

## Chapter 7

# CURVE INTERSECTION

Several algorithms address the problem of computing the points at which two curves intersect. Predominant approaches are the Bézier subdivision algorithm [?], the interval subdivision method adapted by Koparkar and Mudur [?], implicitization [?], and Bézier clipping [?].

### 7.1 Timing Comparisons

A few sample timing comparisons for these four methods are presented in [?]. 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. These 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 ?? 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

Example	Degree	Clip	Impl.	Int	Sub.
1	3	2.5	1	10	15
2	3	1.8	1	5	6
3	5	1	1.7	3	5
4	10	1	na	2	4

Table 7.1: Relative computation times

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. For curves of degree less than five, the implicitization algorithm is typically 1-3 times faster than the Bézier clip algorithm, which in turn is typically 2-10 times faster than the other two algorithms. For curves of degree higher than four, the Bézier clipping algorithm generally wins.

An brief discussion of these curve intersection methods follows.

## 7.2 Bézier subdivision

The Bézier subdivision curve intersection algorithm relies on the convex hull property and the de Casteljau algorithm. Though we overview it in terms of Bézier curves, it will work for any curve which obeys the convex hull property. Figure ?? shows the convex hull of a single Bézier curve, and the convex hulls after subdividing into two and four pieces.

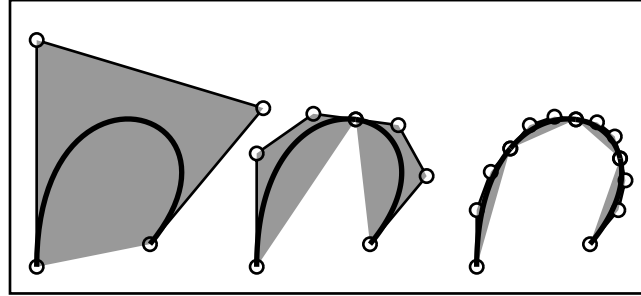


Figure 7.1: Convex Hulls

The intersection algorithm proceeds by comparing the convex hulls of the two curves. If they do not overlap, the curves do not intersect. If they do overlap, the curves are subdivided and the two halves of one curve are checked for overlap against the two halves of the other curve. As this procedure continues, each iteration rejects regions of curves which do not contain intersection points. Figure ?? shows three iterations.

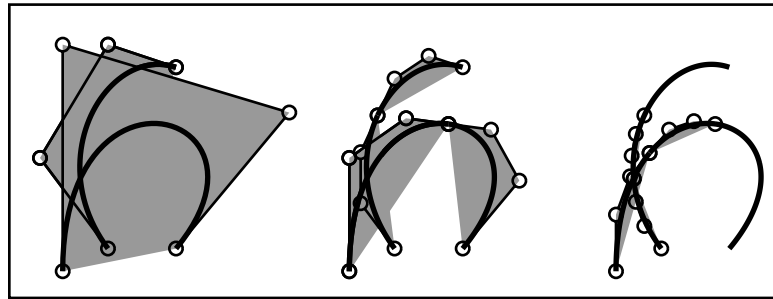


Figure 7.2: Three iterations of Bézier subdivision

Once a pair of curves has been subdivided enough that they can each be approximated by a line segment to within a tolerance  $\epsilon$  (as given in equation 6.10), the intersection of the two approximating line segments is found.

Since convex hulls are rather expensive to compute and to determine overlap, in practice one normally just uses the  $x - y$  bounding boxes.

## 7.3 Interval subdivision

The interval subdivision algorithm method is similar in spirit to the Bézier subdivision method. In this case, each curve is preprocessed to determine its vertical and horizontal tangents, and divided

into “intervals” which have such tangents only at endpoints, if at all. Note that within any such interval, a rectangle whose diagonal is defined by any two points on the curve completely bounds the curve between those two endpoints. The power of this method lies in the fact that subdivision can now be performed by merely evaluating the curve (using Horner’s method instead of the more expensive de Casteljau algorithm) and that the bounding box is trivial to determine. Figure ?? illustrates this bounding box strategy.

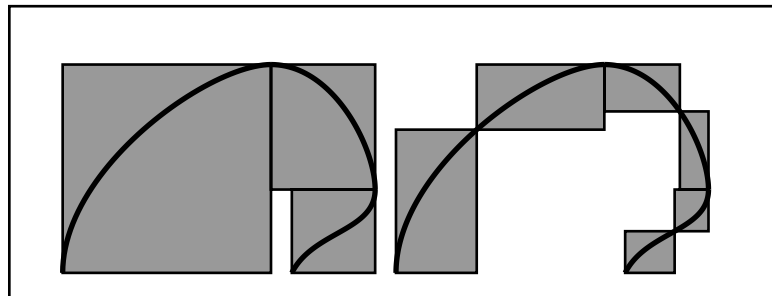


Figure 7.3: Interval preprocess and subdivision

## 7.4 Implicitization

The implicitization curve intersection algorithm is briefly sketched in section ?. For numerical stability, if the computations are to be performed in floating points, one should implicitize in the Bernstein basis as discussed in section ?.

## 7.5 Bézier Clipping method

The method of Bézier clipping has many applications, such as ray tracing trimmed rational surface patches [?], algebraic curve intersection [?], and tangent intersection computation [?]. It can be viewed as kind of an interval Newton’s method, because it has quadratic convergence, but robustly finds all intersections. Since it is such a powerful tool, and since it is based on some ideas that haven’t been discussed previously in these notes, the majority of this chapter is devoted to this method.

### 7.5.1 Fat Lines

Define a *fat line* as the region between two parallel lines. Our curve intersection algorithm begins by computing a fat line which bounds one of the two Bézier curves. Similar bounds have been suggested in references [?, ?].

Denote by  $\bar{L}$  the line  $\mathbf{P}_0 - \mathbf{P}_n$ . We choose a fat line parallel to  $\bar{L}$  as shown in Figure ?. If  $\bar{L}$  is defined in its normalized implicit equation

$$(7.1) \quad ax + by + c = 0 \quad (a^2 + b^2 = 1)$$

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

$$(7.2) \quad d(x, y) = ax + by + c$$

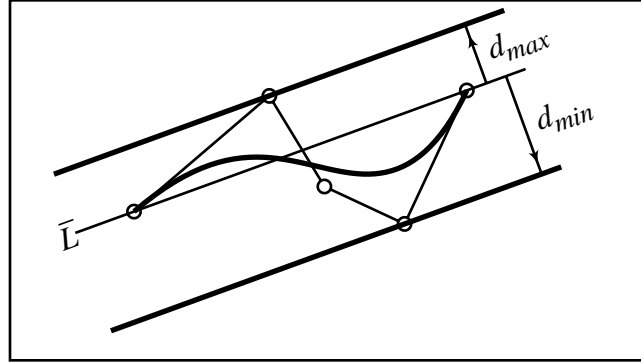


Figure 7.4: Fat line bounding a quartic curve

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:

$$(7.3) \quad \{(x, y) | d_{min} \leq d(x, y) \leq d_{max}\}$$

where

$$(7.4) \quad d_{min} = \min\{d_0, \dots, d_n\}, \quad d_{max} = \max\{d_0, \dots, d_n\}.$$

### Quadratic case

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.

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

$$(7.5) \quad d(t) = 2t(1-t)d_1$$

from which the tight bounds are

$$(7.6) \quad d_{min} = \min\{0, \frac{d_1}{2}\}, \quad d_{max} = \max\{0, \frac{d_1}{2}\}.$$

### 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,

$$(7.7) \quad d(t) = 3t(1-t)[(1-t)d_1 + td_2].$$

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

$$(7.8) \quad t_1 = \frac{d_1}{2d_1 - d_2 + \sqrt{d_1^2 - d_1d_2 + d_2^2}}$$

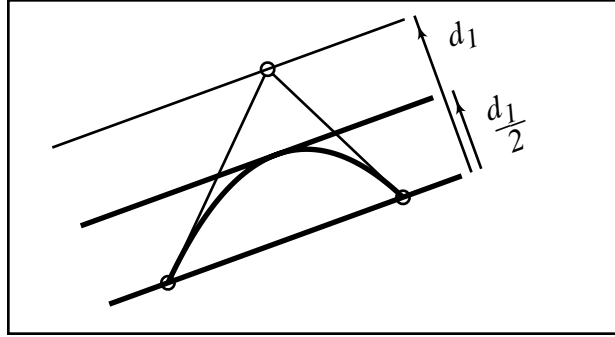


Figure 7.5: Fat line for a polynomial quadratic curve

and

$$(7.9) \quad d_{min} = \min\{0, d(t_1)\}, \quad d_{max} = \max\{0, d(t_1)\}.$$

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

$$(7.10) \quad \begin{aligned} t_1 &= \frac{2d_1 - d_2 + \sqrt{d_1^2 + d_2^2 - d_1 d_2}}{3(d_1 - d_2)} \\ t_2 &= \frac{2d_1 - d_2 - \sqrt{d_1^2 + d_2^2 - d_1 d_2}}{3(d_1 - d_2)} \end{aligned}$$

and

$$(7.11) \quad d_{min} = \min\{d(t_1), d(t_2)\}, \quad d_{max} = \max\{d(t_1), d(t_2)\}.$$

In our experience, 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 ??, if  $d_1 d_2 > 0$ ,

$$(7.12) \quad \min\{0, d_1, d_2\} 3t(1-t)d(t) \leq \max\{0, d_1, d_2\} 3t(1-t).$$

Thus if  $d_1 d_2 > 0$ , use

$$(7.13) \quad d_{min} = \frac{3}{4} \min\{0, d_1, d_2\}, \quad d_{max} = \frac{3}{4} \max\{0, d_1, d_2\}.$$

From equation ??, if  $d_1 \leq 0$  and  $d_2 \geq 0$ , then

$$(7.14) \quad 3t(1-t)^2 d_1 \leq d(t) \leq 3t^2(1-t) d_2.$$

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

$$(7.15) \quad d_{min} = \frac{4}{9} \min\{0, d_1, d_2\}, \quad d_{max} = \frac{4}{9} \max\{0, d_1, d_2\}.$$

These fat lines are illustrated in Figure ??.

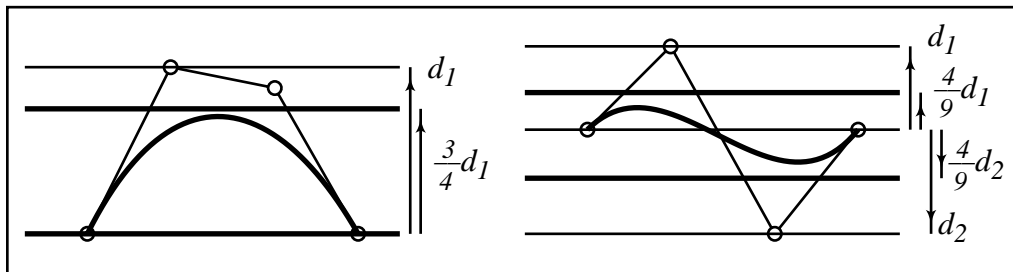


Figure 7.6: Fat lines for polynomial cubic curves

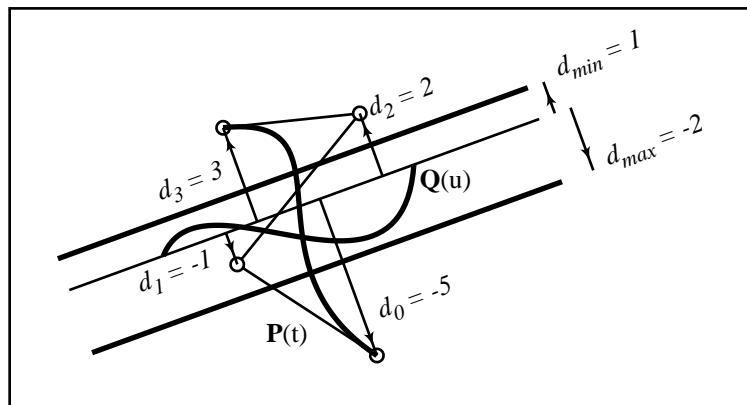


Figure 7.7: Bézier curve/fat line intersection

### 7.5.2 Bézier Clipping

Figure ?? shows two polynomial cubic Bézier curves  $\mathbf{P}(t)$  and  $\mathbf{Q}(u)$ , and a fat line  $\mathbf{L}$  which bounds  $\mathbf{Q}(u)$ . In this section, we discuss how to identify intervals 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

$$(7.16) \quad \mathbf{P}(t) = \sum_{i=0}^n \mathbf{P}_i B_i^n(t)$$

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  $\mathbf{Q}_0 - \mathbf{Q}_n$  is defined by

$$(7.17) \quad ax + by + c = 0 \quad (a^2 + b^2 = 1),$$

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

$$(7.18) \quad d(t) = \sum_{i=0}^n d_i B_i^n(t), \quad d_i = ax_i + by_i + c.$$

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 ??).

The function  $d(t)$  is a polynomial in Bernstein form, and can be represented as an explicit Bézier curve (Section 2.12) as follows:

$$(7.19) \quad \mathbf{D}(t) = (t, d(t)) = \sum_{i=0}^n \mathbf{D}_i B_i^n(t).$$

Figure ?? shows the curve  $\mathbf{D}(t)$  which corresponds to Figure ??.

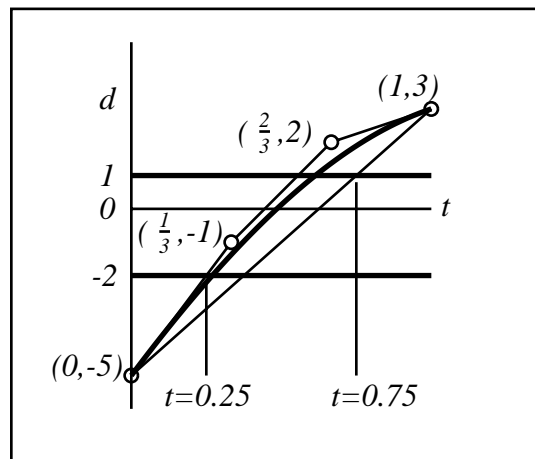


Figure 7.8: Explicit Bézier curve

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  $d = d_{min}$ . We can identify parameter ranges of  $t$  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, we are assured 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, such that portions of  $\mathbf{P}$  over parameter values  $t < 0.25$  and  $t > 0.75$  are removed.

### 7.5.3 Iterating

We have just discussed the notion of Bézier clipping 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. Our Bézier clipping curve intersection algorithm proceeds by iteratively applying the Bézier clipping procedure.

Figure ?? shows curves  $\mathbf{P}(t)$  and  $\mathbf{Q}(u)$  from Figure ?? 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 fine pen width, and a fat line is shown which bounds  $\mathbf{P}(t)$ ,  $0.25 \leq t \leq 0.75$ . 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, we

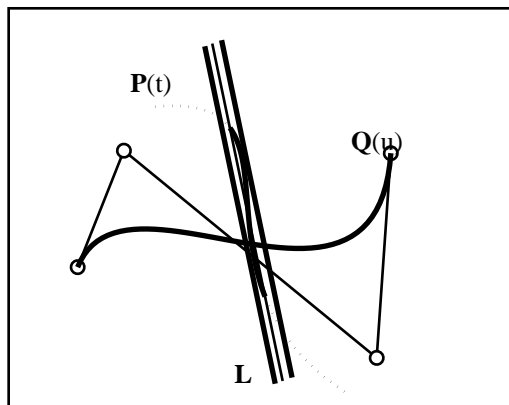


Figure 7.9: After first Bézier clip

define an explicit Bézier curve which expresses the distance from  $\bar{L}$  in Figure ?? to the curve  $\mathbf{Q}(u)$ , from which we conclude that it is safe to clip off regions of  $\mathbf{Q}(u)$  for which  $u < .42$  and  $u > .63$ .

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

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

Table 7.2: Parameter ranges for  $\mathbf{P}(t)$  and  $\mathbf{Q}(u)$ .

### 7.5.4 Clipping to other fat lines

The fat line defined in section ?? 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 fatline. In many cases, a fat line perpendicular to line  $\mathbf{P}_0 - \mathbf{P}_n$  provides a larger Bézier clip than does the fat line parallel to line  $\mathbf{P}_0 - \mathbf{P}_n$ . We suggest that in general it works best to examine both fat lines to determine which one provides the largest clip. This extra overhead reaps a slightly lower average execution time.

### 7.5.5 Multiple Intersections

Figure ?? shows a case where two intersection points exist. In this case, no Bézier clipping is possible

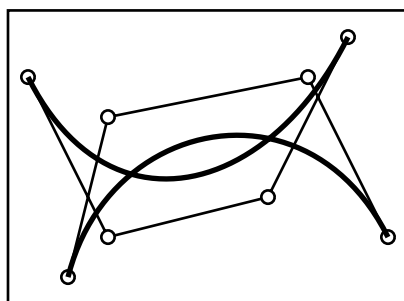


Figure 7.10: Two intersections

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 ???. A stack data structure is used to store pairs of curve segments, as in the conventional

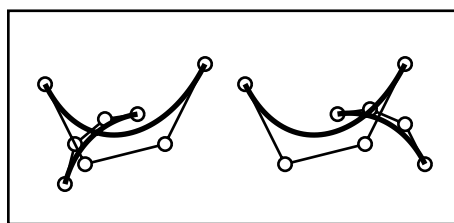


Figure 7.11: Two intersections, after a split

divide-and-conquer intersection algorithm [?].

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

### 7.5.6 Rational Curves

If  $\mathbf{P}$  is a *rational* Bézier curve

$$(7.20) \quad \mathbf{P}(t) = \frac{\sum_{i=0}^n w_i \mathbf{P}_i B_i^n(t)}{\sum_{i=0}^n w_i B_i^n(t)}$$

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 ?? into equation ?? and clearing the denominator yields:

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

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  $\{(x, y) = \mathbf{P}(t) | d_{min} \leq d(t) \leq d_{max}\}$ . Instead,  $\mathbf{P}$  must be clipped independently against each of the two lines bounding the fat line. Thus, we identify ranges of  $t$  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.

## 7.6 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. [?] presents an algorithm for quickly isolating two adjacent intersections, and for computing tangent intersections, based on Bézier clipping. 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  $90^\circ$ , 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 [?].

Thus, if we can compute a line which is perpendicular to both curves, we can thereby isolate two close intersections. We will refer to such a line 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 ?? shows a collinear normal in the case of two distinct intersections, and Figure ?? shows the case of a tangent intersection.

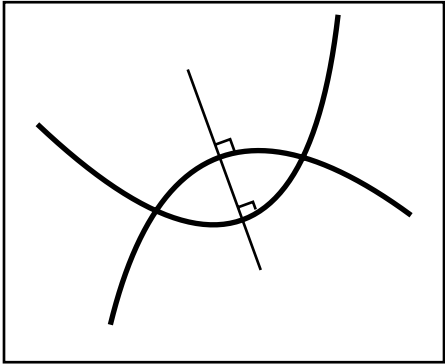


Figure 7.12: Collinear normal line

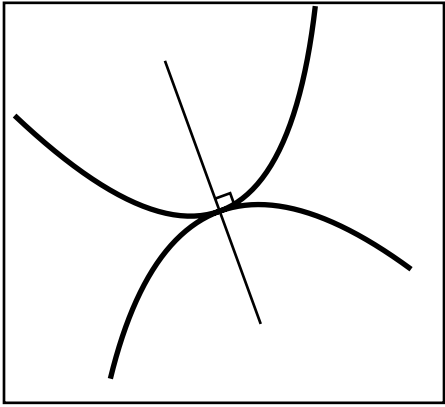


Figure 7.13: Tangent intersection

