Distance Approximations for Rasterizing Implicit Curves

GABRIEL TAUBIN
IBM T. J. Watson Research Center

In this article we present new algorithms for rasterizing implicit curves, i.e., curves represented as level sets of functions of two variables. Considering the pixels as square regions of the plane, a "correct" algorithm should paint those pixels whose centers lie at less than half the desired line width from the curve. A straightforward implementation, scanning the display array evaluating the Euclidean distance from the center of each pixel to the curve, is impractical, and a standard quad-tree-like recursive subdivision scheme is used instead. Then we attack the problem of testing whether or not the Euclidean distance from a point to an implicit curve is less than a given threshold. For the most general case, when the implicit function is only required to have continuous first-order derivatives, we show how to reformulate the test as an unconstrained global root-finding problem in a circular domain. For implicit functions with continuous derivatives up to order k we introduce an approximate distance of order k. The approximate distance of order k from a point to an implicit curve is asymptotically equivalent to the Euclidean distance and provides a sufficient test for a polynomial of degree k not to have roots inside a circle. This is the main contribution of the article. By replacing the Euclidean distance test with one of these approximate distance tests, we obtain a practical rendering algorithm, proven to be correct for algebraic curves. To speed up the computation we also introduce heuristics, which used in conjunction with low-order approximate distances almost always produce equivalent results. The behavior of the algorithms is analyzed, both near regular and singular points, and several possible extensions and applications are discussed.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation - *display algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *curve, surface, solid, and object representations*; J.6 [**Computer Applications**]: Computer-Aided Engineering - *computer-aided design*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Algebraic curve, approximate distance, implicit curve, rendering

1. INTRODUCTION

Planar curves can be represented parametrically or implicitly. A parametric curve is the image set of a two dimensional vector function of one variable $\{(x(t), y(t)) : t \in \mathbb{R}\}$, while an implicit curve is the set of zeros of a function of two variables $Z(f) = \{(x, y) : f(x, y) = 0\}$. Parametric curves are very popular in the graphics and CAD literature, and very efficient methods exist to

render several families of parametric curves. On the other hand, implicit curves are not easy to render.

Since the implicit-function theorem insures that a local parameterization always exists in a neighborhood of a regular point of an implicit curve, i.e., a point p such that $f(p) = 0$ and $\nabla f(p) \neq 0$, one approach that several researchers follow is to approximately parameterize the implicit curve and then render it using the methods for parametric curves [Abhyankar and Bajaj 1987a; 1987b; 1987c; 1988; Mountaudouin 1991; Hobby 1990; Jordan et al. 1973; Van Aken and Novack 1985]. The basic difficulty with this method lies in the fact that implicit curves can be multiply connected and often have singular points, where they intersect themselves, or split into several branches. To render the curve correctly, the singular points and the connected components must be identified [Arnon 1983; Bajaj et al. 1987]. The problem with this approach is that the algorithms are intrinsically complex and computationally expensive.

In this article we present algorithms for rendering implicit curves on raster devices. Since the pixels form a regular grid, except for very particular cases, almost none of the centers of the pixels will lie exactly on the curve. Our algorithms are based on finding the pixels whose centers lie at less than half the desired line width from the curve. In order to obtain practical algorithms, we first reduce the complexity by using a fairly standard quad-tree-like recursive space subdivision scheme [Geisow 1983], but the main problem is to determine whether the Euclidean distance from a point to an implicit curve is less than a threshold or not. We show that in the case of general implicit curves, the test can be reduced to a global unconstrained optimization problem in a circular domain. In the algebraic case, when the defining function is a polynomial, a combination of symbolic and numerical methods can be used to measure the distance [Kriegman and Ponce 1990; Ponce et al. 1992] or just to decide whether or not a polynomial has roots in a circle or square [Milne 1990; Pedersen 1991b]. But these methods are known to be practical only for low-degree polynomials, computationally expensive, and sometimes numerically unstable. Nevertheless, new numerical methods in elimination theory look promising in this area [Manocha 1992; Manocha and Canny 1992]. We then introduce a computationally inexpensive approximate test based on a first-order approximation of the Euclidean distance from a point to an implicit curve, which only requires the evaluation of the function and its first-order derivatives at the point. This first-order distance approximation is not new though; it has been used in the past for algebraic curve and surface fitting [Pratt 1987; Sampson 1982; Taubin 1988; 1991; Taubin et al. 1992; 1993]. The local behavior of this first-order approximate distance near the curve ensures that the rendered curve will have approximately constant width, but it sometimes overestimates the Euclidean distance resulting in missing regions in the rendered curve. Then, and this is the main contribution of the article, we solve the problem for algebraic curves by introducing a family of higher-order approximations to the Euclidean distance from a point to an implicit curve. The approximate distance of order k involves all the partial derivatives of order $\leq k$ of the function at the point; it behaves like the Euclidean distance close to the regular points of the curve; and if the

function is a polynomial of degree $\leq k$, it is a lower bound for the Euclidean distance. This lower bound provides a sufficient condition for rejecting regions which are not cut by the curve. The approximate distance of order k is more expensive to evaluate than the first-order approximate distance, but ensures a correctly rendered algebraic curve without missing points. Combining the first-order approximate distance, the higher-order approximate distances, and some heuristics, we end up with efficient, robust, and correct algorithms for rendering planar algebraic curves. Although now correct, the algorithm is sometimes computationally expensive, particularly for higher-degree algebraic curves. To solve this problem we then introduce two heuristics, which used in conjunction with a low-order approximate distance, usually second order, produces almost equivalent results at substantially less computational cost. There is a tradeoff between accuracy and speed here, and this heuristic approach improves the performance most of the times. When the accuracy of the result is more important, the algorithm based on the higher distance approximations should be used. In a certain sense our algorithms resemble the box-bisection methods for root finding [Baker Kearfott 1987; Eiger et al. 1984; Morgan and Shapiro 1987] and, loosely, the methods based on piecewise linear approximations [Allgower and Georg 1980; Allgower and Schmidt 1985; Bloomenthal 1988; Dobkin et al. 1990].

The article is organized as follows. In Section 2 we define a "correct" algorithm and show how to reduce the complexity of such an algorithm by recursive space subdivision. In Section 3 we describe methods to compute the Euclidean distance from a point to an implicit curve. In Section 4 we introduce the first-order approximate distance; we show that it is asymptotically equivalent to the Euclidean distance in the neighborhood of every regular point and point out its disadvantages. In Section 5 we introduce the higher-order approximate distances; we show that they all behave like the first-order approximate distance in the neighborhood of a regular point and that the approximate distance of order k is a lower bound for the Euclidean distance for polynomials of degree $\leq k$, providing a sufficient condition to safely discard empty pixels (in a sequel to this article [Taubin 1993] we have improved this algorithm correcting its behavior near singular points as well). Since low-order approximate distances are much less expensive to evaluate than the higher-order approximate distances, in Section 6 we introduce heuristics, which combined with low-order approximate distances, usually order one or two, reduce the number of evaluations of the higher-order approximate distances, but still preserve the correctness of the algorithm in most practical cases. In Section 7 we study the behavior of the algorithms near singular points. In Section 8 we present some experimental results, and in Section 9 we discuss several extensions and applications of these methods. Finally, for those readers who would like to reproduce the results, in Appendix A we give all the data necessary to recreate the pictures with the algorithms described in the article.

2. AN IDEAL ALGORITHM

Figure 1 describes an ideal algorithm for rendering the implicit curve $Z(f) = \{(x, y): f(x, y) = 0\}$ in the square of side n pixels centered at (x_0, y_0). The

procedure **IdealPaintZeros** (f, x_0, y_0, n)
　　for $x \leftarrow x_0 - n/2 + 1/2$ to $x_0 + n/2 - 1/2$ step 1 do
　　　　for $y \leftarrow y_0 - n/2 + 1/2$ to $y_0 + n/2 - 1/2$ step 1 do
　　　　　　if $\delta((x, y), Z(f)) <$ half-line-width
　　　　　　　　PaintPixel (x, y)

Fig. 1. Ideal algorithm to render the curve $Z(f)$ in a square grid of $n \times n$ pixels centered at (x_0, y_0). $\delta((x, y), Z(f))$ denotes the Euclidean distance from the point (x, y) to the curve $Z(f)$.

square is scanned, and only those pixels which are cut by the neighborhood of the desired line width of the curve are painted. First of all, the desired line width cannot be smaller than the length of the diagonal of a pixel, which we will assume equal to $\sqrt{2}$. Otherwise, the rasterized version of the curve will probably have gaps. The condition for a pixel to be painted is equivalent to say that the curve must cut the circle of radius half-line-width centered at the center of the pixel, or that the distance from the center of the pixel to the curve is less than half-line-width.

We call this algorithm ideal because every "correct" algorithm should paint exactly the same pixels, but it is impractical. If n is the number of pixels on a side of the square in device space, it requires n^2 distance evaluations, but since in general it will be rendering a smooth curve on a planar region, the number of pixels it is expected to paint is only $O(n)$. We must look for a more efficient algorithm. In fact, there are two issues to look at. The first one is how to reduce the computational complexity of the number of distance evaluations. The second issue is how to efficiently evaluate, or approximate, the distance from a point to an implicit curve. We will discuss the first issue in this section, while subsequent sections will be devoted to the second issue.

The number of distance evaluations can be easily reduced using a quad-tree-like recursive space subdivision scheme described in Figure 2, where n is assumed to be a power of 2. The rationale behind this algorithm is very simple. It starts by considering the initial square region as a single pixel. Then, every square region which had previously satisfied the distance test for its resolution is divided into four equal-size square regions, which are then tested. The squares which pass the distance test are kept into a stack. The size of the squares is divided by two at each iteration; and when the square regions become single pixels, the iteration stops, and the pixels are painted.

Note that as it is described in Figure 2, a block of $n \times n$ pixels is pushed onto the stack, i.e., it is not discarded, if the distance from the center of the block to the curve is less than n times the desired line width for the final result. In fact, as suggested by one of the reviewers, a tighter threshold can be used, producing exactly the same final result, and saving some computation at lower resolutions. For the tighter threshold,

$$\delta((x_i, y_i), Z(f)) < \text{half-line-width} \cdot n$$

should be replaced by

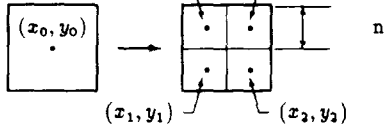$$\delta((x_i, y_i) Z(f)) < \frac{n-1}{\sqrt{2}} + \text{half-line-width}$$

```
procedure RecursivePaintZeros (f, x₀, y₀, n)
    input-stack ← ∅
    output-stack ← ∅
    Push ((x₀, y₀), input-stack)
    while n > 1 do
        n ← n/2
        while input-stack ≠ ∅ do
            (x₀, y₀) ← Pop(input-stack)
```

$$\begin{array}{c} (x_3, y_3) \quad (x_4, y_4) \\ \boxed{(x_0, y_0)} \longrightarrow \boxed{\begin{array}{cc} \bullet & \bullet \\ \bullet & \bullet \end{array}} \quad n \\ (x_1, y_1) \quad (x_2, y_2) \end{array}$$

```
            for i = 1 to 4 step 1 do
                if δ((xᵢ, yᵢ), Z(f)) < half-line-width · n
                    Push((xᵢ, yᵢ), output-stack)
        input-stack ← output-stack
        output-stack ← ∅
    while input-stack ≠ ∅ do
        (x, y) ← Pop(input-stack)
        PaintPixel(x, y)
```

Fig. 2. Improved rendering algorithm. $\delta((x, y), Z(f))$ is the Euclidean distance from the point (x, y) to the curve $Z(f)$.

in Figure 2 and in all the other subdivision algorithms in the rest of the article. A block of $n \times n$ pixels should be kept in the processing stack if the distance from the center of the block to the curve is at most equal to the distance from the center of the block to a corner pixel, $(n - 1)/\sqrt{2}$, plus the distance from a corner pixel to the curve, half-line-width.

Implicitly, this algorithm uses a quad-tree data structure, where the root is the initial square; each node has at most four children; and the leaves are the pixels (for surveys on applications of quad-trees, and other related data structures to graphics, see Overmars [1988] and Samet [1988]). However, since no operation needs to be performed among neighboring squares, it is sufficient to keep the coordinates of the centers of the squares in a list. And since the order in which squares of the same size are processed is not important either, stacks suffice for the implementation. The algorithm of Figure 2 traverses the quad-tree in breath-first order, but if storage limits are important factors, a depth-first traversal can be implemented using recursion. In practice, a mixed strategy is more appropriate, traversing the tree in breath-first order up to a certain resolution, and then in depth-first order, painting in sequence all the pixels in each block. Clearly, if more than one processor is available, the work can be easily divided among them based on these ideas, but we will not elaborate on the subject.

3. EUCLIDEAN DISTANCE

The next problem that we have to deal with is how to answer whether the distance from a certain point $p = (u, v)$ to the set $Z(f)$ of zeros of the function $f : \mathbb{R}^2 \to \mathbb{R}$ is below a certain positive threshold or not. The distance from a point $p = (u, v)$ to the zero set $Z(f)$ of a function $f : \mathbb{R}^2 \to \mathbb{R}$ is defined

as the minimum of the distances from p to points in the zero set

$$\delta(p, Z(f)) = \min\{\|p - q\| : f(q) = 0\}. \tag{3.1}$$

In this section we show that in the general case we will need to explicitly compute the distance using numerical methods, but in the algebraic case (when $f(x, y)$ is a polynomial of low degree) different combinations of symbolic and numerical methods can be used to solve the problem. However, since even in the algebraic case the decision procedure based on the Euclidean distance is computationally expensive, and sometimes numerically unstable, in following sections we study different alternatives to overcome these problems.

3.1 Computing the Euclidean Distance

The problem with computing the Euclidean distance from a point to the zero set of a function, which in this section will be assumed to have continuous first-order partial derivatives ($\mathscr{C}^1$), is that, as the definition (3.1) shows, it is a constrained minimization problem. We describe below how to transform it into an unconstrained minimization problem, for which many well-established numerical methods are available [Dennis and Shnabel 1983; Eiger et al. 1984; Moré et al. 1980].

Since the function is $\mathscr{C}^1$, a necessary condition for the point $\hat{q}$ to minimize $\|p - q\|^2$ constrained by $f(q) = 0$ is that the two vectors $p - \hat{q}$ and $\nabla f(\hat{q})$ be aligned (Lagrange multipliers theorem; see for example Thorpe [1979, Chapter 4] for a geometric interpretation). This condition can be rewritten as $\hat{q} = (\hat{x}, \hat{y})$ being a zero of the function

$$f'(x, y) = (x - u)f_y(x, y) - (y - v)f_x(x, y), \tag{3.2}$$

where f_x and f_y denote the two first-order partial derivatives of f. If we set $\mathbf{f} = (f', f)^T$, every minimizer $\hat{q}$ of the distance from p to $Z(f)$ is a zero of the map $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$. The converse is clearly not true, because not only the minimizers of the distance from p to $Z(f)$ satisfy (3.2). For example, the singular points of $Z(f)$, i.e., the points where not only the function but also the two first-order partial derivatives vanish, are zeros of $\mathbf{f}$. In the general case, we can only say that the zeros of $\mathbf{f}$ are the critical points of the distance from p restricted to $Z(f)$, but very often the set of zeros of $\mathbf{f}$ is finite. Even in the algebraic case the analysis of when the set of zeros of $\mathbf{f}$ is finite is complex, but based on Bezout's theorem [Walker 1950], it can be reduced to find necessary and sufficient conditions for the polynomials f and f' not to have nontrivial common factors. For example, on one extreme is the case of polynomials which are radial at p. We say that a polynomial f is *radial* at $p = (u, v)$ if there exists a univariate polynomial $\phi(t)$, such that $f(x, y) = \phi((x - u)^2 + (y - v)^2)$. If f is radial at p, then f' is identically zero, and the distance from p to $Z(f)$ is equal to the square root of the smallest positive root of ϕ (if any). Also, if the polynomial f has a multiple factor g, i.e., $f = g^2 \cdot h$ for a certain other polynomial h, then g also divides $f' = g(2g' \cdot h + g \cdot h')$, and so $Z(g) \subseteq Z(\mathbf{f})$. Since in general the set of zeros of g is infinite,

so is the set of zeros of **f**. We have decided not to present the analysis of the general case here, because it is long and tedious and will not be used in the rest of the article. Besides, for algebraic curves, a better solution is presented in the next section. When the number of zeros is finite, a method to compute the distance from p to $Z(f)$ is to find all the zeros of **f** in the region of interest (global unconstrained optimization), and then, if more than one is found, choose the one closer to p. In our case, since we only need to determine whether the distance $\delta(p, Z(f))$ is below the threshold or not, the search region should be the circle of center p and radius equal to the threshold. Unless more constraints are imposed on the family of implicit functions, this is the best that we can do. We can do better if the functions are polynomials.

3.2 Euclidean Distance for Algebraic Curves

As shown in Kriegman and Ponce [1990] and Ponce et al. [1992], when the function is a polynomial of low degree, elimination theory can be used to transform the multidimensional minimization problem described in the previous section into a one-dimensional polynomial root-finding problem. Since the points p and $\hat{q}$ and the distance $\delta = \delta(p, Z(f))$ satisfy the following system of algebraic equations

$$\begin{cases} f(x, y) = 0 \\ f'(x, y) = 0 \\ (x - u)^2 + (y - v)^2 - \delta^2 = 0 \end{cases} \tag{3.3}$$

where f' is the polynomial previously defined in Equation (3.2), the variables x and y can be eliminated among these three equations, yielding a single equation

$$D(\delta) = 0,$$

where D is a polynomial in the distance δ, with coefficients polynomials in the coordinates (u, v) of the point p, and the coefficients of f (for other applications of elimination theory to graphics and CAD, see Sederberg et al. [1984; 1985] and Canny [1988] and Manocha and Canny [1992] for multidimensional resultants). The distance δ is the minimum positive root of this polynomial, and it can be found by some numerical root-finding algorithm. In our case, where we need to test whether $\delta(p, Z(f))$ is less than a threshold or not, we can just apply Sturm's theorem [Bochnak et al. 1987, Theorem 1.2.8] to the univariate polynomial $D(\delta)$ to count the number of real zeros it has in the interval [0, threshold]. That is, we look at both the coefficients of the polynomial $f(x, y)$ and the coordinates of the point $p = (u, v)$ not as variables, but as constants, and consider the polynomials in (3.3) as functions of three variables x, y, and δ. The two variables x, y must be eliminated at every point though, yielding the polynomial $D(\delta)$ in a single variable δ. Although efficient and robust methods to evaluate multidimensional resultants have been developed lately [Manocha and Canny 1992], the amount of computation involved in this approach is still impractical for our purposes. Finally, let us mention that other researchers have developed numerical-sym-

bolic methods to count the number of zeros that a polynomial, or system of polynomials, has inside a square [Milne 1990], circle, or a more general semialgebraic region [Pedersen 1991b], but all these methods suffer from the same kind of problems: they are either computationally expensive or numerically unstable.

4. A FIRST-ORDER APPROXIMATE DISTANCE

Since the test based on evaluating the Euclidean distance is computationally expensive, even in the algebraic case because the resultant D is usually a polynomial of high degree, or numerically unstable, we seek alternatives.

We will first perform a less computationally expensive test. This test is to be based on a simple first-order approximation to the Euclidean distance and is asymptotically equivalent to the Euclidean distance test. However, since this first-order approximate-distance test can (and often does) reject regions that otherwise would be accepted by the Euclidean distance test, more expensive tests based on higher-order approximations to the Euclidean distance will be introduced in Section 5. The approximate distance of order k will be shown to be a lower bound to the Euclidean distance for curves defined by polynomials of degree $\leq k$, and so, for these curves a test based on this approximation never rejects a region that would otherwise be accepted by the Euclidean distance test. Also, the approximate-distance test of order k will provide a sufficient condition for a polynomial of degree $\leq k$ not to have roots inside a circle of given radius. To minimize the computation, the first-order approximate-distance test will always be applied first, and the more expensive higher-order distance tests will be applied only to regions previously rejected by the first-order approximate-distance test of this section.

The first-order approximate-distance test is based on replacing the Euclidean distance from a point to a zero set of a smooth function by a first-order approximation, an approximation that has successfully been used in the past within algebraic curve- and surface-fitting algorithms [Pratt 1987; Sampson 1982; Taubin 1988; 1991]. This is a generalization of the update step of the Newton method for root finding [Dennis and Shnabel 1983], and although it can be extended to higher dimensions, as for example the case of surface-surface intersections (briefly discussed in Section 9), here we restrict our derivation only to the case of interest, i.e., the two-dimensional case.

Let p be a point such that $\|\nabla f(p)\| \neq 0$, and let us expand $f(q)$ in Taylor series up to first order in a neighborhood of p

$$f(q) = f(p) + \nabla f(p)^T (q - p) + O\big(\|q - p\|^2\big).$$

We now truncate the series after the linear terms, apply the triangular inequality, and then the Cauchy–Schwarz inequality to obtain

$$|f(q)| \approx |f(p) + \nabla f(p)^T (q - p)| \geq |f(p)| - |\nabla f(p)^T (q - p)|$$
$$\geq |f(p)| - \|\nabla f(p)\| \|(q - p)\|.$$

We now define the first-order approximate distance from p to $Z(f)$ as the value of $\|q - p\|$ that annihilates the expression on the right side

$$\delta_1(p, Z(f)) = \frac{|f(p)|}{\|\nabla f(p)\|}.$$ (4.1)

For rendering curves of approximately constant width based on the space subdivision scheme of Figure 2, an approximation to the Euclidean distance should be a first-order approximation in a neighborhood of every regular point of the curve (a point $p \in \mathbb{R}^2$ such that $f(p) = 0$ but $\|\nabla f(p)\| > 0$). That is, at sufficiently high resolution, the set of pixels which lie at approximate distance not larger than half the desired line width from the curve must be perceived as a curve of approximately constant width. Indeed, besides the fact that it can be computed very fast in constant time, requiring only the evaluation of the function and its first-order partial derivatives, the approximation defined by Equation (4.1) is a first-order approximation to the Euclidean distance.

LEMMA 1. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function with continuous derivatives up to second order, in a neighborhood of a regular point p_0 of $Z(f)$. Let w be a unit length normal vector to $Z(f)$ at p_0, and let $p_t = p_0 + tw$, for $t \in \mathbb{R}$. Then*

$$\lim_{t \to 0} \frac{\delta_1(p_t, Z(f))}{\delta(p_t, Z(f))} = 1.$$

PROOF. In the first place, under the current hypotheses, there exists a neighborhood of p_0 within which the Euclidean distance from the point p_t to the curve $Z(f)$ is exactly equal to $|t|$

$$\delta(p_t, Z(f)) = \|p_t - p_0\| = |t|$$

(see for example Thorpe [1979, Chapter 16]). Also, since p_0 is a regular point of $Z(f)$, we have $\|\nabla f(p_0)\| > 0$, and by continuity of the partial derivatives, $1/\|\nabla f(p_t)\|$ is bounded for small $|t|$. Thus, we can divide by $\|\nabla f(p_t)\|$ without remorse. And since w is a unit length normal vector, $\nabla f(p_0) = \pm\|\nabla f(p_0)\|w$. Finally, by continuity of the second-order partial derivatives, and by the previous facts, we obtain

$$\nabla f(p_t) = \nabla f(p_0) + O(\|p_t - p_0\|) = \pm\|\nabla f(p_0)\|w + O(|t|)$$

and

$$0 = f(p_0) = f(p_t) + \nabla f(p_t)^T(p_0 - p_t) + O\left(\|p_0 - p_t\|^2\right)$$

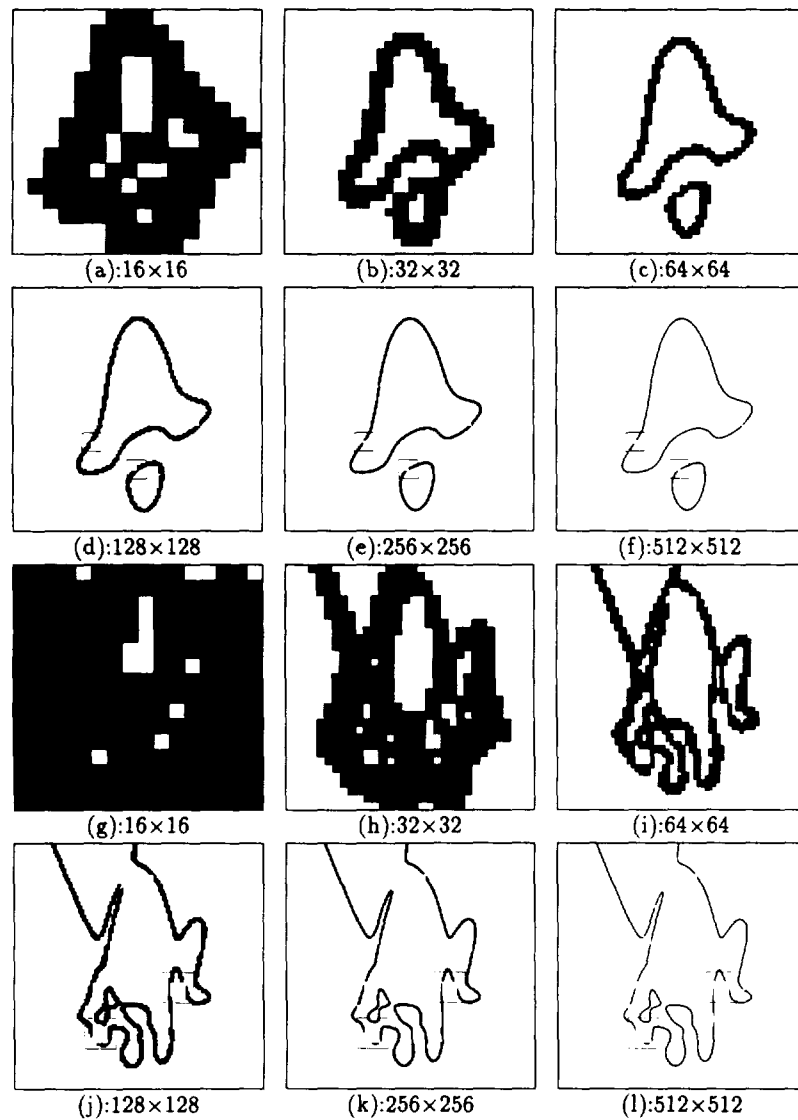$$= f(p_t) \mp \|\nabla f(p_t)\|t + O(|t|^2).$$

Fig. 3. Rendering a fourth-degree and a tenth-degree regular algebraic curves with the algorithm **RecursivePaintZeros**, and the first-order approximate distance instead of the Euclidean distance. The squares in the higher-resolution pictures enclose some regions erroneously discarded at lower resolutions.

Moving $f(p_t)$ to the other member, dividing by $\|\nabla f(p_t)\|\|t\|$, and taking absolute value, we obtain

$$\frac{\delta_1(p_t, Z(f))}{\delta(p_t, Z(f))} = \frac{|f(p_t)|}{\|\nabla f(p_t)\|\,|t|} = 1 + O(|t|),$$

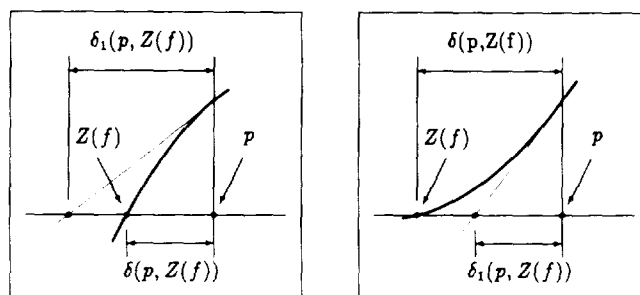which finishes the proof. □

Fig. 4. The first-order approximate distance either overestimates or underestimates the exact distance.

We can now replace the exact distance by the approximate distance in the algorithm **RecursivePaintZeros** (Figure 2). As can be appreciated in Figure 3, the results are not very satisfactory. The sets of rendered pixels clearly represent curves of constant width, but large blocks of pixels are missing.

As it can be seen already in the one-dimensional case in Figure 4, the approximate distance either underestimates or overestimates the exact distance. Underestimation is not a problem for the algorithm described above, because large blocks of pixels will not be erroneously discarded at early stages of the computation, and the error will be corrected at later stages at the expense of more approximate-distance evaluations. But overestimation is a real problem, particularly at low resolution when large blocks of pixel can be discarded at early stages of the algorithm. It is not difficult to find examples where the four blocks of pixels are discarded at the initial first-order approximate-distance evaluation. For example, the zero set of the polynomial $f(x, y) = (x^2 + y^2 - 1)^2$ is a circumference of radius 1, but the gradient of f is zero at the origin; and so, the first-order approximate distance from the origin to $Z(f)$ is ∞. The results are the gaps that can be appreciated in the higher-resolution pictures of Figure 3. For algebraic curves the higher-order approximate distances of Section 5 solve the problem. For a curve defined by a polynomial of degree k the solution will be, as we mentioned above, to first perform the first-order approximate-distance test to minimize the computation, and if a block is discarded, then perform the approximate-distance test of order k, to be sure that no block is erroneously discarded. It is important to observe that, except for this problem, the algorithm based on the first-order approximate distance does a good job.

5. HIGHER-ORDER APPROXIMATE DISTANCES

As we mentioned before, the main disadvantage of the first-order approximate distance is that it sometimes overestimates the Euclidean distance, resulting in regions erroneously rejected, particularly at early stages of the space subdivision scheme. In this section we derive a family of approximate distances. The first-order approximate distance involves the value of the

function and the first-order partial derivatives at the point. This will be the simplest member of our family. In general, the approximate distance of order k will involve all the partial derivatives of order $\leq k$ at the point, and for a polynomial f of degree $\leq k$, will prove to be a lower bound for the Euclidean distance from the point to the algebraic curve $Z(f)$. That is, we will introduce an algorithm to compute a function $\delta_k(p, Z(f))$ of the coordinates of the point, and the partial derivatives of order $\leq k$ of the function f at the point p, satisfying the following inequality

$$0 \leq \delta_k(p, Z(f)) \leq \delta(p, Z(f))$$

when f is a polynomial of degree $\leq k$. Since all the approximate distances will be shown to be asymptotically equivalent to the Euclidean distance near regular points, by replacing the Euclidean distance with the approximate distance of order k in our space subdivision schemes we will obtain correct and practical algorithms for rasterizing curves defined by polynomials of degree $\leq k$.

5.1 Approximate Distance of Order k

In the rest of this section, we will assume (1) that the function f has continuous partial derivatives up to order $k + 1$ in a neighborhood of the point $p = (u, v)$ and (2) that $f(p) \neq 0$, because otherwise the Euclidean distance from p to $Z(f)$ is zero. Taylor's formula affirms that in such a case we can write

$$f(q) = \sum_{0 \leq i, j, i+j \leq k} f_{ij}(x - u)^i (y - v)^j + O(\|q - p\|^{k+1}),$$

where $q = (x, y)$, and

$$f_{ij} = \frac{1}{i! j!} \frac{\partial^{i+j} f(p)}{\partial x^i \partial y^j}.$$

In the case of polynomials, Horner's algorithm can be used to compute these coefficients stably and efficiently [Borodin and Munro 1975], and even parallel algorithms exist to do so [Dowling 1990] (we describe Horner's algorithm and other implementation details below in Section 5.3). Now we truncate the series after the terms of degree k, and we end up with a polynomial of degree $\leq k$, which we will denote $T^k f_p$. Remember that $T^h f_p(q) \equiv f(q)$ for $h \geq k$, if f is a polynomial of degree $\leq k$. We want to find a lower bound for the distance from p to the set $Z(T^k f_p) = \{q : T^k f_p(q) = 0\}$. We first rewrite the polynomial as a sum of forms, i.e., polynomials with all the terms of the same degree

$$T^k f_p(q) = \sum_{h=0}^{k} \left\{ \sum_{i+j=h} f_{ij}(x - u)^i (y - v)^j \right\} = \sum_{h=0}^{k} f_p^h(q), \qquad (5.1)$$

and apply the triangle inequality

$$|T^k f_p(q)| \geq |f_p^0| - \sum_{h=1}^{k} |f_p^h(q)|.$$ (5.2)

We now consider each one of the forms individually, and show upper bounds for them. For this we need to recall the binomial formula

$$\left((x - u)^2 + (y - v)^2\right)^h = \sum_{i+j=h} \binom{h}{i}(x - u)^{2i}(y - v)^{2j} = \|X_h\|^2,$$

where X_h is the vector of monomials $\left\{ \binom{h}{i}^{1/2}(x - u)^i(y - v)^j : i + j = h \right\}$, and $\| \cdot \|$ is the Euclidean norm in $\mathbb{R}^{h+1}$. Now we can write the form f_p^h as an inner product in $\mathbb{R}^{h+1}$

$$d^h f_p(q) = \sum_{i+j=h} \left(f_{ij} \middle/ \binom{h}{i}^{1/2} \right) \cdot \left(\binom{h}{i}^{1/2}(x - u)^i(y - v)^j \right) = \sum_{i+j=h} F_h^T X_h,$$

with F_h the vector of normalized coefficients $\left\{ f_{ij} \middle/ \binom{h}{i}^{1/2} : i + j = h \right\}$, and $F_0 = f_p^0 = f_{00} = f(0,0)$, and apply the Cauchy–Schwarz inequality

$$|f_p^h(q)| = F_h^T X_h \leq \|F_h\| \|X_h\| = \|F_h\| \delta^h,$$

where $\delta = \sqrt{(x - u)^2 + (y - v)^2}$. Now we can return to Equation (5.2) and obtain

$$|T^k f_p(q)| \geq |f_p^0| - \sum_{h=1}^{k} |f_p^h(q)| \geq |F_0| - \sum_{h=1}^{k} \|F_h\| \delta^h.$$

Note that since it attains the positive value $|F_0| = |f(p)|$ at $\delta = 0$, and it is monotonically decreasing for $\delta > 0$, the polynomial $F_p^k(\delta) = |F_0| - \sum_{h=1}^{d} \|F_h\| \delta^h$ has a unique positive root δ_k. This number, δ_k, is a lower bound for the Euclidean distance from p to $Z(T^k f_p)$, because for $\delta < \delta_k$, we have

$$|T^k f_p(q)| \geq |F_0| - \sum_{h=1}^{d} \|F_h\| \delta^h > 0,$$

i.e., if $q = (x, y)$ is a point of $Z(T^k f_p)$ which minimizes the distance from p, then $\delta \geq \delta_k$. If f is a polynomial of degree $\leq k$, we have $f(q) \equiv T^k f_p(q)$, and so δ_k is a lower bound for the Euclidean distance from p to $Z(f)$.

For our application, where we need to know whether or not a certain given value of δ is larger or smaller than δ_k, we do not compute the value of δ_k, we

just evaluate the polynomial $F_p^k(\delta)$ and determine the sign of the result. If $F_p^k(\delta) > 0$ then the distance from p to $Z(T^k f_p)$ is larger than δ, and in the case of polynomials of degree $\leq k$, this means that the circle of radius δ centered at p can be safely discarded, because the curve $Z(f)$ does not cut it.

If the value of δ_k was needed for a different application, due to the monotonicity of the polynomial $F_p^k(\delta)$ and the uniqueness of the positive root, any univariate root-finding algorithm will converge very quickly to δ_k. In particular, a Newton-based algorithm will converge in a few iterations. But in order to make such an algorithm work, we need a practical method to compute an initial point or to bracket the root. Since we already have a lower bound ($\delta = 0$), we only need to show an upper bound for δ_k. For this, note that

$$F_p^k(\delta) = |F_0| - \sum_{h=1}^{k} \|F_h\| \delta^h \leq |F_0| - \sum_{h=1}^{k-1} \|F_h\| \delta^h = F_p^{k-1}(\delta)$$

for every $\delta \geq 0$, and so $\delta_k \leq \delta_{k-1}$. Also note that δ_1 is nothing but the first-order approximate distance of Section 4 because $\|F_1\| = \|\nabla f(p)\|$, i.e., the approximate distance δ_k is also a lower bound for the first-order approximate distance δ_1. If $\|F_2\| \neq 0$, a tighter upper bound for δ_k is δ_2. The polynomial $F_2(\delta)$ has two roots, one positive and one negative. The positive root is

$$\delta_2 = \sqrt{\frac{\|F_1\|^2}{4\|F_2\|^2} + \frac{|F_0|}{\|F_2\|}} - \frac{\|F_1\|}{2\|F_2\|}, \tag{5.3}$$

which satisfies the inequality $0 \leq \delta_k \leq \delta_2 \leq \delta_1$. Replacing the Euclidean distance with this second-order approximate distance very often solves the problems of the first-order approximate distance, as Figure 5 shows, because although it is quite common to hit a point where both first-order derivatives vanish, it is not so easy to hit a point where all the first- and second-order derivatives vanish.

If $f(p) \neq 0$, but all the partial derivatives of f of order $< k$ are zero at p, but at least one partial derivative of f of order k is nonzero at p, we have $\delta_1 = \cdots = \delta_{k-1} = \infty$ and

$$\delta_k = \left(\frac{|F_0|}{\|F_k\|} \right)^{1/k},$$

because $|F_0| \neq 0$, $\|F_1\| = \cdots = \|F_{k-1}\| = 0$, and $\|F_k\| \neq 0$. In general, if $\|F_k\| \neq 0$, from the inequality

$$|F_0| - \sum_{h=1}^{k} \|F_h\| \delta^h \leq |F_0| - \|F_k\| \delta^k$$
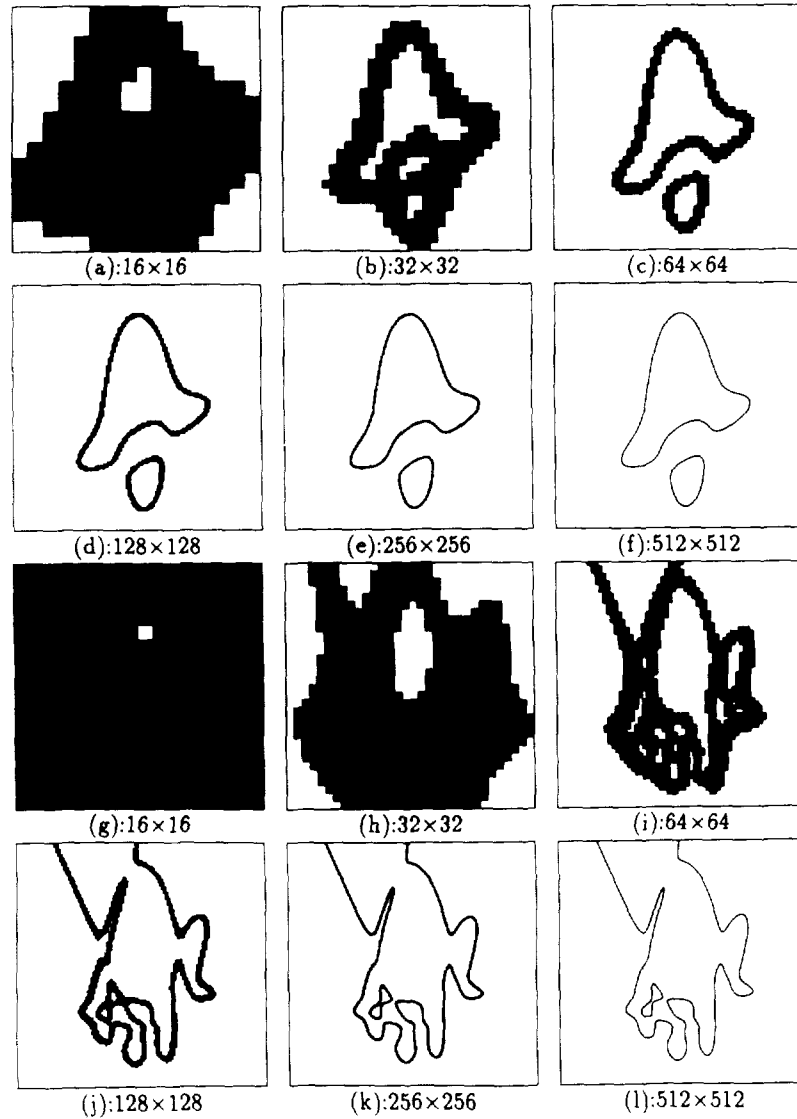
Fig. 5.   Rendering a fourth-degree and a tenth-degree regular algebraic curves correctly with the algorithm **RecursivePaintZeros**, and the second-order approximate distance instead of the Euclidean distance. Compare with Figure 3.

we obtain a new upper bound for the approximate distance of order k:

$$\delta_k \leq \left(\frac{|F_0|}{\|F_k\|} \right)^{1/k}$$

5.2 Asymptotic Behavior

In this section we show that the asymptotic behavior of all the approximate distances near regular points are determined by the behavior of the first-order approximate distance, i.e., the first-order approximate distance.

LEMMA 2. *Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a function with continuous partial derivatives up to order $k + 1$, in a neighborhood of a regular point p_0 of $Z(f)$. Let w be a unit length normal vector to $Z(f)$ at p_0, and let $p_t = p_0 + tw$, for $t \in \mathbb{R}$. Then*

$$\lim_{t \to 0} \frac{\delta_k(p_t, Z(f))}{\delta(p_t, Z(f))} = 1.$$

PROOF. Since the case $k = 1$ was the subject of Lemma 1, let us assume that $k > 1$. For every value of t, let us denote $\delta_k(t)$ the approximate distance of order k from p_t to $Z(f)$, i.e., $\delta_k(t)$ is the unique positive number which satisfies the following equation

$$0 = |F_0(t)| - \sum_{h=1}^{k} \|F_h(t)\| \delta_k(t)^h,$$

where here the coefficients are continuous functions of t, because they are continuous functions of the partial derivatives of f evaluated at p_t. Since for small t $|F_0(t)| = |f(p_t)| \neq 0$ $(t \neq 0)$ and $\|F_1(t)\| = \|\nabla f(p_t)\| \neq 0$, we can divide by $|F_0(t)|$ and by $\|F_1(t)\|$. Remembering that the first-order approximate distance is $\delta_1(t) = |F_0(t)|/\|F_1(t)\|$, we can rewrite the previous equation as follows:

$$1 - \frac{\delta_k(t)}{\delta_1(t)} = \frac{\delta_k(t)}{\delta_1(t)} \left\{ \sum_{h=2}^{k} \frac{\|F_h(t)\|}{\|F_1(t)\|} \delta_k(t)^{h-2} \right\} \delta_k(t).$$

Now we observe that the three factors on the right side are positive; the first factor is bounded by 1 because $0 < \delta_k(t) \le \delta_1(t)$; the second factor is bounded by

$$\sum_{h=2}^{k} \frac{\|F_h(t)\|}{\|F_1(t)\|}$$

for $\delta_1(t) < 1$, which is continuous at $t = 0$, and so bounded for $t \to 0$, and the last factor is bounded by $\delta_1(t)$. That is, we have shown that

$$\frac{\delta_k(t)}{\delta_1(t)} = 1 + O(\delta_1(t))$$

which based on Lemma 1 finishes the proof. □

5.3 The Approximate-Distance Test

As we explained above, the approximate-distance test of order k provides a sufficient condition for the distance from a point p to the set $Z(T^k f_p)$ to be larger than a certain threshold. The procedure **ApproxDistanceTest** (p, f, k, ϵ) should return the value FALSE if $\delta_k(p, Z(f)) < \epsilon$, and TRUE otherwise. Since δ_k is a lower bound for $\delta(p, Z(T^k f_p))$, if δ_k is larger than the threshold, then so is $\delta(p, Z(T^k f_p))$. So, the circular region of radius δ centered at p will be discarded when $\delta_k \ge \epsilon$, or equivalently when $F_p^k(\epsilon) > 0$. Now, in order not to discard a region, it might not be necessary to evaluate

```
procedure ApproxDistanceTest (p, f, k, δ)
    for h ← 0 to k step 1 do
        for d ← degree(f) - 1 to h step -1 do
            for i ← 0 to h step 1 do
                f_{i,d-i-1} ← f_{i,d-i-1} + u · f_{i,d-i}
            for i ← h to d step 1 do
                f_{i,d-i-1} ← f_{i,d-i-1} + v · f_{i,d-i}
        ||F_h|| ← √(Σ_{i+j=h} f²_{i,j} / (ʰᵢ))
        if h = 0
            F_p^h(ε) ← |f(p)|
        else
            F_p^h(ε) ← F_p^{h-1}(ε) - ||F_h||δ^h
        if F_p^k(ε) < 0
            return FALSE
    return TRUE
```

Fig. 6. Practical implementation of the approximate-distance test ($p = (u, v)$).

$F_p^k(\epsilon)$. Since $F_p^k(\epsilon) \leq F_p^{k-1}(\epsilon) \leq \cdots \leq F_p^1(\epsilon)$, if for certain $h \leq k$ we have $F_p^h(\epsilon) < 0$, we certainly have $F_p^k(\epsilon) < 0$. In order to save computation we can evaluate in sequence $F_p^1(\epsilon), F_p^2(\epsilon), \ldots, F_p^k(\epsilon)$. If $F_p^h(\epsilon) < 0$ for a certain value of h, it is not necessary to continue. Also, $F_p^h(\epsilon)$ can be easily computed from $F_p^{h-1}(\epsilon)$ by subtracting $||F_h||\delta^h$, and the elements of the coefficient vector F_h can be computed recursively as well, using Horner's rule. Figure 6 describes a practical implementation of this approximate-distance test. The inner two loops correspond to Horner's algorithm. And if f is a polynomial, for a correct rendering k should be equal to the degree of f in the calling statement.

6. HEURISTICS

In this section we introduce simple heuristics to reduce even further the number of times that the test based on a higher-order approximate distance needs to be performed.

As we have noted above, the basic problem with replacing the exact distance with the first-order approximate distance is that at low resolution, the first-order approximate distance often overestimates the exact distance, and large blocks of pixels are discarded at early stages of the algorithm. A simple heuristic to try to solve this problem is described in Figure 7. A pixel block is not discarded if it does not pass the distance test, unless its father did not pass the distance test either. With this simple heuristic, pixels stay longer in the processing queues, and very often the curves are correctly rendered. Figure 8 shows the same curves of Figure 3, but rendered with this modified algorithm. The gray areas represent the pixels which do not pass the distance test at the current resolution, but are kept in the processing queues because their fathers did pass the distance test. The first curve is drawn correctly, but the second one still has a missing block of pixels. The main advantage of this heuristic is that it is as simple to implement as the algorithm **RecursivePaintZeros**. The disadvantage is clearly that, although in a lesser degree, it still produces errors. In our experience, the algorithm works very well. It is difficult to find examples where it fails to render all the points, but nevertheless, examples exist, as Figure 8 shows.

procedure **KeepOnePaintZeros** (f, x_0, y_0, n)
    input-stack $\leftarrow \emptyset$
    output-stack $\leftarrow \emptyset$
    Push $((x_0, y_0, \boxed{accepted}), \text{input-stack})$
    while $n > 1$ **do**
        $n \leftarrow n/2$
        while input-stack $\neq \emptyset$ **do**
            $(x_0, y_0, \boxed{\text{status-father}}) \leftarrow$ **Pop**(input-stack)

        for $i = 1$ **to** 4 **step** 1 **do**
            if $\delta_k((x_i, y_i), Z(f)) <$ half-line-width $\cdot n$
                Push$((x_i, y_i, \boxed{accepted}), \text{output-stack})$
            $\boxed{\textbf{else if } n > 1 \textbf{ and } \text{status-father} = accepted}$
                $\boxed{\textbf{Push}((x_i, y_i, \textit{rejected}), \text{output-stack})}$
        input-stack $\leftarrow$ output-stack
        output-stack $\leftarrow \emptyset$
    while input-stack $\neq \emptyset$ **do**
        $(x, y) \leftarrow$ **Pop**(input-stack)
        PaintPixel(x, y)

Fig. 7. First approximate rendering algorithm. Differences with the algorithm **Recursive-PaintZeros** are enclosed in boxes.

To understand why this heuristic fails, look at the gray areas in the pictures of Figure 8. These areas can be seen as *safety neighborhoods* of the rendered (black) pixels. In order to be sure that no pixel is erroneously discarded, pixels in these safety neighborhoods are also tested at the next higher resolution. What is apparent from the pictures is that the neighborhoods are not balanced around the rendered pixels. Some discarded pixels are not separated from the rendered pixels by the safety neighborhoods, and those are exactly the cases where low-resolution pixels are erroneously discarded. The next heuristic involves keeping the neighborhoods balanced. This effect can be achieved by applying a dilation operation [Serra 1982] at the beginning of each subdivision level, as described in Figures 9 and 10. For every pixel currently accepted, the four nearest neighbors (4NN) in the vertical and horizontal directions are included as the safety neighborhood, if not yet in the processing queues. The main advantage of this heuristic is that it works better than the previous one. The disadvantages are two. First of all, the data structures must become more complex. We have to represent two-dimensional sets, be able to test membership, and be able to efficiently locate the four nearest neighbors of a pixel. The second disadvantage is that, because of the dependencies on the neighboring pixels, it can only be applied to the breath-first version of the subdivision algorithms. The simplest data structure for representing a two-dimensional grid and for testing neighbor
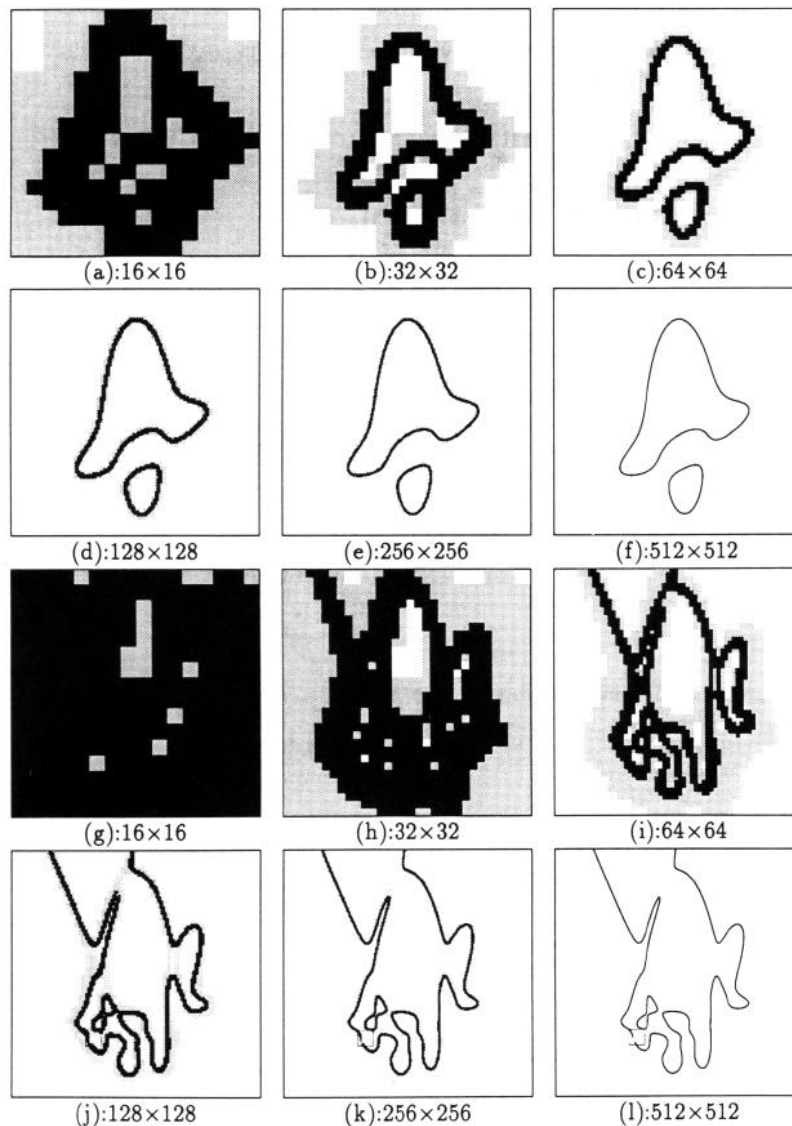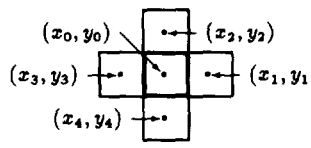
Fig. 8. Rendering the curves of Figure 3 with the algorithm **KeepOnePaintZeros**: No missing points in the first case. In the second case the quality has improved, but it still has some missing points. The gray areas represent the pixels which have been rejected at the current resolution, but kept in the processing queues.

membership is a two-dimensional array, with which the required operations can be executed in constant time, but requiring $O(n^2)$ storage. The alternative is to use an explicit quad-tree structure, for which membership requires $\log_2(\text{depth}) + 1$ operations, and the expected time for testing membership of the four nearest neighbors of a member of the set is constant [Samet 1988].
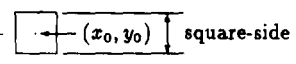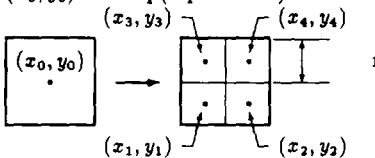
procedure **DilateStack4NN** (input-stack, n, clipping-region)
   set ← $\emptyset$
   for $(x_0, y_0) \in$ input-stack **do**
      set ← set $\cup \{(x_0, y_0)\}$
   output-stack ← $\emptyset$
   while input-stack $\neq \emptyset$ **do**
      (x_0, y_0) ← Pop(input-stack)
      Push$((x_0, y_0)$, output-stack)

   for $i = 1$ **step** 1 **until** $i = 5$ **do**
      if $(x_i, y_i) \notin$ set **and** $(x_i, y_i) \in$ clipping-region
         set ← set $\cup \{(x_i, y_i)\}$
            Push$((x_i, y_i)$, output-stack)
   return output-stack

Fig. 9.   Simple 4-NN dilation procedure.

procedure **RecursivePaintZeros4NN** (f, x_0, y_0, n)

clipping-region ←

input-stack ← $\emptyset$
output-stack ← $\emptyset$
Push$((x_0, y_0)$, input-stack)
while $n > 1$ **do**
   | input-stack ← **DilateStack4NN** (input-stack, n, clipping-region) |
   n ← $n/2$
   while input-stack $\neq \emptyset$ **do**
      (x_0, y_0) ← Pop(input-stack)

      for $i = 1$ **to** $i = 4$ **step** 1 **do**
         if $\delta_k((x_i, y_i), Z(f)) <$ half-line-width $\cdot n$
            Push$((x_i, y_i)$, output-stack)
   n ← $n/2$
   input-stack ← output-stack
   output-stack ← $\emptyset$
while input-stack $\neq \emptyset$ **do**
   (x, y) ← Pop(input-stack)
   PaintPixel(x, y)

Fig. 10.   Approximate rendering algorithm based on the 4-NN dilation heuristic. Differences with the algorithm **RecursivePaintZeros** are enclosed in boxes.
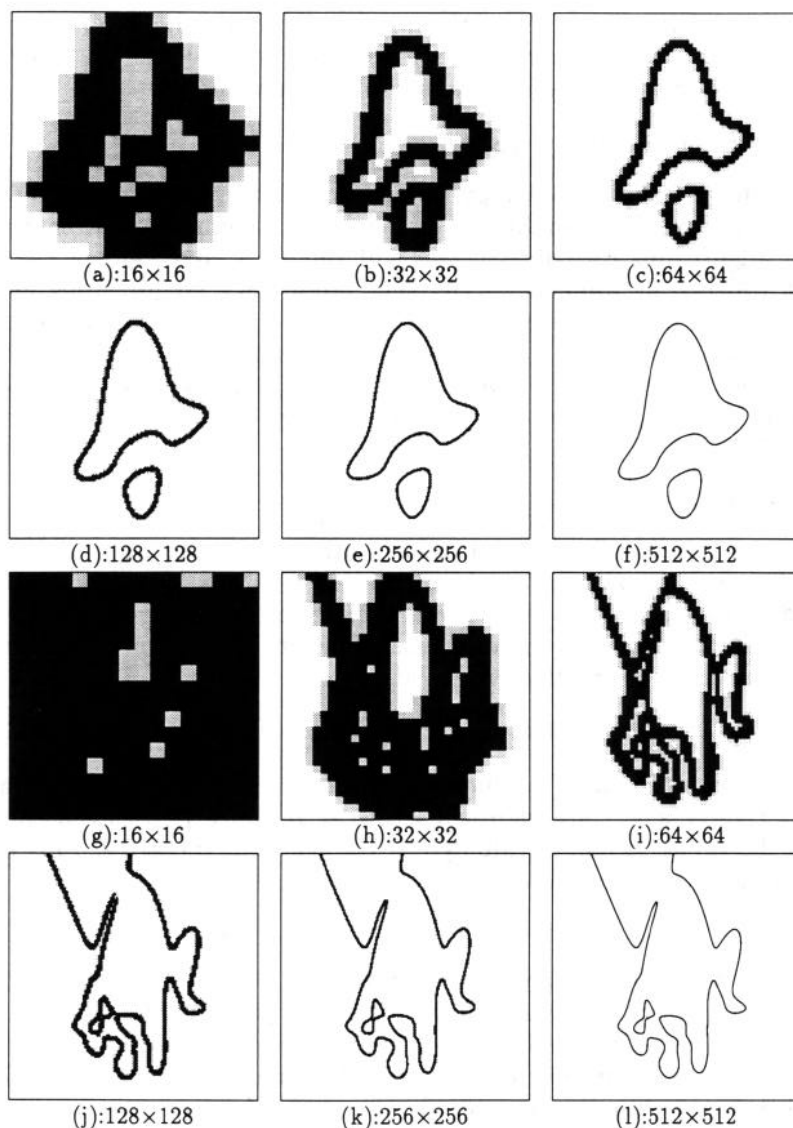
Fig. 11. The curve of Figure 3 correctly rendered with the algorithm **RecursivePaint-Zeros4NN**. The gray areas represent the pixels included by the procedure **DilateStack4NN**.

However, since usually even the display array can be used for this purpose, it is practical to use the two-dimensional array. In the case of a very high-resolution output device, the initial square region can be divided into blocks of manageable size and then process them one by one. Figure 11 shows the same two curves of Figures 3 and 8 correctly rendered by this algorithm.

If we picture the set of different resolution pictures as a pyramid, the first heuristic applies vertically and the second one horizontally. Since neither one

of the two heuristics described in this section precludes the application of the other one, and although the last heuristic works very well in practice, in certain cases it might be necessary to apply both of them.

7. BEHAVIOR NEAR SINGULAR POINTS

So far, we have talked about regular curves. That is, the gradient ∇f of the function f is never zero on $Z(f)$. Lemma 1 and Lemma 2 show that in this case, the algorithms described above will render a curve of approximately constant width. But implicit curves very often have singularities, and our algorithms have to deal with them. In this section we study the behavior of the algorithms described above near singular points. We show that in most cases, they correctly render curves with singular points without modification; we identify those cases where they do not; and we propose solutions to these problems.

7.1 Multiple Points and Tangent Lines

In order not to clutter the exposition with unnecessary details, we will assume in what follows that the function $f(x, y)$ has as many continuous partial derivatives as needed. The *multiplicity* of a point p as a zero of the function f is the minimum index h such that at least one partial derivative of f of order h is not identically zero at p

$$m(p; f) = \min\{h : f_p^h \neq 0\},$$

where the form f_p^h was introduced in Equation (5.1) above or ∞ if all the partial derivatives of f are identically zero at p. Since f_p^0 is a constant, and equal to the function itself evaluated at the point p, the set $Z(f)$ is exactly the set of points with positive multiplicity

$$Z(f) = \{p \in \mathbb{R}^2 : m(p; f) > 0\}.$$

A point p of multiplicity one is called a *simple*, or *regular*, point of $Z(f)$. If $m(p; f) > 1$, p is called a *multiple*, or *singular*, point of $Z(f)$.

Lemma 1 described the behavior of the first-order approximate distance δ_1 in the vicinity of a simple point, while Lemma 2 did the same for the higher-order approximate distance δ_k. The behavior near a multiple point depends not only on the multiplicity of the point itself, but also on the structure of the tangent space at the point. At every simple point p, the curve $Z(f)$ has a unique tangent, whose implicit equation is given by

$$\left\{q \in \mathbb{R}^2 : \nabla f(p)^T(q - p) = f_p^1(q) = 0\right\}.$$

A multiple point has multiple tangents. According to Taylor's formula, if $m(p; f) = k > 1$, then

$$f(q) = f_p^k(q) + O\left(\|q - p\|^{k+1}\right). \tag{7.1}$$

Since f_p^k is a kth degree form in two variables, it can be factorized as a product of some quadratic forms without nontrivial roots $s_1, \ldots, s_r$ and some
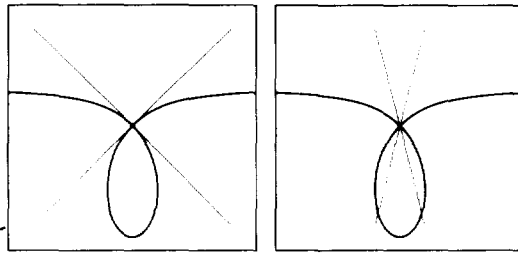
Fig. 12.   Tangent lines and transversal lines at a singular point.

linear forms $l_{r+1}, \ldots, l_k$, [Walker 1950], all of these with real coefficients:

$$f_p^k(q) = \prod_{i=1}^{r} s_i(q - p) \cdot \prod_{i=r+1}^{k-r} l_i(q - p).$$ (7.2)

In the complex domain, each quadratic factor s_i also splits into two linear forms l_i and $\bar{l}_i$, now with complex coefficients. Each linear factor defines a tangent line to $Z(f)$ at p. Each quadratic factor determines a pair of conjugate "imaginary" tangent lines, two tangent lines in the complex domain which cannot be observed in the real domain, except for increasing the multiplicity of p. Note that both the quadratic and the linear factors can be repeated, in which case we can talk about multiplicity of tangent lines. If $l(q - p)$ is a linear form, either with real or complex coefficients, and not necessarily one of the l_i above or one of the complex factors of the s_i, its multiplicity at p is defined as the exponent of $l(q - p)$ in the product (7.2), and it is denoted $m_p(l; f)$. Tangent lines, either real or imaginary, correspond to forms with positive multiplicity. Forms with $m_p(l; f) = 0$ are called "transversal" to $Z(f)$ at p. Look at Figure 12 for an illustration.

7.2 The Approximate Distance Near Multiple Points

Now we can go back to analyzing the behavior of our algorithms in the vicinity of singular points. Since there are many cases to consider, instead of enunciating and proving a detailed proposition, we will show and analyze typical cases.

Figure 13 shows examples of implicit curves, algebraic in all the cases, with isolated singular points. In (a), (b), (c), (f), (g), (h), and (i), the origin $p = (0, 0)$ is the only singular point. It is a double point, i.e., $m(p; f) = 2$, in (a), (b), (c), and (i); a quadruple point in (f), and a sextuple point in (g) and (h). In (d) there are four double points, all intersections of pairs of lines. And in (e) there are four double points, two triple points, and two quadruple points. In (i), the origin is isolated as an element of $Z(f)$, but still a double point. In (a) and (b) the origin has two simple tangents. In (c) the origin has one double tangent, a "cusp." In (d) the four singular points have two simple tangents each, which are contained in the the curve itself. In (e) the two double points and the two triple points have simple tangents, but the two quadruple points have one
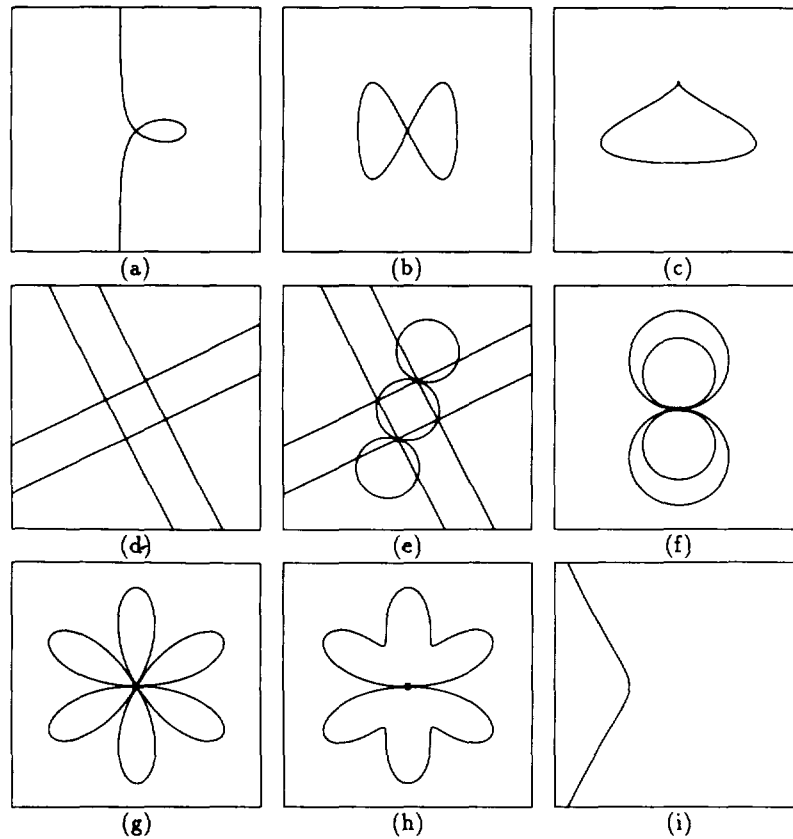
Fig. 13. Examples of algebraic curves with singular points rendered in a grid of 512 × 512 pixels with the algorithm **RecursivePaintZeros4NN**. Almost identical results are obtained using the second-order approximate-distance test instead of the Euclidean distance test in the algorithm **RecursivePaintZeros**.

double and two simple tangents each. The origin has a quadruple tangent in (f) and three double tangents in (g). In (h) the origin has one double and four imaginary tangents, and in (i) the origin is an isolated point of $Z(f)$; and so, the two tangents are imaginary.

As we said before, these are typical cases. The algorithm **Recursive-PaintZero4NN** was used to render them, but the behavior near singular points is the same for the other algorithms as well. We can see in the pictures that multiple points with simple real tangents are rendered correctly, as well as some of the cases of multiple tangents. However, where the multiplicity of the tangents increase, or when imaginary tangents are also present, as in (h) the approximate distance seems to underestimate the exact distance producing thicker lines.

If p is a point of multiplicity k of f, according to Equation (7.1), the behavior of the function f in the neighborhood of p is governed by the form
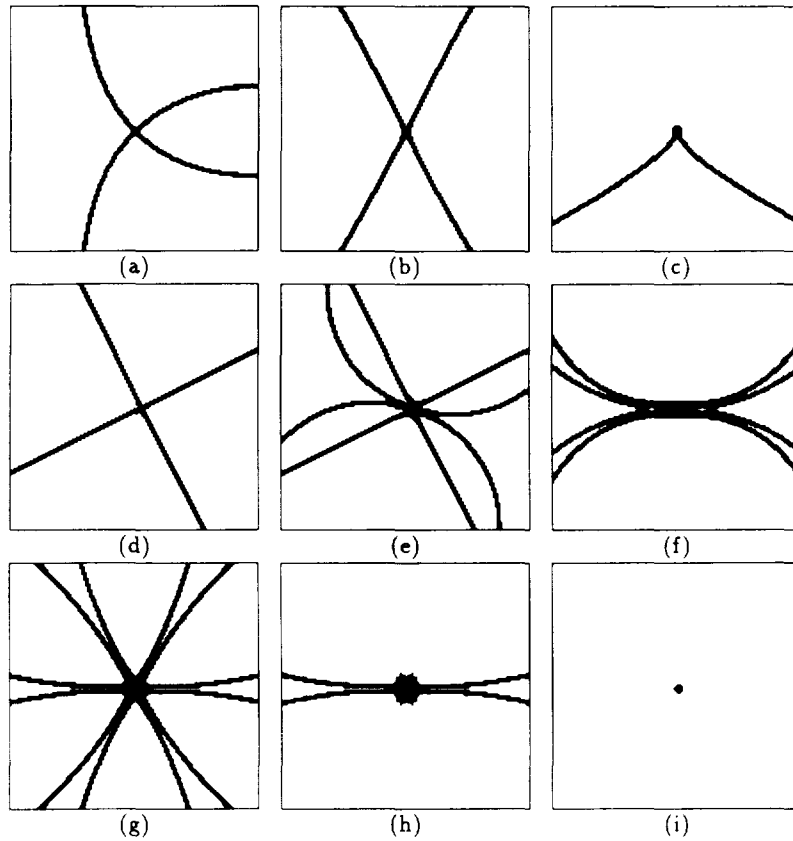
Fig. 14. Closeups of curves in Figure 13 near singular points. Neighborhoods of 128×128 pixels.

f_p^k. With a reasoning similar to the proof of Lemma 1, we can show that

$$\delta_1(q, Z(f)) = \frac{|f(q)|}{\|\nabla f(q)\|} \approx \frac{|f_p^k(q)|}{\|\nabla f_p^k(q)\|} + O(\|q - p\|^2)$$

for every point q such that $f_p^k(q) \neq 0$. But from Euler's theorem [Walker 1950], since f_p^k is a form of degree k, we have

$$f_p^k(q) = \frac{1}{k} \nabla f_p^k(q)^T \cdot (q - p),$$

and applying the Cauchy–Schwarz inequality, we obtain

$$\frac{|f_p^k(q)|}{\|\nabla f_p^k(q)\|} \leq \frac{1}{k}\|q - p\|.$$

This inequality explains the behavior of the first-order approximate distance near an isolated singular point, because in that case $\|q - p\|$ eventually
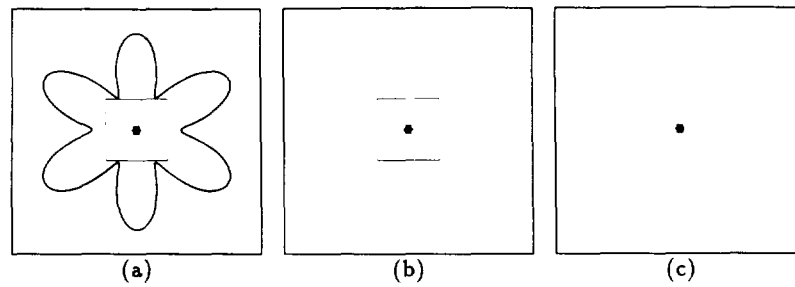
Fig. 15. An isolated sextuple point with three imaginary double tangents. The regions inside the squares in (a) and (b) are rendered at four times the resolution in (b) and (c), respectively.
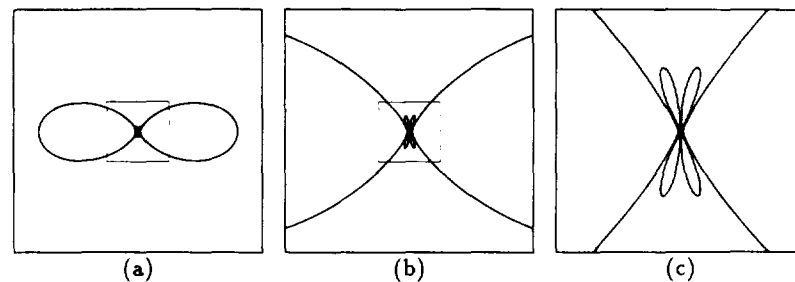

Fig. 16. Zooming up on a part of the curve that falls below the resolution of the picture. The regions inside the squares in (a) and (b) are rendered at four times the resolution in (b) and (c), respectively.

becomes equal to the exact distance from q to $Z(f)$. It also explains the behavior of the higher-order approximate distances near an isolated singular point, because $0 \leq \delta_h \leq \delta_1$ for every value of h. Figure 15 shows an example of isolated multiple point, sextuple in this case, where the neighborhood of the singular point is rendered at three different resolutions.

In Figure 13h we have a related case. The singular point is not isolated here, but the curve still shows similar behavior near to the origin. In Figure 16 we can see an example of a curve with a singular point at the origin, which seems to be showing the same behavior at low resolution, but we discover that it is not the case when we increase the resolution near the singularity.

Although we have explained the behavior of the first-order approximate distance near an isolated singular point, no general result can be established about when it overestimates or underestimates the exact distance near an arbitrary singular point. For the simplest example, let us consider the polynomial $f(x, y) = y^2 - \lambda^2 x^2$, for certain constant λ. The set $Z(f)$ is the union of two lines. Figure 17 shows this curve for three different values of λ.

A simple algebraic computation shows that the relation between the first-order approximate distance and the exact distance from a point $p = (0, y)$ on
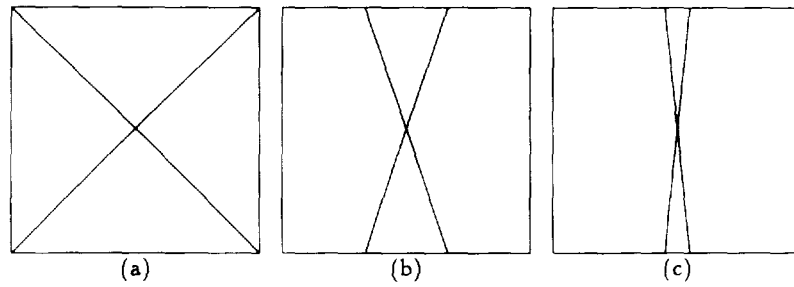
Fig. 17.   The set $Z(y^2 - a^2x^2)$ for three different values of a: (a) $\lambda = 1$, (b) $\lambda = 3$, (c) $\lambda = 10$.
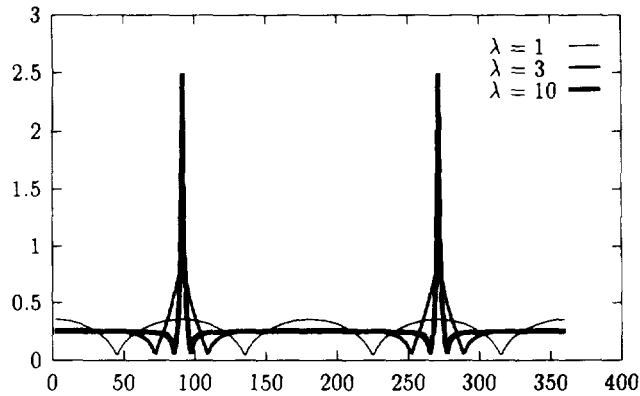

Fig. 18.   Plot of the ratio of the approximate distance over the exact distance from a point $p = (\cos(t), \sin(t))$ to the curve $Z(y^2 - a^2x^2)$ for three different values of λ.

the vertical axis is given by the following expression

$$\frac{\delta_1(p, Z(f))}{\delta(p, Z(f))} = \frac{\sqrt{1 + \lambda^2}}{2}.$$

Depending on the value of the constant a, this ratio goes from $1/2$ to ∞. It is important to note though, that even when the value of a is large, the region where the ratio is larger than 1 is limited to a small neighborhood of the vertical axis, as can be seen in Figure 18, which shows a plot of the ratio as a function of the angular parameter t, for $p = (\cos(t), \sin(t))$, and three different values of the parameter λ (since the ratio is a homogeneous function of zeroeth order, for a general point $p = (x, y)$ it only depends on the angle $t = \arctan(y/x)$).

8. EXPERIMENTS

We have seen that replacing the Euclidean distance test with the first-order approximate-distance test in the algorithm **RecursivePaintZeros** does not
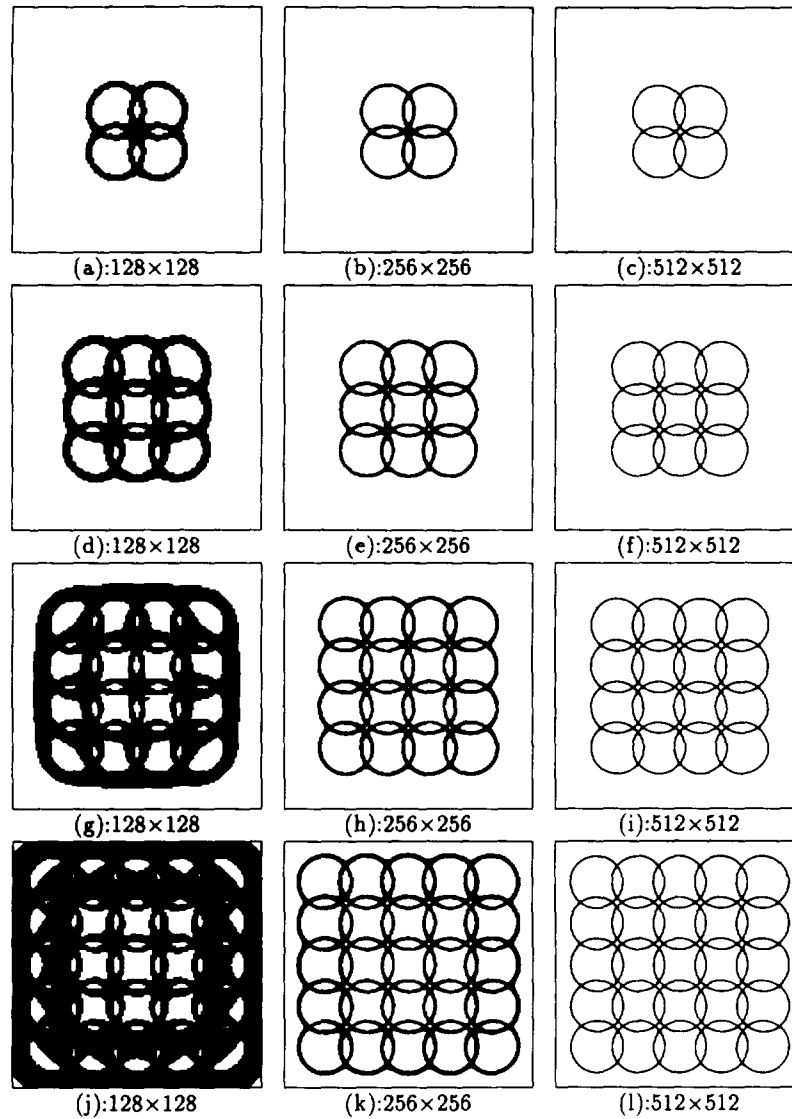
Fig. 19.   Examples of algebraic curves of degree 8 with 45 coefficients (top row), degree 18 with 190 coefficients (second row), degree 32 with 561 coefficients (third row), and degree 50 with 1326 coefficients (bottom row), correctly rendered with the algorithm **RecursivePaintZeros**, and the second-order approximate-distance test instead of the Euclidean distance test, at three different resolutions.

produce correct algorithms, not even using the heuristics introduced above. Although for algebraic curves the algorithm based on the approximate-distance test of the same order as the degree of the polynomial is correct, it is very often too expensive to evaluate. The algorithm based on the second-order approximate-distance test, maybe with the addition of the heuristics de-
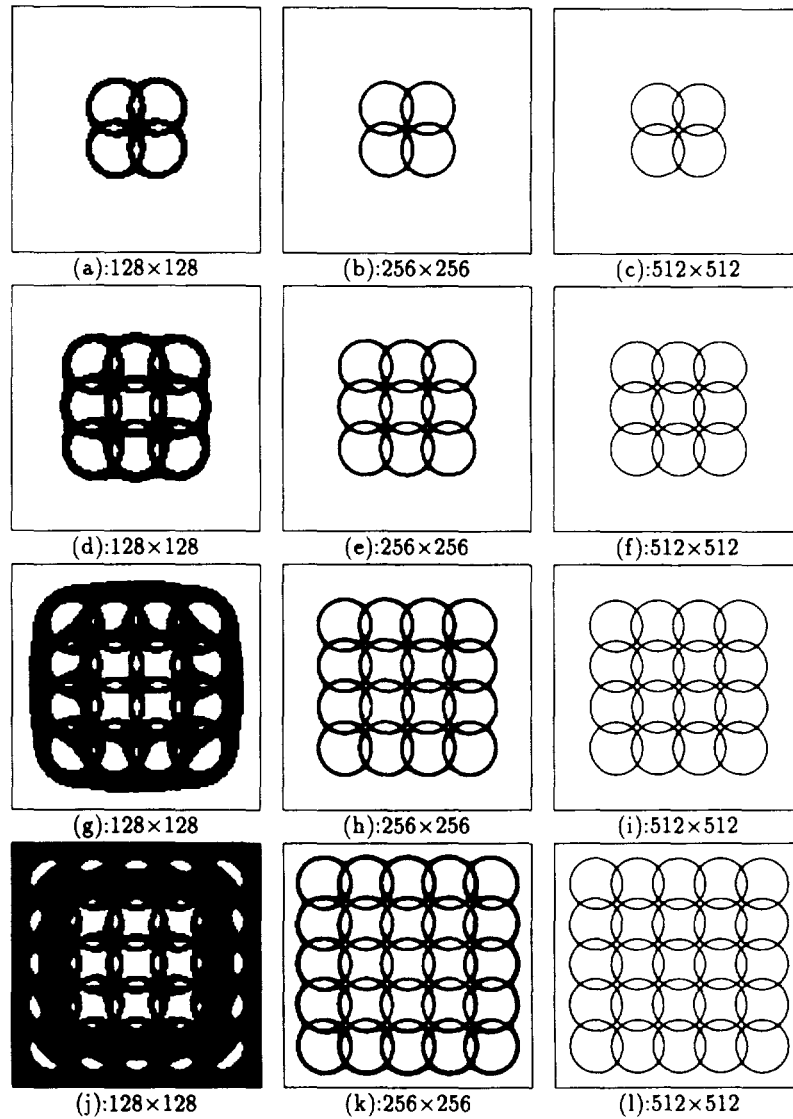
Fig. 20. The same curves of Figure 19 correctly rendered with the algorithm **Recursive-PaintZeros**, and the approximate-distance test of the same order as the degree of the polynomial instead of the Euclidean distance test.

scribed above, has been able to render correctly all the examples of algebraic curves presented in the article, and with significantly less computation. We even know when such an algorithm fails. It rejects a region centered at a point p if the function value is nonzero at the point, but all the first and second derivatives vanish, or are very close to zero, at the point. In general this is very unlikely to happen, but in any case it can be tested; and a

		128 × 128	256 × 256	512 × 512	1024 × 1024
Degree	8	1.5	2.5	4.6	9.0
Degree	18	13.6	15.5	41.3	69.0
Degree	32	38.6	135.7	225.8	366.6
Degree	50	81.5	345.2	757.1	1,294.0

Fig. 21.   Execution times for the pictures in Figure 19.

		128 × 128	256 × 256	512 × 512	1024 × 1024
Degree	8	2.7	4.5	7.6	14.1
Degree	18	41.4	86.4	137.4	221.5
Degree	32	83.8	746.7	1,269.6	2,010.2
Degree	50	165.0	2,303.5	6,234.6	10,503.7

Fig. 22.   Execution times for the pictures in Figure 20.

higher-order approximate distance be used instead. The algorithm based on the third-order approximate-distance test will solve the problem for most practical cases. Depending on the application, either correctness is the most serious concern or speed. If correctness is more important, then algebraic curves must be rendered using the test of the same order as the degree of the polynomial. Otherwise, we recommend to use the algorithm based on the second or third approximate distance and either (or both) heuristics.

To push these algorithms to their limits we will first compare execution times for curves of different degrees. Figures 19 and 20 show four different curves of degree 8 (45 coefficients), 18 (190 coefficients), 32 (561 coefficients), and 50 (1326 coefficients) composed of different numbers of circumferences of the same radius located in a regular fashion. Figure 19 shows these four curves rendered at three different resolutions with the algorithm based on the second-order approximate-distance test, and no heuristic, while Figure 20 shows the same curves rendered with the correct algorithm based on the approximate-distance of the same degree as the polynomial, at the same three different resolutions. No major difference can be observed at the highest resolution, with the correct algorithm being more conservative at lower resolutions as expected.

The execution times are very different though. Figures 21 and 22 show the timing results for Figures 19 and 20 respectively, and at an extra higher resolution. The figures are in seconds and a decimal fraction of a second. These timing results correspond to actual running times in a lightly loaded IBM RS/6000 model 930 workstation, including I/O time and system overhead. The improvement in running time is more obvious at higher degrees. In general, we can observe that running times initially grow quadratically with the resolution, but at a certain point they start to grow linearly, as expected. So that, doubling the resolution, approximately doubles the running time.

Running times for reasonably low-degree curves are not as high as those quoted for these high-degree examples. Figures 23 and 24 show the corre-

	128 × 128	256 × 256	512 × 512	1024 × 1024
(a)	0.2	0.4	1.1	3.6
(b)	0.4	0.7	1.6	4.0
(c)	0.3	0.6	1.3	3.5
(d)	0.6	1.3	2.7	5.8
(e)	6.4	13.1	24.9	46.5
(f)	3.0	5.6	10.3	20.0
(g)	3.1	6.0	11.5	22.6
(h)	2.8	4.6	7.8	14.3
(i)	0.2	0.3	0.9	2.9

Fig. 23. Execution times for the curves in Figure 13 at four different sizes, using the approximate distance of the same order as the polynomial in algorithm **RecursivePaintZeros** instead of the Euclidean distance.

	128 × 128	256 × 256	512 × 512	1024 × 1024
(a)	0.2	0.4	1.0	3.5
(b)	0.3	0.6	1.4	3.7
(c)	0.3	0.5	1.2	3.2
(d)	0.5	1.1	2.3	5.1
(e)	3.3	6.6	12.5	23.9
(f)	1.7	3.2	6.1	12.7
(g)	1.8	3.5	6.9	14.0
(h)	1.5	2.5	4.6	9.2
(i)	0.1	0.3	0.9	2.9

Fig. 24. Execution times for the curves in Figure 13 at four different sizes, using the approximate distance of order two in algorithm **RecursivePaintZeros** instead of the Euclidean distance.

sponding timing results for the nine curves of Figure 13, rendered again with the algorithm based on the second-order approximate distance and the approximate distance of the same order as the polynomial. Since the addition of any one of the two heuristics to the algorithm based on the second-order approximate-distance test do not significantly increase the running times, we have decided not to show the timings for these cases. In general, for low degree the rate is already appropriate for interactive devices and clearly practical for noninteractive rendering devices such as laser printers.

9. EXTENSIONS AND APPLICATIONS

In this section we briefly discuss several extensions and applications of the algorithms introduced in this article. These extensions and applications are the subject of our current research.

9.1 Union of Planar Curves

Let us consider rendering the union of two implicit curves $Z(h)$ and $Z(g)$, where $h(x, y)$ and $g(x, y)$ are two functions of two variables with at least

continuous first-order partial derivatives, and let $f = g \cdot h$. Since

$$f(x, y) = 0 \Leftrightarrow g(x, y) = 0 \quad \text{or} \quad h(x, y) = 0,$$

we have $Z(f) = Z(gh) = Z(g) \cup Z(h)$. We can clearly use the algorithms discussed above to render $Z(f)$, but if the two factors g and h are available, or can be easily computed, this does not make sense from the computational point of view. There are two computationally effective approaches to this problem. The first one is to render $Z(g)$ and $Z(h)$ independently, and in two different memory arrays, and then take the logical "or" of the two two-dimensional arrays as the rasterized version of $Z(f)$. This approach clearly minimizes the evaluation time, but maximizes memory usage. The second approach is based on the following fact

$$\delta(p, Z(g) \cup Z(h)) = \min\{\delta(p, Z(g)), \delta(p, Z(h))\}.$$

Only one array will be used, and in principle the two distances must be evaluated to decide whether to discard a square region or not. However, if information about which one of the two distances was small for the father is kept, and the same distance is evaluated first for the four children of a region, most of the painted pixels will eventually require only one distance evaluation, and all the rejected pixels will require two. In the case of polynomials, we can do even better. If g is a polynomial of degree k, and the approximate distance of order k is evaluated at a point p during the execution of the algorithm, no pixel inside the circle of radius δ_k minus the desired line width centered at p will be part of the rasterized version of $Z(g)$, and only the approximate distance to $Z(h)$ needs to be evaluated inside this region.

Clearly, the union of more than two curves can be handled in a similar way.

9.2 Intersection of Planar Curves

Now, let us consider rendering the intersection of two implicit curves $Z(h)$ and $Z(g)$, where $h(x, y)$ and $g(x, y)$ are again two functions of two variables with at least continuous first-order partial derivatives. As in the previous case, we can use the algorithms discussed above to render the curves $Z(g)$ and $Z(h)$ independently, and in two different memory arrays, and then take the logical "and" of the two two-dimensional arrays as the rasterized version of $Z(g) \cap Z(h)$. This approach clearly maximizes both evaluation time and memory usage. A better approach can be based on the following inequality

$$\delta(p, Z(g) \cap Z(h)) \geq \max\{\delta(p, Z(g)), \delta(p, Z(h))\}.$$

That is, if one of the two distances is large, the distance to the intersection is large. The idea is to measure the two distances and discard the region only if one of them is larger than the threshold. In the algebraic case, and with the same bookkeeping discussed in the previous section, most of the discarded regions will require only one distance evaluation, and all the accepted regions
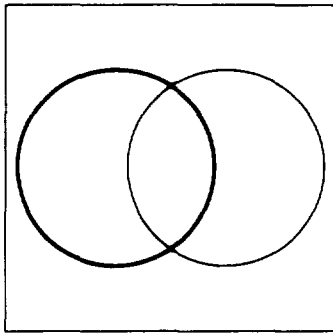
Fig. 25. Example of curve with a multiple component $(f(x, y) = ((x + 1)^2 + y^2 - 5)^2((x - 1)^2 + y^2 - 5))$, which can be identified by the different thickness.

will require two. But since in general two curves intersect in a finite set of points (they can have a common component as well), this approach is clearly the most cost effective. If the intersection of the two curves is a finite set of points, the rasterized version will be formed by a few clusters of neighboring pixels. If the resolution is high enough, each of these clusters will correspond to one intersection point in the region. Then, if the locations of the intersection points are required at higher resolution, a numerical root-finding algorithm can be used to improve the estimate. However, the main problem which remains to be solved is how to determine if a cluster corresponds to a single intersection point or not. In the algebraic case we could use symbolic methods to count the number of zeros inside a box [Milne 1990; Pedersen 1991a], and if the number is larger than one, keep refining until each cluster corresponds to a single intersection point. However, as we mention above, these methods are only practical for very low-degree polynomials and can also be unstable. We will leave this subject for further research. And clearly, as in the case of union of curves, the intersection of more than two curves can be handled in a similar way.

9.3 Approximation of Singular Points

As we discussed in Section 7, the algorithms introduced above tend to render the neighborhoods of singular points at larger width. In certain applications, as for example when the user wants to locate the singular points by looking at the display, this can be seen as a feature. Also, multiple components of algebraic curves can be identified by looking at the rasterized representation produced by these algorithms, as it is illustrated in Figure 25. Even the relative thickness of the lines with respect to the regular parts can be used to determine the multiplicity.

However, in other applications this property of the approximate distances could be a disadvantage. In those cases it is necessary to locate the singular regions and solve the problem. Since the singular points of the curve $Z(f)$ are the set of points q which satisfy $f(q) = f_x(q) = f_y(q) = 0$, a possible solution is to rasterize the intersection of the three curves $Z(f) \cap Z(f_x) \cap Z(f_y)$, locate clusters which correspond to one or more singular points, continue
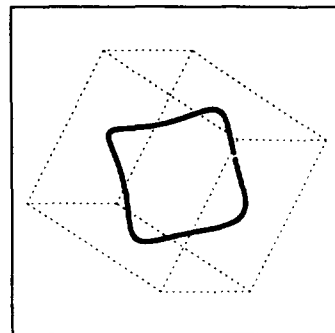
Fig. 26.   Surface-surface intersection rendered with the algorithms described in the text based on the first-order approximate distance.

with the recursive space subdivision scheme only in those regions for one or two more steps at the subpixel level, and then paint only those pixels which contain at least one of these high-resolution subpixels.

9.4 Rendering Surface-Surface Intersections

Our next task is to generalize the algorithms described above to higher-dimensional cases. That is, we want to extend the previous methods to render two-dimensional projections of higher-dimensional implicit curves. This is particularly interesting in the CAD arena, where nonplanar three-dimensional curves, intersection of two implicit surfaces, have to be rendered.

The approach here is to work in the original space, where the curve lives, and apply the projection at rendering time. In this way the user could also interact with the system to choose the viewpoint, i.e., once the points are computed, different views of the curve can be rendered in linear time (proportional to the number of points in the approximating set). Figure 26 shows an example of a three-dimensional curve rendered with an extension of the algorithms described above to the three-dimensional case, and the extension of the first-order approximate distance described below. As in the planar case, some missing points can be observed in this example as well.

Both the space subdivision scheme, and the 4NN dilation procedure have clear extensions to higher-dimensional spaces. For example, in three-dimensional space, every voxel is divided into eight lower-resolution voxels, and each of them has six nearest neighbors. In order to properly extend the algorithms to higher dimensions, we just have to show how to measure the approximate distances from a point to a higher-dimensional implicit set. In Taubin [1988; 1991], we have shown how to extend the first-order approximate distance to the higher-dimensional case. If $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^k$ is a vector-valued function with continuous first-order derivatives, and the Jacobian $D\mathbf{f}(p)$ of $\mathbf{f}$ at $p \in \mathbb{R}^n$ has maximum rank k, then the following expression is a first-order approximation of the Euclidean distance from p to $Z(\mathbf{f})$

$$\delta_1(p, Z(\mathbf{f}))^2 = \mathbf{f}(p)^T \left(D\mathbf{f}(p) D\mathbf{f}(p)^T \right)^{-1} \mathbf{f}(p).$$

The proper extension of the higher-order approximate distances to the higher-dimensional case requires a deeper analysis, and we will also leave it for further research.

9.5 Parallel Algorithms

Except for **RecursivePaintZeros4NN**, all the algorithms described in this article are ideal for parallel implementation due to the nonexistent communication between different pixels. Clearly, if as many processors as pixels are available, a parallel version of the ideal rendering algorithm of Figure 1, with the Euclidean distance test replaced by the higher-order approximate-distance test, can be trivially implemented. With this algorithm, an algebraic curve defined by a polynomial of degree k can be rendered in $O(1)$ time, the time required to evaluate the polynomial and all its partial derivatives at a point using Horner' algorithm, which itself can be parallelized [Dowling 1990]. If not so many processors are available, we just need to subdivide the original region in as many processors are available and assign each subregion to a different processor for rendering. If the side of the original square is a power of two, the most symmetric way to subdivide it is using a power of four number of processors.

APPENDIX

Equations

In this appendix we list all the polynomials used for creating the figures and the specifications for the bounding boxes:

Figures 3, 5, 8, and 11, (a) to (f). Box centered at (0.0, 0.5), side 5.0.

$$f(x, y) = 0.004 + 0.110x - 0.177y - 0.174x^2 + 0.224xy - 0.303y^2$$
$$- 0.168x^3 + 0.327x^2y - 0.087xy^2 - 0.013y^3 + 0.235x^4$$
$$- 0.667x^3y + 0.745x^2y^2 - 0.029xy^3 + 0.072y^4.$$

Figures 3, 5, 8, and 11, (g) to (l). Box centered at (0.0, 0.6), side 4.5.

$$f(x, y) = -0.139 - 0.179x - 1.798y + 0.482x^2 - 0.399xy - 6.367y^2$$
$$+ 0.084x^3 + 13.770x^2y + 0.314xy^2 - 8.279y^3 + 0.910x^4$$
$$- 0.762x^3y + 41.679x^2y^2 - 0.533xy^3 - 1.938y^4 + 2.297x^5$$
$$- 32.303x^4y - 0.658x^3y^2 + 34.719x^2y^3 - 3.995xy^4 + 3.147y^5$$
$$- 4.106x^6 + 10.536x^5y - 92.138x^4y^2 + 12.892x^3y^3$$
$$- 0.440x^2y^4 - 4.514xy^5 + 0.898y^6 - 2.588x^7 + 22.109x^6y$$
$$+ 1.609x^5y^2 - 36.923x^4y^3 + 8.600x^3y^4 - 7.411x^2y^5$$
$$+ 0.251xy^6 - 0.542y^7 + 3.079x^8 - 13.236x^7y + 71.933x^6y^2$$
$$- 33.078x^5y^3 + 8.256x^4y^4 + 8.130x^3y^5 + 0.418x^2y^6$$
$$+ 0.975xy^7 + 0.037y^8$$

Figure 13(a). Box centered at $(0.0, 0.0)$, side 5.0.

$$f(x, y) = x^3 + 3x^2y - x^2 + y^2$$

Figure 13(b). Box centered at $(0.0, 0.0)$, side 5.0.

$$f(x, y) = 4x^4 - 4x^2 + y^2$$

Figure 13(c). Box centered at $(0.0, -1.0)$, side 5.0.

$$f(x, y) = 3y^4 - 5y^3 + x^2$$

Figure 13(d). Box centered at $(0.0, 0.0)$, side 5.0.

$$f(x, y) = (2y - x - 1)(2y - x + 1)(2x + y + 1)(2x + y - 1)$$

Figure 13(e). Box centered at $(0.0, 0.0)$, side 5.0.

$$f(x, y) = (2y - x - 1)(2y - x + 1)(2x + y + 1)(2x + y - 1)$$
$$\left((5x - 2)^2 + (5y - 6)^2 - 10\right)\left((5x + 2)^2 + (5y + 6)^2 - 10\right)$$
$$(25(x^2 + y^2) - 10)$$

Figure 13(f). Box centered at $(0.0, 0.0)$, side 6.0.

$$f(x, y) = \left((x + 1)^2 + y^2 - 1\right)\left((x - 1)^2 + y^2 - 1\right)$$
$$\left((x + 1.1)^2 + y^2 - 1.21\right)\left((x - 1.1)^2 + y^2 - 1.21\right)$$

Figure 13(g). Box centered at $(0.0, 0.0)$, side 2.5.

$$f(x, y) = (3x^2 - y^2)^2 y^2 - (x^2 + y^2)^4$$

Figure 13(h). Box centered at $(0.0, 0.0)$, side 2.5.

$$f(x, y) = (8x^4 - 4x^2y^2 + y^4)y^2 - (x^2 + y^2)^4$$

Figure 13(i). Box centered at $(0.0, 0.0)$, side 5.0.

$$f(x, y) = x^2 + y^2 + y^3$$

Figure 14, (a) to (i), are just regions of Figure 13, (a) to (i), rendered at higher magnification.

Figure 15, (a), (b), and (c) share the same polynomial. The three boxes are centered at $(0.0, 0.0)$, and the sides are 2.5, 0.625, and 0.1562.

$$f(x, y) = +0.2x^6 + 9.0x^4y^2 - 5.0x^2y^4 + 1.0y^6 - (x^2 + y^2)^4$$

Figure 16, (a), (b), and (c) share the same polynomial. The three boxes are centered at $(0.0, 0.0)$, and the sides are 10.0, 2.5, and 0.625.

$$f(x, y) = x^2(4x^2 - y^2)^2 - (x^2 + y^2)^4$$

Figure 17 is explained in the corresponding caption.

Figures 19 and 20, (a) to (l), share the same center (0.0, 0.0), and side 6.0.
Figures 19 and 20, (a) to (c).

$$f(x, y) = \left((x - 0.5)^2 + (y + 0.5)^2 - 0.4\right)\left((x + 0.5)^2 + (y + 0.5)^2 - 0.4\right)$$
$$\left((x - 0.5)^2 + (y - 0.5)^2 - 0.4\right)\left((x + 0.5)^2 + (y - 0.5)^2 - 0.4\right)$$

Figures 19 and 20, (d) to (f).

$$f(x, y) = \left((x + 1)^2 + (y + 1)^2 - 0.4\right)\left((x + 1)^2 + y^2 - 0.4\right)$$
$$\left((x + 1)^2 + (y - 1)^2 - 0.4\right)\left(x^2 + (y + 1)^2 - 0.4\right)$$
$$\left(x^2 + y^2 - 0.4\right)\left(x^2 + (y - 1)^2 - 0.4\right)\left((x - 1)^2 + (y + 1)^2 - 0.4\right)$$
$$\left((x - 1)^2 + y^2 - 0.4\right)\left((x - 1)^2 + (y - 1)^2 - 0.4\right)$$

Figures 19 and 20, (g) to (i).

$$f(x, y) = \left((x - 1.5)^2 + (y + 1.5)^2 - 0.4\right)\left((x - 0.5)^2 + (y + 1.5)^2 - 0.4\right)$$
$$\left((x + 0.5)^2 + (y + 1.5)^2 - 0.4\right)\left((x + 1.5)^2 + (y + 1.5)^2 - 0.4\right)$$
$$\left((x - 1.5)^2 + (y + 0.5)^2 - 0.4\right)\left((x - 0.5)^2 + (y + 0.5)^2 - 0.4\right)$$
$$\left((x + 0.5)^2 + (y + 0.5)^2 - 0.4\right)\left((x + 1.5)^2 + (y + 0.5)^2 - 0.4\right)$$
$$\left((x - 1.5)^2 + (y - 0.5)^2 - 0.4\right)\left((x - 0.5)^2 + (y - 0.5)^2 - 0.4\right)$$
$$\left((x + 0.5)^2 + (y - 0.5)^2 - 0.4\right)\left((x + 1.5)^2 + (y - 0.5)^2 - 0.4\right)$$
$$\left((x - 1.5)^2 + (y - 1.5)^2 - 0.4\right)\left((x - 0.5)^2 + (y - 1.5)^2 - 0.4\right)$$
$$\left((x + 0.5)^2 + (y - 1.5)^2 - 0.4\right)\left((x + 1.5)^2 + (y - 1.5)^2 - 0.4\right)$$

Figures 19 and 20, (j) to (l).

$$f(x, y) = \left((x + 2)^2 + (y + 2)^2 - 0.4\right)\left((x + 2)^2 + (y + 1)^2 - 0.4\right)$$
$$\left((x + 2)^2 + y^2 - 0.4\right)\left((x + 2)^2 + (y - 1)^2 - 0.4\right)$$
$$\left((x + 2)^2 + (y - 2)^2 - 0.4\right)\left((x + 1)^2 + (y + 2)^2 - 0.4\right)$$
$$\left((x + 1)^2 + (y + 1)^2 - 0.4\right)\left((x + 1)^2 + y^2 - 0.4\right)$$
$$\left((x + 1)^2 + (y - 1)^2 - 0.4\right)\left((x + 1)^2 + (y - 2)^2 - 0.4\right)$$
$$\left(x^2 + (y + 2)^2 - 0.4\right)\left(x^2 + (y + 1)^2 - 0.4\right)$$
$$\left(x^2 + y^2 - 0.4\right)\left(x^2 + (y - 1)^2 - 0.4\right)$$
$$\left(x^2 + (y - 2)^2 - 0.4\right)\left((x - 1)^2 + (y + 2)^2 - 0.4\right)$$
$$\left((x - 1)^2 + (y + 1)^2 - 0.4\right)\left((x - 1)^2 + y^2 - 0.4\right)$$

$$\left((x - 1)^2 + (y - 1)^2 - 0.4\right)\left((x - 1)^2 + (y - 2)^2 - 0.4\right)$$

$$\left((x - 2)^2 + (y + 2)^2 - 0.4\right)\left((x - 2)^2 + (y + 1)^2 - 0.4\right)$$

$$\left((x - 2)^2 + y^2 - 0.4\right)\left((x - 2)^2 + (y - 1)^2 - 0.4\right)$$

$$\left((x - 2)^2 + (y - 2)^2 - 0.4\right)$$

Figure 25. Box centered at $(0.0, 0.0)$, side 8.0.

$$f(x, y) = \left((x + 1)^2 + y^2 - 5\right)^2\left((x - 1)^2 + y^2 - 5\right)$$

REFERENCES

ABHYANKAR, S. S. AND BAJAJ, C. 1988. Automatic parameterization of rational curves and surfaces IV: Algebraic space curves. Tech. Rep. CSD-TR-703, Purdue Univ., Computer Sciences Dept., West Lafayette, Ind.

ABHYANKAR, S. S. AND BAJAJ, C. 1987a. Automatic parameterization of rational curves and surfaces I: Conics and Conicoids. *Comput. Aided Des. 19*, 1 (Jan./Feb.), 11-14.

ABHYANKAR, S. S. AND BAJAJ, C. 1987b. Automatic parameterization of rational curves and surfaces II: Cubics and Cubicoids. *Comput. Aided Des. 19*, 9 (Nov.), 499-502.

ABHYANKAR, S. S. AND BAJAJ, C. 1987c. Automatic parameterixation of rational curves and surfaces III: Algebraic plane curves. Tech. Rep. CSD-TR-619, Purdue Univ., Computer Sciences Dept., West Lafayette, Ind.

ALLGOWER, E. AND GEORG, K. 1980. Simplicial and continuation methods for approximating fixed points and solutions to systems of equations. *SIAM Rev. 22*, 1 (Jan.), 28-85.

ALLGOWER, E. L. AND SCHMIDT, P. H. 1985. An algorithm for piecewise-linear approximation of an implicitly defined manifold. *SIAM J. Num. Anal. 22*, 2 (Apr.), 322-346.

ARNON, D. S. 1983. Topologically reliable display of algebraic curves. *Comput. Graph. 17*, 3 (July), 219-227.

BAJAJ, C. L., HOFFMANN, C. M., HOPCROFT, J. E., AND LYNCH, R. E. 1987. Tracing surface intersections. Tech. Rep. CSD-TR-728, Dept. of Computer Science, Purdue Univ., West Lafayette, Ind.

BAKER KEARFOTT, R. 1987. Some tests of generalized bisection. *ACM Trans. Math. Softw. 13*, 3 (Sept.), 197-220.

BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Comput. Aided Geo. Des. 5*, 4 (Nov.), 341-355.

BOCHNAK, J., COSTE, M., AND ROY, M.-F. 1987. Gémétrie algébrique réelle. In *Ergebnisse der Mathematik un ihrer Grenzgebiete*. Vol. 12. Springer-Verlag, Berlin.

BORODIN, A. AND MUNRO, I. 1975. *The Computational Complexity of Algebraic and Numeric Problems*. American Elsevier, New York.

CANNY, J. F. 1988. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, Mass.

DE MOUNTAUDOUIN, Y. 1991. Résolution of $p(x, y) = 0$. *Comput. Aided Des. 23*, 9 (Nov.), 653-654.

DENNIS, J. E. AND SHNABEL, R. B.   1983.   *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, N.J.

DOBKIN, D. P., LEVY, S. V. F., THURSTON, W. P., AND WILKS, A. R.   1990.   Contour tracing by piecewise linear approximation. *ACM Trans. Graph. 9*, 4 (Oct.), 389–423.

DOWLING, M. L.   1990.   A fast parallel horner algorithm. *SIAM J. Comput. 19*, 1 (Feb.), 133–142.

EIGER, A., SIKORSKI, K., AND STENGER, F.   1984.   A bisection method for systems of nonlinear equations. *ACM Trans. Math. Softw. 10*, 4 (Dec.), 367–377.

GEISOW, A.   1983.   Surface interrogations. Ph.D. thesis, Univ. of East Anglia, School of Computing Studies, U.K.

HOBBY, J. D.   1990.   Rasterization of nonparametric curves. *ACM Trans. Graph. 9*, 3 (July), 162–277.

JORDAN, B. W., JR., LENNON, W. J., AND HOLM, B. D.   1973.   An improved algorithm for the generation of nonparametric curves. *IEEE Trans. Comput. C-22*, 12 (Dec.), 1052–1060.

KRIEGMAN, D. J. AND PONCE, J.   1990.   On recognizing and positioning curved 3D objects from image contours. *IEEE Trans. Patt. Anal. Mach. Intell. 12*, 12 (Dec.), 1127–1137.

MANOCHA, D.   1992.   Algebraic and numeric techniques in modeling and robotics. Ph.D. thesis, Univ. of California at Berkeley, Berkeley, Calif.

MANOCHA, D. AND CANNY, J. F.   1992.   Multipolynomial resultant algorithms. Tech. Rep., Computer Science Div., Univ. of California at Berkeley, Berkeley, Calif.

MILNE, P.   1990.   On the solution of a set of polynomial equations. Tech. Rep., Bath Univ., Bath, U.K.

MORÉ, J. J., GARBOW, B. S., AND HILLSTROM, K. E.   1980.   User guide for minpack-1. Tech. Rep. ANL-80-74, Argone National Laboratories.

MORGAN, A. AND SHAPIRO, V.   1987.   Box-bisection for solving second-degree systems and the problem of clustering. *ACM Trans. Math. Softw. 13*, 2 (June), 152–167.

OVERMARS, M. H.   1988.   Geometric data structures for computer graphics: an overview. In *Theoretical Foundations of Computer Graphics and CAD.* NATO ASI Series, vol. F40, Springer-Verlag, 21–49.

PEDERSEN, P.   1991a.   Counting real zeros. Ph.D. thesis, New York Univ., New York.

PEDERSEN, P.   1991b.   Mutlivariate Sturm theory. In *Proceedings of the 9th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes.* Lecture Notes in Computer Science, vol. 539. Springer-Verlag, Berlin, 318–332.

PONCE, A., HOOGS, J., AND KRIEGMAN, D.   1992.   On using CAD models to compute the pose of curved 3D objects. *Comput. Vis. Graph. Image Proc. 55*, 2, 184–197.

PRATT, V.   1987.   Direct least squares fitting of algebraic surfaces. *Comput. Graph. 21*, 4 (July), 145–152.

SAMET, H.   1988.   An overview of quadtrees, octrees, and related hierachical data structures. In *Theoretical Foundations of Computer Graphics and CAD.* NATO ASI Series, vol. F40. Springer-Verlag, Berlin, 51–68.

SAMPSON, P. D.   1982.   Fitting conic sections to very scattered data: An iterative refinement of the bookstein algorithm. *Comput. Vis. Graph. Image Proc. 18*, 97–108.

SEDERBERG, T. W., ANDERSON, D. C., AND GOLDMAN, R. N.   1984.   Implicit representation of parametric curves and surfaces. *Comput. Vis. Graph. Image Proc. 28*, 1, 72–84.

SEDERBERG, T. W., ANDERSON, D. C., AND GOLDMAN, R. N.   1985.   Implicitization, inversion, and intersection of planar rational cubic curves. *Comput. Vis. Graph. Image Proc. 31*, 1 (July), 89–102.

SERRA, J.   1982.   *Image Analysis and Mathematical Morphology.* Academic Press, New York.

TAUBIN, G.   1988.   Nonplanar curve and surface estimation in 3-space. In *Proceedings of the IEEE Conference on Robotics and Automation.* IEEE, New York.

TAUBIN, G.   1991.   Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations, with applications to edge and range image segmentation. *IEEE Trans. Patt. Anal. Mach. Intell. 13*, 11 (Nov.), 1115–1138.

TAUBIN, G.   1993.   An accurate algorithm for rasterizing algebraic curves. In the *ACM Symposium on Solid Modeling and Applications.* ACM, New York.

TAUBIN, G., CUKIERMAN, F., SULLIVAN, S., PONCE, J., AND KRIEGMAN, D. J.   1993.   Parameterized families of polynomials for bounded algebraic curve and surface fitting. *IEEE Trans. Patt. Anal. Mach. Intell.* To be published.

TAUBIN, G., CUKIERMAN, F., SULLIVAN, S., PONCE, J., AND KRIEGMAN, D. J.   1992.   Parameterizing and fitting bounded algebraic curves and surfaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, New York, 103–108.

THORPE, J. A.   1979.   *Elementary Topics in Differential Geometry.* Springer-Verlag, New York.

VAN AKEN, J. AND NOVAK, M..   Curve-drawing algorithms for raster displays. *ACM Trans. Graph. 4,* 2, 147–169.

WALKER, R   1950.   *Algebraic Curves.* Princeton University Press, Princeton, N.J.