat line. In many cases, a fat line perpendicular to the line $\mathbf{P}_0 - \mathbf{P}_n$ provides a larger Bézier clip than does the fat line parallel to the line $\mathbf{P}_0 - \mathbf{P}_n$. Figure 8 shows

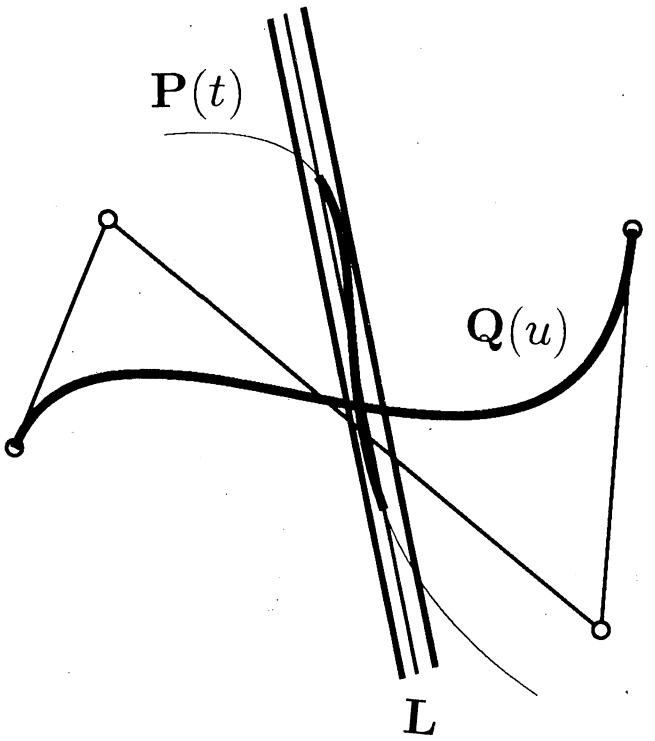


Figure 6. After the first Bézier clip

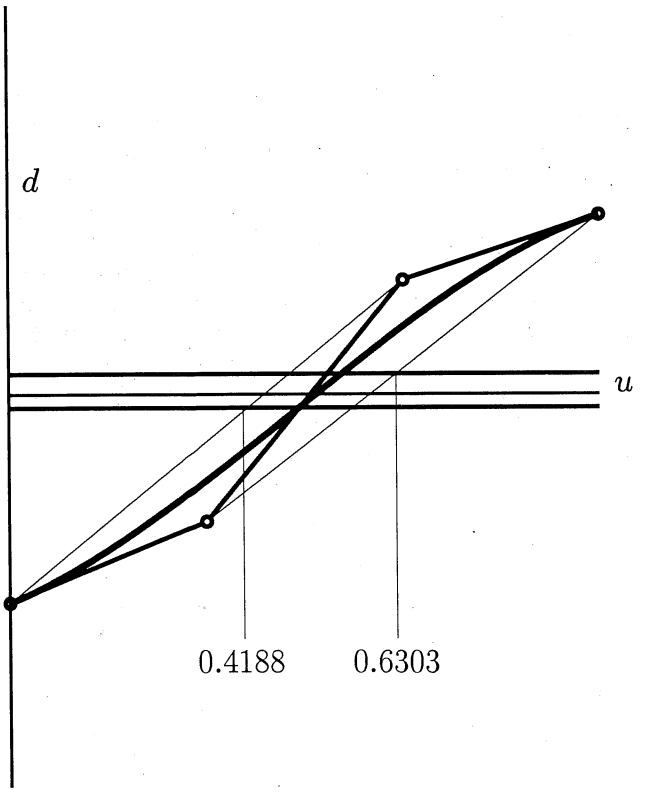


Figure 7. Distance from **Q**(u) *to* $\overline{L}$

such a case. It is suggested that in general it works best to examine both fat lines to determine which one provides the largest clip. This extra overhead results in a slightly lower average execution time.

Table 1. Parameter ranges for P(t) and Q(u)

Step	t_{min}	t_{max}	u_{min}	u_{max}
0	0	1	0	1
1	0.25	0.75	0.4188	0.6303
2	0.3747	0.4105	0.5121	0.5143
3	0.382079	0.382079	0.512967	0.512967

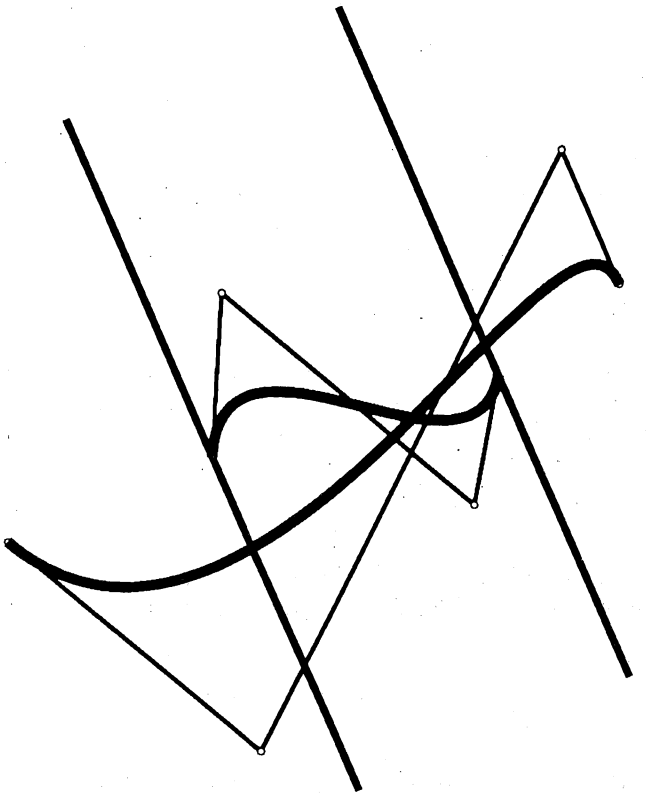


Figure 8. Alternative fat lines

Multiple intersections

Figure 9 shows a case where two intersection points exist. In this case, no Bézier clipping is possible because the endpoints of each curve lie within the fat line of the other. The remedy is to split one of the curves in half and to compute the intersections of each half with the other curve, as suggested in Figure 10. A stack data structure is used to store pairs of curve segments, as in the conventional divide-and-conquer intersection algorithm[1].

Experimentation suggests the following heuristic. If a Bézier clip fails to reduce the parameter range of either

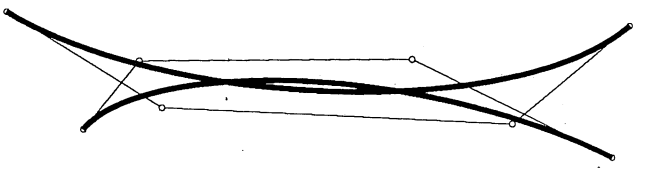


Figure 9. Two intersections

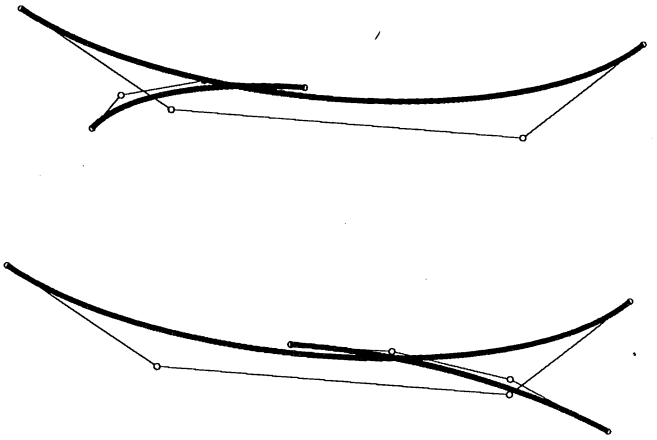


Figure 10. Two intersections after a split

curve by at least 20%, subdivide the 'longest' curve (largest remaining parameter interval) and intersect the shorter curve, respectively, with the two halves of the longer curve. This heuristic, applied recursively if needed, allows computation of arbitrary numbers of intersections.

Rational curves

If **P** is a *rational* Bézier curve

$$\mathbf{P}(t)=\frac{\Sigma_{i=0}^{n} w_i \mathbf{P}_i B_i^n(t)}{\Sigma_{i=0}^{n} w_i B_i^n(t)} \qquad (20)$$

with control point coordinates $\mathbf{P}_i=(x_i, y_i)$ and corresponding non-negative weights w_i, the Bézier clip computation is modified as follows. Substituting equation (20) into equation (17) and clearing the denominator yields:

$$d(t)=\sum_{i=0}^{n} d_i B_i^n(t)$$

$$d_i = w_i(ax_i + by_i + c)$$

The equation $d(t)=0$ expresses the intersection of $\mathbf{P}(t)$ with a line $ax+by+c=0$. However, unlike the non-rational case, the intersection of $\mathbf{P}(t)$ with a fat line *cannot* be represented as $\{(x, y)=\ \mathbf{P}(t)\,|\,d_{\min} \leqslant d(t) \leqslant d_{\max}\}$. Instead, **P** must be clipped independently against each of the two lines bounding the fat line. Thus, ranges of t are identified for which

$$\sum_{i=0}^{n} w_i(ax_i + by_i + c - d_{\max})B_i^n(t) > 0$$

or for which

$$\sum_{i=0}^{n} w_i(ax_i + by_i + c + d_{\min})B_i^n(t) < 0$$

These ranges are identified using the Bézier clipping technique as previously outlined.

TANGENT INTERSECTIONS

The solution to multiple intersections just discussed works well if the intersections are well spaced. If the difference between the parameter values of two intersections is small, a large number of subdivisions may be needed to isolate the intersections, and the algorithm tends to degenerate to a divide-and-conquer binary search. An algorithm is now presented for quickly isolating two adjacent intersections, and for computing tangent intersections. This algorithm can compute a tangent intersection to high precision in few iterations.

The algorithm is based on the following theorem.

Collinear normal theorem

If two curve segments, each C^1 smooth, intersect in two points, and neither curve turns more than 90°, then there exists a line which is mutually perpendicular to both curves. Further, the two intersection points lie on opposite sides of the line.

Proof. See Sederberg *et al.*[10].

Thus, if a line can be computed which is perpendicular to both curves, it will be possible to isolate two close intersections. Such a line will be referred to as a *collinear normal* (as opposed to parallel normal lines, of which there are typically an infinite number). If the two curves are tangent, then a collinear normal meets both curves at that point of tangency. Figure 11 shows a collinear normal in the case of two distinct intersections, and Figure 12 shows the case of a tangent intersection.

An algorithm is presented here for computing collinear normals which uses geometric insight. This algorithm adapts Bézier clipping and introduces the notion of a *focus*.

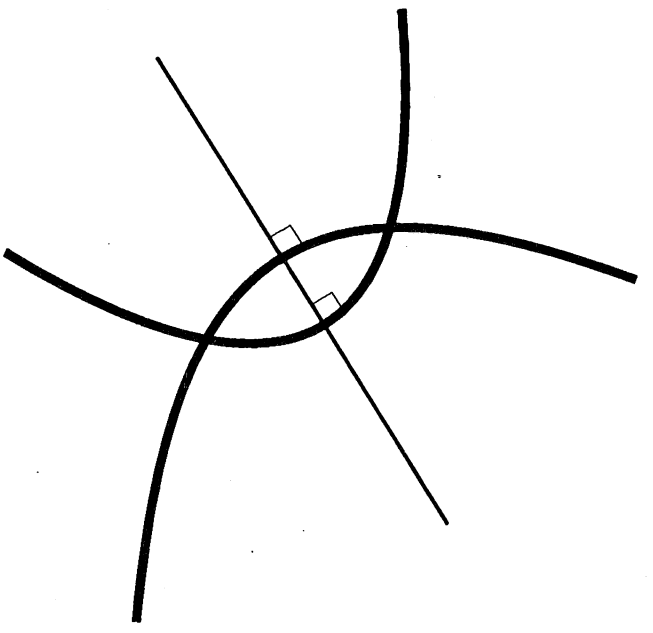


Figure 11. Collinear normal line

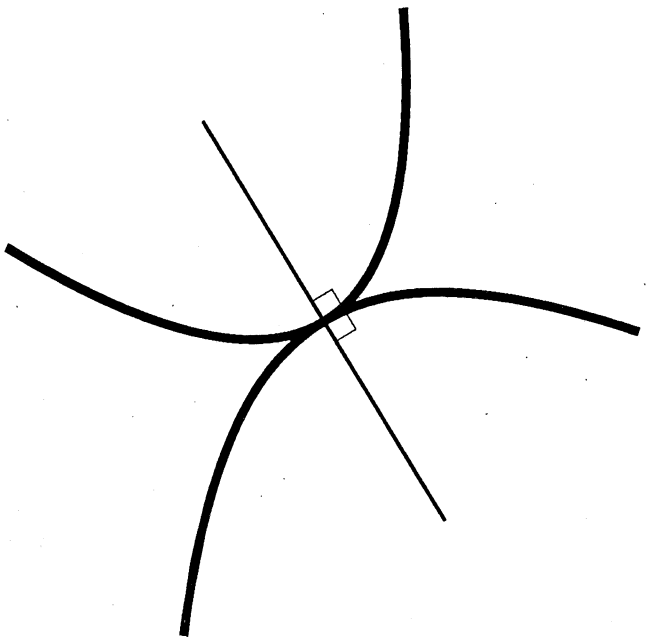


Figure 12. Tangent intersection

Bézier curve focus

A focus of a Bézier curve $\mathbf{P}(t)$ is defined to be any curve through which all lines perpendicular to $\mathbf{P}(t)$ pass. By this definition, a focus is not unique. For example, $\mathbf{P}(t)$ is a focus of itself. The algorithm for computing collinear normals works best if the focus curve is relatively small, and somewhat removed from the curve. One heuristic for creating such a focus which has been found to work well is discussed here. Other possible heuristics for obtaining small foci are mentioned below.

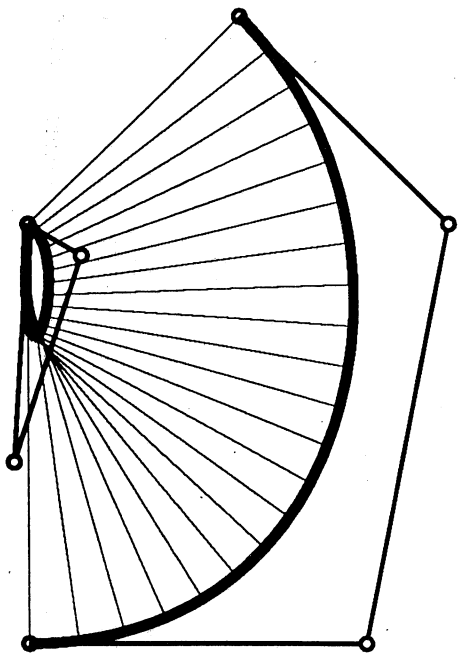
Figures 13–15 show some examples of focus curves created using the method described. Notice the families of lines perpendicular to the given curves. These normal lines all pass through the focus curve, though are not generally perpendicular to the focus curve.

The focus construction proceeds as follows. If $\mathbf{N}(t)$ defines any vector function (not necessarily of unit length) which is perpendicular to $\mathbf{P}(t)$ for all values of t, then the curve $\mathbf{F}(t) = \mathbf{P}(t) + c(t)\mathbf{N}(t)$ (where $c(t)$ is any function of t) can serve as a focus curve.

$\mathbf{N}(t)$ is taken to be the rotated hodograph[11] of $\mathbf{P}(t)$. The hodograph $\mathbf{H}(t)$ of a degree n Bézier curve $\mathbf{P}(t)$ is the first derivative of that curve, $\mathbf{P}'(t)$. The hodograph is itself a Bézier curve of degree $n-1$, whose control points $\mathbf{H}_i$ are $n(\mathbf{P}_{i+1} - \mathbf{P}_i)$ (see Figure 16). This discussion deals only with hodographs and foci of polynomial curves. Hodographs of rational Bézier curves are discussed in Reference 12, following which the construction of a focus parallels the present discussion.

Graphically, the tangent vector $\mathbf{P}'(t)$ is expressed as a vector from the hodograph origin to the point $\mathbf{H}(t)$ on the hodograph (see Figure 17). By rotating the hodograph through 90° about its origin, a vector function $\mathbf{N}(t)$ is obtained which defines vectors perpendicular to $\mathbf{P}(t)$ (see Figure 18). In creating the focus $\mathbf{F}(t) = \mathbf{P}(t) + c(t)\mathbf{N}(t)$, $c(t)$ is chosen to be a degree-one polynomial $c_0(1-t) + c_1 t$ which satisfies the condition $\mathbf{F}(0) = \mathbf{F}(1)$. This is a heuristic decision, motivated by the observation that if $\mathbf{F}(t)$ begins and ends at the same point, it will tend to cover a relatively small area. The examples in Figures 13–15 illustrate that this choice generally works well. The coefficients c_0 and c_1 are solved from the linear equation

$$\begin{bmatrix} x_1 - x_0 & x_{n-1} - x_n \\ y_0 - y_1 & y_n - y_{n-1} \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \end{Bmatrix} = \begin{Bmatrix} y_n - y_0 \\ x_n - x_0 \end{Bmatrix} \tag{21}$$



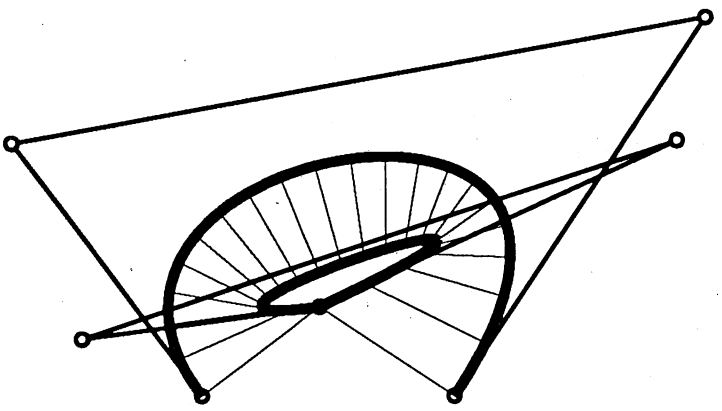


Figure 13. Focus example 1

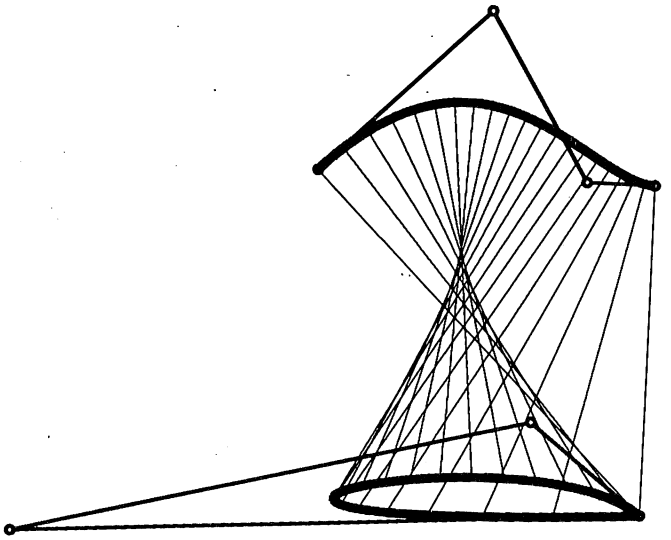


Figure 14. Focus example 2

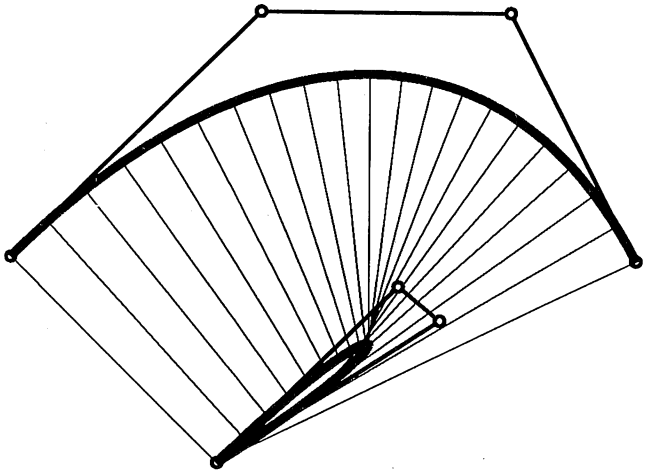


Figure 15. Focus example 3

It can happen in regions of zero curvature that the magnitudes of c_0 and c_1 can get excessively large, in which case their magnitude is simply limited to some large value. Thus, even curves with inflection points (as in Figure 14) can have a reasonable focus.

This focus construction behaves well when the curve is subdivided into smaller segments, quickly converging to the centre of curvature of their respective curve segments. Thus, even though a curve may initially have an impracticably large focus, the size of the focus will shrink quickly as the curve is subdivided. Figures 19 and 20 show the curve in Figure 15 after it has been split into two and four segments. The resulting foci appear nearly optimally small.

Other focus heuristics

After some experimentation, the authors are confident that the focus heuristic is competitive. A few other choices present themselves. First, one could simply compute a line segment through which all lines normal to a given curve pass. This has the virtue that the focus

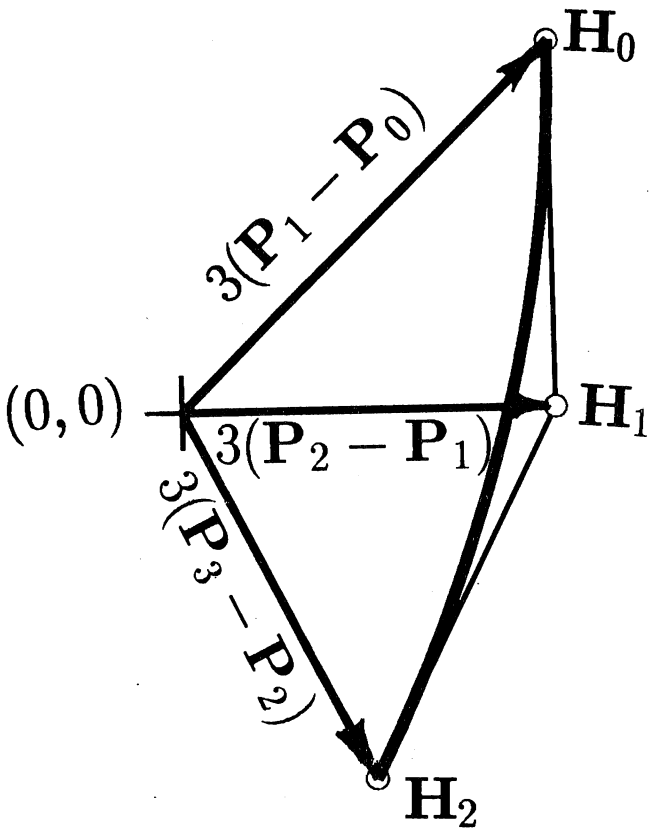


Figure 16. Curve and hodograph

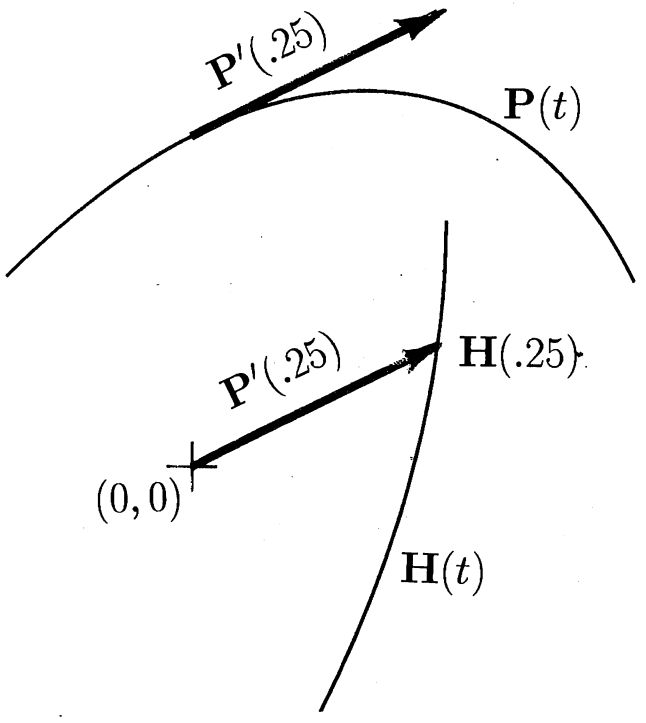


Figure 17. Tangent vector

is a simpler entity than our degree n focus, but more computation must go into computing the focus.

A second choice is to use the *evolute*[13] of the given curve as its focus. The evolute is the locus of the centres of curvature of the given curve, and is itself a rational

curve. Some drawbacks are that the degree of the evolute is larger than the degree of the focus described here, the evolute goes to infinity for segments which have inflection points, and the evolute does not seem to be as small as the focus in the examples presented here. For example, the evolute for the curve in Figure 14 can be seen as the envelope of the normal lines.

Bézier clip application

We next consider the problem of identifying regions of a Bézier curve for which lines perpendicular to the curve $\mathbf{P}(t)$ do not pass through a given point $\mathbf{F}$. The parameter values t for which lines normal to $\mathbf{P}(t)$ pass through $\mathbf{F}$ are the zeros of the equation

$$d(t) = \mathbf{P}'(t) \cdot (\mathbf{P}(t) - \mathbf{F}) = 0 \tag{22}$$

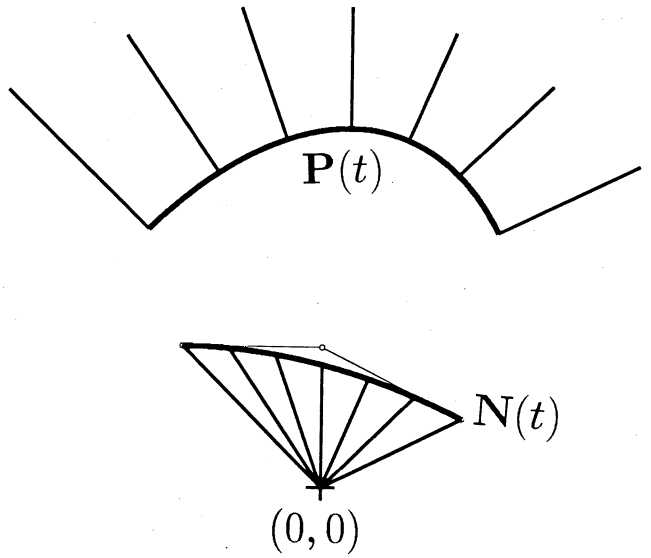


Figure 18. Rotated hodograph, $\mathbf{N}(t)$

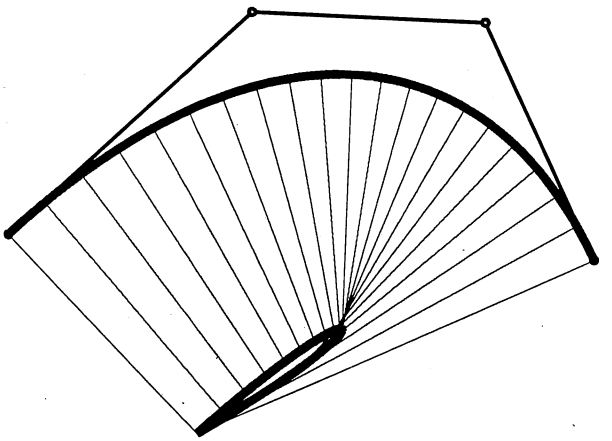


Figure 19. Curve of Figure 15 (left) split into two segments

The polynomial $d(t)$ can be expressed in Bernstein form, $d(t) = \sum_{i=0}^{2n-1} d_i B_i^{2n-1}(t)$, where

$$d_i = \sum_{\substack{j+k=i \\ j\in\{0,\ldots,n\} \\ k\in\{0,\ldots,n-1\}}} \frac{\binom{n}{j}\binom{n-1}{k}}{\binom{2n-1}{i}} n(\mathbf{P}_{k+1} - \mathbf{P}_k) \cdot (\mathbf{P}_j - \mathbf{F}) \tag{23}$$

For the cubic Bézier curve in Figure 21, $d(t)$ is shown as a non-parametric Bézier curve in Figure 22. Applying the convex hull property, it is observed that for parameter ranges $t < 0.374$ and $t > 0.5$, there are no normals to $\mathbf{P}(t)$ passing through $\mathbf{F}$.

Next consider the case where $\mathbf{F}$ is not a point, but is itself a Bézier curve. Specifically, $\mathbf{F}_\mathbf{Q}(u) = \sum_{j=0}^{m} \mathbf{F}_j B_j^m(u)$ is a focus of a Bézier curve $\mathbf{Q}(u)$ (see Figure 23). It is now desired to identify regions of $\mathbf{P}(t)$ whose normal lines do not pass through any point on $\mathbf{F}_\mathbf{Q}(u)$ – that is,

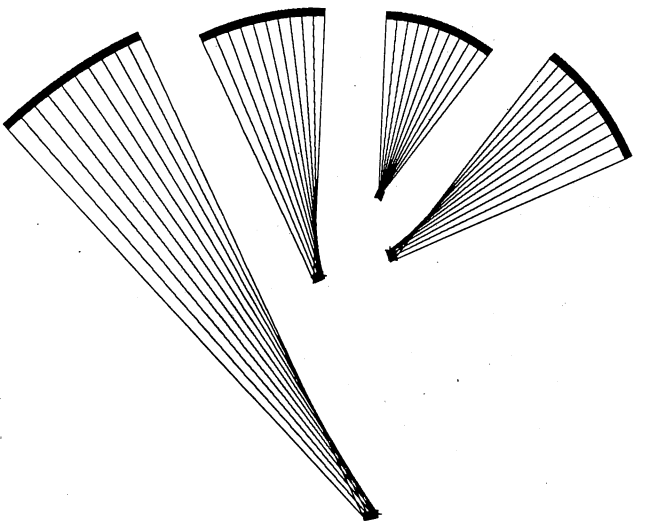
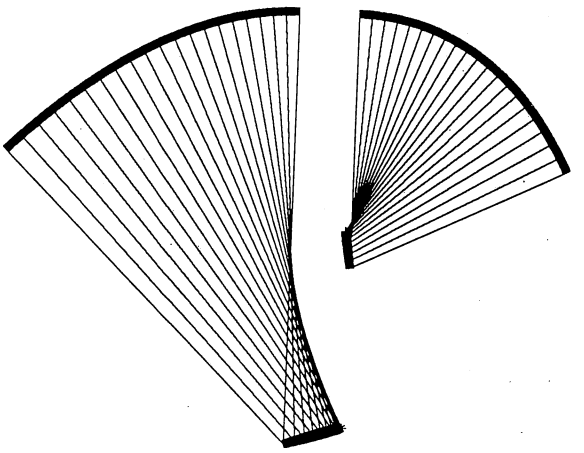


Figure 20. Curve of Figure 15 split into four segments



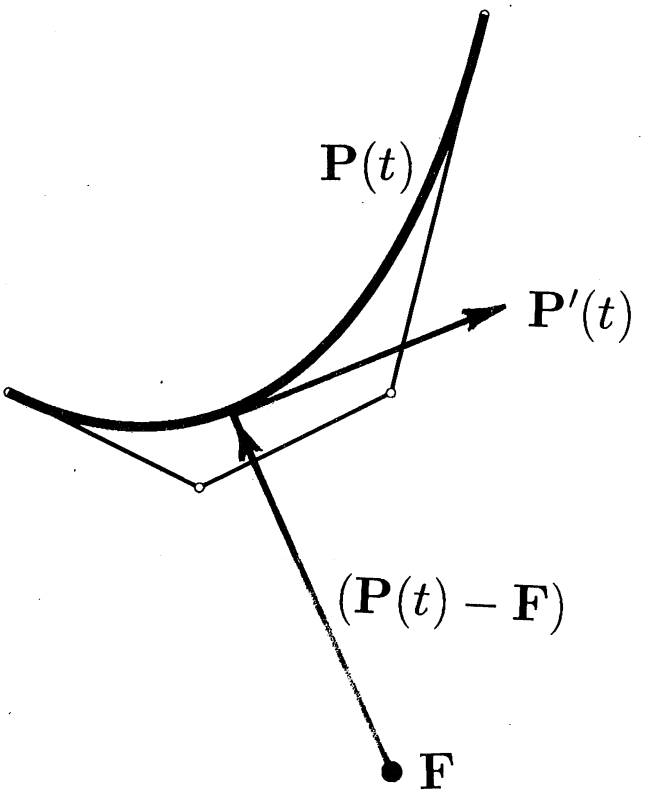


Figure 21. Normal to **P**(t) *passing through* **F**

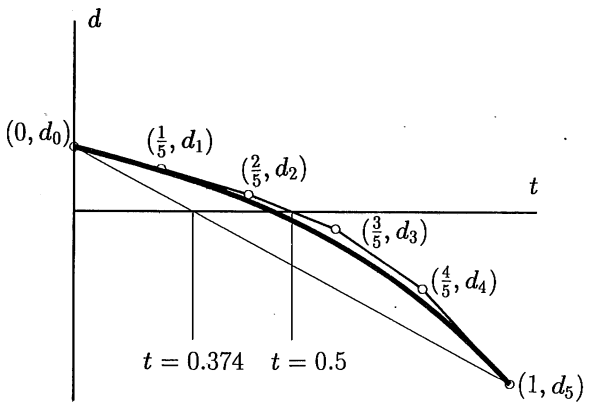


Figure 22. Non-parametric Bézier curve

to find ranges of t for which

$$D(t, u) = (\mathbf{P}(t) - \mathbf{F}_{\mathbf{Q}}(u)) \cdot \frac{\mathbf{P}'(t)}{n} \neq 0,\ 0 \leqslant u \leqslant 1 \qquad (24)$$

$D(t, u)$ can be expressed as a tensor product polynomial in Bernstein form in t and u:

$$d(t, u) = \sum_{i=0}^{2n-1} \sum_{j=0}^{m} B_i^{2n-1}(t) B_j^m(u) d_{ij} \qquad (25)$$

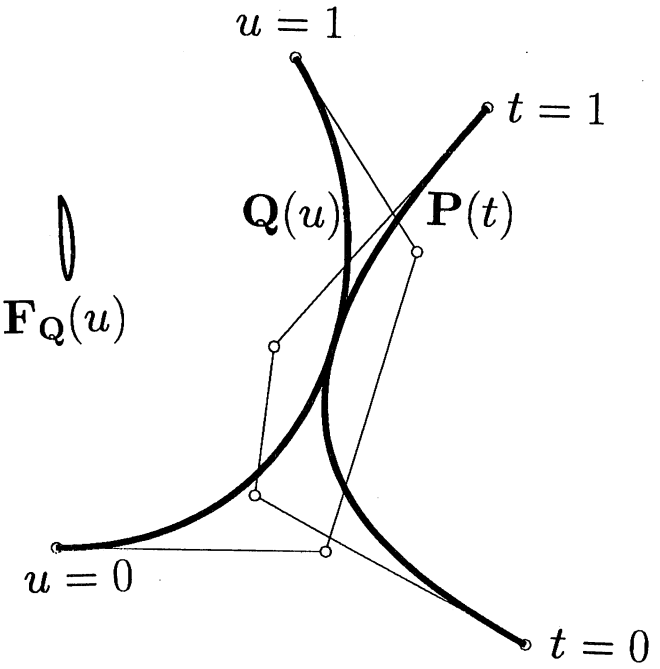


Figure 23. **P**, **Q** *and* $\mathbf{F}_{\mathbf{Q}}$

where

$$d_{ij} = \sum_{\substack{k+l=i \\ l \in \{0,\ldots,n\} \\ k \in \{0,\ldots,n-1\}}} \frac{\binom{n}{j}\binom{n-1}{k}}{\binom{2n-1}{j}} n(\mathbf{P}_{k+1} - \mathbf{P}_k) \cdot (\mathbf{P}_l - \mathbf{F}_j) \qquad (26)$$

The function $D(t, u)$ can be represented, in a (t, u, D) coordinate system, as a so-called 'non-parametric' Bézier surface patch[9] whose control points $\mathbf{D}_{ij} = (t_{ij}, u_{ij}, D_{ij})$ are evenly spaced in t and u: $t_{ij} = [i/(2n-1)]$,

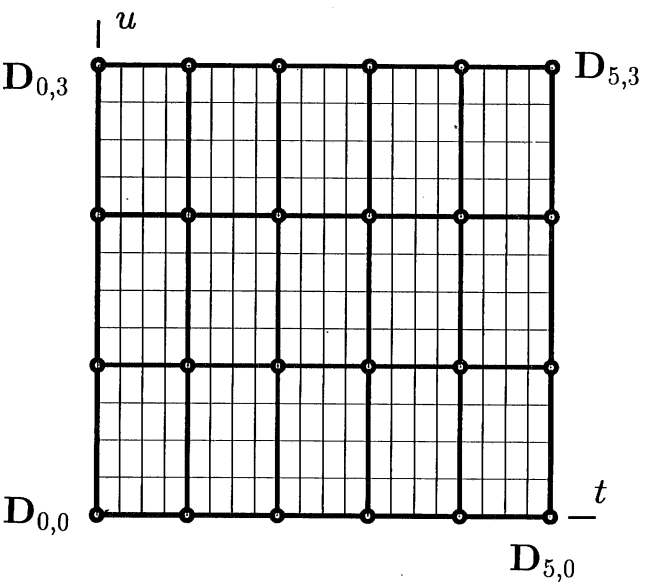


Figure 24. Top view of **D**(t, u) *patch*

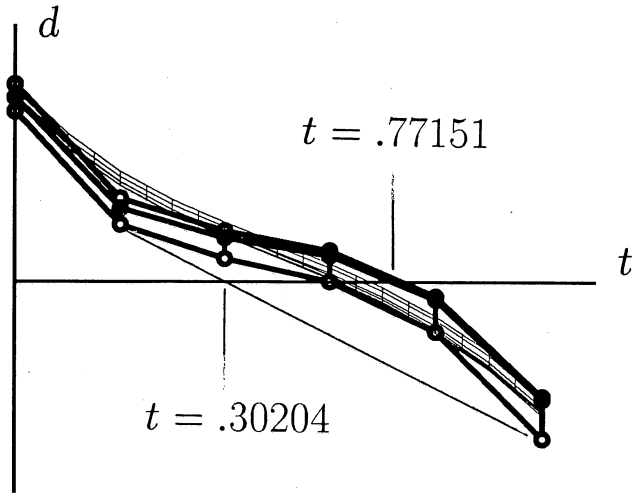


Figure 25. Side view of **D**(t, u) *patch*

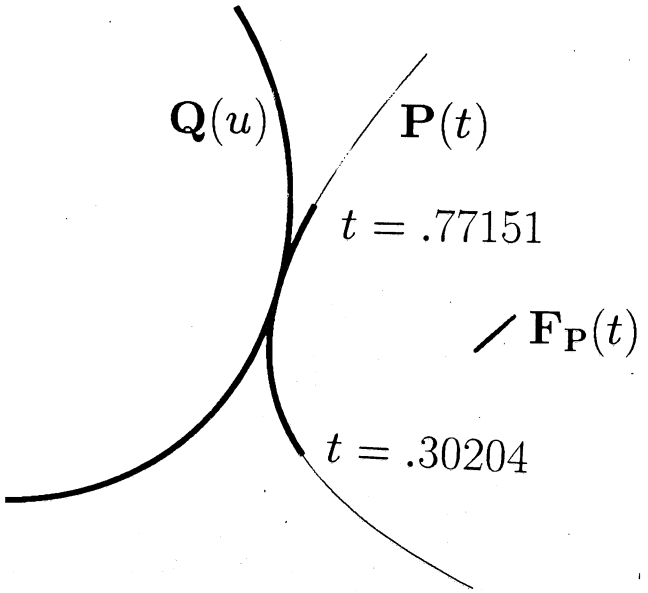


Figure 26. **P** *after first clipping, and its focus*

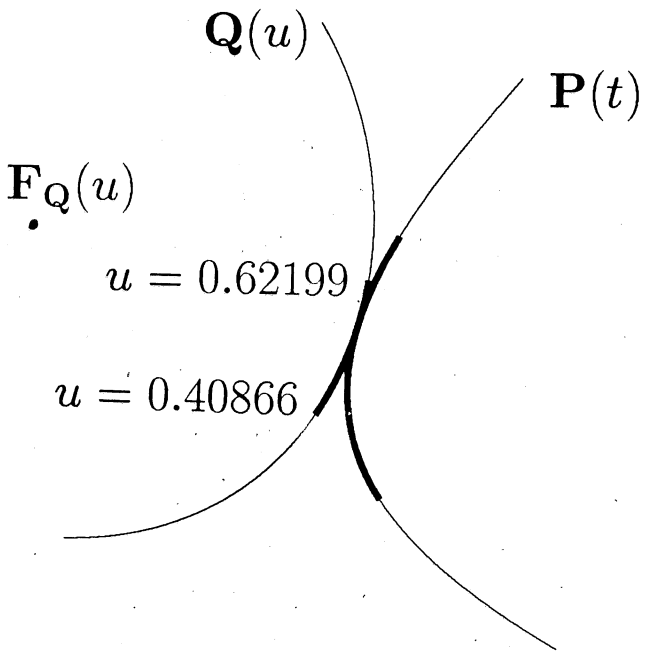


Figure 27. **Q** *after first clipping, and its focus*

$u_{ij} = j/m$. A point on such a patch has coordinates

$$\mathbf{D}(t, u) = \sum_{i=0}^{2n-1} \sum_{j=0}^{m} B_i^{2n-1}(t) B_j^m(u) \mathrm{D}_{ij}$$
$$= (t, u, d(t, u)) \qquad (27)$$

The top view of the patch $\mathbf{D}(t, u)$ corresponding to Figure 23 is shown in Figure 24.

A side view of the $\mathbf{D}(t, u)$ patch, looking down the u axis, is shown in Figure 25. In this side view, portions of $\mathbf{D}(t, u)$ which are completely above or beneath the t-axis correspond to portions of $\mathbf{P}(t)$ for which no normal line intersects any point on $\mathbf{F_Q}(u)$. In Figure 25, the convex hull of the projected control points bounds the projection of the $\mathbf{D}(t, u)$ patch. Therefore, it is certain that regions of the t-axis which lie completely outside the convex hull of the projected $\mathbf{D}(t, u)$ control points represent portions of $\mathbf{P}(t)$ for which no normal line intersects any point on $\mathbf{F_Q}(u)$. In this example, that convex hull intersects the t-axis at points $t_{min} = 0.3020$ and $t_{max} = 0.7712$. It is concluded that $d(t, u) \neq 0$, and therefore lines normal to $\mathbf{P}(t)$ do not intersect any point on $\mathbf{F_Q}(u)$, for $t < 0.3020$ and $t > 0.7712$.

Collinear normal algorithm

All the tools have now been gathered to create an algorithm for computing all lines which are simultaneously perpendicular to two curves. Continuing the example in the previous subsection, we clip away portions of **P** which do not have normals through $\mathbf{F_Q}$. Those portions are shown in fine pen width in Figure 26.

Next, compute a focus for the remaining segment of **P**, as shown in Figure 26, and clip away regions of **P** whose normal lines do not pass through **Q**'s focus. The remaining portion of **P** is shown in heavy pen width in Figure 27, and also its focus $\mathbf{F_P}$. This iteration continues as recorded in Table 2. After four iterations, the point of tangency is actually determined to eight digits of accuracy.

If an iteration fails to reduce the parameter range of either curve by at least, say, 20%, there may be more than one collinear normal. The remedy is to split one of the curves in half and to compute the collinear normals of each half with the other curve, using a stack data structure to store pairs of curve segments.

TIMING COMPARISONS

The Bézier trim curve intersection algorithm has been implemented and some timing comparisons have been

Table 2. Parameter ranges for P(*t*) and Q(*u*)

Step	t_{min}	t_{max}	u_{min}	u_{max}
0	0	1	0	1
1	0.30204	0.77151	0.40866	0.62199
2	0.58194	0.64566	0.55427	0.55657
3	0.62443	0.62530	0.55556	0.55556
4	0.62500	0.62500	0.55556	0.55556

run against the conventional Bézier subdivision divide-and conquer algorithm. Koparkar's interval algorithm, and the implicitization algorithm. The implementation details we used for these algorithms are discussed in Reference 3. Some example timings are presented to suggest general patterns of behaviour. Comparative algorithm timings can, of course, change somewhat as the implementations are fine-tuned, if tests are run on different computers, or even if different compilers are used. In fact, the authors have improved their root finder for polynomials in Bernstein form so that it computes roots of degree 25 polynomials twice as quickly as does the root finder used in the timings in Reference 3. This resulted in a significant decrease of execution times for the implicitization curve intersection algorithm. Timing tests were run on a Macintosh II using double precision arithmetic, computing the answers to eight decimal digits of accuracy.

The columns in Table 3 indicate the relative execution time for the algorithms *Clip* = Bézier clipping algorithm, *Impl* = implicitization, *Int* = Koparkar's interval algorithm and *Sub* = the conventional Bézier subdivision algorithm. In general, the implicitization intersection algorithm is only reliable for curves of degree up to five, using double precision arithmetic. For higher degrees, it is possible for the polynomial condition to degrade so that no significant digits are obtained in the answers.

Table 3. Relative computation times

Figure	Degree	Clip	Impl.	Int	Sub.
28	3	2.5	1	10	15
29	3	1.8	1	5	6
30	5	1	1.7	3	5
31	10	1	na	2	4

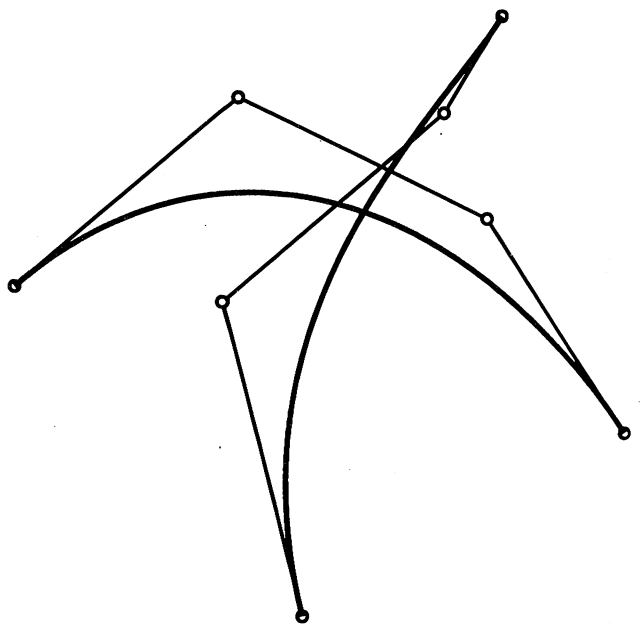


Figure 28. Degree-three example

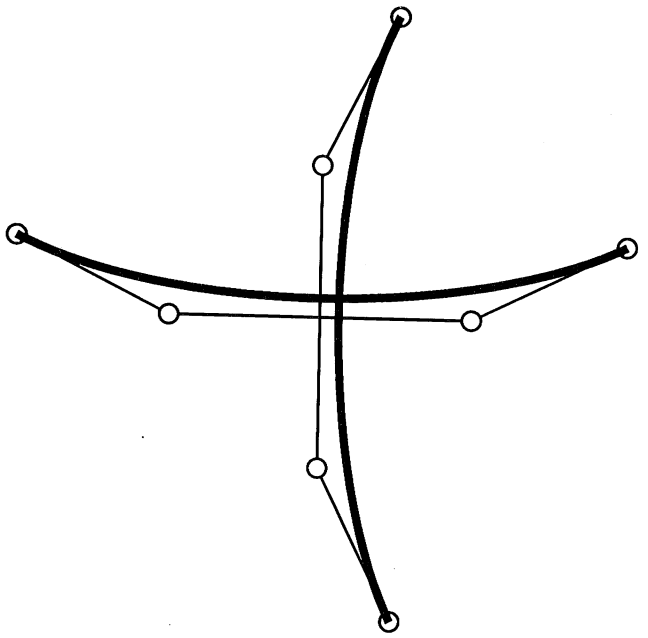


Figure 29. Degree-three example

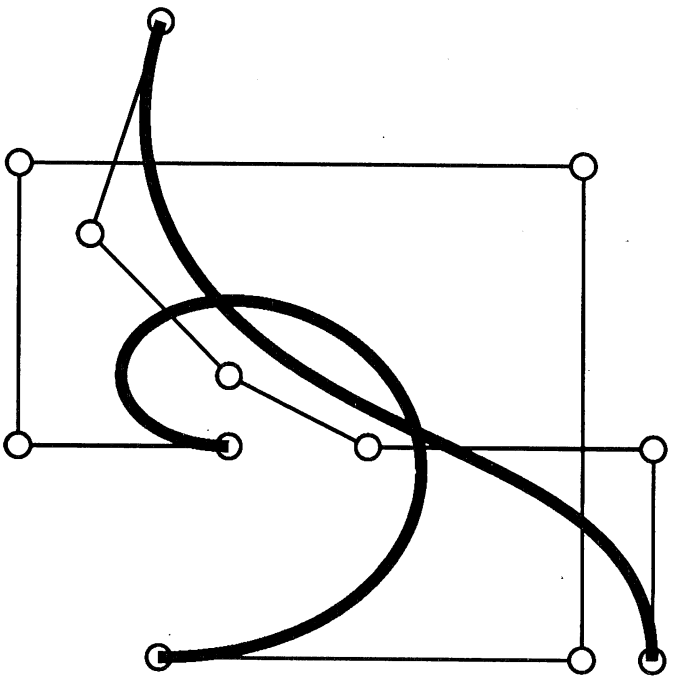


Figure 30. Degree-five example

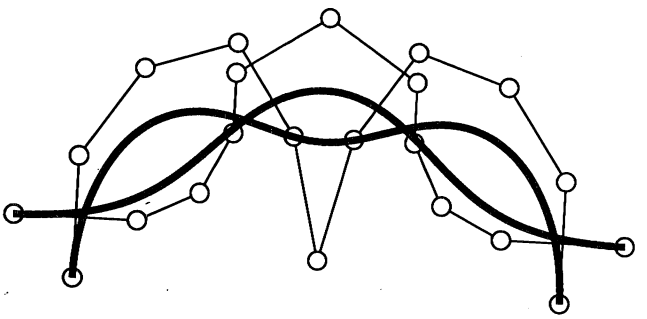


Figure 31. Degree-ten example

Our experience is that, for curves of degree less than five, the implicitization algorithm is typically between one and three times faster than the Bézier clip algorithm, which in turn is typically between two and ten times faster than the other two algorithms. For curves of degree higher than four, the Bézier clipping algorithm generally wins.

The algorithm for computing tangent intersections can typically compute a tangent intersection faster than the Bézier clip curve intersector can compute two well-spaced simple intersections. The algorithm for computing tangent intersections is designed for first-order tangencies, and its performance degrades for higher-order tangencies.

ACKNOWLEDGEMENTS

The first-named author was supported in part by the National Science Foundation under grant DMC-8657057.

REFERENCES

1 **Lane, J M and Riesenfeld, R** 'A theoretical development for the computer generation and display of piecewise polynomial surfaces' *IEEE Trans. RAMI* Vol 2 (1980) pp 35–46

2 **Koparkar, P A and Mudur, S P** 'A new class of algorithms for the processing of parametric curves' *Comput.-Aided Des.* Vol 15 (1983) pp 41–45

3 **Sederberg, T W and Parry, S R** 'Comparison of three curve intersection algorithms' *Comput.-Aided Des.* Vol 18 (1986) pp 58–63

4 **Markot, R P and Magedson, R L** 'Solutions of tangential surface and curve intersections' *Comput.-Aided Des.* Vol 21 (1989) pp 421–427

5 **Ballard, D H** 'Strip trees: a hierarchical representation for curves' *Comm. ACM* Vol 24 (1981) pp 310–321

6 **Carlson, W E** 'An algorithm and data structure for 3-D object synthesis using surface patch intersections' *Computer Graphics* Vol 16 No 3 (1982) pp 255–263

7 **Sederberg, T W, White, S C and Zundel, A K** 'Fat arcs: A bounding region with cubic convergence' *Comput. Aided Geometric Des.* Vol 6 (1989) pp 205–218

8 **Farin, G** *Curves and Surfaces for Computer Aided Geometric Design*, Academic Press (1988)

9 **Böhm, W, Farin, G and Kahmann, J** 'A survey of curve and surface methods in CAGD' *Comput. Aided Geometric Des.* Vol 1 (1984) pp 1–60

10 **Sederberg, T W, Christiansen, H N and Katz, S** 'Improved test for closed loops in surface intersections' *Comput.-Aided Des.* Vol 21 (1989) pp 505–508

11 **Bézier, P** *The Mathematical Basis of the UNISURF CAD System* Butterworths, UK (1986)

12 **Sederberg, T W and Wang, X** 'Rational hodographs' *Comput. Aided Geometric Des.* Vol 4 (1987) pp 333–335

13 **Salmon, G** *Higher Plane Curves*, G E Stechert & Co (1934)

Bézier brushstrokes

Yap Siong Chua

Traditional Chinese calligraphy and painting rely heavily on the artist's skill in executing brushstrokes. While CAD systems have been able to help the user in a variety of graphics design activities ranging from architectural and engineering drawings to very realistic 3D renderings, the infinite variations of the brushstroke still present a challenging modelling problem. The author shows some possibilities in modelling brushstrokes using Bézier techniques. Cubic Bézier functions are used to model the following attributes of brushstrokes: variation of shape or outline of a brushstroke; shade or colour variation along and across the brushstroke; some effects due to the variation of moisture content of the brush as well as paper characteristics. Idealized brushstrokes can be produced by supplying a few parameters (primarily, Bézier control points). A program processes these parameters and outputs a PostScript program which is then executed by a PostScript equipped printer.*

computer-aided design, brushstrokes, Bézier curves

CAD systems make it possible for many users to create graphics designs in a reasonable amount of time. Without CAD systems, some designs could either be very time consuming and expensive to produce using traditional methods or simply impossible. CAD systems provide powerful tools to help users create designs ranging from simple line and curve drawings to very complex architectural and engineering drawings and even very realistic 3D renderings. Users who have trouble using a compass to draw a circle can produce a perfect drawing of a circle by pressing a few keys. Those who never progressed beyond crayons can produce beautiful 3D renderings of machine parts or fictional scenery with the help of CAD systems.

Traditional Chinese painting and calligraphy share the same minimal set of material, namely, paper, brush, ink stick and ink stone. Liquid ink is produced by placing some water on the ink stone and rubbing the ink stick against the water puddle on the ink stone. The jet black ink diluted with water can produce an infinite number of shades of grey. In the hands of trained artists, bamboo leaves, pine needles, and lotus flowers come alive with skilful brushwork. Many factors influence the behaviour of the ink as it is transferred from the brush to the painting surface (usually paper or silk but simply referred to as 'paper' hereafter). Some of these factors are wetness of the brush, absorbency of the paper, inclination of the brush and pressure exerted on it, and resiliency of the hair on the brush. Using a computer to model brushstrokes taking into consideration all of these factors is extremely difficult. Previous attempts at modelling brushstrokes include Strassman[1] and Chua and Winton[2]. More work needs to be done in both the mathematical modelling of brushstrokes and the move towards a more natural user interface.

Bézier curves and surfaces have been used in designing the bodies and other parts of automobiles, aircraft, and ships. They have also been used in designing character fonts[3–5]. In this paper, the author shows some possibilities in using Bézier techniques to model brushstrokes typically found in Chinese calligraphy and painting. Cubic Bézier curves are used throughout the entire modelling process. The shape (outline) of a brushstroke depends, among many factors, on the way the brush is manipulated on the paper, particularly the inclination of the brush and the amount of pressure between the brush and the paper. The outlines of a brushstroke are modelled using piecewise cubic Bézier curves. Variation of grey shades (or colours) are also modelled with cubic Bézier functions. For further realism, the wetness of the brushstroke is also taken into consideration. A cubic Bézier function is used to model the effect of decreasing amount of moisture in a continuous stroke. The mathematics behind Bézier and other parametric curves can be found in many sources[6–9] and will not be duplicated here. All cubic Bézier functions used in this paper are specified by providing four control points $\{\mathbf{P}_i, i = 0, 1, 2, 3\}$ resulting in a set of parametric curves passing through points $\mathbf{P}_0$ and $\mathbf{P}_3$. Cubic Bézier functions in 2D are used in most of the modelling process. A 3D cubic Bézier function in RGB (Red, Green, Blue) space is used in modelling the variation of colours.

SHAPE OR OUTLINE OF BRUSHSTROKE

Consider the process of executing a simple brushstroke. The brush is dipped in ink (and water for shade variations) and lowered to the paper. With varying pressure, inclination and direction of movement, ink is transferred from the brush to the paper. At the end of the stroke, the brush is lifted. The stroke may start with a fine point, with just the very tip of the brush touching,

College of Computer & Information Sciences, University of North Florida, Jacksonville, FL 32216, USA

*PostScript is a registered trademark of Adobe Systems Inc.

Paper received: 15 December 1989. Revised: 26 May 1990

 0010–4485/90/090550–06