The Visual Computer

A localized method for intersecting plane algebraic curve segments

J.K. Johnstone[*] and M.T. Goodrich[**]

Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218, USA

We present a local method for the computation of the intersections of plane algebraic curve segments. The conventional method of intersection is global, because it must first find all of the intersections between two curves before it can restrict the segments in question; hence, it cannot take advantage of situations dealing with the intersection of short-curve segments on complex curves. Our local method, on the other hand, will directly find only those intersections that lie on the segments, as it is based upon an extension of methods for tracing along a curve.

Key words: Intersection – Algebraic curves – Curve tracing – Plane sweep – Resultants – Theory of elimination – Geometric modeling

[*] This author's research was supported by the National Science Foundation under Grant IRI-8910366
[**] This author's research was supported by the National Science Foundation under Grant CCR-8810568

1 Introduction

Intersection is one of the most universal and basic problems in geometric modeling. Although all of the Boolean operations are axiomatic to geometric modeling, intersection is particularly important and particularly challenging. It is required for the definition of a geometric model [e.g., CSG (Requicha 1980)] and is fundamental to the model's applications, such as interference detection or hidden-surface elimination (Sechrest and Greenberg 1982; Mortenson 1985; McKenna 1986). The classic view of the intersection of algebraic curves and surfaces is that it is equivalent to the solution of a simultaneous system of equations, such as $\{f(x, y)=0, g(x, y)=0\}$ for two plane algebraic curves.[1] Canonically, the system of equations is reduced to a single univariate equation, the univariate equation is solved, and full solutions are built from these partial solutions. An artifact of this approach is that all of the intersections are found. However, in geometric modeling a person is usually interested only in the intersections between two short segments of the curves. Thus, following this approach, one must first compute all of the intersections between the curves and then decide which of these intersections actually lie on the segments [a decidedly nontrivial decision involving the sorting of points along a curve (Johnstone 1987; Johnstone and Bajaj 1990)]. Therefore, segment intersection is actually more complex than curve intersection when the traditional approach is adopted.

Therefore, a method that makes segment intersection simpler than curve intersection, especially if the segments are short, is needed. This is especially urgent, because as geometric models become more complex intersection of higher-degree curves with the global system of equations method becomes prohibitively expensive, while the curve segments involved remain short. In this paper, we present a new method for intersecting plane algebraic curve segments. The method is input-sensitive: the simpler and shorter the segment, the more efficient the intersection computation.

Our method is based upon crawling (or tracing), a method for moving along a curve that has received much attention of late (Timmer 1977; Dobkin et al. 1986; Hoffmann 1987; Owen and Rockwood 1987; Bajaj et al. 1988). We find intersections by crawling along the two segments in a coordinat-

[1] In this paper, we deal with curves that are defined using implicit representation. Thus, methods of intersection that rely on parametric representation (see Mortenson 1985) are not applicable

ed fashion. Because crawling is a method for moving along a single curve, we must adapt it to two curves (Sect. 3). This crawling is easiest if both segments are *xy-monotone* (monotone with respect to both coordinate axes), so that curve segments are first decomposed into xy-monotone segments (Sect. 7). Two methods of coordinated crawling along xy-monotone segments are presented: the simultaneous and the staircase crawl (Sects. 4 and 5). Several optimizations are also suggested, including a method of recognizing when two xy-monotone segments cannot intersect and two methods of eliminating long crawls within a coordinated crawl (Sect. 8). A variant of the plane-sweep method (familiar from computational geometry: Bentley and Ottmann 1979; Preparata and Shamos 1985; Edelsbrunner 1987) is used to find the intersections of the collection of xy-monotone segments comprising the two segments we are intersecting, using several calls to coordinated crawling (Sect. 9). We end with some conclusions.

2 Elimination method of intersection

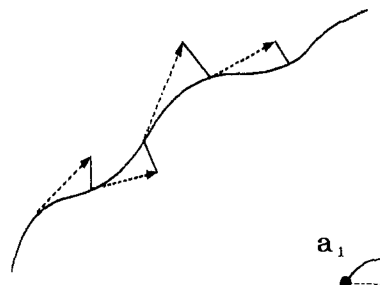
In this section, we review the global method of intersection, which we refer to as the *elimination* method of intersection. An example will clarify the details of this method. It should be noted that the use of Groebner bases, rather than elimination, to perform curve intersection is also an essentially global method.

Example 2.1 Suppose that we wish to find the intersection of the two plane algebraic curves $f(x, y)=0$ and $g(x, y)=0$. The two equations are reduced to a single equation and a variable is eliminated by taking the Sylvester or Bezout/Cayley resultant $h(x)$ of f and g with respect to y (Walker 1950; van der Waerden 1953; Sederberg et al. 1984). Even if there are more than two equations, techniques from the theory of elimination can be used to reduce the system to a single univariate equation by using several rounds of Sylvester resultants or a multivariate resultant. Next, the univariate equation (which encodes the common roots) is solved (e.g., by Newton's method), yielding one coordinate for each solution of the original system. For example, if x_0 is a root of $h(x)$, then there exists y_0 such that $f(x_0, y_0)=0=g(x_0, y_0)$. Finally, the full solutions are built up from these partial solutions by solving more univariate equations. For example, $f(x_0, y)$ is solved, yielding $y_1, \ldots, y_k$, and y_is such that $g(x_0, y_i)\neq 0$ are discarded; the remaining y_js define intersections (x_0, y_i). Alternatively, full solutions can be found from partial solutions by computing a birational map that maps the projections x_i of the roots back to the actual curve intersections, as described by Abhyankar and Bajaj (1989) and Garrity and Warren (1989).

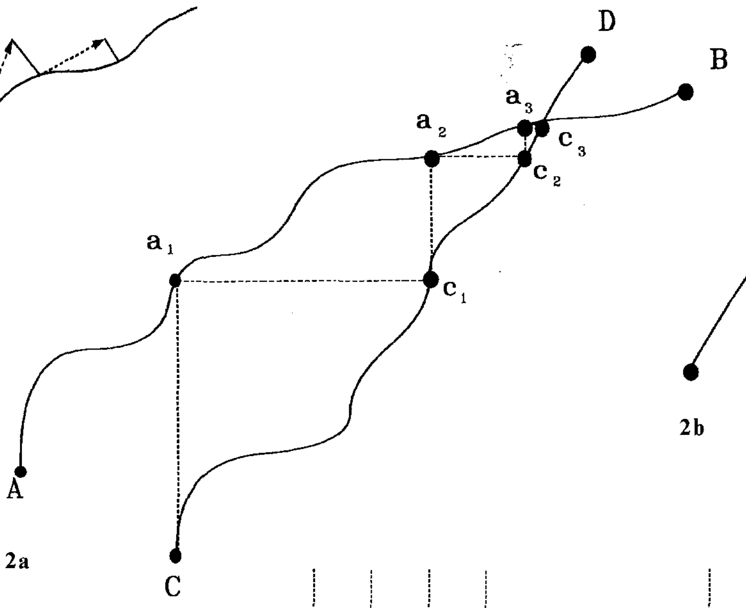
Note that all of the intersections between the curves are found. In particular, it is impossible to find only the intersections on a given segment of each curve with this method. It is possible to restrict the intersections in a certain range of x and range of y. However, even if it is known that the desired segment lies in this range, there may be many other segments that also lie in this range. There is no way of predicting where the intersection will be until it is fully computed. Moreover, segment intersection requires expensive postprocessing procedures. The intersections must be sorted along each curve (Johnstone 1987; Johnstone and Bajaj 1990) and those that are not between the endpoints of the appropriate segment must be discarded. Thus, with the elimination method, segment intersection is actually more complex than curve intersection. Another problem with the elimination method is that it requires the solution of a univariate equation of high degree. In particular, the degree of the resultant polynomial $h(x)$ is potentially the product of the degree of $f(x, y)$ and the degree of $g(x, y)$.

3 Coordinated crawling

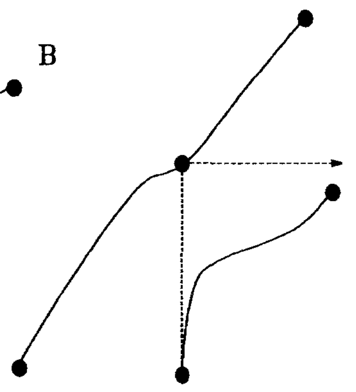
In this section, we give a short introduction to crawling and a general overview of coordinated crawling. The reader is referred to Hoffmann (1987) and Bajaj et al. (1988) for the details of crawling. Crawling is a method of traversing a curve. Progress is made by repeatedly making short steps away from the curve and relaxing back onto the curve (Fig. 1). There are various ways of stepping away from the curve, such as stepping along the tangent or in a direction parallel to the axes. The relaxation back onto the curve can be achieved with Newton's method. One of the useful properties of crawling is its locality: it relies only on the behavior of the curve in a restricted neighborhood of the current position.



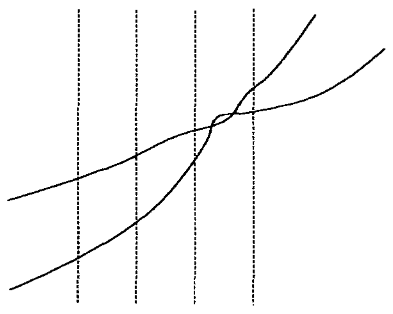
1



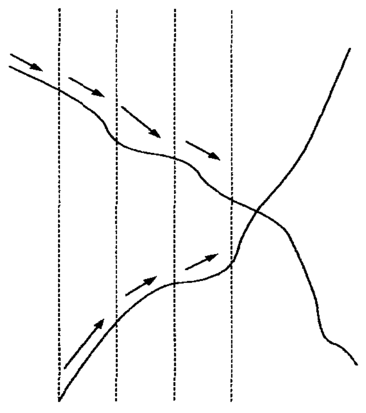
2a



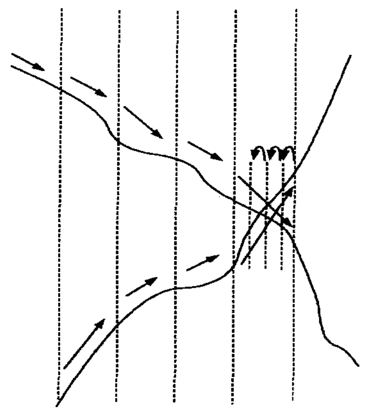
2b



4



3a



3b

Fig. 1. Crawling along a curve

Fig. 2a, b. Two staircase crawls

Fig. 3a, b. Simultaneous crawl (**a**); at the end (**b**) it must back up in fine increments

Fig. 4. The danger of a simultaneous crawl along two rising segments

The size of each step of a crawl can be adjusted. It cannot be too large, because it might lose the curve, and it must be particularly small near singularities and other places where confusion is likely. However, within the bounds of these restrictions, it is possible to talk of coarse crawls with large steps and fine crawls with small steps. We shall be intent upon keeping the crawl as coarse as possible, because the larger the steps, the faster the crawl.

One of the contributions of this paper is to show that crawling can also be used to discover the intersections between two xy-monotone curve segments. Let *AB* and *CD* be two xy-monotone curve segments. Starting at the beginning of each segment, we shall crawl along the two segments in tandem, alternating the crawl along *AB* with the crawl along *CD* so that, at any given time, progress is made along only one of the segments (Fig. 2). The crawl along a segment continues until a *switching condition* becomes true. The alternation between segments continues until an *end condition* becomes true, signalling that an intersection has been found or that the two segments do not intersect. The segment along which one is presently crawling (or, not crawling) is called the *active* (or, *dormant*) segment. A crawl along an active segment between switching conditions is often referred to simply as a crawl (of the coordinated crawl).

Example 3.1 Consider the coordinated crawl of Fig. 2a. Horizontal and vertical lines have been added to the picture to reveal the structure of the crawls. The first crawl is along Aa_1 of AB. AB then becomes dormant and the second crawl is made along Cc_1 of CD, and so on. Eventually, the crawls get progressively smaller and converge to an intersection x. If no intersection exists, then the coordinated crawl reaches the end of one of the segments (Fig. 2b).
The coordinated crawl, as we have presented it, will only find the first intersection. The second intersection is found by starting another coordinated crawl from the first intersection. A new coordinated crawl should be begun from each intersection until it is determined by the end-condition that the segments do not intersect any further.
We shall present two variants of coordinated crawling, because there are two types of xy-monotone segments. A *rising* xy-monotone segment increases in y as it increases in x, while a *falling* xy-monotone segment decreases in y as it increases in x. The first method of coordinated crawling, which we call the *simultaneous crawl*, can be used to crawl along any pair of xy-monotone segments; however, it is best suited to crawling along one rising and one falling segment. The second method, which we call the *staircase crawl*, will only apply to two rising or two falling segments.
We need to define some notations and assumptions. Our notation for a curve segment will not only specify the endpoints, but also the order of the endpoints, in the sense that it is assumed that $x(A) \leq x(B)$ is always true of the segment AB, where $x(A)$ denotes the x-coordinate of the point A. P_{active} (or, P_{dormant}) is our notation for the present point on the active (or, dormant) segment during a coordinated crawl. Finally, in the remainder of this paper, we assume that all curves are nonlinear, irreducible, plane algebraic curves.

3.1 Approximate vs exact methods

It is often difficult to be exact in geometric computations. The reason for this is twofold: (1) because of the use of numerical methods that converge rather than compute exactly, and (2) because of finite machine precision. This is certainly the case with the elimination method of intersection, which uses numerical methods, such as Newton's method. Coordinated crawling is no different. Therefore, we make two natural assumptions, both of which can be removed if desired, as discussed below.
a) Two intersections that are less than ε distance apart are considered to be the same, where $\varepsilon > 0$ is very small and must be part of the input of an intersection problem.
b) If the distance between the two segments decreases below ε, then we are free to decide that there is an intersection near this point. In other words, if there is an intersection, then we will always recognize it; but if there is no intersection, then we may sometimes make a mistake and diagnose an intersection.
ε may be chosen as small as desired without affecting the efficiency of the coordinated crawl (see Lemma 6.1 below).
These two assumptions simplify the presentation of coordinated crawling considerably. Moreover, for the purposes of applications such as graphics, the two assumptions are valid not only because of inherent error in computations but also because of the inherent crudeness of algorithms, e.g., two segments may also intersect if they are closer to each other than a pixel. However, it is possible to do without them. Indeed, for a coordinated crawl along a rising and a falling segment, neither of the assumptions is necessary at all. (That is, intersections will always be found within the accuracy of the crawling method that is being used, and a near-intersection will never be mistaken for an intersection.)
The first assumption can always be removed by the choice of a proper ε. For example, one can use Canny's gap theorem (Lemma 3.1), which reveals that the intersections of two algebraic curves are never too close together.

Lemma 3.1 (Canny's gap theorem (Canny 1987)) *Let $\wp(d, c)$ be the class of polynomials of degree d and coefficient magnitude c. Let $f_1(x_1, \ldots, x_n)$, $\ldots, f_n(x_1, \ldots, x_n) \in \wp(d, c)$ be a collection of n polynomials in n variables, which has only finitely-many solutions when projectivized. Then if $(\alpha_1, \ldots, \alpha_n)$ is a solution of the system, for any j either $\alpha_j = 0$ or $|\alpha_j| > (3dc)^{-nd^n}$.*

Corollary 3.1 *Let $f_1(x, y) = 0, f_2(x, y) = 0$ be two irreducible plane algebraic curves of degree d and coef-*

ficient magnitude c. If $\varepsilon<(3dc)^{-2d^2}$, then no two intersections will be within an ε-distance of each other.

Proof. In order to apply the lemma, translate one of the two intersections to the origin.

The second assumption can be removed as follows. Whenever, the distance between the segments decreases below ε, construct a two-dimensional box at the point that contains at most one intersection (using Canny's gap theorem or other methods) and test whether this box contains an intersection, using a technique for testing whether a box contains an intersection (Sakkalis 1989) or a technique for counting the number of intersections in a box (e.g., Pedersen 1990). Next jump past the segments in the box and consider another box at this new point. Continue in this way until the distance between the segments increases above ε.
Note that, due to the expense of exact methods, in most cases it will be preferable to make the two simplifying assumptions.

3.2 Coarse vs fine crawls

In coordinated crawling, we shall distinguish between coarse and fine crawls, depending on the size of each crawl step. A fine crawl will be used to find something accurately and to avoid skipping over an intersection. Thus, fine crawl steps are less than ε in length. Coarse crawl steps are as long as possible without losing the curve. Obviously, for reasons of efficiency, it is important that coarse crawls be used as much as possible. The desired paradigm is to use coarse crawls to get close to the intersection and fine crawls only at the end to accurately find the intersection.
For any pair of xy-monotone segments, we shall present a coordinated crawling method such that most of the crawling is in coarse steps. Later sections (Sects. 8.2 and 8.3) will investigate the use of even coarser traversals of the curve, where one skips over a large subsegment of the curve in a single step (using a line-curve intersection).

4 Simultaneous crawl

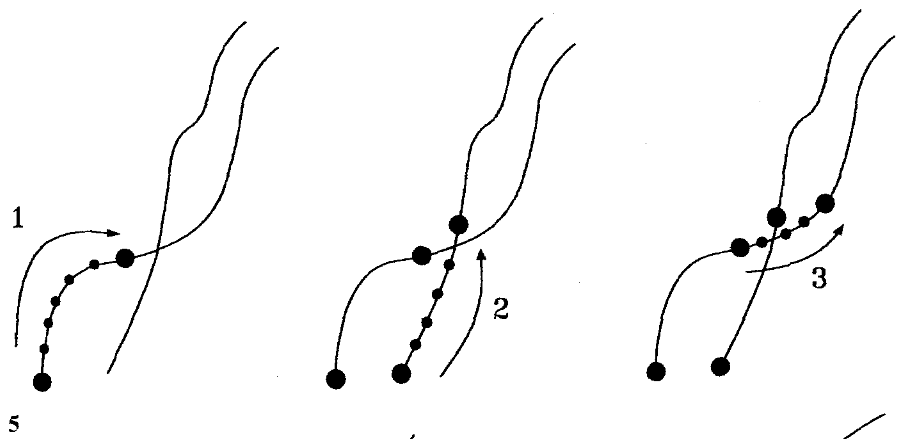
In order to fully define a coordinated crawl, the switching and end conditions must be defined. In this section, we present the first of our two coordinated crawling methods, the *simultaneous crawl.* With the simultaneous crawl, one simulates crawling along both segments at the same time while maintaining the same velocity with respect to the x-axis, hence its name (Fig. 3a). The associated switching condition is $x(P_{\text{active}})>x(P_{\text{dormant}})$.
The end-condition must signal an intersection or the end of a segment. In the neighborhood of an intersection between a rising segment and a falling segment, the relative vertical order of the segments is reversed. Therefore, a simultaneous crawl along a rising and falling segment can proceed with coarse steps until the relative vertical order of the segments is reversed, and then crawl backwards with finer steps to accurately find the intersection (Fig. 3b). The backwards crawl should continue until the relative vertical order switches once more, which is where the intersection is placed. There is no danger of skipping over two intersections with the coarse crawl, because a rising segment and a falling segment can have only one intersection.
Because of x-monotonicity, it is simple to recognize the end of a segment AB or CD: the condition is $x(P_{\text{active}})\geq\min\{x(B), x(D)\}$. Thus, for a simultaneous crawl along a rising segment AB and a falling segment CD, the entire end condition is

$$((y(A')<y(C'))\neq(y(A)<y(C)))\\ \vee x(P_{\text{active}})\geq\min\{x(B), x(D)\},$$

where A' (or, C') is the present point on AB (or, CD) during the crawl. Recall that the end condition signaling an intersection actually signals only the passing of an intersection, so that one must retrace steps back to the intersection before outputting it.
Theoretically, the simultaneous crawl can also be used for two rising (or two falling) segments. However, a simultaneous crawl along two rising segments is dangerous, because it is possible to skip over a pair of intersections without noticing them (Fig. 4). Using Lemma 3.1, this danger could be avoided if the crawl is fine enough. That is, the crawl steps must be finer than 2ε where $\varepsilon<(3dc)^{-2d^2}$. Because a simultaneous crawl along two rising or two falling segments would require fine steps at all times, the staircase crawl of the next section is preferred for these segments.

5 Staircase crawl

In this section, we present the second method of coordinated crawling, which is used for crawling



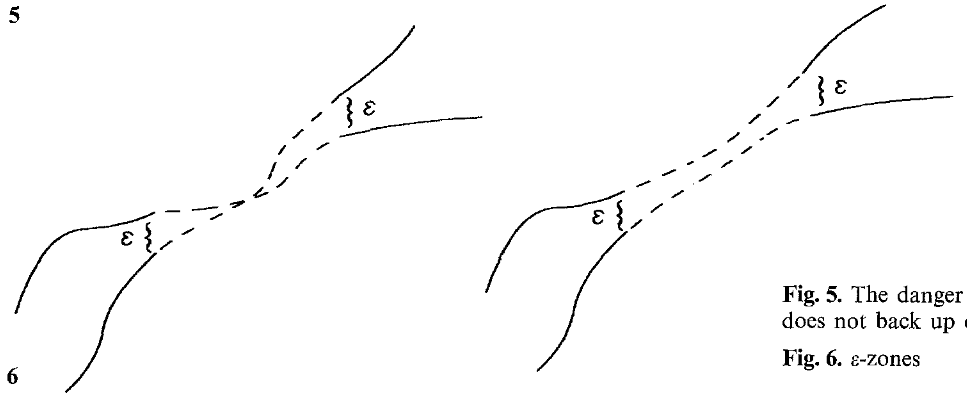


Fig. 5. The danger of a staircase crawl that does not back up one step after each crawl

Fig. 6. ε-zones

along two rising or two falling segments. Consider a coordinated crawl along two rising segments. Rather than switching segments as soon as the x-coordinate of the active segment exceeds the x-coordinate of the dormant segment, in the staircase crawl one waits until both the x-coordinate *and* the y-coordinate of the active segment exceed those of the dormant segment before switching. We call this a *staircase crawl*, because if the endpoints of the crawls are joined by straight lines, a staircase leading towards an intersection will result (Fig. 2 and Lemma 5.1). The associated switching condition is

$$x(P_{\text{active}}) > x(P_{\text{dormant}}) \wedge y(P_{\text{active}}) > y(P_{\text{dormant}}).$$

Because of the granularity of the crawl, an intersection might be overlooked with this switch condition (Fig. 5). To correct this, one should back up one step before switching segments so that the active segment remains behind the staircase.

An intersection is signalled when a stair less than ε in height is encountered. (This is the only place that the second assumption from Sect. 3.1 is used.) Every intersection will be found with this end-condition, because the staircase converges to an intersection (see Lemma 5.1), and we do not jump past the staircase. Thus, the end-condition for a staircase crawl is simply

$$|y(P_{\text{active}}) - y(P_{\text{dormant}})| < \varepsilon$$
$$\vee\, x(P_{\text{active}}) > \min\{x(B), x(D)\}.$$

Coarse crawls are used until the coordinated crawl approaches an intersection, at which point fine crawls are used. If a coarse crawl is used near an intersection, it is possible to enter an infinite loop: continually going forward one step (at which point both x and y coordinates of the active segment exceed those of the dormant) and then back one step (to stay behind the staircase). This will not happen with a fine crawl, because if only one fine step (of length less than ε) separates the x and y-coordinates, the height of the present stair must be less than ε and an intersection will be signalled. Because an infinite loop, as described above, is only possible when the length of a stair becomes as short as a step of a coarse crawl, fine crawls are only necessary near the intersection. In an intermediate phase, when strictly coarse crawls are too crude, but strictly fine crawls are too slow, one can use a coarse crawl to crawl forward and a fine crawl to go backwards.

We must show that the staircase crawl converges to the first intersection of the segments, if one exists.

Lemma 5.1 *Let AB and CD be two rising segments. The staircase crawl along AB and CD will converge to the first intersection of AB and CD, if such an intersection exists. Otherwise, it will reach the endpoint B or D of one of the segments.*

Proof. First, one does not jump over an intersection. This is best seen by considering the perfect staircase: the staircase consisting of true horizontal and vertical line segments. (The stairs of the staircase connecting the endpoints of the actual staircase crawl will not be perfectly horizontal or vertical.) It is easy to see that the perfect staircase converges to an intersection. The staircase crawl is guided by the perfect staircase. Because the staircase crawl backs up before switching segments, it is indeed constrained by the perfect staircase.

Second, progress is made with each crawl. If there is no progress, then the stair must be of a height less than ε and we say that an intersection has been found. In particular, progress of at least ε (usually much more) is made with each crawl. Thus, the staircase crawl must eventually find the first intersection, if one exists.

A staircase crawl diagnoses an intersection when the stairs become shorter than ε. Two questions arise: where should the intersection be placed and where should the crawl start from to look for the next intersection? The crawl cannot place the intersection where it stopped and continue from there, because it will immediately stop and diagnose another intersection. We introduce the concept of an ε-zone to provide the answer to these questions. The *ε-zone* is a pair of subsegments of the curves that stay within a vertical distance of ε. We enter an ε-zone when the vertical distance between the segments becomes less than ε (in practice, the ε-zone actually begins when a stair of a height $<\varepsilon$ is found) and exit it when the vertical distance becomes greater than ε (Fig. 6).

Now the two questions can be answered. When a stair of a staircase crawl becomes shorter than ε, we skip over the associated ε-zone and restart the staircase crawl there. The intersection is placed in the middle of the ε-zone. A simultaneous crawl is used to cross the ε-zone. (As mentioned at the end of Sect. 4, a simultaneous crawl of two rising segments will use fine crawls, so it is perfect for crossing the ε-zone.)

The first crawl after an intersection must be treated as a special case, because neither segment dominates. In order to get things started, one should make a crawl of length ε along one of the segments. It might seem that this crawl is dangerous, because it may skip over an intersection, being blind and unconstrained by any staircase. However, recall that any two intersections that are within ε of each other are considered equivalent; thus, it is impossible to skip over a relevant intersection.

We end this section by noting that the staircase crawl cannot be used to find the intersection of a rising segment and a falling segment, as illustrated by Fig. 7. In as much as we have already noted that the simultaneous crawl is not well suited for the intersection of two rising segments (because fine crawls are always necessary), it can be seen that both types of coordinated crawling are necessary. With a simultaneous crawl and a staircase crawl, one can find the intersection(s) of any pair of xy-monotone segments, and most of the crawling uses coarse steps.

6 Efficiency

A simultaneous crawl continually switches from one segment to the other. Indeed, of the two segments between any two adjacent vertical lines in Fig. 3(a), one will be a single crawl step long. It might appear that this large number of switches will make the crawl expensive. The following lemma shows that this is not the case, because switches are essentially free.

Lemma 6.1 *The number of switches in a coordinated crawl is irrelevant.*

Proof. The number of switches does not matter because stopping and starting a crawl takes no time. This can be seen as follows. Let *AB* and *CD* be xy-monotone segments. We can keep two separate regions in memory, one set up for crawling along *AB*, the other for crawling along *CD*. Switching crawls merely involves jumping to the other part of memory.

It might seem that if the segments remain very close for a long time, then a staircase crawl will be slow because the staircase is very fine with very short stairs. Similarly, it might appear that a staircase crawl will slow down as its stairs get very short during convergence to an intersection. Lemma 6.1 shows that these intuitions are wrong.

In the worst case, a coordinated crawl along two segments seems slightly less efficient than crawling independently along the entire first segment and then crawling independently along the entire second segment, because one must test the switching and end conditions at each step. However, in Sect. 8 we shall show that a coordinated crawl need not crawl along all of the two segments, so that the complexity of a coordinated crawl is actually less than the complexity of making two independent crawls with condition testing.

7 XY-Monotone decomposition

As our coordinated crawling methods work upon xy-monotone segments, the first step in intersecting two segments with the coordinated crawling method is to partition each curve segment into xy-monotone curve segments. Observe that a curve segment is xy-monotone if and only if it contains no local extrema (no changes in direction with respect to the x-axis or y-axis). An xy-monotone decomposition of a segment can be computed by crawling along the segment. One simply marks points at which $x(\mathrm{P})$ or $y(\mathrm{P})$ changes direction, where $\mathrm{P}=(x(\mathrm{P}), y(\mathrm{P}))$ is the present point on the crawl. The properties of crawling guarantee that one will not miss any directional changes during a crawl. Therefore, the complexity of the xy-monotone decomposition of a segment is the complexity of crawling along the segment.

An alternative method is to compute the local extrema of the segment algebraically, using the fact that the local extrema of a curve $f(x, y)=0$ are the solutions of $\{f_x=0, f=0\}$ and $\{f_y=0, f=0\}$, where f_x and f_y are the derivatives of f with respect to x and y, respectively. After computing the extrema, they must be sorted along the curve in order to pair them into xy-monotone segments. Note that the local extrema may already be known, because it is trivial to compute the local extrema of a curve as part of computing the singularities of a curve (the singularities are the solutions of $\{f_x=0, f_y=0, f=0\}$); and the singularities of an algebraic curve are fundamental to many geometric modeling algorithms (e.g., Abhyankar and Bajaj 1986; Johnstone 1987). If local extrema are not known, the elimination method of Sect. 2 can be used to compute them. This may appear to lead to a circularity in our method. However, xy-monotone decomposition is a one-time overhead. The expense of preprocessing is well warranted for curves in a solid model, because they are relatively permanent and intersection is a common operation. For example, consider the pairwise intersection of a large collection of segments. The cost of xy-monotone decomposition can be amortized over all of the intersections, whereas the elimination method would require a curve-curve intersection for each of the pairwise intersections.

Although the elimination method of xy-monotone decomposition is mentioned in the interests of completeness, the first method using crawling will usually be the best choice, because a local method for xy-monotone decomposition is appropriate for a local method of curve intersection.

8 Improvements

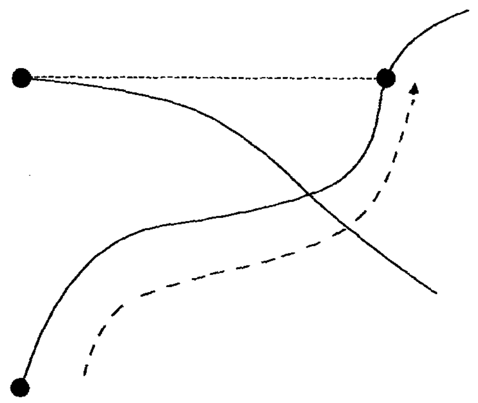
8.1 Early abortion

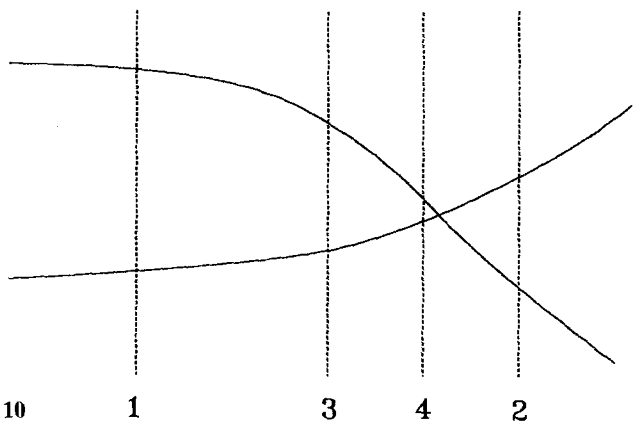
In this section, we outline some methods for improving the efficiency of a coordinated crawl. A coordinated crawl can be aborted as soon as it becomes apparent that the two segments cannot intersect. We begin with a set of conditions that guarantees the distinctness of two xy-monotone segments.

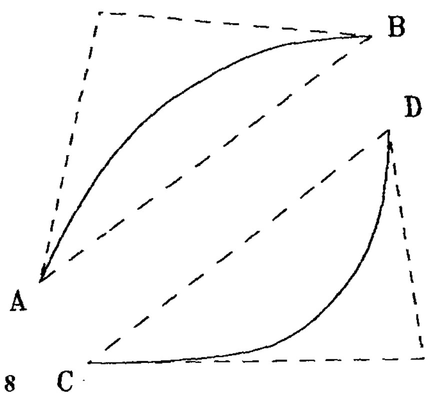
Lemma 8.1 *Let AB and CD be xy-monotone segments. If any of the following conditions is true, then AB and CD do not intersect.*

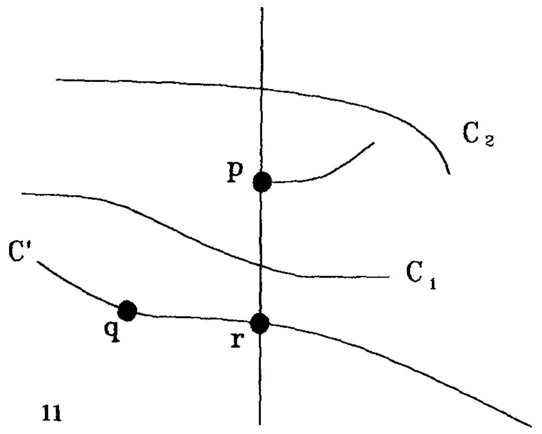
a) $x(B)<x(C)$.
b) $x(D)<x(A)$.
c) $\{y(A), y(B)\}<\{y(C), y(D)\}$.
d) $\{y(C), y(D)\}<\{y(A), y(B)\}$.
e) *AB and CD are convex (a segment is convex if no line has more than two distinct intersections with it) and $\Delta AB \cap \Delta CD=\emptyset$, where ΔAB is the triangle whose sides are the tangent at A, the tangent at B, and $\overline{AB}$ (Fig. 8).*

Proof. The sufficiency of conditions a–d is a straightforward consequence of the xy-monotonicity of the segments. (Recall that $x(A) \leq x(B)$ is implicit from the notation AB.) The sufficiency of the fifth condition follows from noting that if AB is convex, then ΔAB contains AB.


7


10


8


11

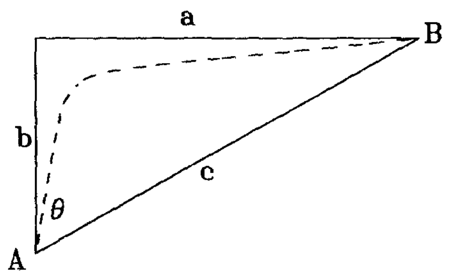

'9

Fig. 7. A staircase crawl cannot be used for a rising segment and a falling segment

Fig. 8. *AB* and *CD* cannot intersect

Fig. 9. The length of *AB* is bounded by $a+b$

Fig. 10. A binary search for the intersection

Fig. 11. A curve comparison between *C'* and p: crawl from q to r

These conditions should be tested throughout the coordinated crawl. (The condition involving convex segments should only be tested if there is prior knowledge that the segments are convex as well as xy-monotone.) In order to lighten the computational burden, they might only be tested intermittently, rather than after every step.

8.2 Eliminating long crawls in the staircase crawl

We have noted that a coordinated crawl should use coarse crawls whenever possible. In this section, we show how to make even larger jumps in a staircase crawl. In particular, it is possible to replace a crawl (from one stair endpoint to the next) by a single line-curve intersection.

In a staircase crawl, one climbs a staircase towards an intersection. This process can be fully characterized by the series of endpoints of the stairs. For example, the staircase crawl of Fig. 2 can be represented by $C, a_1, c_1, a_2, c_2, \ldots$ The act of climbing a stair (i.e., finding the next endpoint in the series) is equivalent to finding the intersection of a line with one of the curve segments. In particular, the endpoint that follows endpoint E on curve segment

1 is the intersection with curve segment 2 of a horizontal or vertical line through E. This suggests another method for climbing the stair: find the intersection of the line and curve segment with the elimination method of Sect. 2.

Because a staircase crawl is being used, it can be assumed that the elimination method is inferior for the intersection of the two curve segments. However, it may still be feasible for the simpler intersection of a line and one of the curve segments. The time to climb a stair E_iE_{i+1} of the staircase $E_1, E_2, \ldots, E_n$ by crawling depends upon the length of the segment $E_{i-1}E_{i+1}$, whereas the time to climb a stair with a line-curve intersection depends on the degree of the curve to which we are climbing. Therefore, the higher the stair and the lower the degree of the curve, the more attractive it is to climb the stair with a line-curve intersection.

It may be difficult to decide when the next stair should be climbed with a line-curve intersection rather than a crawl. The following lemma could be used to approximate the cost of climbing it with a crawl.

Lemma 8.2 *The length of an xy-monotone segment AB is bounded by* $\sqrt{2}\,\mathrm{dist}(A, B)$.

Proof. Consider the right triangle with hypotenuse AB, whose other sides are horizontal and vertical (Fig. 9). By xy-monotonicity, it is easy to see that the length of the curve segment AB is bounded by the sum of the lengths of two of the sides of the triangle $|x(B)-x(A)|+|y(B)-y(A)|=a+b$. The result follows by noticing that $a+b=c(\sin\theta+\cos\theta)$ and $\max_\theta(\sin\theta+\cos\theta)=\sqrt{2}\left(\text{at } \theta=\frac{\pi}{4}\right)$.

8.3 Accelerating the simultaneous crawl

The simultaneous crawl can also benefit from the use of line-curve intersections. In this case, the analogy is to root-finding of univariate polynomials, where binary search is used to isolate a region for the root before Newton's method is applied. In finding the unique intersection of a rising segment and a falling segment, it may be useful to use a binary search for the intersection with line-curve intersections before beginning the actual simultaneous crawl (Fig. 10). (A binary search is not possible for the intersection of two rising segments, because there may be more than one intersection, and a binary search only makes sense when searching for a single element.) A probe of this 'binary search' is the intersection of a vertical line with both curve segments to determine their relative vertical order. If the relative vertical order is the same as the beginning of the segments, then the intersection must lie to the right of the probe.

The binary search may allow the simultaneous crawl to begin closer to the intersection. It is difficult to determine the number of probes that should be made. There is a tradeoff between the number of line-curve intersections that are computed and the amount of crawling that is saved. As a general rule, the binary search should continue longer if the degree of the curves is low (because line-curve intersections will be inexpensive) or if the curve segments are long (because a lot of crawling is more likely).

9 Intersecting arbitrary curve segments

We have discussed how to intersect two xy-monotone segments. However, the original goal was to intersect two arbitrary segments. In as much as the two original segments were decomposed into xy-monotone segments, we must show how to find the intersections of a collection of xy-monotone segments. Rather than using the naive $O(n^2)$ algorithm of intersecting every pair, we shall use a variant of plane sweep to reduce this to looking at only $O(n+k)$ pairs, where k is the number of intersections. The advantage of this plane-sweep method (based on the familiar plane sweep of Bentley and Ottmann 1979) is it avoids testing pairs that are never vertically adjacent.

We begin by inserting all of the xy-monotone segment endpoints into a priority queue E (sorted by x-coordinate). We will be sweeping a vertical line L through the plane from left to right. As we sweep, we will maintain a database D, which consists of all curve segments that intersect L, stored in sorted order by their intersections with L. We represent D as a (2,3)-tree (Aho et al. 1974) (or some equivalent efficient dynamic-search structure). Note that because the segments are xy-monotone, each segment will intersect L at most once. As we sweep L to the right we need to stop at various *event* points to maintain the consistency of the database

D. The priority queue E determines the events. An event is either an endpoint or an intersection point. With each curve C, we also keep a priority queue E(C), which stores the names of all the curves that we have compared with C already. These lists will prevent us from performing any redundant intersection tests.
A generic step in the plane-sweep algorithm is as follows. Remove the point in E with minimum x-coordinate. Let p be this point. Intuitively, this corresponds to moving L to the right until it hits p. We must then update D, depending on the identity of p. We identify each of the possible cases below.

Case 1. The point p is the left endpoint of a curve segment C (Fig. 11). In order to maintain the consistency of our database, we must insert C in to D. To do this, we must find the curve segment C_1 in D such that C_1 intersects L in the highest point below p, i.e., C_1 is directly below p. We can do this by making $O(\log n)$ curve comparisons to find a path in the tree D from the root to the place where C belongs. Each such curve comparison determines whether a curve C' intersects L above or below p and is implemented by crawling along C' from the previous event point on C' until reaching L, as in Fig. 11 (or by performing a line-curve intersection). After locating where C belongs in D, suppose that C_1 (or, C_2) is C's predecessor (or, successor) curve in D. We check if C_1 is already in $E(C)$ and, if not, intersect C with C_1 (using coordinated crawling). Similarly, we check if C_2 is already in $E(C)$ and, if not, intersect C with C_2. We add all discovered intersection points p to the priority queue E as long as the two curves cross at p (as opposed to simply 'touching'). We also add C to $E(C_1)$ and $E(C_2)$ and add C_1 and C_2 to $E(C)$. At first glance, one might worry that the crawling involved in the curve comparisons might become prohibitive. However, even in the pathological worst case, the entire time required for inserting C into D is bounded by the time to crawl along $\log n$ segments, which is not a significant expense when compared with the alternative of finding the intersections between all $O(n^2)$ segment-pairs.
The coordinated crawl for intersecting two xy-monotone segments should be started from L, not from the beginning of the segments.

Case 2. The point p is an intersection point. If C_1 and C_2 are the two curves that intersect at p, then we swap them in D. Without loss of generality, assume C_2 now occurs before C_1 in the list D. Let C_0 be the new predecessor of C_2 and let C_3 be the new successor of C_1. Provided C_0 is not in $E(C_2)$, we find the intersections of C_0 and C_2 (and insert them into the event queue E). Similarly, we intersect C_1 and C_3, provided C_3 is not in $E(C_1)$. We then update $E(C_0)$, $E(C_1)$, $E(C_2)$, and $E(C_3)$ as necessary.

Case 3. The point p is a right endpoint of a curve C. In this case, we delete C from D. We then need to intersect the two neighbors C_1 and C_2 of C at p (which are now adjacent), after checking if C_1 is already in $E(C_2)$. Of course, we then update $E(C_1)$ and $E(C_2)$ as necessary.
Because these are all the possible cases, this completes the algorithm. We summarize with the following theorem:

Theorem 9.1 *Given n xy-monotone curve segments in the plane, one can compute all of their intersection points with $O(n+k)$ segment-segment intersections, where k is the number of intersection points.*

Proof. The only time that segment-segment intersections are made is when inserting or deleting an event from the event queue E. Each (of $2n$) segment endpoints and each (of k intersections) are inserted and deleted from E, and at most two segment-segment intersections are performed with each insertion or deletion.
The benefits of this algorithm will be most strongly felt when the segments S are of a different order of complexity from the curves $C \supset S$. Note that the same plane-sweep algorithm can be applied to the intersection of any number of algebraic curve segments, as well as two algebraic curve segments.

10 Conclusions

By extending the technique of crawling along one segment to a technique of coordinated crawling along two segments, we have introduced a new method for intersecting plane algebraic curve segments. It takes advantage of the locality and simplicity of the segments, unlike the elimination method of intersection. Rather than first finding all of the intersections between the curves, our method directly finds only the intersections between the segments.

The coordinated crawling method can be especially useful when degree explosion is encountered in the elimination method. That is, among other things, the conventional elimination method entails the solution of a univariate equation (the resultant of the two curves) whose degree is the product of the degrees of the curves, and the solution of an equation of high degree soon becomes prohibitive. The coordinated crawling method only involves the evaluation of equations whose degree is the degree of the curves (for crawling), which is simpler for two reasons: because of the lower degree of the equation and because evaluation is easier than solution.

Acknowledgements. We would like to thank C. Bajaj and M.S. Kim for helpful conversations.

References

Aho A, Hopcroft J, Ullman J (1974) The design and analysis of computer algorithms. Addison-Wesley, Reading, Mass

Abhyankar S, Bajaj C (1988) Automatic parameterization of rational curves and surfaces III: algebraic braic plane curves. Comput-Aided Geom Des 5:309–321

Abhyankar S, Bajaj C (1989) Automatic parameterization of rational curves and surfaces IV: algebraic space curves. ACM Trans Graphics 8:325–334

Bajaj C, Hoffmann C, Hopcroft J, Lynch R (1988) Tracing surface intersections. Comput-Aided Geom Des 5:285–307

Bentley JL, Ottmann TA (1979) Algorithms for reporting and counting geometric intersections. IEEE Trans on Computers C-28:643–647

Canny JF (1987) The complexity of robot motion planning. PhD thesis, Massachusetts Institute of Technology

Dobkin DP, Thurston WP, Wilks AR (1986) Robust contour tracing. Tech Rep CS-TR-054-86, Princeton University

Edelsbrunner H (1987) Algorithms in combinatorial geometry. Springer, New York

Garrity T, Warren J (1989) On computing the intersection of a pair of algebraic surfaces. Comput-Aided Geom Des 6:137–153

Hoffmann CM (1987) Algebraic curves. Tech Rep CSD-TR-675, Purdue University

Johnstone JK (1987) The sorting of points along an algebraic curve. Tech Rep 87–841; PhD Thesis, Cornell University

Johnstone JK, Bajaj C (1990) Sorting points along an algebraic curve. SIAM J Computing 19:925–967

McKenna M (1987) Worst-case optimal hidden-surface removal. ACM Trans Graphics 6:19–28

Mortenson ME (1985) Geometric modeling. Wiley, New York

Owen JC, Rockwood AP (1987) Intersection of general implicit surfaces. In: Farin G (ed) Geometric modeling: algorithms and new trends. SIAM, Philadelphia, pp 335–345

Pedersen P (1990) Generalizing Sturm's theorem to *n* dimensions. Talk at the DIMACS Workshop on Algebraic Issues in Geometric Computation, Rutgers University, May 21–25

Preparata FP, Shamos MI (1985) Computational geometry: an introduction. Springer, New York

Requicha AAG (1980) Representations for rigid solids: theory, methods, and systems. Computing Surveys 12:437–464

Sakkalis T (1989) Signs of algebraic numbers. In: Kaltofen E, Watt S (eds) Computers and mathematics. Springer, New York, 130–134

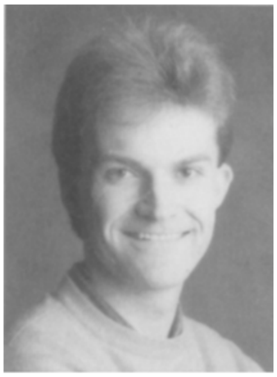
Sechrest S, Greenberg DP (1982) A visibility polygon reconstruction algorithm. ACM Trans Graphics 1:25–42

Sederberg TW, Anderson DC, Goldman RN (1984) Implicit representation of parametric curves and surfaces. Computer Vision, Graphics, and Image Processing 28:72–84

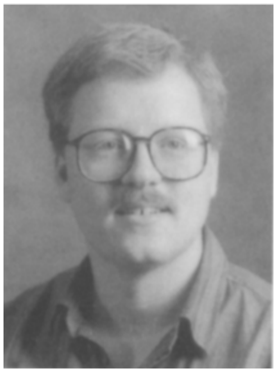
Timmer HG (1977) Analytical background for computation of surface intersections. Douglas Aircraft Company Technical Memorandum C1-250-CAT-77-036, cited in Mortenson 1985

Van der Waerden BL (1953) Modern Algebra. Frederick Ungar, New York

Walker RJ (1950) Algebraic curves. Springer, New York



JOHN K. JOHNSTONE is assistant professor of computer science at The Johns Hopkins University. He earned a PhD in computer science from Cornell University in 1987, where he was supported by a Canadian NSERC Scholarship and a Cornell Sage Fellowship. Professor Johnstone conducts research in geometric modeling and robotics, supported by the National Science Foundation. Methods that can be applied to models involving curves and surfaces of high degree are of especial interest.



MICHAEL T. GOODRICH is assistant professor of computer science at The Johns Hopkins University. He earned a PhD in computer science from Purdue University in 1987, where he was a Compere Loveless Fellow. With National Science Foundation support, Professor Goodrich conducts research in computational geometry and parallel algorithms.