# A Derivative-Free Tracking Algorithm for Implicit Curves with Singularities

José F.M. Morgado and Abel J.P. Gomes

Dept. Computer Science and Engineering, Univ. Beira Interior
6200-001 Covilhã, Portugal
{fmorgado,agomes}@di.ubi.pt

**Abstract.** This paper introduces a new algorithm for rendering implicit curves. It is curvature-adaptive. But, unlike most curve algorithms, no differentiation techniques are used to compute self-intersections and other singularities (e.g. corners and cusps). Also, of theoretical interest, it uses a new numerical method for sampling curves pointwise.

## 1 Introduction

An implicit curve $\Gamma = \{\mathbf{p} \in \mathbb{R}^2 : f(\mathbf{p}) = 0\}$ is a level set (or zero set) of some function $f$ from $\mathbb{R}^2$ to $\mathbb{R}$. The algorithm proposed in this paper focuses on the representation of planar implicit curves defined by real, but not necessarily analytic, functions (e.g. polynomial and transcendental functions). There are three major categories of algorithms to render implicit curves, namely:

- *Representation conversion.* Rarely, a global parameterization exists for an implicit curve. But, a local parameterization always exists in a neighborhood of a regular point of an implicit curve, i.e. a point $\mathbf{p} = (u, v)$ such that $f(\mathbf{p}) = 0$ and $\nabla f \neq 0$. This allows us to render an implicit curve by using the algorithms for parametric curves [10,2,4,6,1].
- *Space subdivision.* Recursively, it splits the ambient space into subspaces, discarding those not intersecting the curve. The subdivision terminates soon after we obtain a good approximation to the curve by a set of small subspaces (e.g. rectangles) [4,15,9]. Robust approximations can be implemented by using interval arithmetic [14], algebraic or rational techniques [8,7], and floating-point arithmetic [13].
- *Curve tracking.* It consists of sampling the curve pointwise [5,11]. This approach has its roots in the Bresenham's algorithm for rendering circles, which is essentially a continuation method in screen image space. Continuation methods are attractive because they concentrate computational processing where it is needed. However, they need a starting point on each curve component. Finding a starting point on a component can be a frustrating experience, in particular for degenerated components consisting of a single point. A way to compute these curve components is by means of the cylindrical algebraic decomposition technique from computer algebra [3].

This paper deals with the rendering of implicit curves possibly with singularities, but no derivatives are used at all.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 221–228, 2004.

## 2 Curve Sampling through Numerical Approximation

The basic idea behind the curve-tracking algorithm proposed in this paper is, given the previous and current points $P$, $Q$ of the curve $\Gamma$, to determine the next point belonging to the intersection $N_Q \cap \Gamma$, where $N_Q$ is the frontier of a small circular neighborhood centered at $Q$ (Fig. 1(a)). The algorithm does not evaluate the intersection points analytically. Instead, any intersection point of $N_Q \cap \Gamma$ is computed by a new approximation method inspired in the false position numerical method, called *angular false position method*.

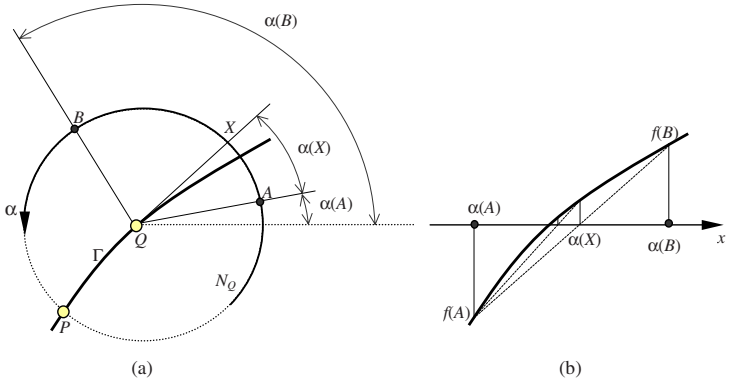


**Fig. 1.** The angular false position method

### 2.1 False Position Method: A Brief Review

The false position method is a root-finding algorithm which starts from two distinct estimates $A$ and $B$ for the root of $f(x) = 0$, being $f$ a function from $\mathbb{R}$ to $\mathbb{R}$, so that $f(A) \, . \, f(B) < 0$, i.e. $f(A)$ and $f(B)$ have opposite signs or, equivalently, a root is initially bracketed in the interval $]A, B[$. The next estimate $X$ is iteratively determined by linear interpolation given by the following formula:

$$X = B - \frac{f(B)}{f(B) - f(A)}(B - A) \tag{1}$$

This numerical method retains the prior estimate, either $A$ or $B$, that together with $X$ continue to bracket the root [12]. The formula (1) shows that the false position is an adequate sampling numerical method for generic curves, not necessarily differentiable or analytic, provided that no derivatives are used at all.

### 2.2 Angular False Position Method

As described above, the false position method calculates the roots of some function from $\mathbb{R}$ to $\mathbb{R}$ in the product space $\mathbb{R}^2$. But, for the curve $\Gamma$ defined implicitly

by $f : \Omega \subseteq \mathbb{R}^2 \to \mathbb{R}$, we are not interested in the roots of $f$ in the product space $\mathbb{R}^3$, but in the zero set of $f$ in $\mathbb{R}^2$, i.e. the curve itself in the domain space $\mathbb{R}^2$. For that, the curve is sampled by intersecting the zero set and a small circle $N_Q$ centered at the current point $Q$ (Fig. 1(a)). But, the intersection $\Gamma \cap N_Q$ occurs in the domain space of $f$, not in the product space. To overcome this difficulty we have first to transform the co-ordinates of both point estimates $A$ and $B$ on the circle onto their corresponding angle estimates $\alpha(A)$ and $\alpha(B)$ defined by a function $\alpha : \mathbb{R}^2 \to \mathbb{R}$. Then, as illustrated in (Fig. 1(b)), the false position method is used to determine an intermediate angle given by

$$\alpha(X) = \alpha(B) - \frac{f(B)}{f(B) - f(A)}[\alpha(B) - \alpha(A)] \tag{2}$$

Thus, the angular approximation method occurs in the product space of $f \circ \alpha^{-1}$, i.e. $\mathbb{R}^2$, according to the following diagram:

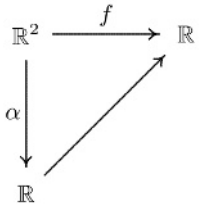


**Fig. 2.** Diagram of the angular false position method

## 3 Curve Tracking Algorithm

This algorithm confines all computations to the neighborhood $N_Q$ to determine the intersecting points $\Gamma \cap N_Q$. The main difficulty is to correctly choose the next point $X$ amongst those in $\Gamma \cap N_Q$. The main criterion for choosing the next point is based on the variance of curve curvature within $N_Q$, what is given by $\angle PQX$. Remarkably, this criterion works for any local shape, no matter whether or not the curve self-intersects, it has a cusp or a corner, it almost touches itself, or it oscillates.

### 3.1 Computing Neighborhood Points

The curve points $N_Q \cap \Gamma$ are numerically determined by the angular method introduced above. Instead of using the $x$-axis for computing the points approaching an intersection point, we use a small circular neighborhood $N_Q$ (Fig. 3).

We could think of a uniform distribution of points on the neighborhood $N_Q$ separated by an angle $\theta$, and then apply the angular method to every pair of

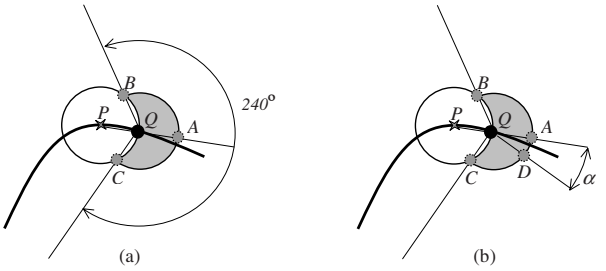


**Fig. 3.** Distribution of points on the neighborhood of the current point $Q$

points in order to determine possible intersection points. Such a distribution of points is given by $(x, y) = (x_c + r\cos\theta, y_c + r\sin\theta)$, with $\theta \in [-180°, 180°[$, being $(x_c, y_c)$ the center of $N_Q$, i.e. the current point $Q$. To speed up our algorithm, the circle points are computed for $\theta \in [-120°, 120°]$ because the others were calculated before in the neighborhood of the previous point $P$ (Fig. 3).

We start by determining three points, $A = Q + \overrightarrow{PQ}$, $B = Q + M_{2\pi/3}.\overrightarrow{QA}$, and $C = Q + M_{-2\pi/3}.\overrightarrow{QA}$ on $N_Q$, where $M$ is the rotation matrix.

Then, one determines the point $D = Q + M_\alpha.\overrightarrow{QA}$, with $\alpha \in [5°, 10°]$, such that $f(D).f(A) < 0$, by applying the angular approximation method to the arc $\overset{\frown}{AD}$. Note that for curves with small curvature, a small $\alpha$ leads to very fast search for intersection or solution points. In particular, for a straight-line curve, the point $A$ is the solution point itself. Nevertheless, we have to look for solution points along the arcs $\overset{\frown}{AB}$ and $\overset{\frown}{CD}$ in case we have more curve points on $N_Q$.

### 3.2 Cusp and Other High-Curvature Points

Most curve-tracking algorithms break down at singularities (see, for example, [5]). Nevertheless, the algorithm proposed by [11] works for curves with bifurcation points by analysis of the sign changes of the partial derivatives in a rectangle neighborhood. However, it breaks down at other singularities such as, for example, cusps, which belong to the function domain, but not to the domain of the partial derivatives. For example, it is not capable of rendering the curve $|x| + |y| - 2 = 0$ (Fig. 6(b)) with four singularities at $(0, 2)$, $(2, 0)$, $(0, -2)$ and $(-2, 0)$, where the partial derivatives do not exist.

In contrast, our algorithm needs not compute derivatives at all. This allows us to draw curves defined by both differentiable and non-differentiable functions. This reduces somehow the computation time of each point in the curve. In fact, the computation of the partial derivatives of an analytic function such as, for example, $y(9 - xy)(x + 2y - y^2)((x - 10)^2 + (y - 4)^2 - 1) = 0$ (Fig. 6(f)) may be more time-consuming than the function itself.

Cusps and corners are points at which the curvature flips. A cusp point (Fig. 4(a)), or a quasi-cusp point (Fig. 4(b)), is characterized by having a high curvature variance along the curve within $N_Q$. To be sure that there is a cusp

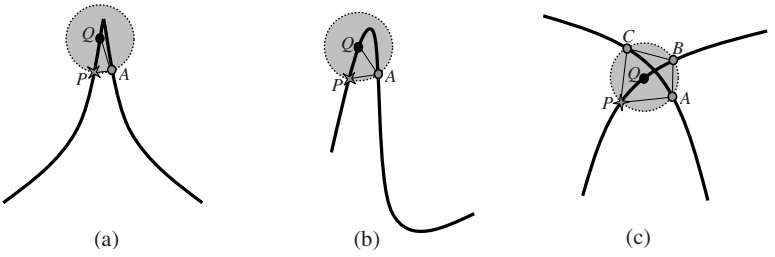


**Fig. 4.** (a) A cusp; (b) a quasi-cusp; (c) a self-intersection point

(or a quasi-cusp) point in $N_Q$, we have to check that the mediatrix of $\overline{PA}$ in $N_Q$ intersects the curve at exactly a single point. But, first, we have to check that the angle $\angle PQA$ is relatively small. A small angle $\angle PQA$ means that $A$ is not an appropriate point next to $Q$.

The strategy is then to assume that the points after the cusp (or quasi-cusp) are image points of those before it in $N_Q$. For example, $A$ is the image of $P$. The image of $Q$ is $B$ by tracing a line segment parallel to $\overline{PA}$. The next point $R$ is determined by intersecting the curve with the mediatrix of $\overline{QB}$ in $N_Q$. $R$ is possibly a cusp, but if it has an image point $C$, we determine again the next point by computing the intersection between the curve and the mediatrix of $\overline{RC}$, stopping when the distance between the latest next point and its image is under $10^{-6}$. The latest next point is nearly the cusp point (or the quasi-cusp point).

### 3.3 Self-Intersection Points

A self-intersection point can be seen as a generalized cusp point (Fig. 4(c)). In fact, every two adjacent curve branches incident at the crossing point can be viewed as meeting at a cusp point. As for a cusp (or a quasi-cusp), no curve point on $N_Q$ is the next point because, with the exception of $B$, the points $A$ and $C$ form small angles with $P$, $Q$. But, $B$ cannot be the next point either because the segment $\overline{PB}$ intersects the curve at a point. This intersecting point between $P$ and $B$ is a better candidate to next point than $B$ because it is before $B$. It can be determined by applying the false position method between $P$ and $B$.

Similar to a cusp point, we have to compute the image point $D$ of $Q$ by intersecting the curve with a line segment parallel to $\overline{PA}$ passing through $Q$ in $N_Q$. Then, we determine the remaining image points $E$ of $D$ and $F$ of $E$ by using the segments parallel to $\overline{AB}$ and $\overline{BC}$, respectively. At this point, we can generalize the convergence process to a cusp, so that the next point $R$ is determined by intersecting the curve with the mediatrix of $\overline{QD}$ in $N_Q$. This process converging to the self-intersection point stops when the distance between the latest next point and its image is under $10^{-6}$. This latest point is nearly the self-intersection point.

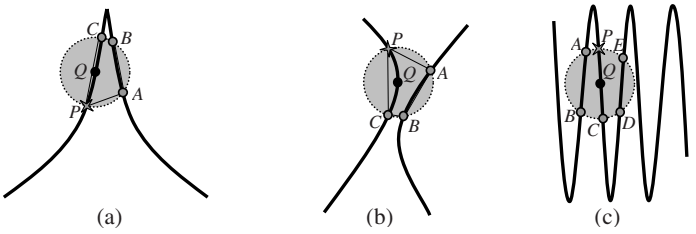


**Fig. 5.** Near-branch points

### 3.4 Near-Branch Points

Sometimes a curve almost touches itself, i.e. some of its points are very close (Fig. 5). Unlike the previous cases, one of the neighborhood-intersecting points is the point next to $Q$. Note that determining the next point is done without changing the neighborhood radius, even under ripples and undulations.

Let us look again at Fig. 5, where $P$ and $Q$ are the previous and current points, respectively. To see that the neighborhood-intersecting point $C$ is the next point after $Q$, we use two criteria: *angle* (or curvature) *criterion* as above, and the *neighbor-branch criterion*. In Fig. 5(a) and (b), $A$ cannot be the next point because the angle $\angle PQA$ is far from 180°. But, both angles $\angle PQB$ and $\angle PQC$ are close to 180°, and neither $\overline{PB}$ nor $\overline{PC}$ crosses the curve, so either $B$ or $C$ can be the next point. To pick up the right point, either $B$ or $C$, we use the neighbor-branch criterion. This criterion is basically an elimination criterion amongst candidates to the next point, and can be described as follows:

1. Determine the midpoints of the segments $\overline{PC}$, $\overline{CB}$, and $\overline{BA}$ in Fig. 5(a)(b) (the midpoint of $\overline{PA}$ is not calculated because $A$ is, by the angle criterion, no longer a candidate next point).
2. For each segment with midpoint $M$, discard its endpoints if the segment $\overline{QM'}$ intersects the curve at one or more points, being $M'$ the projection of $M$ on the neighborhood circle by prolonging $\overline{QM}$. This eliminates $B$ as a candidate next point in Fig. 5(a). Note that the point $B$ in Fig. 5(b) cannot be the next point for a different reason. In fact, the segment $\overline{PB}$ crosses the curve at a point, preventing it from being the next point.

In Fig. 5(c), the angle criterion eliminates $A$ and $E$ as candidate next points, whereas $B$ and $D$ are eliminated by the near-branch criterion. So, the point $C$ is the next point. Note that the neighborhood radius holds constant independently of whether the curve oscillates or not.

### 3.5 The Algorithm

The NEXTPOINT algorithm sketched below determines the point $X$ next to the current point $Q$ according to criteria described above.

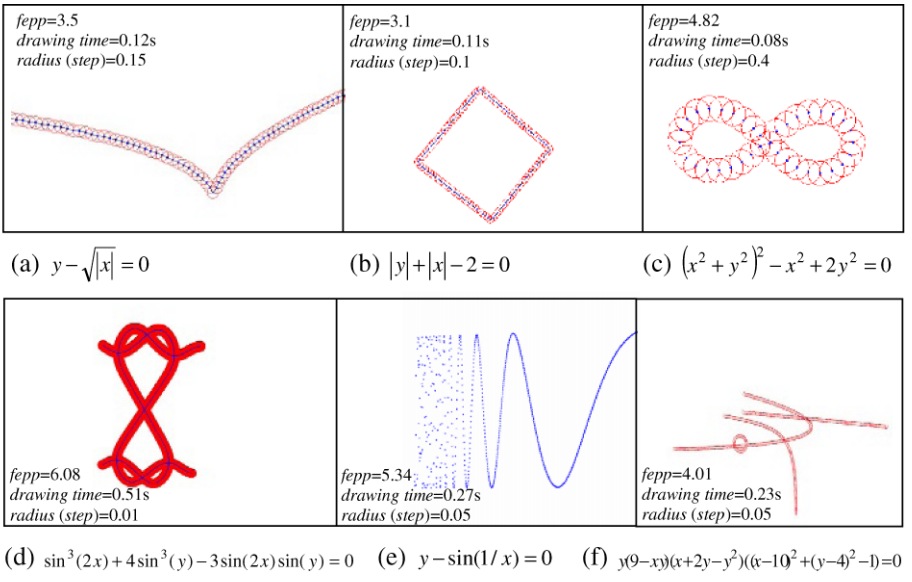


**Fig. 6.** Implicit plane curves

**Algorithm** (NEXTPOINT). *The inputs are the previous and current points $P$, $Q$, the radius $r$ of $N_Q$ and the function defining the curve $\Gamma$. The output is the point $X$ next to $Q$.*

1. Compute the intersection points $\Gamma \cap N_Q$ by means of the angular numerical method described in Sect. 2.2.
2. **if** $(\#(\Gamma \cap N_Q) = 1)$ —a single candidate point
   (a) $X \leftarrow$ get such a single point from $\Gamma \cap N_Q$
   (b) **if** $(\angle(\overline{PQ}, \overline{PX}) \not\approx 180^o)$
   - $X \leftarrow$ compute cusp or quasi-cusp through the convergence technique described in Sect. 3.2

   **else** —two or more candidate points
   (a) $X \leftarrow$ get such a single point from $\Gamma \cap N_Q$ by applying elimination criteria described in Sect. 3.4
   (b) **if** $(X = NULL)$ —there is a self-intersection point about $Q$
   - $X \leftarrow$ compute self-intersection point through the convergence technique described in Sect. 3.3
3. **return** $X$

This algorithm is part of the 2DCURVE algorithm that is able to render generic implicit curves in $\mathbb{R}^2$ as those in Fig. 6, which is not detailed here because of space limitations.

## 4 Conclusions and Experimental Results

The algorithm 2DCURVE was written in C++. Its major contributions are:

- It works for general curves defined by real functions.
- It is derivative-free. So, it does not break down at other singularities.
- It does not break down under shape oscillations and ripples either.
- It introduces a new numerical method for sampling curve points.

Fig. 6 shows interesting performance results for some curves defined by real functions in $\mathbb{R}^2$. The term 'fepp' stands for 'function evaluations per point', i.e. the average number a function is evaluated for each sampled curve point. The runtime tests were performed on a mere Windows PC equipped with a 500MHz Intel Pentium and 128MB RAM, but even so we got really fast rendering times.

## References

1. Allgower, E., Gnutzmann, S.: Simplicial Pivoting for Mesh Generation of Implicitly Defined Surfaces. Comp. Aid. Geom. Des. **8** (1991) 30–46
2. Abhyankar, S., Bajaj, C.: Automatic parameterization of rational curves and surfaces III: algebraic plane curves. Technical Report CSD-TR-619, Purdue University, Computer Science Department, USA (1987)
3. Arnon, D.: Topologically reliable display of algebraic curves. Comp. Graph. **17** (1983) 219–227
4. Bloomenthal, J.: Poligonisation of implicit surfaces. Comp. Aid. Geom. Des. **5** (1988) 341–355
5. Chandler, R.: A tracking algorithm for implicitly defined curves. IEEE Comp. Graph. and Appl. **8** (1988) 83–89
6. Hobby, J.: Rasterization of nonparametric curves. ACM Trans. on Graph. **9** (1990) 262–277
7. Keyser, J., Culver, T., Manocha, D., Krishnan, S.: MAPC: a library for efficient and exact manipulation of algebraic points and curves. In Proceedings of the 15th ACM Symposium on Computational Geometry, ACM Press (1999) 360–369
8. Krishnan, S., Manocha, D.: Numeric-symbolic algorithms for evaluating one-dimensional algebraic sets. In Proceedings of the ACM Symposium on Symbolic and Algebraic Computation (1995) 59–67
9. Lopes, H., Oliveira, J., Figueiredo, L.: Robust adaptive polygonal approximation of implicit curves. In Proceedings of the SibGrapi 2001, IEEE Computer Society (2001)
10. Lorensen, W., Cline, W.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. Comp. Graph. **21** (1987) 163–169
11. Moeller, T., Yagel, R.: Efficient rasterization of implicit functions. http://citeseer.nj.nec.com/357413.html (1995)
12. Press, W., Flannery, B., Teukolsky, S., Vetterling, W.: Numerical recipes in C: the art of scientific computing. Cambridge University Press, 2nd edition, 1992.
13. Shewchuk, J.: Adaptive precision floating-point arithmetic and fast robust geometric predicates. Disc. and Comp. Geom. **18** (1997) 305–363
14. Snyder, J.: Interval arithmetic for computer graphics. In Proceedings of ACM SIGGRAPH'1992, ACM Press (1992) 121–130
15. Taubin, G.: An accurate algorithm for rasterizing algebraic curves. In Proceedings of the 2nd ACM Solid Modeling and Applications, ACM Press (1993) 221–230