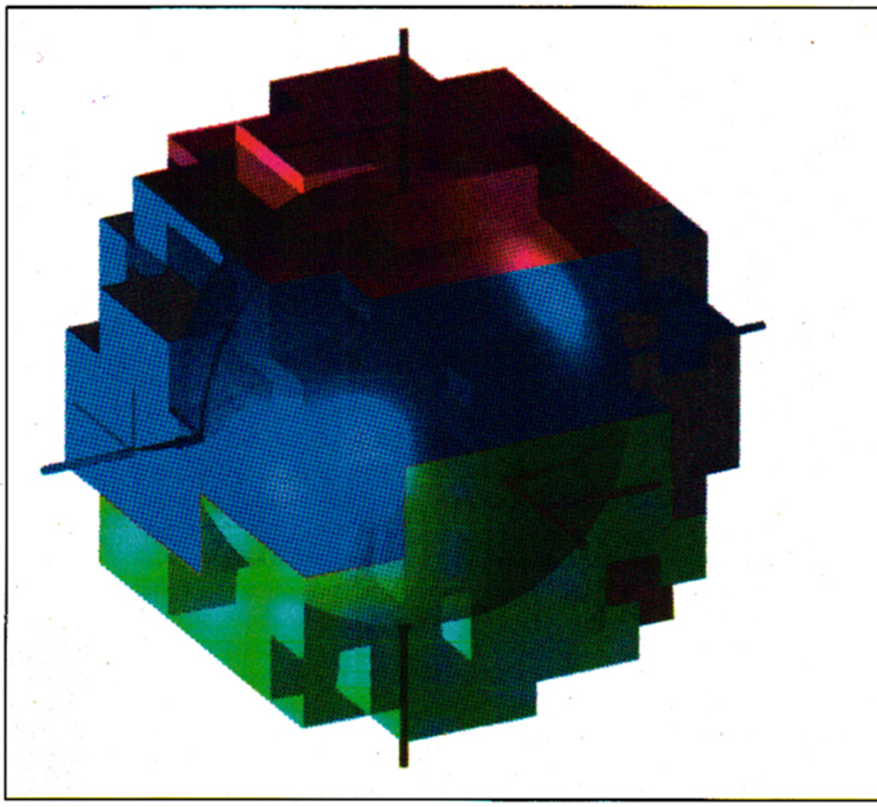
# Rasterizing Algebraic Curves and Surfaces

**Gabriel Taubin**
*IBM T.J. Watson Research Center*



***A new, recursive, space-subdivision algorithm for rasterizing algebraic curves and surfaces gets its accuracy from a newly devised, computationally efficient, and asymptotically correct test.***

We can represent planar curves parametrically or implicitly. A parametric curve is the image set of a 2D 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 $Z(f) = \{(x, y) : f(x, y) = 0\}$. Very efficient methods exist to render several families of parametric curves, but implicit curves are difficult to render.

A local parameterization always exists in a neighborhood of a *regular* point of an implicit curve (that is, a point $p = (u, v)$ such that $f(p) = 0$). Therefore, several researchers use an approach called *tracing* to approximately parameterize the implicit curve, then render it using methods designed for parametric curves.[1] The basic difficulty with this method is that implicit curves can be multiply connected, often having singular points (where $f(p) = 0$ and $\nabla f(p) = 0$) where they intersect themselves or split into several branches. To render the curve correctly, an algorithm must identify the singular points and the connected components.[2] Unfortunately, the algorithms are intrinsically complicated, computationally expensive, and can only handle curves with isolated singularities.

To find all the singular points, an algorithm must compute all the solutions of a system of polynomial equations. Tracing algorithms follow a bottom-up approach, while recursive subdivision algorithms follow a top-down approach. Somewhere in between is a family of algorithms that compute piecewise linear approximations.[3] These algorithms use a single tessellation of the initial box into a regular mesh of cells, triangles, or squares in the plane and cubes or tetrahedra in space, using a marching cubes[4] type of algorithm to move from an occupied cell to a neighboring one. Inside an occupied cell the curve is approximated by a straight line segment, or by a polyhedron in the spatial case.

This kind of approach has two basic problems. The first is how to choose the resolution of the mesh. The second is how to find an occupied cell to start the marching process. In principle, all the cells must be visited in order not to miss disconnected components.[5] Recursive space-subdivision algorithms are generally based on subdividing boxes—the subject of this article—and on subdividing triangles or tetrahedra.[6] I believe that so far no algorithm has dealt with arbitrary singular curves and surfaces correctly. The algorithm described in this article does.

By using a space-subdivision scheme—that is, looking at the original square as a low-resolution pixel, discarding pixels not cut by the curve, and subdividing those that might be cut—we can reduce the problem of rendering an algebraic curve in a raster device to determining whether the curve cuts a given square or not. In a previous work,[7] I introduced a correct algorithm for rasterizing algebraic curves based on this scheme. Instead of using a yes-or-no test to determine whether the curve cuts the square (that is, evaluating a necessary and sufficient

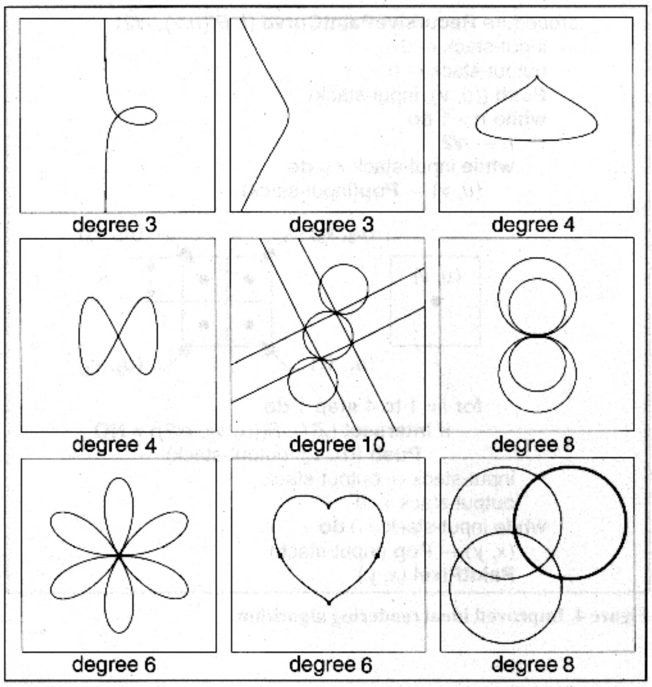
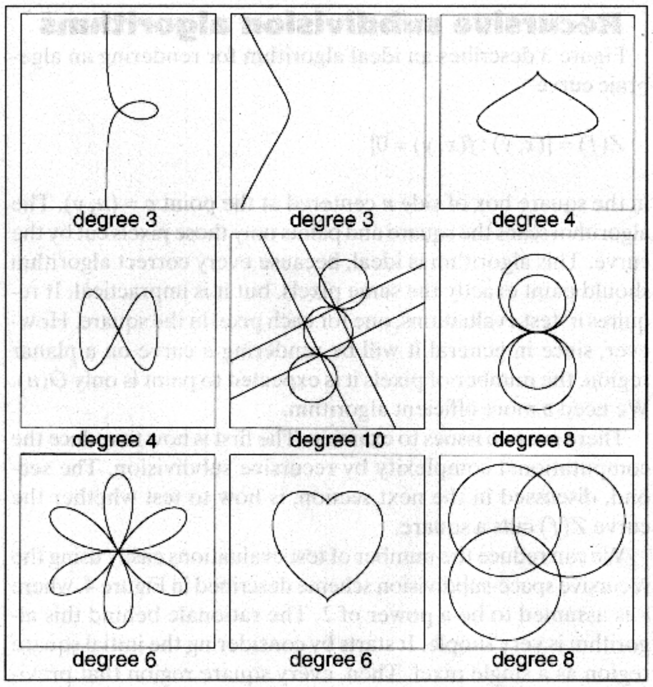


**Figure 1. Examples of algebraic curves rasterized with the algorithm described elsewhere,[7] on a grid of 512 × 512 pixels. Note that isolated zeros and singularities of low order are correctly rendered, but neighborhoods of singularities are thicker. The curve on the lower right corner is the union of three circumferences, but all the points in one of them are singular.**

**Figure 2. The same algebraic curves as in Figure 1, rasterized with the algorithm described in this article. Note the curves have constant width, even in the neighborhoods of singular points.**



condition for zeros of the associated polynomial in the box, a difficult problem[8]), I introduced a much less computationally expensive test based on a simple polynomial inequality. This no-or-maybe test can determine that a polynomial does not have roots inside a circle, although an empty circle could pass the test. You can test squares by testing the circumscribed circles. Squares that do not pass the test can be safely discarded, but the curve might not cut the squares that pass the test. However, the test is asymptotically correct near regular points of algebraic curves; the results are rendered curves of constant width in the neighborhood of regular points. Figure 1 shows some examples of algebraic curves rendered with this algorithm.

Although practical and simple (it produces correct rasterizations of regular curves and almost correct rasterizations of curves with low-order isolated singularities), this rendering algorithm does not behave correctly near singular points. Here I introduce an improved version of the intersection test that solves the problem. First, switching from the 2-norm $\|(x, y)\|_2 = (x^2 + y^2)^{1/2}$ to the $\infty$-norm $\|(x, y)\|_\infty = \max\{|x|, |y|\}$ and modifying the corresponding inequality yields a new test that is a sufficient condition for an algebraic curve not to cut a square instead of a circle. This test takes fewer arithmetic operations to evaluate, and the results are equivalent to those obtained with the old test. Second, the previous algorithm is too conservative near singularities. It does not discard many empty squares near a singularity, which results in thicker lines in the neighborhood of a singular point. Desingularizing the polynomial—constructing a rational function (a ratio of two polynomials) that is continuous everywhere, with exactly the same zeros as the original polynomial, but with constant multiplicity—solves the problem. Essentially, this rational function behaves locally as a second-degree polynomial, even in the neighborhood of a singular point. I extended the polynomial test to this rational function to determine whether a box should be discarded.

With this new test we now have a practical algorithm that renders a curve of constant width, even in neighborhoods of singularities. Figure 2 shows the curves of Figure 1 rendered with this new algorithm. Note that neighborhoods of singular points, even when they are not isolated, are rendered correctly.

Many readers might be skeptical about the robustness and numerical stability of the algorithm, particularly near singular points. I had the same concerns originally, mainly after the work of Farouki and Rajan,[9] but the experimental results clearly show that this algorithm answers these concerns in practice. Due to lack of space, I show only a few examples here, but I have experimented extensively with both versions of this algorithm, and their performance is very good. In a previous paper,[7] I showed curves of degree 50 with many singular points correctly rendered with the previous version of the algorithm. The improved version described here produces even better pictures.

The issue of speed might also concern some readers. Again, due to lack of space, I do not present timing information here, but the running times are acceptable. For low-degree curves, the running times for the pictures shown range from a fraction of a second to a few seconds on a typical technical workstation (IBM RS/6000 Model 530, in my case). I gave timing information for the old algorithm elsewhere.[7] I recognize that this is an area that needs further exploration, and I am currently studying ways to speed up the process.

Finally, note that the approach followed here is essentially the interval arithmetic method for rendering implicit curves.[10] My contribution is a particularly efficient way to construct inclusion functions for polynomials.

```
procedure IdealPaintCurve ($f$, $B((u,v), n/2)$)
    for $x \leftarrow u - n/2 + 1/2$ to $u + n/2 - 1/2$ step 1 do
        for $y \leftarrow v - n/2 + 1/2$ to $v + n/2 - 1/2$ step 1 do
            if Intersect ( $Z(f)$, $B((x,y), 1/2)$) $\neq$ NO
                PaintPixel ( $x,y$)
```

**Figure 3. Ideal algorithm for rendering the curve $Z(f) = \{(x, y) : f(x, y) = 0\}$ on a square of side $n$ centered at the point $(u, v)$. $B((u, v), \delta)$ denotes the square box $\{(x, y) : \max\{|x - u|, |y - v|\} \leq \delta\}$.**

## Recursive subdivision algorithms

Figure 3 describes an ideal algorithm for rendering an algebraic curve

$$Z(f) = \{(x, y) : f(x, y) = 0\}$$

in the square box of side $n$ centered at the point $p = (u, v)$. The algorithm scans the square and paints only those pixels cut by the curve. This algorithm is ideal, because every correct algorithm should paint exactly the same pixels, but it is impractical. It requires $n^2$ test evaluations, one for each pixel in the square. However, since in general it will be rendering a curve on a planar region, the number of pixels it is expected to paint is only $O(n)$. We need a more efficient algorithm.

There are two issues to examine. The first is how to reduce the computational complexity by recursive subdivision. The second, discussed in the next section, is how to test whether the curve $Z(f)$ cuts a square.

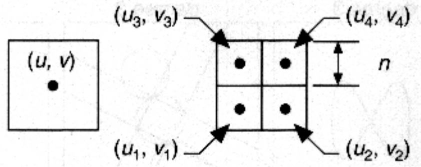
We can reduce the number of test evaluations easily using the recursive space-subdivision scheme described in Figure 4, where $n$ is assumed to be a power of 2. The rationale behind this algorithm is very simple. It starts by considering the initial square region as a single pixel. Then, every square region that previously satisfied the test for its resolution is divided into four equal-sized square regions, which are then tested. The squares that pass the distance test are kept in a stack. Each iteration halves the sides of the squares; when the square regions become single pixels, the iteration stops and the pixels are painted.

## Tests for zeros of an algebraic curve

This section tackles the problem of testing whether the algebraic curve $Z(f)$ defined by a polynomial of two variables $f(x, y)$ intersects the box $B((u, v), \delta) = \{(x, y) : \max\{|x - u|, |y - v|\} \leq \delta\}$. Tests that give a yes or no answer, that is, tests for necessary and sufficient conditions for the curve to intersect the box, exist.[8] However, they are computationally very expensive, involving elimination techniques, and probably not numerically well behaved. You could use approximate tests instead; the main problem with approximate tests, though, is proving the correctness of the algorithm. An approximate test should satisfy three main properties. First, it should never discard a box that intersects the curve; otherwise, parts of the curve could be missing. That is, the test should be a sufficient condition for a curve not to intersect the box. Second, it should be asymptotically correct; that is, if a box does not intersect the curve and is not rejected by the test, after a number of subdivision steps all the resulting subboxes should be rejected. Third, and very much related to the previous property, it should render curves of approximately constant width.

The last two properties are difficult to achieve, in particular when close to singular points. There is also the issue of how

```
procedure RecursivePaintCurve ($f$, $B((u,v), n/2)$)
    input-stack $\leftarrow 0$
    output-stack $\leftarrow 0$
    Push (($u$, $v$), input-stack)
    while $n > 1$ do
        $n \leftarrow n/2$
        while input-stack $\neq 0$ do
            ($u$, $v$) $\leftarrow$ Pop(input-stack)
```



```
            for $i = 1$ to 4 step 1 do
                if Intersect ($Z(f)$, $B((u_i, v_i), n/2)$) $\neq$ NO
                    Push (($u_i$, $v_i$), output-stack)
        input-stack $\leftarrow$ output-stack
        output-stack $\leftarrow 0$
    while input-stack $\neq 0$ do
        ($x$, $y$) $\leftarrow$ Pop (input-stack)
        PaintPixel ($x$, $y$)
```

**Figure 4. Improved ideal rendering algorithm.**

well the test performs in practice, because even if it is asymptotically correct, it could converge very slowly.

Now I will briefly describe a slightly modified version of a test that I recently introduced.[7] It satisfies the first property, and the second and third ones in the neighborhoods of the regular points of the curve. The test fails to discard enough pixels close to singular points, resulting in thicker regions. That is, the width of the rendered curve increases in the neighborhood of singular points. Figure 1 has already shown us examples of algebraic curves rendered with the recursive subdivision algorithm of Figure 4, using this test. I then modify the test further to correct its behavior near singular points. This is the main contribution of this article.

Note that by first translating the origin to $(u, v)$, that is, by expanding the polynomial in Taylor series around $(u, v)$, the problem reduces to the case $(u, v) = (0, 0)$. An efficient algorithm for transforming the polynomial

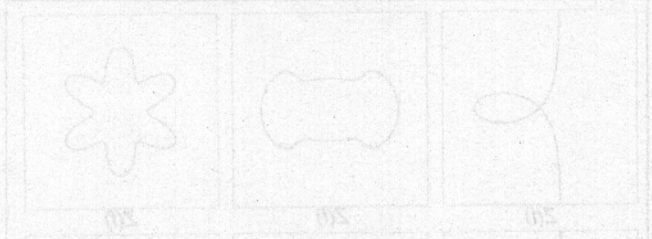
$$f(x,y) = \sum_{0 \leq i,j;\, i+j \leq d} f_{ij} x^i y^j \tag{1}$$

where $d$ is the degree of $f$, to the form

$$f(x,y) = f'(x-u, y-v) = \sum_{0 \leq i,j;\, i+j \leq d} f'_{ij} (x-u)^i (y-u)^j \tag{2}$$

is Horner's algorithm.[11] Going from Equation 1 to Equation 2 is equivalent to evaluating the polynomial and all its partial derivatives at $(u, v)$. Horner's algorithm requires $O(m^2)$ operations, where $m$ is the number of coefficients of the polynomial. To evaluate a polynomial in Bernstein form requires the same order of operations.

So, from now on we will concentrate on finding a sufficient condition for the polynomial in Equation 1 not to have zeros in the box $B((0, 0), \delta) = \{(x, y) : \|(x, y)\|_\infty \leq \delta\}$; we can assume that $f(0, 0) = f_{00} \neq 0$, because otherwise the curve clearly intersects the box. The approach is as follows: Construct a polynomial of one



variable $F(\varepsilon)$ whose coefficients are functions of the coefficients $f_{ij}$ of $f$ such that $F(0) = |f(0,0)| > 0$, and

$$\forall\, \delta > 0\ \forall\, (x, y) \in B((0, 0), \delta) : |f(x, y)| \geq F(\delta)$$

Clearly, if $\delta$ is smaller than the first positive root of $F$ (if any), the polynomial $f$ does not have zeros in $B((0, 0), \delta)$.

In the past I constructed the bounding polynomial $F(\delta)$ to provide a sufficient condition for the polynomial in Equation 1 not to have zeros in a circle $C((0, 0), \delta) = \{(x, y) : (x^2 + y^2)^{1/2} \leq \delta\}$.[7] The following construction provides a test for boxes that requires fewer arithmetic operations to evaluate. First rewrite Equation 1 as a sum of homogeneous terms, or forms

$$f(x, y) = \sum_{h=0}^{d}\left\{\sum_{i+j=h} f_{ij}x^i y^j\right\} = \sum_{h=0}^{d} f^h(x, y) \qquad \textbf{(3)}$$

and apply the triangular inequality to the last sum.

$$|f(x, y)| \geq |f^0| - \sum_{h=1}^{d}|f^h(x, y)| \qquad \textbf{(4)}$$

Now consider each one of the forms $f^1, \ldots, f^d$ individually and observe that

$$\begin{aligned} |f^h(x, y)| &\leq \sum_{i+j=h}|f_{ij}x^i y^j| \\ &\leq \left\{\sum_{i+j=h}|f_{ij}|\right\}\left\{\max_{i+j=h}|x^i y^j|\right\} \\ &\leq F_h \varepsilon^h \end{aligned} \qquad \textbf{(5)}$$

where

$$\begin{aligned} \varepsilon = \|(x, y)\|_\infty &= \max\{|x|, |y|\} \\ F_h = \|(f_{h0}, \ldots, f_{0h})\|_1 &= \sum_{i+j=h}|f_{ij}| \end{aligned}$$

because it is trivial to verify

$$\max_{i+j=h}|x^i y^j| = \{\max\{|x|, |y|\}\}^h = \varepsilon^h$$

If we define $F(\varepsilon) = F_0 - \Sigma_{h=1}^{d} F_h\varepsilon^h$, from Equations 4 and 5, we obtain

$$|f(x, y)| \geq F(\varepsilon)$$

Since $F(\varepsilon)$ is positive at the origin ($F(0) = F_0 = |f(0, 0)| > 0$), is monotonically decreasing for $\varepsilon > 0$, the coefficients $F_1, \ldots, F_d$ are nonnegative, and $F_d$ is positive, $F(\varepsilon)$ has a unique positive root $\delta_d$. Since $F(\varepsilon) > 0$ for $0 < \varepsilon < \delta_d$, we have a sufficient condition for the polynomial $f$ not to have zeros inside a box: If $0 < \varepsilon < \delta_d$, then $F$ does not have zeros inside the box $B((0, 0), \varepsilon)$. In fact, although the number $\delta_d$ can be computed very quickly with a couple of Newton iteration steps, we do not need to compute it. Since the polynomial $F(\varepsilon)$ is positive at the origin and is strictly decreasing for $\varepsilon \geq 0$, to test whether the curve intersects the box $B((0, 0), \delta)$, we just evaluate $F(\delta)$. If the value is positive, the curve does not cut the box.

In the old test,[7] the coefficients $F_1, \ldots, F_d$ were defined as follows:

$$F_h = \left\{\sum_{i+j=h}\binom{h}{i}^{-1}|f_{ij}|^2\right\}^{1/2}$$

In this case, $F(\varepsilon) > 0$ instead implies that $f$ does not have zeros in the circle $C((0, 0), \varepsilon)$. The difference between the old and new tests is related to the switching from the 2-norm $\|(x, y)\|_2 = (x^2 + y^2)^{1/2}$ to the $\infty$-norm $\|(x, y)\|_\infty = \max\{|x|, |y|\}$. In fact, these are just two particular instances of a more general construction. For every real number $r$ such that $0 < r < \infty$, we can construct an $r$-norm test in a similar way by defining the coefficients $F_1, \ldots F_d$ as follows:

$$F_h = \left\{\sum_{i+j=h}\binom{h}{i}^{1-r}|f_{ij}|^r\right\}^{1/r}$$

In this case $F(\varepsilon) > 0$ is a sufficient condition for the polynomial $f$ not to have zeros in the $r$-ball

$$B_r((0, 0), \varepsilon) = \{(x, y) : \|(x, y)\|_r = (|x|^r + |y|^r)^{1/r} < \varepsilon\}$$

However, since we will not use these tests and it is more expensive to evaluate them, I will not describe them in detail here. I only mention this family of tests because the only positive root $\delta_d$ of the bounding polynomial $F(\varepsilon)$ associated with the $r$-norm is a lower bound estimate for the $r$-distance from the origin to the curve. Since all the norms are equivalent to each other (in the sense that, when one goes to zero, all of them go to zero at the same rate), using any one of these tests as the basis for the rendering algorithm should produce almost the same results. Owing to lack of space I will not give a formal proof for this, but I have confirmed it experimentally. Although the results are not identical, there is no significant perceptual difference. So, to understand the local behavior of a rendering algorithm based on these tests, it is sufficient to analyze any one of them.

When analyzing the case of the 2-norm test,[7] I proved that the 2-norm approximate distance $\delta_d(p)$ (defined as above by first translating the origin to the point $p = (u, v)$, then evaluating the coefficients of the bounding polynomial, and finally finding its unique nonnegative root) has the following property, which explains the behavior of the algorithm near regular points:

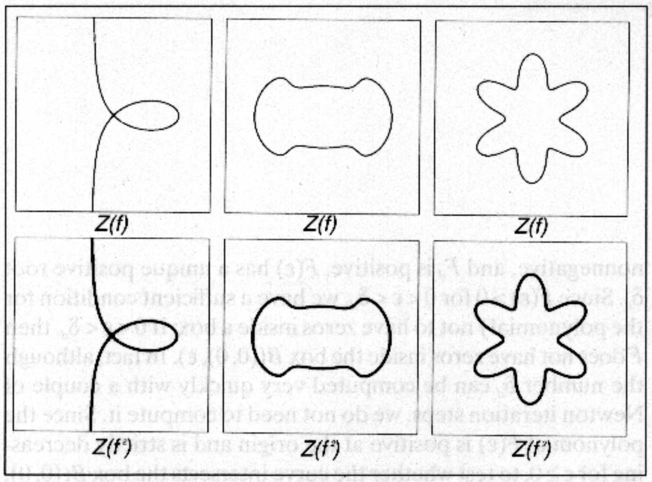


Figure 5. Examples of algebraic curve pairs $(Z(f), Z(f^2))$. $Z(f^2)$ is rendered at almost twice the width of $Z(f)$.

*Lemma 1.* If $f$ is a polynomial of degree $d \geq 1$, $p_0$ is a regular point of $Z(f)$, $w$ is a unit length normal vector to $Z(f)$ at $p_0$, and $p_t = p_0 + tw$, for $t \in \mathbb{R}$, then

$$\lim_{t \to 0} \frac{\delta_d(p_t)}{\operatorname{dist}(p_t, Z(f))} = 1$$

where $\operatorname{dist}(p, Z(f))$ is the Euclidean distance from the point $p = (u, v)$ to the curve $Z(f)$.

The behavior of the algorithm near singular points is more difficult to analyze. Near a singular point the approximate distance $\delta_d(p)$ always underestimates the Euclidean distance, and in general the width of the curve increases proportionally to the multiplicity of the singularity. I will be more specific in the next section, but for example, if $Z(f)$ is a regular curve, $\nabla f(x, y) \neq 0$ for every point $(x, y) \in Z(f)$. When we set $g(x, y) = f(x, y)^2$, the algorithm based on the test described above will still render the curve $Z(g)$ correctly—that is, without erroneously discarding occupied boxes—but at almost twice the width. Most rendering algorithms, in particular tracing algorithms, cannot deal with this kind of situation. Figure 5 shows some examples of these curves.

When a polynomial has a multiple factor, as in the case of

$$\begin{aligned} f(x, y) = &((x+1)^2 + (y+1)^2 - 1) \\ &((x+1)^2 + (y-1)^2 - 1) \\ &((x-1)^2 + (y-1)^2 - 1)^2 \end{aligned}$$

shown in the lower right corner of Figure 1, this algorithm renders the multiple components at larger width.

## Desingularization

My goal in this section is to modify the test described above so as to correct its behavior near singular points.

### Multiple points

The multiplicity of a point $p = (u, v)$ as a zero of the polynomial $f$ is the minimum index $h$, such that at least one partial derivative of $f$ of order $h$ is not identically zero at $p$:

$$m(p; f) = \min\{h : f_p^h \not\equiv 0\}$$

where the form $f_p^h$ is the form $f^h$ of Equation 3, after translating the origin to $p$, as in Equation 2. We can define the multiplicity of a point as a zero of an analytic function in a similar way. In this case, we must expand the function in Taylor series around $p$ to obtain the forms $f_p^h$. Thus, the multiplicity of a point with respect to a rational function is the multiplicity of the point with respect to the numerator minus the multiplicity of the same point with respect to the denominator. Points with positive multiplicity are called *zeros*, and points with negative multiplicity are called *poles*.

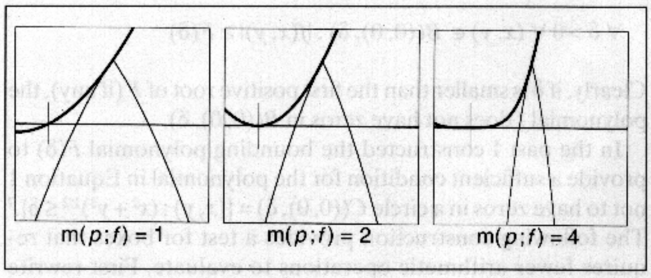
Since $f_p^0 = f(p)$ is a constant, the set $Z(f)$ is exactly the set of points with positive multiplicity

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

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 $Z(f)$. The set of singular points of the curve $Z(f)$ is exactly the set of points with multiplicity at least two:

$$Z'(f) = \{p \in \mathbb{R}^2 : m(p; f) > 1\} \subseteq Z(f)$$

Figure 6. Behavior of $\delta_d$ near regular and singular points. The thick curve is the graph of $f$, and the thin curve is the graph of the bounding polynomial $F$ at the origin. The closest zero to the origin is $p$, marked with a vertical segment on the left-hand side of the pictures.



### Reducing multiplicities

Figure 6 illustrates the main reason why the approximate distance $\delta_d$ underestimates the Euclidean distance near a singular point. The graphs of the polynomial $f(x, y)$ and of its bounding polynomial $F(\varepsilon)$ at the origin are represented along a line that cuts a zero point $p$ (marked with a vertical line segment on the left side of the figures). When approaching a regular point, the value of $|f(x, y)|$ goes to zero linearly; as the multiplicity increases, $|f|$ goes to zero much faster. Note that the graph of $|f|$ is concave near a zero of $f$, while the graph of $F(\varepsilon)$, which is always below the graph of $|f|$, is convex. For a good approximation of the former by the latter, we need both graphs to be almost linear. If the concavity of the graph of $|f|$ is too pronounced, it is just not possible to obtain an accurate approximation with a convex function.

My approach is to linearize the behavior of $f$ near $Z(f)$, or to *desingularize* it. By this I mean to construct a new rational function $g(x, y)$, not a polynomial, with exactly the same set of zeros $Z(g) = Z(f)$, but such that all the points in $Z(g)$ have exactly the same multiplicity with respect to $g$. First set $\phi(x, y) = f(x, y)^2$. The degree of $\phi$ is $2d$, and the multiplicity of a point as a zero of $\phi$ is twice the multiplicity of the same point as a zero of $f$. Now I show how to construct a polynomial $\psi(x, y)$ with the following properties:

1. The degree of $\psi$ is $2d - 2$.
2. The set of zeros of $\psi$ is a subset of the set of zeros of $\phi$. In fact, $Z(\psi)$ will be the set $Z'(f)$ of singular points of $f$.
3. If $p$ is a zero of $\psi$, and thus of $\phi$, the multiplicity of $p$ as a zero of $\psi$ is exactly equal to the multiplicity of $p$ as a zero of $\phi - 2$.

Under these conditions, the rational function $g(x, y) = \phi(x, y)/\psi(x, y)$ is well defined and continuous everywhere ($g$ does not have poles). It has exactly the same zeros as $f$, and the multiplicity of every point $p$ in $Z(g) = Z(f)$ as a zero of $g$ is exactly 2.

So, let us construct $\psi$. First, denote with $f_x$ and $f_y$ the two partial derivatives of $f$. A point $p$ is singular if and only if it is a common zero of $f, f_x$, and $f_y$. Since the degrees of $f_x$ and $f_y$ are at most $d - 1$ while the degree of $f$ is $d$, let us define a new polynomial:

$$f_\infty(x, y) = d\,f(x, y) - (x\,f_x(x, y) + y\,f_y(x, y))$$

The degree of $f_\infty$ is at most $d-1$ because of Euler's formula,[12] that is, the leading form $f^d$ of $f$ satisfies the following identity:

$$d\,f^d(x, y) = x\,f_x^d(x, y) + y\,f_y^d(x, y)$$

The common zeros of $f, f_x$, and $f_y$ are exactly the same as the common zeros of $f_\infty$, $f_x$, and $f_y$, because the construction is reversible:

$$f(x, y) = 1/d(f_\infty(x, y) + x\,f_x(x, y) + y\,f_y(x, y))$$

Another way to construct $f_\infty$ is to first homogenize $f(x, y)$ with a new variable $w$,

$$f(w,x,y) = \sum_{h=0}^{d} w^{d-h} f_h(x,y)$$

obtaining a homogeneous polynomial of degree $d$ in three variables, then differentiate with respect to the new variable:

$$f_w(w,x,y) = \sum_{h=0}^{d-1} (d-h) w^{d-h-1} f_h(x,y)$$

Finally, dehomogenize and divide by $d$:

$$f_\infty(x,y) = \frac{1}{d} f_w(1,x,y) = \sum_{h=0}^{d-1} \frac{d-h}{d} f_h(x,y)$$

Let $\psi(x, y) = f_\infty(x, y)^2 + f_x(x, y)^2 + f_y(x, y)^2$. The degree of $\psi$ is exactly $2d - 2$, because at least one of $f_x$ or $f_y$ should have degree $d - 1$, and none of the three has degree larger than $d - 1$. By construction, the set of zeros of $\psi$ is exactly equal to the set $Z(f')$ of singular points of $f$, and so, a subset of $Z(f) = Z(\phi)$. The only property that remains to be proved is the following:

*Lemma 2.* If $\phi(p) = 0$, then $m(p; \psi) = m(p; \phi) - 2$.

Proof: If $m(p; f) = k > 0$, after translating the origin to $p$, we can write the polynomial $f$ as a sum of forms:

$$f(x,y) = \sum_{h=k}^{d} f_h(x,y)$$

with the form $f^k$ not identically zero. The two partial derivatives are then

$$f_x(x,y) = \sum_{h=k}^{d} f_x^h(x,y)$$

$$f_y(x,y) = \sum_{h=k}^{d} f_y^h(x,y)$$

where $f_x^k$ and $f_y^k$ are forms of degree $k - 1$, and at least one of them is nonzero, again from Euler's formula.[12] It follows that either $m(p; f_x) = k - 1$ or $m(p; f_y) = k - 1$, that is,

$$\min\{m(p; f_x), m(p; f_y)\} = m(p; f) - 1$$

This is also true for functions of more than two variables. Finally, by construction, $m(p; f_\infty) \geq k - 1$ as well. We can conclude that $m(p; \psi) = 2k - 2$.

Note that the construction of $f_\infty$, as well as the construction of $\psi$, depends on the location of the origin of the coordinate system. Once we choose a point $p = (u, v)$, we first translate the origin of the polynomial to $p$, then construct $\phi$ and $\psi$. Different rational functions are used at different locations.

Now we need to extend our intersection test for polynomials to the rational function $g = \phi/\psi$. As before, after translating the origin we can just assume that $p = (0, 0)$. A second assumption is that $\psi(p) \neq 0$, and so $\phi(p) \neq 0$, because if $\psi(p) = 0$, the curve clearly cuts the box. Since rational functions are analytic, in a certain neighborhood of the origin we can expand $g$ as a convergent series:

$$g(x,y) = \sum_{h=0}^{\infty} g^h(x,y)$$

where $g^h$ is a form of degree $h$ or identically zero. We can compute the forms $g^0, g^1, \ldots$ from the following recursive equations:

$$
\begin{aligned}
\phi^0 &= \psi^0 g^0 \\
\phi^1 &= \psi^0 g^1 + \psi^1 g^0 \\
&\vdots \\
\phi^{2d-2} &= \psi^0 g^{2d-2} + \ldots + \psi^{2d-2} g^0 \\
\phi^{2d-1} &= \psi^0 g^{2d-1} + \ldots + \psi^{2d-2} g^1 \\
\phi^{2d} &= \psi^0 g^{2d} + \ldots + \psi^{2d-2} g^2 \\
0 &= \psi^0 g^{2d+k} + \ldots + \psi^{2d-2} g^{2+k}
\end{aligned}
\qquad \textbf{(6)}
$$

with the last equation valid for $k = 1, 2, 3, \ldots$. Then we generalize the inequality $|f(x, y)| \geq F_0 - \Sigma_{h=1}^{d} F_h \varepsilon^h$ to an infinite sum:

$$|g(x,y)| \geq G_0 - \sum_{h=1}^{\infty} G_h \varepsilon^h = G(\varepsilon)$$

where the coefficient $G_h$ is computed from the coefficients of the form $g^h(x, y)$ as before:

$$G_h = \sum_{i+j=h} |g_{ij}| \ \text{ where } \ g^h(x,y) = \sum_{i+j=h} g_{ij} x^i y^j \qquad \textbf{(7)}$$

The analytic function of one variable $G(\varepsilon)$ behaves as the polynomial $F(\varepsilon)$. That is, it is positive at the origin and decreases monotonically for $\varepsilon > 0$. Thus, it has a unique positive root. We don't need to compute this root either. To test whether $g$ has no zeros in a box $B((0, 0), \delta)$, we just need to evaluate $G(\delta)$. If $G(\delta) > 0$, then $g$ has no zeros in the box. And here we have a problem. If a partial sum $\Sigma_{h=0}^{k} G_h \delta^h$ is negative, we can stop the evaluation and accept the box for further subdivision. However, to be sure that $G(\delta) > 0$, we need to perform an infinite number of operations, and we cannot do that. The solution is to truncate the series and redefine $G(\varepsilon)$ as a polynomial plus an extra term:

$$G(\varepsilon) = G_0 - \sum_{h=1}^{2d} G_h \varepsilon^h - G_{2d+1}(\varepsilon)$$

where $G_0, \ldots G_{2d}$ are defined in Equation 7, but $G_{2d+1}(\varepsilon)$ will bound the tail of the series expansion of $g$:

$$\left| \sum_{h=2d+1}^{\infty} g^h(x,y) \right| \leq G_{2d+1}(\varepsilon)$$

for $\|(x, y)\|_\infty \leq \varepsilon$. To construct $G_{2d+1}$, we need the following result:

*Lemma 3.* Let $\xi^1, \ldots, \xi^n, \nu^1, \ldots, \nu^n, M$ be real numbers. Let us define $\nu^{j+n} = \xi^1 \nu^{j+n-1} + \ldots + \xi^n \nu^j$, and $K_j = \max\{|\nu^j|, \ldots, |\nu^{j+n-1}|\}$, for $j \geq 1$. If $M$ satisfies the inequality $|\xi^1| + \ldots + |\xi^n| \leq M < 1$, then

$$\left| \sum_{j=r\,n+1}^{\infty} \nu^j \right| \leq n K_1 M^r / (1-M)$$

for $r \geq 1$.

Proof: First, note that for every $j \geq 1$ we have

$$
\begin{aligned}
|\nu^{j+n}| &= |\xi^1 \nu^{j+n-1} + \ldots + \xi^n \nu^j| \\
&\leq |\xi^1| + \ldots + |\xi^n|\} \max\{|\nu^j|, \ldots, |\nu^{j+n-1}|\} \\
&= M\, K_j \\
&< K_j
\end{aligned}
$$

Now, from the previous two inequalities we obtain

$$
\begin{aligned}
K_{j+n} &= \max\{|\nu^{j+n}|, \ldots, |\nu^{j+2n-1}|\} \\
&\leq \max\{M\, K_j, \ldots, M\, K_{j+n-1}\} = M\, K_j
\end{aligned}
$$

and by induction, $K_{j+hn} \leq M^h K_j$, for $h \geq 1$. Finally, for $r \geq 1$ we have

$$
\begin{aligned}
\left| \sum_{j=r\,n+1}^{\infty} \nu^j \right| &\leq \sum_{h=0}^{\infty} \left\{ \sum_{j=1}^{n} \left| \nu^{j+(r+h)n} \right| \right\} \leq \\
\sum_{h=0}^{\infty} \left\{ n K_{1+(r+h)n} \right\} &\leq \sum_{h=0}^{\infty} \left\{ n M^{r+h} K_1 \right\} = \\
n K_1 M^r \sum_{h=0}^{\infty} M^h &= n K_1 M^r / (1-M)
\end{aligned}
$$

Now we compute $G_{2d+1}(\varepsilon)$ as follows. First compute the coefficients of the forms $g^0, \ldots, g^{2d}$ with the recursive Equation 6, then evaluate the coefficients $G_0, \ldots, G_{2d}$ of the bounding polynomial. Then apply Lemma 3 for $n = 2d$, $r = 1$, $\xi^j = -\psi_j/\psi_0$ for $j = 1, \ldots, 2d-2$, $\xi^{2d-1} = \xi^{2d} = 0$, and $\nu^j = g^j$, for $j = 1, \ldots, 2d$. Note that the recursive equation of the lemma is nothing but the last line of Equation 6 and that everything here is a function of the point $p = (x, y)$, even the bounds $M$ and $K_1$. We have some freedom in choosing $M$, which we will make a function of $\varepsilon$. The form $\xi^h(x, y)$ satisfies the inequality $\xi^h(x, y) \leq \Xi_h \|(x, y)\|_\infty^h$, where $\Xi_h = \Sigma_{i+j=h} |\xi_{ij}|$, and $\xi^h = \Sigma_{i+j=h} \xi_{ij} x^i y^j$. We take

$$M(\varepsilon) = \Xi_1\, \varepsilon + \ldots + \Xi_{2d}\, \varepsilon^{2d}$$

Note that $M(\varepsilon)$ is a monotone function, and $M(0) = 0$. So, for small $\varepsilon$, the hypothesis $M(\varepsilon) < 1$ is satisfied. If it is not satisfied for the current value of $\varepsilon$, subdivide the cube even further. Clearly, the condition $M(\varepsilon) < 1$ will eventually be satisfied after a number of subdivisions. We can always replace $K_1 = \max\{|g^1|, \ldots, |g^{2d-2}|\}$ with an upper-bound function of $\varepsilon$. Since $\xi_{2d-1} = \xi_{2d} = 0$, we set

$$K_1(\varepsilon) = \max\{G_1 \varepsilon, \ldots, G_{2d-2} \varepsilon^{2d-2}\}$$

This is also a monotone function of $\varepsilon$, and $K_1(0) = 0$. Observe that $K_1(\varepsilon)$ is independent of $G_0$, which is assumed to be different from zero. Finally,

$$G_{2d+1}(\varepsilon) = 2d\, K_1(\varepsilon)\, M(\varepsilon) \,/\, (1 - M(\varepsilon))$$

Note that at least $G_{2d+1}(\varepsilon) = O(\varepsilon^2)$.

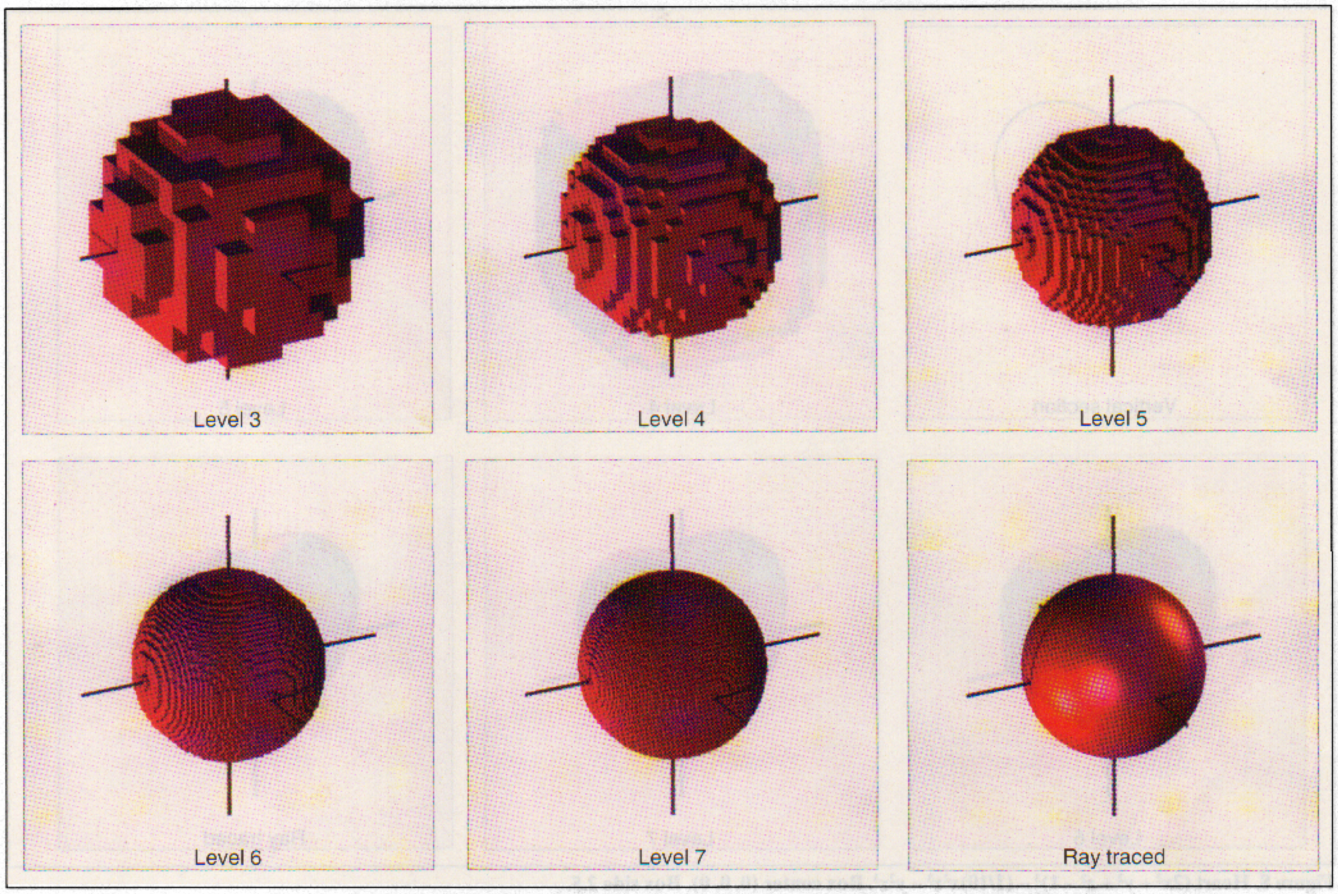


**Figure 7. Sphere: $x^2 + y^2 + z^2 - 2.25 = 0$. Box center (0, 0, 0). Box side 4.**

Figure 2 shows examples of curves rasterized with the algorithm of Figure 4 based on this test.

## Rasterizing surfaces and surface-surface intersections

The algorithms described above can be trivially extended to dimension three and above. In dimension three we start with a cubical box

$$B((u, v, w), \delta) = \{(x, y, z) : \|(x - u, y - v, z - w)\|_\infty \leq \delta\}$$

and a polynomial of three variables $f(x, y, z)$ of degree $d$. We can extend the recursive subdivision algorithm to three dimensions by subdividing the cube into eight subcubes. I introduced the tests in dimension two to avoid overloading the reader with notation, but we can use Horner's algorithm in dimension three and above. To determine whether the polynomial $f$ has zeros in a box $B((u, v, w),\delta)$, translate the origin to $p = (u, v, w)$, then test the box $B((0, 0, 0), \delta)$. After translating the origin, write the polynomial $f(x, y, z)$ as a sum of forms of increasing degree:

$$f(x,y,z) = \sum_{h=0}^{d} f^h(x,y,z)$$

where

$$f^h(x, y, z) = \sum_{i+j+k=h} f_{ijk} x^i y^j z^k$$

For the first test, we define the coefficients of the bounding polynomial $F(\varepsilon) = F_0 - \Sigma_{h=1}^{d} F_h \varepsilon^h$ as follows:

$$F_h = \sum_{i+j+k=h} \left| f_{ijk} \right|$$

and we obtain the corresponding test for dimension three: If $F(\delta) > 0$, then the polynomial $f$ does not have zeros in the box $B((0, 0, 0), \delta)$. Figure 7 shows the result of applying the algorithm based on this test to the simple case of a sphere. I rendered the results at the end of each subdivision level with a standard ray-tracing algorithm. You can extend the new test introduced in the previous section to 3D in a similar way, but I leave the details to the reader.

The set of regular zeros of a polynomial define a surface (if not empty). However, the set of singular points can be a union of isolated points, curves, and even surfaces. For example, the surface of the heart in Figure 8 (next page) has two isolated singular points (the cusps at the top and the bottom). It also has a curve of singular points: The intersection of the surface with the plane $z = 0$ is an ellipse, and all the points on this ellipse are singular. You can observe a fattening around that curve due to using the simpler test for this picture.

In the case of the Cartan's umbrella in Figure 9 (next page), the $z$ axis is the set of singular points. Unlike the heart, not all the singular points belong to the boundary of the set of regular points. The algorithm correctly keeps track of a neighborhood of all the points of the surface, including the singular points.

In fact, this algorithm can render the intersection of two regular surfaces by representing it as a completely singular sur-

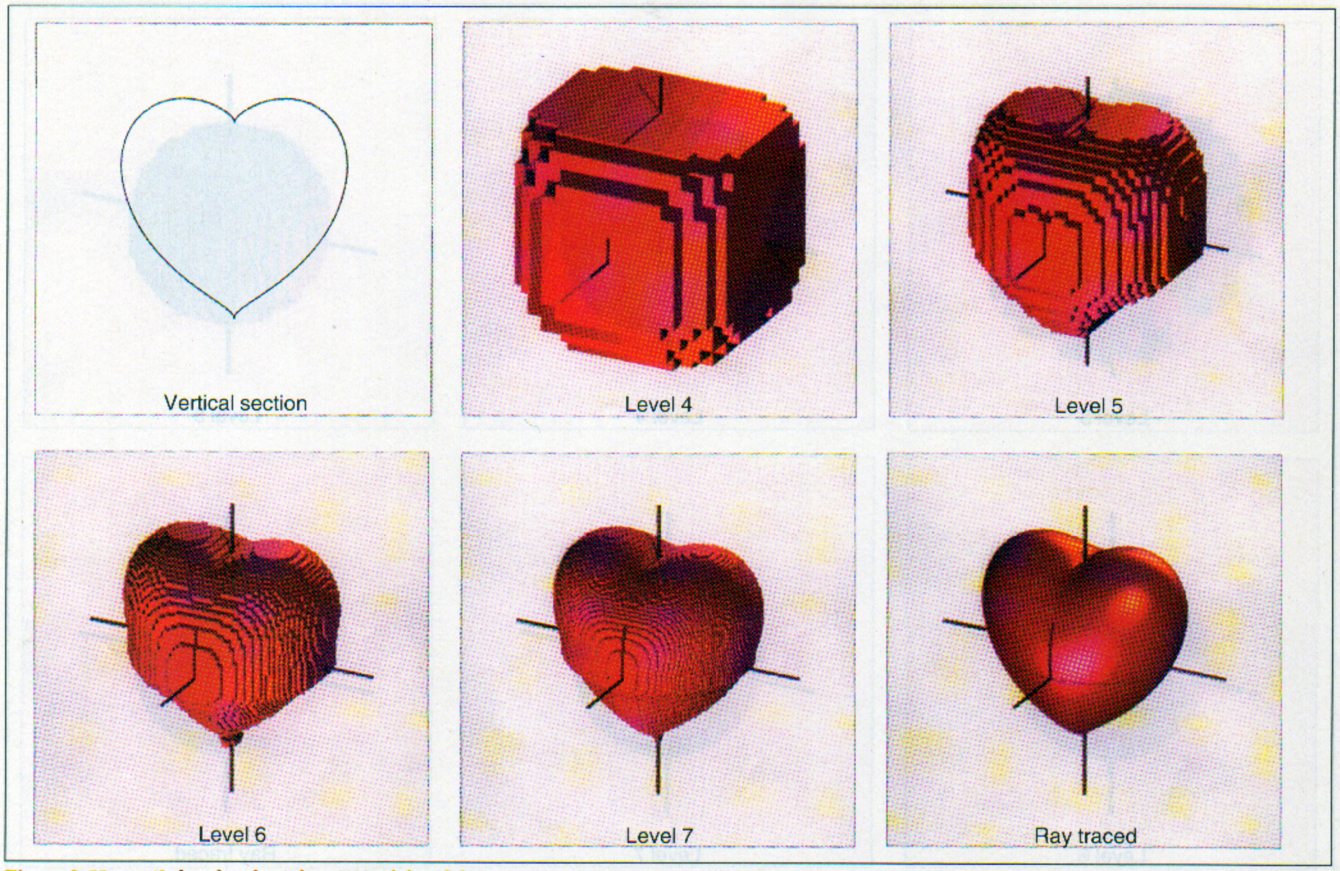


**Figure 8. Heart $(2x^2 + y^2 + z^2 - 1)^3 - (1/10)x^2z^3 - y^2z^3$. Box center (0, 0, 0). Box side 2.5.**

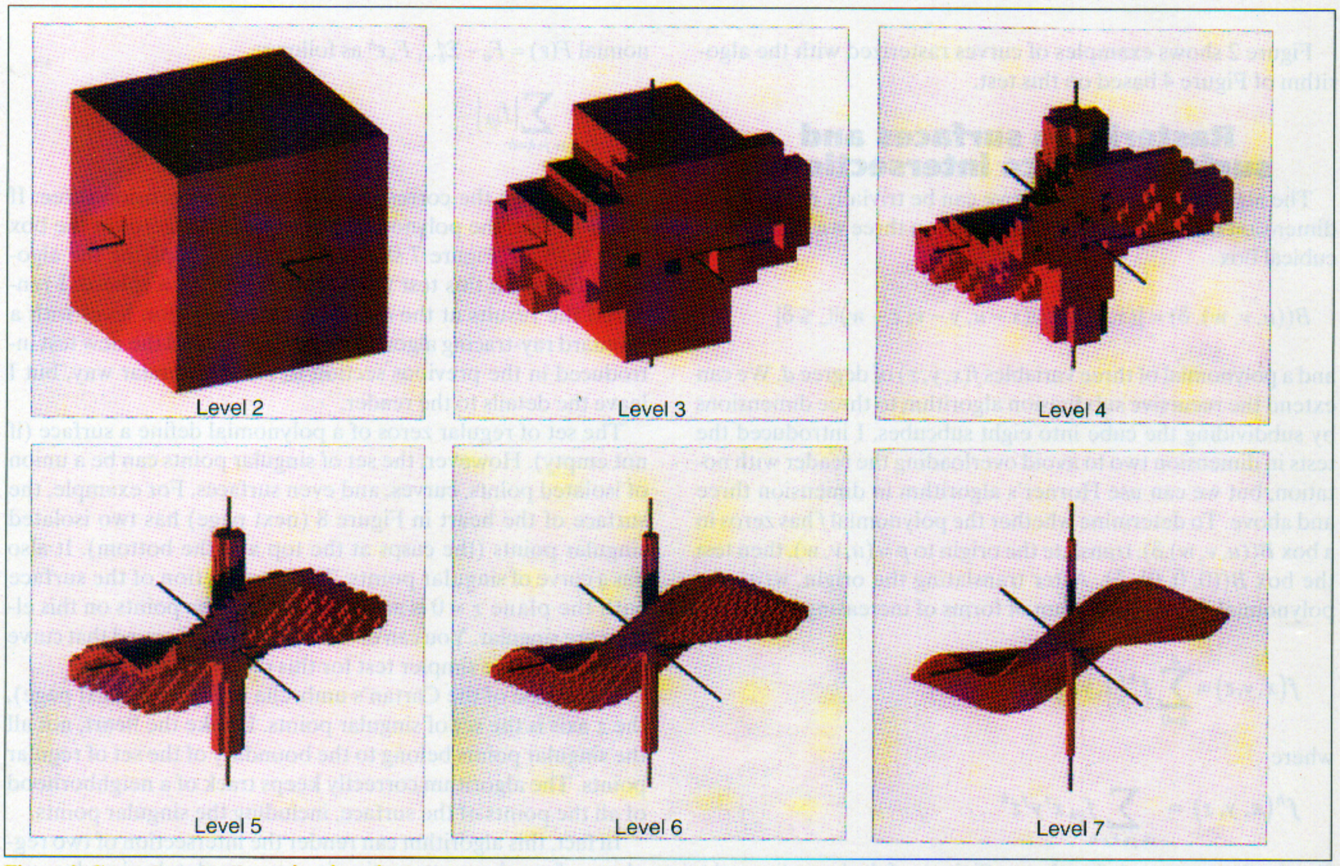


**Figure 9. Cartan's umbrella: $z(x^2 + y^2) - x^3 = 0$. Box center (0, 0, 0). Box side 4.**

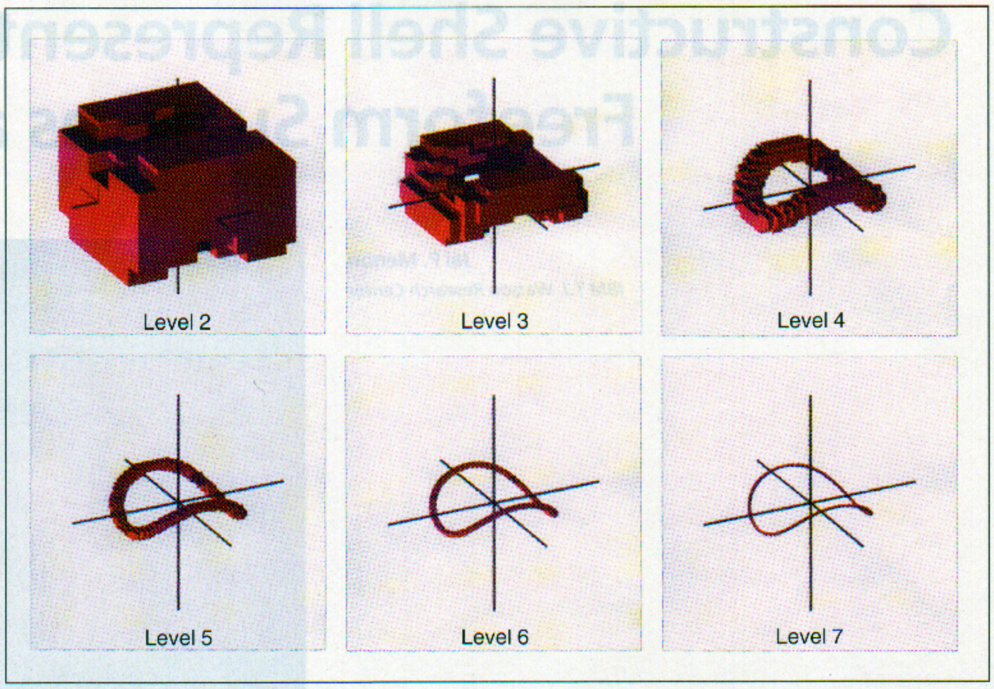


**Figure 10. Intersection of two cylinders $x^2 + (z-1)^2 - 2 = 0$, $y^2 + (z+1)^2 - 2 = 0$, as a singular surface $(x^2 + (z-1)^2 - 2)^2 + (y^2 + (z+1)^2 - 2)^2 = 0$. Box center (0, 0, 0). Box side 4.**

face. Figure 10 shows an example of this process. The intersection of two cylinders is rendered by applying the subdivision algorithm to the sum of squares of the two polynomials defining the individual surfaces.

As a particular case, we can use the algorithm to visualize and explore surfaces with singularities. Simultaneously, we can render the singular space, defined as the set of zeros of the polynomial

$$\psi(x,y,z) = f_\infty(x,y,z)^2 + f_x(x,y,z)^2 + f_y(x,y,z)^2 + f_z(x,y,z)^2$$

used in the desingularization test. The user can choose regions of the original surface to explore in more detail based on the results of rendering the singular points.

This algorithm provides a primitive tool, both to visualize algebraic surfaces and space curves in 3D and to search for singularities. Right now, the algorithm results in a set of cubes containing the curve or surface. For regular surfaces, we can use a marching cubes type of algorithm[4] to compute an approximate triangulation of the surface. The problem is how to deal with singular cases. I am currently working on how to deal with this problem.

Extensions to dimensions higher than 3 are left to the reader. This approach is not practical in very high-dimensional space, though, because each box must be subdivided into $2^n$ subboxes, where $n$ is the dimension. Storage problems could arise easily. As one of the reviewers pointed out, you can also extend this work from squares and cubes to rectangles and cuboids. Then, binary rather than octree division could be performed. This would conform more tightly to the zeros of the polynomials with less division, which would somewhat ameliorate the storage problem when working in higher dimensions. ❑

## References

1. C.L. Bajaj et al., "Tracing Surface Intersections," *Computer-Aided Geometric Design*, Vol. 5, No. 4, Nov. 1988, pp. 285-307.
2. D.S. Arnon, "Topologically Reliable Display of Algebraic Curves," *Computer Graphics*, Vol. 17, No. 3, July 1983, pp. 219-227.
3. E.L. Allgower and P.H. Schmidt, "An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold," *SIAM J. Numerical Analysis*, Vol. 22, No. 2, Apr. 1985, pp. 322-346.
4. W. Lorenson and H. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Computer Graphics* (Proc. Siggraph), Vol. 21, No. 4, July 1987, pp. 163-169.
5. E.L. Allgower and S. Gnutzmann, "Simplicial Pivoting for Mesh Generation of Implicitly Defined Surfaces," *Computer-Aided Geometric Design*, Vol. 8, No. 4, Oct. 1991, pp. 305-325.
6. M. Hall and J. Warren, "Adaptive Polygonization of Implicitly Defined Surfaces," *IEEE CG&A*, Vol. 10, No. 6, Nov. 1990, pp. 33-42.
7. G. Taubin, "Distance Approximations for Rasterizing Implicit Curves," *ACM Trans. Graphics*, Vol. 13, No. 1, Jan. 1994 (to appear).
8. P. Pedersen, *Counting Real Zeros*, doctoral thesis, Computer Science Dept., New York Univ., New York, 1991.
9. R.T. Farouki and V.T. Rajan, "On the Numerical Condition of Berstain Polynomials," *Computer-Aided Geometric Design*, Vol. 4, No. 3, Sept. 1987, pp. 191-216.
10. T. Duff, "Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry," *Computer Graphics* (Proc. Siggraph), Vol. 26, No. 2, July 1992, pp. 131-138.
11. A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.
12. R. Walker, *Algebraic Curves*, Princeton Univ. Press, Princeton, N.J., 1950.

**Gabriel Taubin** is a research staff member at the IBM T.J. Watson Research Center, where he is part of the Exploratory Computer Vision Group. In the recent past, his research efforts were using algebraic curves and surfaces and algebraic invariants as building blocks for 2D and 3D object recognition systems. His current interests also include graphics and applications of computer vision methods to visualization problems. Taubin obtained his Licenciado en Ciencias Matematicas degree in pure mathematics from the University of Buenos Aires, Argentina, and his PhD in electrical engineering from Brown University, Providence, R.I.

Readers can contact Taubin at IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, e-mail taubin@watson.ibm.com.