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, locus, 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, the interval subdivision method adapted by Koparkar and Mudur, and implicitization. Implementations of those algorithms have suggested that implicitization is easiest to implement of those algorithms for curves of degree less than five. For higher degrees, the interval algorithm is generally fastest.

An algorithm for computing points of tangency between two parametric curves has recently been proposed, 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.

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 $P_0 - P_r$. A fat line is chosen parallel to $\bar{L}$ as shown in Figure 1. If $\bar{L}$ is defined in its normalized implicit equation

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

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

$$d(x, y) = ax + by + c$$

Denote by $d_i = d(x_i, y_i)$ the signed distance from control point $P_i = (x_i, y_i)$ 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:

$$\{(x, y) | d_{\text{min}} \leq d(x, y) \leq d_{\text{max}}\}$$

where

$$d_{\text{min}} = \min\{d_0, \ldots, d_n\}$$

$$d_{\text{max}} = \max\{d_0, \ldots, d_n\}$$

Figure 1. Fat line bounding a quartic curve

---

Engineering Computer Graphics Lab, Brigham Young University, Provo, UT 84602, USA
*Department of Electrical Engineering, Fukuyama University, Fukuyama, Japan

0010-4485/90/090538-12 © 1990 Butterworth-Heinemann Ltd

computer-aided design
These values for $d_{\text{min}}$ and $d_{\text{max}}$ are conservative. For polynomial Bézier curves (all weights = 1) of degree two and three (the most common cases), values of $d_{\text{min}}$ and $d_{\text{max}}$ 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 $P(t)$ to $L$, then, for polynomial quadratic Bézier curves (see Figure 2)

$$d(t) = 2(1 - t)d_1$$

from which the tight bounds are

$$d_{\text{min}} = \min \left\{ 0, \frac{d_1}{2} \right\}$$

$$d_{\text{max}} = \max \left\{ 0, \frac{d_1}{2} \right\}$$

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

$$d(t) = 3(1 - t)\sqrt{(1 - t)d_1 + td_2}$$

(7)

The function $d(t)$ has an extremum where $d'(t) = 0$. If $d_1d_2 > 0$, there is one extremum at

$$t = \frac{d_1}{2d_1 + \sqrt{d_1^2 - d_1d_2 + d_2^2}}$$

(8)

and

$$d_{\text{min}} = \min \left\{ 0, d(t_1) \right\}$$

$$d_{\text{max}} = \max \left\{ 0, d(t_1) \right\}$$

(9)

If $d_1d_2 \leq 0$, there are two extrema, at

$$t_1 = \frac{2d_1 - d_2 + \sqrt{d_1^2 + d_2^2 - d_1d_2}}{3(d_1 - d_2)}$$

and

$$t_2 = \frac{2d_1 - d_2 - \sqrt{d_1^2 + d_2^2 - d_1d_2}}{3(d_1 - d_2)}$$

(10)

and

$$d_{\text{min}} = \min \left\{ d(t_1), d(t_2) \right\}$$

$$d_{\text{max}} = \max \left\{ d(t_1), d(t_2) \right\}$$

(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 $d_{\text{min}}$ and $d_{\text{max}}$. From equation (7), if $d_1d_2 > 0$,

$$\min \left\{ 0, d_1, d_2 \right\} 3t(1 - t) \leq d(t) \leq \max \left\{ 0, d_1, d_2 \right\} 3t(1 - t)$$

(12)

Thus if $d_1d_2 > 0$, use

$$d_{\text{min}} = \frac{3}{4} \min \left\{ 0, d_1, d_2 \right\}$$

$$d_{\text{max}} = \frac{3}{4} \max \left\{ 0, d_1, d_2 \right\}$$

(13)

From equation (7), if $d_1 \leq 0$ and $d_2 \geq 0$, then

$$3t(1 - t)d_1 \leq d(t) \leq 3t'(1 - t)d_1$$

(14)

Thus, if $d_1d_2 \leq 0$, use

$$d_{\text{min}} = \frac{4}{9} \min \left\{ 0, d_1, d_2 \right\}$$

$$d_{\text{max}} = \frac{4}{9} \max \left\{ 0, d_1, d_2 \right\}$$

(15)

These fat lines are illustrated in Figure 3.

**Bézier clipping**

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

![Figure 2. Fat line for a polynomial quadratic curve](image)

![Figure 3. Fat lines for polynomial cubic curves](image)
of $t$ for which $P(t)$ lies outside of $L$, and hence for which $P(t)$ does not intersect $Q(u)$.

$P$ is defined by its parametric equation

$$P(t) = \sum_{i=0}^{n} P_i B_i^n(t) \quad (16)$$

where $P_i = (x_i, y_i)$ are the Bézier control points, and $B_i^n(t) = \binom{n}{i}(1-t)^n t^i$ denote the Bernstein basis functions. If the line $L$ through $P_0 - P_n$ is defined by

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

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

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

Note that $d(t) = 0$ for all values of $t$ at which $P$ intersects $L$. Also, $d_i$ is the distance from $P_i$ to $L$ (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 as follows:

$$D(t) = (t, d(t)) = \sum_{i=0}^{n} D_i B_i^n(t) \quad (19)$$

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

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

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

**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 curve intersection algorithm proceeds by iteratively applying the Bézier clipping procedure.

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

Next, $P(t)$ is again clipped against $Q(u)$, 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 $P_0 - P_n$ provides a larger Bézier clip than does the fat line parallel to the line $P_0 - P_n$. Figure 8 shows...
Table 1. Parameter ranges for $P(t)$ and $Q(u)$

<table>
<thead>
<tr>
<th>Step</th>
<th>$t_{\min}$</th>
<th>$t_{\max}$</th>
<th>$u_{\min}$</th>
<th>$u_{\max}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
<td>0.75</td>
<td>0.4188</td>
<td>0.6303</td>
</tr>
<tr>
<td>2</td>
<td>0.3747</td>
<td>0.4105</td>
<td>0.5121</td>
<td>0.5143</td>
</tr>
<tr>
<td>3</td>
<td>0.382079</td>
<td>0.382079</td>
<td>0.512967</td>
<td>0.512967</td>
</tr>
</tbody>
</table>

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.

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

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.
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 \( \mathbf{P} \) is a rational Bézier curve

\[
\mathbf{P}(t) = \frac{\sum_{i=0}^{n} w_i \mathbf{B}_i^p(t)}{\sum_{i=0}^{n} w_i \mathbf{B}_i^p(t)}
\]  

(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:

\[
d(t) = \sum_{i=0}^{n} d_i \mathbf{B}_i^p(t)
\]

\[
d_i = w_i (ax_i + by_i + c)
\]

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 \( \{ (x, y) = \mathbf{P}(t) \mid d_{\text{min}} \leq d(t) \leq d_{\text{max}} \} \). Instead, \( \mathbf{P} \) must be clipped independently against each of the two lines bounding the fat line. Thus, ranges of \( t \) are identified for which

\[
\sum_{i=0}^{n} w_i (ax_i + by_i + c - d_{\text{max}}) \mathbf{B}_i^p(t) > 0
\]

or for which

\[
\sum_{i=0}^{n} w_i (ax_i + by_i + c + d_{\text{max}}) \mathbf{B}_i^p(t) < 0
\]

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 locus.

Figure 11. Collinear normal line
Bézier curve focus

A focus of a Bézier curve $P(t)$ is defined to be any curve through which all lines perpendicular to $P(t)$ pass. By this definition, a focus is not unique. For example, $P(t)$ 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 $N(t)$ defines any vector function (not necessarily of unit length) which is perpendicular to $P(t)$ for all values of $t$, then the curve $F(t) = P(t) + c(t)N(t)$ (where $c(t)$ is any function of $t$) can serve as a focus curve.

$N(t)$ is taken to be the rotated hodograph of $P(t)$. The hodograph $H(t)$ of a degree $n$ Bézier curve $P(t)$ is the first derivative of that curve, $P'(t)$. The hodograph is itself a Bézier curve of degree $n-1$, whose control points $H$, are $n(P_{n-1} - P)$ (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 $P'(t)$ is expressed as a vector from the hodograph origin to the point $H(t)$ on the hodograph (see Figure 17). By rotating the hodograph through $90^\circ$ about its origin, a vector function $N(t)$ is obtained which defines vectors perpendicular to $P(t)$ (see Figure 18). In creating the focus $F(t) = P(t) + c(t)N(t)$, $c(t)$ is chosen to be a degree-one polynomial $C(t) = c_0(1-t) + c_1t$ which satisfies the condition $F(0) = F(1)$. This is a heuristic decision, motivated by the observation that if $F(t)$ 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

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

(21)
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 impractically 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 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*\(^3\) 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 \( P(t) \) do not pass through a given point \( F \). The parameter values \( t \) for which lines normal to \( P(t) \) pass through \( F \) are the zeros of the equation

\[
d(t) = P'(t) \cdot (P(t) - F) = 0
\]

(22)

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

\[
d_i = \sum_{j+k=i \atop j \in \{0, \ldots, n\}} \binom{n}{j} \binom{n-1}{k} \frac{1}{(2n-1)j!} n(P_{k+1} - P_j) \cdot (P_j - F)
\]

(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 \( P(t) \) passing through \( F \).

Next consider the case where \( F \) is not a point, but is itself a Bézier curve. Specifically, \( F_Q(u) = \sum_{m=0}^{n} F_m B_m^u(u) \) is a focus of a Bézier curve \( Q(u) \) (see Figure 23). It is now desired to identify regions of \( P(t) \) whose normal lines do not pass through any point on \( F_Q(u) \) — that is,
to find ranges of $t$ for which

$$D(t, u) = \frac{(P(t) - F_Q(u)) \cdot P'(t)}{n} \neq 0, \ 0 \leq u \leq 1$$

(D.24)

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

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

(D.25)

where

$$d_i = \sum_{k+l=i} \binom{n}{k} \binom{n-1}{l} \frac{1}{(2n-1)} \binom{2n-1}{j} n(P_{k+1} - P_i) \cdot (P_l - F_i)$$

(D.26)

The function $D(t, u)$ can be represented, in a $(t, u, D)$ coordinate system, as a so-called ‘non-parametric’ Bézier surface patch² whose control points $D_i = (t_i, u_i, D_i)$ are evenly spaced in $t$ and $u$: $t_i = l/(2n-1)$,
\[ u_i = \frac{j}{m}. \] A point on such a patch has coordinates

\[ D(t, u) = \sum_{i=0}^{2n-1} \sum_{j=0}^{m} B_i^{2n-1}(t) B_j^m(u) D_{ij} = (t, u, d(t, u)) \]  

(27)

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

A side view of the \( D(t, u) \) patch, looking down the \( u \) axis, is shown in Figure 25. In this side view, portions of \( D(t, u) \) which are completely above or beneath the \( t \)-axis correspond to portions of \( P(t) \) for which no normal line intersects any point on \( F_Q(u) \). In Figure 25, the convex hull of the projected control points bounds the projection of the \( D(t, u) \) patch. Therefore, it is certain that regions of the \( t \)-axis which lie completely outside the convex hull of the projected \( D(t, u) \) control points represent portions of \( P(t) \) for which no normal line intersects any point on \( F_Q(u) \). 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 \( P(t) \) do not intersect any point on \( F_Q(u) \), 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 \( F_Q \). 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 \( F_P \). 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 made.

<table>
<thead>
<tr>
<th>Step</th>
<th>( t_{\min} )</th>
<th>( t_{\max} )</th>
<th>( u_{\min} )</th>
<th>( u_{\max} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.3020</td>
<td>0.77151</td>
<td>0.40866</td>
<td>0.62199</td>
</tr>
<tr>
<td>2</td>
<td>0.58194</td>
<td>0.64566</td>
<td>0.55427</td>
<td>0.55657</td>
</tr>
<tr>
<td>.3</td>
<td>0.62443</td>
<td>0.62330</td>
<td>0.55556</td>
<td>0.55556</td>
</tr>
<tr>
<td>4</td>
<td>0.62500</td>
<td>0.62500</td>
<td>0.55556</td>
<td>0.55556</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Figure</th>
<th>Degree</th>
<th>Clip</th>
<th>Impl.</th>
<th>Int</th>
<th>Sub.</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>3</td>
<td>2.5</td>
<td>1</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>1.8</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>1</td>
<td>1.7</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>31</td>
<td>10</td>
<td>1</td>
<td>na</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 29. Degree-three example
Figure 28. 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 intersecter 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


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


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