



Curve-Drawing Algorithms for Raster Displays

JERRY VAN AKEN and MARK NOVAK

Texas Instruments, Inc.

The *midpoint* method for deriving efficient scan-conversion algorithms to draw geometric curves on raster displays is described. The method is general and is used to transform the nonparametric equation $f(x, y) = 0$, which describes the curve, into an algorithm that draws the curve. Floating-point arithmetic and time-consuming operations such as multiplies are avoided. The maximum error of the digital approximation produced by the algorithm is one-half the distance between two adjacent pixels on the display grid. The midpoint method is compared with the *two-point* method used by Bresenham, and is seen to be more accurate (in terms of the linear error) in the general case, without increasing the amount of computation required. The use of the midpoint method is illustrated with examples of lines, circles, and ellipses. The considerations involved in using the method to derive algorithms for drawing more general classes of curves are discussed.

Categories and Subject Descriptors: I.3.3 [Computing Methodologies]: Computer Graphics—Picture/image generation

General Terms: Algorithms

Additional Key Words and Phrases: Display algorithms, curve representations

1. INTRODUCTION

In this paper, we describe the midpoint method for converting a nonparametric equation for a curve into a scan-conversion algorithm that draws the curve into a bit-mapped frame buffer that drives a raster display. The goals in designing a curve-drawing algorithm are (1) to achieve a representation that closely approximates the true curve, while (2) reducing the amount of computation required to plot each point. The midpoint method produces algorithms that are computationally efficient, eliminating the need for floating-point arithmetic, and eliminating or significantly reducing the number of multiplication operations required. The maximum error of the digital approximation of the curve, drawn by an algorithm based on the midpoint method, is bounded at half the distance between vertically or horizontally adjacent pixels on the raster display grid if an 8-way-stepping algorithm is used or is bounded at half the distance between diagonally adjacent pixels if a 4-way-stepping algorithm is used.

The method presented here is general and can be applied in straightforward fashion to derive efficient algorithms for drawing various curves. Examples are

Author's address: Texas Instruments, Inc., P.O. Box 1443, Houston, TX 77001

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1985 ACM 0730-0301/85/0400-0147 \$00.75

presented to demonstrate the derivation of algorithms for drawing lines, circles, and ellipses. Finally, some of the limitations of the method, as understood by the authors, are discussed.

A significant body of work in curve-drawing algorithms for raster display devices has already been published. Algorithms based on various empirical methods have been described for approximating curves, such as lines and circles to varying degrees of accuracy and computational efficiency [3, 4, 6, 8]. The relative accuracies of several algorithms are determined experimentally in [10]. Bresenham [1] solved the problem of constructing an algorithm to draw the best-fit approximation to an arbitrary straight line. Pitteway [8] described an algorithm for drawing a general conic section based on a technique to be referred to in this paper as the midpoint method.¹ Horn [5] used a similar technique to derive an algorithm for the special case of a circle. Jordan, Lennon and Holm [6] described algorithms for drawing various conic sections based on a technique to be referred to in this paper as the two-point method. Bresenham [2] used a similar technique to derive a circle-drawing algorithm, and proved that the algorithm gave best-fit accuracy in the case of integer radii. McIlroy [7] extended the proof of best-fit accuracy of Bresenham's algorithm to the more general case in which the square of the circle's radius is an integer. The same paper compared the two approaches used in the circle-drawing algorithms of Bresenham and Horn. A comparison of the midpoint method and two-point method was presented by one of the authors [11] for the case of an ellipse in standard position. The superior accuracy of the midpoint method was demonstrated.

BASIC CONCEPTS

This paper discusses the derivation of algorithms for drawing curves described by nonparametric equations of the form $f(x, y) = 0$, where $f(x, y)$ is a polynomial in x and y . The fundamental problem to be solved in developing an incremental curve-drawing algorithm is to determine at each step of the plotting process which of a pair of pixels lies closer to the curve being approximated. The 8-way-stepping ellipse-drawing algorithm presented later in this paper in fact selects between more than two pixels at each step, but does so by considering the pixels a pair at a time.

The plotting of a straight line on a raster display is depicted in Figure 1a and b. The small circles represent pixels on the screen of a raster display, and the filled circles are those pixels of the display that are turned on to represent the line. Owing to the discrete nature of the raster display, the representation can only be an approximation to the true line. Both Figure 1a and b represent the same line but have been drawn by two different line-drawing algorithms that have selected slightly different sets of pixels to represent the line. Assume that each line in Figure 1 is plotted beginning at the lower left corner of the figure. At each step of the plotting process, the line-drawing algorithm must decide

¹ In a previous paper [11], one of the authors incorrectly reported that the accuracy of the algorithms of Pitteway [8] and Horn [5] had not been demonstrated for the general case. In fact, the error of the algorithm in both instances is bounded at $1/2$ the distance between adjacent pixels on the display grid. The author regrets the inaccuracy, and is indebted to M.L.V. Pitteway for bringing the problem to his attention.

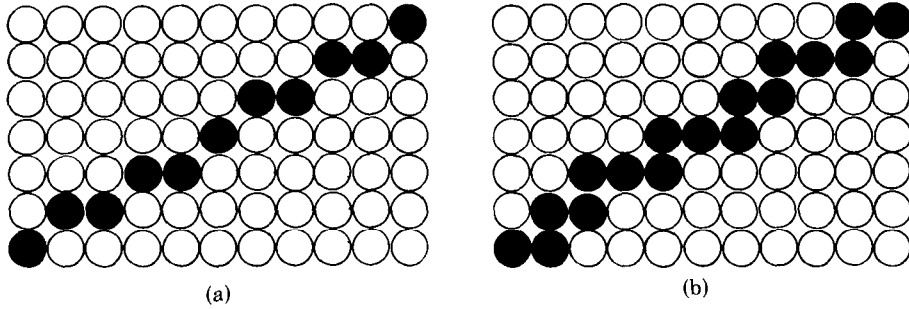


Fig. 1. Two ways of representing a line on a raster display.

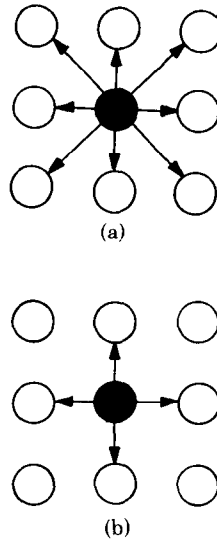


Fig. 2. Comparison of (a) 8-way and (b) 4-way stepping.

where the next pixel is to be plotted. For the example in Figure 1a, the next pixel is located either directly to the right of the current position, or upward and to the right. In Figure 1b, the next pixel is located directly to the right or directly above the current position.

Two categories of curve-drawing algorithms are treated in this paper; *8-way stepping* (8WS) and *4-way stepping* (4WS). Figure 2 demonstrates how 8WS and 4WS algorithms differ. As indicated in Figure 2b, 4WS algorithms are constrained to step in one of four directions: vertically, up or down; or horizontally, left or right. As indicated in Figure 2a, 8WS algorithms can step in any of 8 directions, that is, diagonally, as well as horizontally or vertically. The line in Figure 1a is drawn by an 8WS algorithm, and the line in Figure 1b by a 4WS algorithm.

Figure 3a represents the decision that must be made at each step of the 8WS line-drawing algorithm that generates the line approximation shown in Figure 1a. Pixel A was drawn during the previous step, and a decision must be made

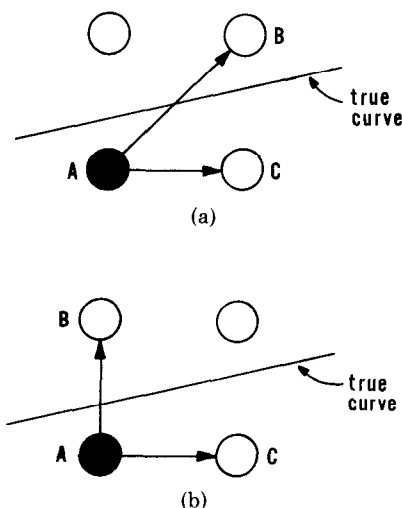


Fig. 3. Comparison of decision processes for (a) 8WS and (b) 4WS algorithms.

whether to draw pixel B or C next. Accuracy is maintained to the extent that the algorithm is capable of selecting the pixel, B or C , that lies closer to the line (or in the general case, the curve) being approximated. The corresponding decision process for the 4WS line-drawing algorithm used to draw the line in Figure 1b is shown in Figure 3b.

Bresenham's circle-drawing algorithm [2] draws an 8WS approximation using an error-control technique that is referred to here as the *two-point method*. Using the example of Figure 3a for reference, the *two-point method* for selecting between two pixels B and C at each step is expressed in general terms as follows: The equation for the curve being approximated, $f(x, y) = 0$, is evaluated at both B and C , where B and C are located at coordinates (x_B, y_B) and (x_C, y_C) , respectively. The values $f(x_B, y_B)$ and $f(x_C, y_C)$ will both be nonzero unless (at least) one of them is fortunate enough to lie directly on the curve $f(x, y) = 0$. The magnitudes of $f(x_B, y_B)$ and $f(x_C, y_C)$ can be taken as an indication of how far pixels B and C are from the curve; that is, the greater the magnitude, the more distant the pixel. If $|f(x_B, y_B)| < |f(x_C, y_C)|$, then pixel B is assumed to be closer. Since $f(x_B, y_B)$ and $f(x_C, y_C)$ are nonlinear indicators of distance for curves other than straight lines, the accuracy of this assumption is not immediately apparent. However, Bresenham proves that his 8WS circle-drawing algorithm, by minimizing this nonlinear error term, minimizes the resulting linear error as well. (This proof does not hold for certain noninteger radii [2, 7].) Whether a similar proof can be constructed for any curve other than an 8WS circle seems to remain an open question. In general, although the two-point method can be extended to curves other than lines and circles, the fact that error terms $f(x_B, y_B)$ and $f(x_C, y_C)$ are nonlinear makes them unreliable indicators of linear error.

An alternate technique, called the midpoint method, is used in this paper as a means of controlling the linear error at each step of the curve-drawing algorithm. Again using the example of Figure 3a as a reference, the approach is to evaluate

the equation for the curve, $f(x, y) = 0$, at the point M located midway between pixels B and C . The coordinates of midpoint M are

$$(xM, yM) = \left(\frac{x_B + x_C}{2}, \frac{y_B + y_C}{2} \right).$$

Should the curve $f(x, y) = 0$ pass directly through midpoint M , then $f(xM, yM) = 0$. The intersection of the curve with an imaginary line drawn from pixel B to pixel C occurs at a point equally distant from B and C and represents the worst-case error that can be generated by the algorithm. In the more likely event that the curve passes to one side or the other of M , $f(xM, yM)$ will be nonzero and its sign will indicate to which side of M it passes. If the sign of $f(xM, yM)$ indicates that the curve passes to the side of M that is closer to B , then B is selected by the algorithm; otherwise, C is selected. In the following discussion, $f(xM, yM)$ will be called the *decision variable* of the algorithm.

Given that the midpoint method is capable of determining to which side of midpoint M the curve passes, the maximum linear error at each step, E_{\max} , must be bounded at one-half the distance between pixels B and C . The convention used in this paper is that the horizontal distance between horizontally or vertically adjacent pixels on the display grid is unity. For 8WS algorithms, this means that $|E_{\max}| \leq 1/2$, and for 4WS algorithms, $|E_{\max}| \leq \sqrt{2}/2$. In contrast to Bresenham's approach, these bounds are valid for curves other than lines and circles. Given the improved control over E_{\max} afforded by the midpoint method, it is the method used exclusively in this paper.

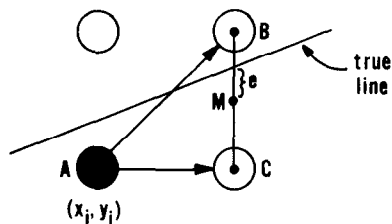
From the discussion above, an 8WS curve-drawing algorithm can typically be expected to produce a more accurate approximation to a given curve than its 4WS counterpart. In practice, curves drawn by 4WS algorithms tend to appear more jagged than those drawn by 8WS algorithms. One potential advantage of 4WS algorithms is that they tend to be simpler, that is, require fewer computations per step, than their 8WS counterparts. (Comparison of the two ellipse-drawing algorithms presented later in this paper illustrates this tendency.) This advantage may be outweighed by the fact that a 4WS algorithm will, in general, plot more pixels than an 8WS algorithm drawing the same curve.

In the following sections, several examples of 8WS and 4WS algorithms are derived to illustrate the method. Some of these have appeared in only slightly different form elsewhere (see references) but are described here in a uniform manner for the sake of illustration. The emphasis is on 8WS algorithms because of their superior accuracy. The curves used as examples are all conic sections that are symmetric about the x axis or the y axis, or both. The general case introduces additional considerations which are discussed briefly.

8WS LINE-DRAWING ALGORITHM

The first example is the derivation of an 8WS algorithm for drawing a straight line similar to that of Figure 1. The resulting algorithm will not differ from Bresenham's [1], but it will serve as a simple illustration of our method. Some simplifying assumptions will be helpful. Assume that the line is drawn from the origin to endpoint (a, b) , where $a \geq b \geq 0$, and a and b are integers. The equation for the line can be expressed as $f(x, y) = bx - ay = 0$. These assumptions will not

Fig. 4. Decision process for 8WS line in octant $x \geq y \geq 0$.



detract from the generality of the resulting algorithm, since symmetry can be used to draw lines in octants other than the one for which $x \geq y \geq 0$, and a simple coordinate transformation will permit the line to begin at points other than the origin.

The decision process at each step of the line-drawing algorithm is represented in Figure 3a and is shown in more detail in Figure 4. In Figure 4, the displacement e of the line $f(x, y) = 0$ from midpoint M is indicated for the case in which the line passes above the midpoint. Pixel A , representing the current position during the i th step of the algorithm (where $i = 0, 1, \dots$), is assigned coordinates (x_i, y_i) . This places pixel B at $(x_i + 1, y_i + 1)$, pixel C at $(x_i + 1, y_i)$, and midpoint M at $(x_i + 1, y_i + \frac{1}{2})$. The point at which line $f(x, y) = 0$ crosses the line between B and C is therefore $(x_i + 1, y_i + \frac{1}{2} + e)$, and the value of $f(x, y)$ at this point must be zero:

$$\begin{aligned} f(x_i + 1, y_i + \tfrac{1}{2} + e) &= b(x_i + 1) - a(y_i + \tfrac{1}{2} + e) \\ &= b(x_i + 1) - a(y_i + \tfrac{1}{2}) - ae \\ &= f(x_i + 1, y_i + \tfrac{1}{2}) - ae \\ &= 0. \end{aligned}$$

The term $f(x_i + 1, y_i + \frac{1}{2})$ is designated the decision variable, d_i , of the algorithm. Rearranging the last equation above yields the relationship

$$\begin{aligned} d_i &= f(x_i + 1, y_i + \tfrac{1}{2}) \\ &= ae. \end{aligned}$$

The sign of d_i is the same as the sign of e and determines to which side of midpoint M the line $f(x, y) = 0$ falls. If the line passes above M , then $e > 0$, $d_i > 0$ and pixel B is selected as lying closer to the line. If the line passes below M , then $e < 0$, $d_i < 0$ and pixel C is selected. If the line passes directly through M , then $e = d_i = 0$, and pixel B is arbitrarily selected.

A relatively inefficient form of line-drawing algorithm, based on the decision variable described above, is presented in Figure 5. The algorithm is written in the Pascal programming language. The function *setpixel*(x, y) turns on the pixel located at device coordinates (x, y) . The drawback with this particular form of the algorithm is that at each step the calculation of the value of decision variable d requires two multiply operations.

A more efficient approach is to calculate d incrementally, such that its value at each step is calculated in terms of its value at the previous step. The initial

```

x := 0;
y := 0;
for i := 0 to a do
  begin
    setpixel(x, y);
    d := b * (x + 1) - a * (y + 1/2);
    if d < 0 then      (* step to pixel C *)
      x := x + 1
    else              (* step to pixel B *)
      begin
        x := x + 1;
        y := y + 1
      end
    end
  end
end

```

Fig. 5. Inefficient form of 8WS line-drawing algorithm.

value of decision variable d is a function of the initial x and y values, $x_0 = 0$ and $y_0 = 0$:

$$\begin{aligned}
 d_0 &= f(x_0 + 1, y_0 + \tfrac{1}{2}) \\
 &= b(x_0 + 1) - a(y_0 + \tfrac{1}{2}) \\
 &= b - \frac{a}{2}.
 \end{aligned}$$

The value of d at each subsequent step depends on which of the two pixels B and C is selected at that step. For example, if pixel B is selected during the i th iteration, then $x_{i+1} = x_i + 1$, and $y_{i+1} = y_i + 1$, and d_{i+1} are calculated in terms of d_i as follows:

$$\begin{aligned}
 d_{i+1} &= f(x_{i+1} + 1, y_{i+1} + \tfrac{1}{2}) \\
 &= b(x_{i+1} + 1) - a(y_{i+1} + \tfrac{1}{2}) \\
 &= b((x_i + 1) + 1) - a((y_i + 1) + \tfrac{1}{2}) \\
 &= f(x_i + 1, y_i + \tfrac{1}{2}) - a + b \\
 &= d_i - a + b.
 \end{aligned}$$

Similarly, if pixel C is selected instead, then $x_{i+1} = x_i + 1$, $y_{i+1} = y_i$, and d_{i+1} is calculated in terms of d_i as follows:

$$\begin{aligned}
 d_{i+1} &= f(x_{i+1} + 1, y_{i+1} + \tfrac{1}{2}) \\
 &= b(x_{i+1} + 1) - a(y_{i+1} + \tfrac{1}{2}) \\
 &= b((x_i + 1) + 1) - a(y_i + \tfrac{1}{2}) \\
 &= d_i + b.
 \end{aligned}$$

The above expressions are used to form the more efficient line-drawing algorithm shown in Figure 6, which requires no multiplies. Assuming that a and b are both integers, only integer add and subtract operations are required. The division of a by 2 can be eliminated by substituting $d' = 2d$ for d . The signs of d

```

x := 0;
y := 0;
d := b - a/2;
for i := 0 to a do
  begin
    setpixel(x, y);
    if d < 0 then      (* step to pixel C *)
      begin
        x := x + 1;
        d := d + b
      end
    else              (* step to pixel B *)
      begin
        x := x + 1;
        y := y + 1;
        d := d - a + b
      end
    end
  end
end

```

Fig. 6. Efficient form of 8WS line-drawing algorithm.

and d' are the same, and the line drawn by the algorithm is unaffected by the substitution. Multiplies by 2 are converted to simple shift operations. To reduce the number of computations per step, the statement " $d := d - a + b$ " within the loop can be replaced with " $d := d + t$ " where t is a variable assigned the value $b - a$. The other curve-drawing algorithms presented in this paper have similar properties, and the obvious optimizations are omitted for the sake of brevity.

8WS CIRCLE-DRAWING ALGORITHM

The methodology developed in the derivation of the line-drawing algorithm can be applied to the derivation of algorithms for other types of curves that can be represented by polynomial equations. The next example is the circle represented by the equation $f(x, y) = x^2 + y^2 - r^2 = 0$ where r is the radius. For simplicity, the center lies at the origin, and only the arc contained in the octant $x \geq y \geq 0$ is drawn. As before, the resulting algorithm can be extended to the general case using symmetry and coordinate transformation.

Figure 7 shows an arc of a circle of radius $r = 10$ drawn on a raster display. The starting point of the algorithm is the pixel located at (10, 0). With each subsequent step, the algorithm selects either the pixel located directly above or the pixel above and to the left of the current pixel.

The decision process at each step is indicated in Figure 8. The current position is pixel A, located at (x_i, y_i) . The next will be either pixel B, located at $(x_i - 1, y_i + 1)$, or pixel C, located at $(x_i, y_i + 1)$. Midpoint M, which bisects the line from B to C, is located at $(x_i - \frac{1}{2}, y_i + 1)$. Circle $f(x, y) = 0$ intersects the line between B and C at a distance e from M. The equation for the circle at that point is

$$\begin{aligned}
 f(x_i - \tfrac{1}{2} + e, y_i + 1) &= (x_i - \tfrac{1}{2} + e)^2 + (y_i + 1)^2 - r^2 \\
 &= (x_i - \tfrac{1}{2})^2 + (y_i + 1)^2 - r^2 + 2(x_i - \tfrac{1}{2})e + e^2 \\
 &= f(x_i - \tfrac{1}{2}, y_i + 1) + 2(x_i - \tfrac{1}{2})e + e^2 \\
 &= 0.
 \end{aligned}$$

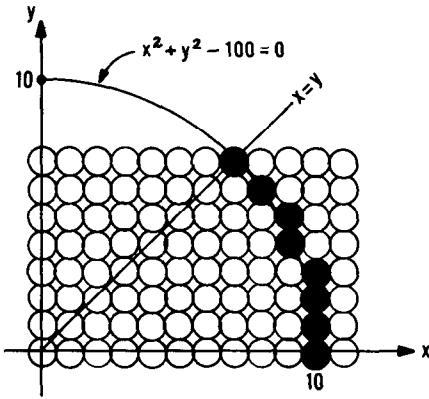
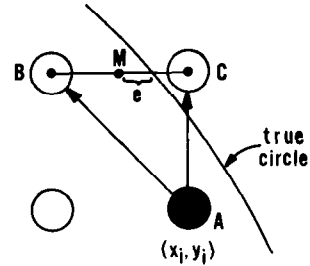


Fig. 7. Circular arc drawn on raster display.

Fig. 8. Decision process for 8WS circle in octant $x \geq y \geq 0$.

The term $f(x_i - \frac{1}{2}, y_i + 1)$ is designated the decision variable, d_i , of the algorithm. Rearranging the last equation above yields the relationship

$$\begin{aligned} d_i &= f(x_i - \tfrac{1}{2}, y_i + 1) \\ &= -2(x_i - \tfrac{1}{2})e - e^2. \end{aligned}$$

As in the previous example, the sign of d_i indicates to which side of midpoint M the circle $f(x, y) = 0$ passes, and determines whether pixel B or pixel C lies closer to the circle. If the circle passes to the left of M , then $e < 0$, $d_i > 0$ and pixel B will be drawn next. If the circle passes to the right of M , then $e > 0$, $d_i < 0$ and pixel C will be drawn next. If $e = d_i = 0$, then pixel B is selected arbitrarily.

Rather than calculate the value of decision variable d explicitly at each step, we calculate it incrementally in terms of its value at the previous step. The initial value of d is determined from the initial values $x_0 = r$ and $y_0 = 0$ as follows:

$$\begin{aligned} d_0 &= f(x_0 - \tfrac{1}{2}, y_0 + 1) \\ &= (r - \tfrac{1}{2})^2 + (0 + 1)^2 - r^2 \\ &= \tfrac{5}{4} - r. \end{aligned}$$

```

x := trunc(r + 1/2);
y := 0;
d := 5/4 - r;
repeat
  setpixel(x, y);
  if d < 0 then          (* step to pixel C *)
    begin
      y := y + 1;
      d := d + 2 * y + 1
    end
  else                  (* step to pixel B *)
    begin
      x := x - 1;
      y := y + 1;
      d := d - 2 * x + 2 * y + 1
    end
until x < y

```

Fig. 9. An 8WS circle-drawing algorithm.

Assuming that pixel *B* is selected during the *i*th iteration, then $x_{i+1} = x_i - 1$, $y_{i+1} = y_i + 1$, and the value d_{i+1} is calculated in terms of d_i as follows:

$$\begin{aligned}
 d_{i+1} &= f(x_{i+1} - \tfrac{1}{2}, y_{i+1} + 1) \\
 &= (x_{i+1} - \tfrac{1}{2})^2 + (y_{i+1} + 1)^2 - r^2 \\
 &= ((x_i - 1) - \tfrac{1}{2})^2 + ((y_i + 1) + 1)^2 - r^2 \\
 &= (x_i - \tfrac{1}{2})^2 + (y_i + 1)^2 - r^2 - 2(x_i - \tfrac{1}{2}) + 2(y_i + 1) + 2 \\
 &= f(x_i - \tfrac{1}{2}, y_i + 1) - 2(x_i - 1) + 2(y_i + 1) + 1 \\
 &= d_i - 2x_{i+1} + 2y_{i+1} + 1.
 \end{aligned}$$

Similarly, if pixel *C* is chosen instead, then $x_{i+1} = x_i$, $y_{i+1} = y_i + 1$, and $d_{i+1} = d_i + 2y_{i+1} + 1$.

The expressions above are utilized to form the 8WS circle-drawing algorithm shown in Figure 9. While *x* and *y* are constrained to be integers, the value of *r* may be real, although the programmer may wish to constrain *r* also to integer values for the sake of faster computation. When *r* is an integer, the initial value of *d* can be truncated to $1 - r$ without affecting the circle drawn by the algorithm. This observation has been made by Horn [5]. In fact, when *r* is an integer, *d* can be assigned an initial value in the range $1 - r \leq d < 2 - r$ without altering the circle drawn by the algorithm. For instance, a circle-drawing algorithm similar to that in Figure 9, but derived using the two-point method, assigns *d* the initial value $\frac{3}{2} - r$. This means that the algorithm described by Bresenham [2], which is based on the two-point method, draws the same circle as the algorithm in Figure 9 when *r* is an integer.

8WS ELLIPSE-DRAWING ALGORITHM

In addition to lines and circles, another useful curve in graphics applications is the ellipse. On certain raster displays, for instance, the distances between adjacent pixels differ in the horizontal and vertical directions. The figure drawn on such

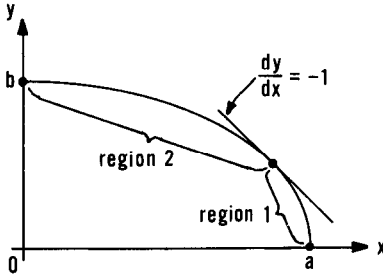


Fig. 10. Ellipse partitioned into two regions.

a display by the circle-drawing algorithm of Figure 9 will appear to the viewer as an ellipse rather than a circle. To compensate for the distortion, an ellipse must be drawn in the display device's coordinate space such that the resulting figure will appear circular to the viewer. In the next example, an algorithm is developed to draw an ellipse whose major and minor axes are parallel to the x and y axes.

For simplicity, the ellipse is centered at the origin, and only the portion of the ellipse in the quadrant $x \geq 0, y \geq 0$ is drawn. As in the previous examples, the resulting algorithm can be made more general using symmetry and translation. The equation for the ellipse is

$$\begin{aligned} f(x, y) &= b^2x^2 + a^2y^2 - a^2b^2 \\ &= 0, \end{aligned}$$

where $2a$ is the diameter of the ellipse along the x axis, and $2b$ is the diameter along the y axis. Assuming that a and b are integer values, the first pixel drawn by the algorithm is located at $(a, 0)$. Subsequently, the algorithm continues stepping in a counterclockwise direction until it reaches the pixel at $(0, b)$.

The algorithm partitions the arc of the ellipse within the quadrant $x > 0, y \geq 0$ into two regions, with the boundary between regions being the point at which the tangent to the ellipse has the value -1 . This is indicated in Figure 10. To the right of this point, identified as region 1, the tangent is less than -1 .

In drawing the portion of the curve contained within region 1 of Figure 10, a decision must be made at each step to choose one of two pixels, indicated as C and D in Figure 11a. Pixel A is darkened to indicate that it has been selected by the algorithm in the previous step.

Over the region of the ellipse for which the tangent is greater than -1 , identified as region 2 in Figure 10, the algorithm selects one of the two pixels identified as B and C in Figure 11b.

Using the convention that pixel A is located at (x_i, y_i) again, the device coordinates for pixels B, C , and D are $(x_i - 1, y_i)$, $(x_i - 1, y_i + 1)$, and $(x_i, y_i + 1)$, respectively. Two decision variables, $d1$ and $d2$, will be maintained by the algorithm to make the two decisions represented in Figure 11a and b.

While traversing region 1 of the ellipse, the decision variable used to decide between pixels C and D in Figure 11a is $d1$. At the point where the ellipse $f(x, y)$

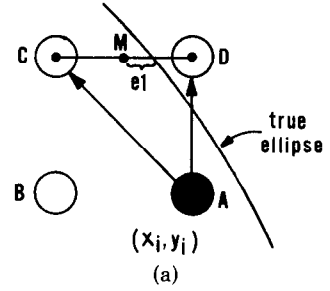
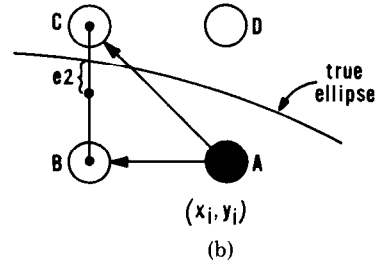


Fig. 11. Decision processes in (a) region 1 and (b) region 2 of 8WS ellipse.



$= 0$ intersects the line connecting C and D ,

$$\begin{aligned} f(x_i - \tfrac{1}{2} + e1, y_i + 1) &= b^2(x_i - \tfrac{1}{2} + e1)^2 + a^2(y_i + 1)^2 - a^2b^2 \\ &= f(x_i - \tfrac{1}{2}, y_i + 1) + 2b^2(x_i - \tfrac{1}{2})e1 + b^2e1^2 \\ &= 0, \end{aligned}$$

where $e1$ represents the displacement of the point of intersection from the midpoint of pixels C and D , located at $(x_i - \frac{1}{2}, y_i + 1)$. Decision variable $d1_i$ is defined as $2f(x_i - \frac{1}{2}, y_i + 1)$. (The factor of 2 does not affect the sign of $d1$, and is included for the sake of consistency with the form of the algorithm presented in [11].) From the equation above, $d1_i$ is related to $e1$ as follows:

$$\begin{aligned} d1_i &= 2f(x_i - \tfrac{1}{2}, y_i + 1) \\ &= -2(2b^2(x_i - \tfrac{1}{2})e1 + b^2e1^2). \end{aligned}$$

As in previous examples, the sign of $d1_i$ indicates to which side of the midpoint the intersection with $f(x, y) = 0$ occurs. If the intersection lies to the right of the midpoint, then $e1 > 0$, $d1_i < 0$, and pixel D is selected. Otherwise, either pixel B or C is selected.

The decision variable used to select between pixels B and C in Figure 11b is $d2$. The equation for the ellipse at the point of intersection with the line connecting B and C is

$$\begin{aligned} f(x_i - 1, y_i + \tfrac{1}{2} + e2) &= b^2(x_i - 1)^2 + a^2(y_i + \tfrac{1}{2} + e2)^2 - a^2b^2 \\ &= f(x_i - 1, y_i + \tfrac{1}{2}) + 2a^2(y_i + \tfrac{1}{2})e2 + a^2e2^2 \\ &= 0, \end{aligned}$$

where $e2$ represents the displacement of the point of intersection from the midpoint of pixels B and C , located at $(x_i - 1, y_i + \frac{1}{2})$. Decision variable $d2_i$ is defined as $2f(x_i - 1, y_i + \frac{1}{2})$, and from the last equation above, $d2$ is related to $e2$ as follows:

$$\begin{aligned} d2_i &= 2f(x_i - 1, y_i + \frac{1}{2}) \\ &= -2(2a^2(y_i + \frac{1}{2})e2 + a^2e2^2). \end{aligned}$$

If the intersection occurs at or below the midpoint, then $e2 \leq 0$, $d2 \geq 0$, and pixel B is selected. Otherwise pixel C is selected.

Decision variables $d1$ and $d2$ can be calculated incrementally to reduce the amount of computation per step. The initial value of $d1$ is calculated in terms of $x_0 = a$ and $y_0 = 0$ as follows:

$$\begin{aligned} d1_0 &= 2f(x_0 - \frac{1}{2}, y_0 + 1) \\ &= 2a^2 - 2ab^2 + \frac{b^2}{2}. \end{aligned}$$

If pixel D is selected during the i th iteration, then $x_{i+1} = x_i$ and $y_{i+1} = y_i + 1$. The value of d_{i+1} is calculated in terms of d_i as follows:

$$\begin{aligned} d1_{i+1} &= 2f(x_{i+1} - \frac{1}{2}, y_{i+1} + 1) \\ &= 2b^2(x_{i+1} - \frac{1}{2})^2 + 2a^2(y_{i+1} + 1) - 2a^2b^2 \\ &= 2b^2(x_i - \frac{1}{2})^2 + 2a^2((y_i + 1) + 1) - 2a^2b^2 \\ &= 2f(x_i - \frac{1}{2}, y_i + 1) + 4a^2(y_i + 1) + 2a^2 \\ &= d1_i + 4a^2y_{i+1} + 2a^2. \end{aligned}$$

Similarly, if pixel C is selected, then $x_{i+1} = x_i - 1$, $y_{i+1} = y_i + 1$, and

$$d1_{i+1} = d1_i - 4b^2x_{i+1} + 4a^2y_{i+1} + 2a^2.$$

After pixel B has been selected once by the algorithm, updates to $d1$ are no longer necessary—this occurs when region 2 of the ellipse (Figure 10) is entered.

Updates to decision variable $d2$ are calculated in similar fashion. The initial value of $d2$ is calculated as

$$\begin{aligned} d2_0 &= 2f(x_0 - 1, y_0 + \frac{1}{2}) \\ &= \frac{a^2}{2} - 4ab^2 + 2b^2. \end{aligned}$$

If pixel D is selected during the i th iteration, $d2$ is updated as follows:

$$d2_{i+1} = d2_i + 4a^2y_{i+1}.$$

If pixel C is selected, $d2$ is updated as follows:

$$d2_{i+1} = d2_i - 4b^2x_{i+1} + 4a^2y_{i+1} + 2b^2.$$

If pixel B is selected, $d2$ is updated as follows:

$$d2_{i+1} = d2_i - 4b^2x_{i+1} + 2b^2.$$

```

x := trunc(a + 1/2); y := 0;
d1 := 2 * a * a - 2 * a * b * b + b * b/2;
d2 := a * a/2 - 4 * a * b * b + 2 * b * b;
while d2 < 0 do          (* region 1 of ellipse *)
  begin
    setpixel(x, y);      (* turn on pixel at (x, y) *)
    if d1 < 0 then        (* step to pixel D *)
      begin
        y := y + 1;
        d1 := d1 + 4 * a * a * y + 2 * a * a;
        d2 := d2 + 4 * a * a * y
      end
    else                  (* step to pixel C *)
      begin
        x := x - 1;
        y := y + 1;
        d1 := d1 - 4 * b * b * x + 4 * a * a * y + 2 * a * a;
        d2 := d2 - 4 * b * b * x + 2 * b * b
          + 4 * a * a * y
      end
    end;
  repeat                (* region 2 of ellipse *)
    setpixel(x, y);      (* turn on pixel at (x, y) *)
    if d2 < 0 then        (* step to pixel C *)
      begin
        x := x - 1;
        y := y + 1;
        d2 := d2 - 4 * b * b * x + 2 * b * b
          + 4 * a * a * y
      end
    else                  (* step to pixel B *)
      begin
        x := x - 1;
        d2 := d2 - 4 * b * b * x + 2 * b * b
      end
    end
  until x < 0

```

Fig. 12. Inefficient form of 8WS ellipse-drawing algorithm.

While traversing region 1, the value of $d2$ is maintained for the sole purpose of detecting the transition into region 2.

A relatively inefficient ellipse-drawing algorithm based on the above expressions for decision variables $d1$ and $d2$ appears in Figure 12. The algorithm can be made more efficient, however, by replacing most of the multiplies with incremental calculations. The strategy is to replace a sequence such as

```

x := x + 1;
d := d + q*x

```

occurring within the main loop of the algorithm with the equivalent sequence

```

x := x + 1;
t := t + q;
d := d + t

```

```

x := trunc(a + 1/2); y := 0;
t1 := a * a; t2 := 2 * t1; t3 := 2 * t2;
t4 := b * b; t5 := 2 * t4; t6 := 2 * t5;
t7 := a * t5; t8 := 2 * t7; t9 := 0;
d1 := t2 + t7 + t4/2; d2 := t1/2 + t8 + t5;
while d2 < 0 do (* region 1 of ellipse *)
  begin
    setpixel(x, y); (* turn on pixel at (x, y) *)
    y := y + 1; (* increment y regardless *)
    t9 := t9 + t3;
    if d1 < 0 then (* step to pixel D *)
      begin
        d1 := d1 + t9 + t2;
        d2 := d2 + t9
      end
    else (* step to pixel C *)
      begin
        x := x - 1;
        t8 := t8 - t6;
        d1 := d1 - t8 + t9 + t2;
        d2 := d2 - t8 + t5 + t9
      end
    end;
  end;
repeat (* region 2 of ellipse *)
  setpixel(x, y); (* turn on pixel at (x, y) *)
  x := x - 1; (* decrement x regardless *)
  t8 := t8 - t6;
  if d2 < 0 then (* step to pixel C *)
    begin
      y := y + 1;
      t9 := t9 + t3;
      d2 := d2 - t8 + t5 + t9
    end
  else (* step to pixel B *)
    d2 := d2 - t8 + t5
  end;
until x < 0

```

Fig. 13. Efficient form of 8WS ellipse-drawing algorithm.

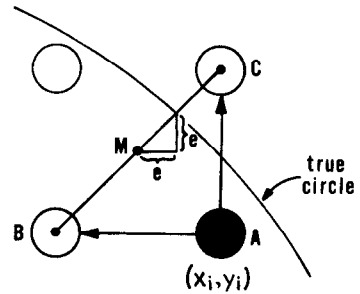
where the temporary variable t is introduced to store the incrementally calculated product of q and x from one loop to the next.

The improved ellipse-drawing algorithm is shown in Figure 13. Temporary variables $t1$ through $t9$ have been introduced to eliminate the need for multiply operations within the main loop of the routine, where

$$\begin{aligned}
 t1 &= a^2, & t2 &= 2a^2, & t3 &= 4a^2, \\
 t4 &= b^2, & t5 &= 2b^2, & t6 &= 4b^2, \\
 t7 &= 2ab^2, & t8 &= 4b^2x, & t9 &= 4a^2y.
 \end{aligned}$$

Although the ellipse-drawing algorithm is more complex than the circle-drawing algorithm developed previously, no multiplies are required within the two loops. The width and height of the ellipse are controlled by selecting the appropriate values for a and b , respectively. Although x and y are constrained to be integers, a and b can be allowed to be real numbers without exceeding the linear error

Fig. 14. Decision process for 4WS circle.



bound of $\frac{1}{2}$. The programmer may choose to constrain a and b to be integers for the sake of faster computation.

An iteration of the first loop in Figure 13 updates both $d1$ and $d2$, and performs more computation than the second loop, which updates only $d2$. This form of the algorithm is efficient when the number of pixels to be drawn in region 2 is greater than the number in region 1, as is the case in the example of Figure 10. When this is not the case, a second form of the algorithm that begins at $(0, b)$ and draws in a counterclockwise direction is more efficient. Symmetry is used to construct this form of the algorithm from that shown in Figure 13.

The derivation of the algorithm in Figure 13 is also presented in [11], along with a demonstration of the advantages of the midpoint method over the two-point method in controlling linear error.

SOME 4WS CURVE-DRAWING ALGORITHMS

The three examples presented thus far have all been 8WS curve-drawing algorithms. In this section three 4WS algorithms are presented for comparison with their 8WS counterparts. The first of these is a 4WS circle-drawing algorithm. For simplicity, the circle is centered about the origin, and only the arc lying in the quadrant $x \geq 0, y \geq 0$ is drawn. The equation for a circle of radius r centered about the origin is $f(x, y) = x^2 + y^2 - r^2 = 0$. Assuming that r is an integer, the circle-drawing algorithm begins by plotting the pixel located at $(r, 0)$, and proceeds in a counterclockwise direction until it reaches the pixel at $(0, r)$.

The decision process at each step of the 4WS circle-drawing algorithm is represented in Figure 14. Pixels A , B , and C are located at coordinates (x_i, y_i) , $(x_i - 1, y_i)$, and $(x_i, y_i + 1)$, respectively. The pixel last drawn is A , and the next to be drawn is B or C , whichever lies closer to the circle, as measured at the intersection with the line connecting B and C . Point M lies midway along the line connecting B and C and has coordinates $(x_i - \frac{1}{2}, y_i + \frac{1}{2})$. Circle $f(x, y) = 0$ intersects the line connecting B and C at coordinates $(x_i - \frac{1}{2} + e, y_i + \frac{1}{2} + e)$, located at a distance $e\sqrt{2}$ from M . The equation for the circle at the point of

intersection is

$$\begin{aligned}
 f(x_i - \tfrac{1}{2} + e, y_i + \tfrac{1}{2} + e) &= (x_i - \tfrac{1}{2} + e)^2 + (y_i + \tfrac{1}{2} + e)^2 - r^2 \\
 &= (x_i - \tfrac{1}{2})^2 + (y_i + \tfrac{1}{2})^2 - r^2 \\
 &\quad + 2(x_i - \tfrac{1}{2})e + 2(y_i + \tfrac{1}{2})e + 2e^2 \\
 &= f(x_i - \tfrac{1}{2}, y_i + \tfrac{1}{2}) \\
 &\quad + 2(x_i + y_i)e + 2e^2 \\
 &= 0.
 \end{aligned}$$

The term $f(x_i - \frac{1}{2}, y_i + \frac{1}{2})$ is designated the decision variable, d_i , of the algorithm. Rearranging the last equation above yields the relationship

$$\begin{aligned}
 d_i &= f(x_i - \tfrac{1}{2}, y_i + \tfrac{1}{2}) \\
 &= -2(x_i + y_i)e - 2e^2.
 \end{aligned}$$

The sign of d_i is opposite to the sign of displacement e and can be used to determine whether the intersection occurs closer to B or to C . If $d_i < 0$ then $e > 0$ and pixel C is selected; otherwise, pixel B is selected.

During the i th iteration, the value of d_{i+1} is calculated in terms of d_i , as in the previous examples. The initial value of d_1 is calculated from initial values $x_0 = r$ and $y_0 = 0$ as follows:

$$\begin{aligned}
 d_0 &= f(x_0 - \tfrac{1}{2}, y_0 + \tfrac{1}{2}) \\
 &= (r - \tfrac{1}{2})^2 + (0 + \tfrac{1}{2})^2 - r^2 \\
 &= \tfrac{1}{2} - r.
 \end{aligned}$$

If pixel B is selected, then $x_{i+1} = x_i - 1$, $y_{i+1} = y_i$, and d_{i+1} is calculated in terms of d_i as follows:

$$\begin{aligned}
 d_{i+1} &= f(x_{i+1} - \tfrac{1}{2}, y_{i+1} + \tfrac{1}{2}) \\
 &= ((x_i - 1) - \tfrac{1}{2})^2 + (y_i + \tfrac{1}{2})^2 - r^2 \\
 &= d_i - 2x_{i+1}.
 \end{aligned}$$

If pixel C is selected, then $x_{i+1} = x_i$, $y_{i+1} = y_i + 1$, and

$$d_{i+1} = d_i + 2y_{i+1}.$$

The above expressions are combined to form the 4WS circle-drawing algorithm in Figure 15.

To permit comparison with their 8WS counterparts, 4WS line- and ellipse-drawing algorithms are presented in Figures 16 and 17. Their derivations are similar to that of the 4WS circle-drawing algorithm of Figure 15, but are not presented here for the sake of brevity.

The line-drawing algorithm of Figure 16 draws a line from the origin to endpoint (a, b) , where $a \geq 0$ and $b \geq 0$. This form of the algorithm assumes that a and b are integers.

Fig. 15. A 4WS circle-drawing algorithm.

```

x := r;
y := 0;
d := 1/2 - r;
for i := 0 to 2 * r do
  begin
    setpixel(x, y);
    if d < 0 then      (* step to pixel C *)
      begin
        y := y + 1;
        d := d + 2 * y
      end
    else              (* step to pixel B *)
      begin
        x := x - 1;
        d := d - 2 * x
      end
    end
  end
end

```

```

x := 0;
y := 0;
d := b - a;
for i := 0 to a + b do
  begin
    setpixel(x, y);
    if d < 0 then      (* step to pixel C *)
      begin
        x := x + 1;
        d := d + 2 * b
      end
    else              (* step to pixel B *)
      begin
        y := y + 1;
        d := d - 2 * a
      end
    end
  end
end

```

Fig. 16. A 4WS line-drawing algorithm.

The ellipse-drawing algorithm of Figure 17 draws an ellipse centered at the origin, whose diameter along the x axis is $2a$ and whose diameter along the y axis is $2b$. This form of the algorithm assumes that a and b are integers. Only the arc of the ellipse lying in the quadrant $x \geq 0, y \geq 0$ is drawn by the algorithm. Temporary variables $t1$ – $t7$ represent the following quantities:

$$\begin{aligned}
 t1 &= a^2, & t2 &= 2a^2, & t3 &= b^2, & t4 &= 2b^2, \\
 t5 &= ab^2, & t6 &= 2b^2x, & t7 &= 2a^2y.
 \end{aligned}$$

These temporary variables are introduced for reasons similar to those described in the derivation of the 8WS ellipse-drawing algorithm in the preceding section.

BEST-FIT VERSUS BOUNDED ERROR

The tracking error of curve-drawing algorithms based on the midpoint method is known to be bounded at half the distance between two adjacent pixels. This is not to say that the midpoint method results in the best-fit approximation in all cases. (The term *best fit* is used here to mean that the linear error is minimized.)

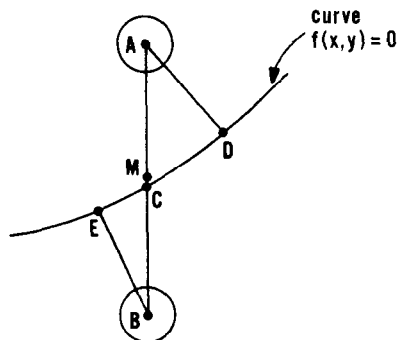
```

 $x := a;$     $y := 0;$ 
 $t1 := a * a;$    $t2 := 2 * t1;$ 
 $t3 := b * b;$    $t4 := 2 * t3;$ 
 $t5 := a * t3;$    $t6 := 2 * t5;$ 
 $t7 := 0;$    $d := (t1 + t3)/4 - t5;$ 
for  $i := 0$  to  $a + b$  do
  begin
    setpixel( $x, y$ );
    if  $d < 0$  then      (* step to pixel C *)
      begin
         $y := y + 1;$ 
         $t7 := t7 + t2;$ 
         $d := d + t7$ 
      end
    else                (* step to pixel B *)
      begin
         $x := x - 1;$ 
         $t6 := t6 - t4;$ 
         $d := d - t6$ 
      end
    end
  end
end

```

Fig. 17. A 4WS ellipse-drawing algorithm.

Fig. 18. The linear error at pixels A and B.



In some instances the algorithm may fail to select the pixel that lies closer to the curve.

Consider the situation shown in Figure 18. The curve $f(x, y) = 0$ intersects the line between pixels A and B at point C, just below midpoint M. The closest point on the curve to pixel A is D, and the closest point on the curve to pixel B is E. In the special case in which the curve is a straight line, $BC < AC$ always implies that $BE < AD$. In other words, the midpoint method always results in the best-fit approximation to a straight line. However, when the curve is not a straight line, the curvature of the function $f(x, y) = 0$ may be such that $BC < AC$ and $BE > AD$ occur simultaneously. This is unlikely, but possible. In such a case, the error of the midpoint method remains bounded as before but is not best fit.

This does not seem to be a very serious drawback, however. First, consider that the midpoint line-drawing algorithm always draws the best-fit straight line—the algorithm is in fact identical to Bresenham's [1], which is known to be best-fit. Second, given a section of a curve with a large radius of curvature, a microscopic view of the portion of the curve passing through a region the size of a pixel appears to be a straight line. This means, for example, that, as the radius of the circle being drawn is allowed to grow large, the midpoint circle-drawing algorithm rapidly converges toward being best fit. (Assume that noninteger radii

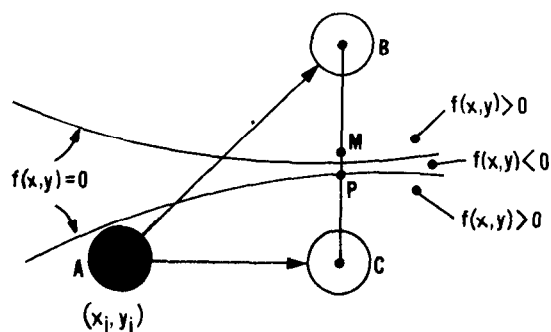


Fig. 19. Curve intersects line from B to C in two places.

are allowed.) Consequently, an algorithm (including Bresenham's circle drawer [2]) that assigns to its control parameters values that differ from those of the midpoint algorithm will not necessarily draw a best-fit circle under all circumstances. In fact, Bresenham's circle drawer is less accurate than the midpoint algorithm for certain noninteger radii.

EXTENDING THE METHOD TO OTHER CURVES

The midpoint method has been used to derive algorithms for drawing lines, circles, and ellipses. This method can be extended to the construction of algorithms for the broader class of curves that includes conic sections in general, as well as equations of higher order. The construction of such algorithms has been described by Pitteway and others [6, 8]. These have been based on curve-tracking methods other than those presented in this paper, but the general approach is the same. The advantage offered by the midpoint method over these methods is in the improved control over the linear error at each step.

In attempting to treat the broader class of curves, however, a drawing algorithm based on the midpoint method is subject to the same failure mode as these other curve-tracking methods. This failure mode, indicated in Figure 19, must be comprehended in the algorithm to be avoided. In Figure 19, the curve $f(x, y) = 0$ intersects the line from pixel B to pixel C in two places below midpoint M. The two edges of the curve could represent, for example, the two sides of a very thin ellipse or the two lobes of a hyperbola.

Assume that $f(x, y) < 0$ in the region between the upper and lower edges of the curve, and that $f(x, y) > 0$ above and below the two edges. Also assume that a curve-drawing algorithm has been tracking the bottom edge of the curve, and that pixel A has been selected by the algorithm during the previous step. The algorithm will now attempt to determine whether pixel B or pixel C lies closer to the edge (or rather, to its intersection at point P) on the basis of the sign-of-decision variable $d_i = f(x_i + 1, y_i + \frac{1}{2})$, which is the function evaluated at midpoint M. As in the previous examples in this paper the algorithm operates on the assumption that the condition $d_i > 0$ implies that B lies closer to the edge than C. Pixel C is, in fact, the better choice, but the algorithm mistakenly selects pixel B because $f(x, y)$ unexpectedly changes sign twice in the interval between M and B.

This failure mode can often be avoided by building into the algorithm the ability to detect situations such as that shown in Figure 19. Consider, for instance, the family of curves described by the general equation for a conic section:

$$\begin{aligned} f(x, y) &= Ax^2 + By^2 + Cxy + Dx + Ey + F \\ &= 0. \end{aligned}$$

The decision process at a particular step might be similar to that shown in Figure 4. At the point where the curve intersects the line from B to C , the equation of the curve is

$$\begin{aligned} f(x_i + 1, y_i + \tfrac{1}{2} + e) &= A(x_i + 1)^2 + B(y_i + \tfrac{1}{2} + e)^2 \\ &\quad + C(x_i + 1)(y_i + \tfrac{1}{2} + e) \\ &\quad + D(x_i + 1) + E(y_i + \tfrac{1}{2} + e) + F \\ &= f(x_i + 1, y_i + \tfrac{1}{2}) \\ &\quad + 2B(y_i + \tfrac{1}{2})e + C(x_i + 1)e + Ee + Be^2 \\ &= 0, \end{aligned}$$

where e is the displacement of the intersection from M . The sign of e determines whether B or C lies closer to the curve. The decision variable d_i is defined as the value of the function evaluated at midpoint M . The equation above is used to relate d_i to e :

$$\begin{aligned} d_i &= f(x_i + 1, y_i + \tfrac{1}{2}) \\ &= -(2B(y_i + \tfrac{1}{2}) + C(x_i + 1) + E + Be)e \\ &= -k_i e, \end{aligned}$$

where the definition

$$k_i = 2B(y_i + \tfrac{1}{2}) + C(x_i + 1) + E + Be$$

has been used. The sign of e is easily determined when the signs of d_i and k_i are known. Also define $h_i = k_i - Be$. Although the evaluation of k_i may involve complex calculations, h_i is of interest because (1) its value is relatively easy to calculate, and (2) the sign of h_i is in most circumstances the same as the sign of k_i , allowing h_i to be used in place of k_i in determining the sign of e .

The sign of k_i is the same as the sign of h_i as long as the condition $|h_i| > |Be|$ is satisfied. When the signs of k_i and h_i differ, this indicates that a critical region of the curve, such as that indicated in Figure 19, has been entered. This can be explained as follows. Assume that the signs of k_i and h_i differ. Allow point M in Figure 19 to move downward from its midpoint position toward P . At some point before reaching P , the function $f(x, y)$ at point M changes sign because the sign of k_i becomes the same as the sign of h_i . This must be so because at an arbitrarily small distance e above P the term Be in k_i can be neglected, and $k_i = h_i$.

The algorithm should be designed to detect situations in which the signs of k_i and h_i differ. Given that the curve-drawing algorithm is able to keep the linear

error $|E_{\max}|$ bounded at $\frac{1}{2}$, this means that $|e|$ is at least bounded at 1. Rather than evaluate e at each step, which may involve time-consuming calculations, we may use the upper bound of 1 in its place. Hence, the condition $|h_i| > |B|$, which is relatively easy to evaluate, is sufficient to verify that the signs of k_i and h_i are the same and that the situation shown in Figure 19 has not occurred. When the algorithm enters a region of the curve for which the condition $|h_i| > |B|$ is no longer satisfied, the signs of k_i and h_i are no longer guaranteed to be the same, and the algorithm must proceed cautiously to avoid selecting the wrong pixel.

When the algorithm detects that it may have entered a region such as that shown in Figure 19, it can respond by cutting its step size temporarily in half. Instead of stepping directly from pixel A to pixel B or C , it first selects as an intermediate point either $(x_i + \frac{1}{2}, y_i)$ or $(x_i + \frac{1}{2}, y_i + \frac{1}{2})$. If the algorithm still cannot ensure that $f(x, y)$ does not change sign more than once in the interval between these two points, the step size can be halved again, and so on.

This approach may not be successful, however, in a region in which two edges of a curve actually meet or cross each other.

CONCLUSIONS

The midpoint method has been described for deriving curve-drawing algorithms for generating such curves as lines, circles, and ellipses on raster display devices. The approach is general and can be extended to other curves (e.g., parabolas and hyperbolas) described by nonparametric equations. The algorithms derived using this method require only a modest amount of computation per step. Although the ellipse-drawing algorithm contains two multiplications that cannot be converted to simple shift operations, the two main loops require only integer additions and subtractions. Additionally, the algorithms are derived in a manner that ensures accuracy within known bounds. Using the convention that the distance between adjacent pixels on the display grid is unity, the maximum absolute error is bounded at $1/2$ for 8WS curve-drawing algorithms, and at $\sqrt{2}/2$ for 4WS algorithms.

ACKNOWLEDGMENTS

The authors are grateful to Andy Heilveil and Karl Gutttag for their useful discussions of this and other material related to curve-drawing algorithms. They also wish to thank J. E. Bresenham for pointing out problems encountered in the design of algorithms for drawing the general class of conic sections.

REFERENCES

1. BRESENHAM, J. E. Algorithm for computer control of a digital plotter. *IBM Syst. J.* 4, 1 (1965), 25-30.
2. BRESENHAM, J. E. A linear algorithm for incremental digital display of digital arcs. *Commun. ACM* 20, 2 (Feb. 1977), 100-106.
3. DANIELSSON, P. E. Incremental curve generation. *IEEE Trans. Comput.* C-19 (1970), 783-793.
4. DE ROO, J., WAMBACQ, P., VAN EYCKEN, L., OOSTERLINCK, A., AND VAN DEN BERGHE, H. A hardware implemented universal graphics generator. In *Proceedings of the Conference on Pattern Recognition and Image Processing* (Dallas, Tex., Aug. 3-5), IEEE, New York, 1981, pp. 300-305.
5. HORN, B. K. P. Circle generators for display devices. *Comput. Graph. Image Process.* 5 (1976) 280-288.

6. JORDAN, B. W., LENNON, W. J., AND HOLM, B. C. An improved algorithm for the generation of nonparametric curves. *IEEE Trans. Comput.* C-22, 12 (Dec. 1973), 1052-1060.
7. MCILROY, M. D. Best approximate circles on integer grids. *ACM Trans. Graph.* 2, 4 (Oct. 1983), 237-263.
8. PITTEWAY, M. Algorithm for drawing ellipses or hyperbolae with a digital plotter. *Comput. J.* 10, 3 (Nov. 1967), 282-289.
9. SPROULL, R. F. Using program transformations to drive line-drawing algorithms. *ACM Trans. Graph.* 1, 4 (Oct. 1982), 259-273.
10. SUENAGA, Y., KAMAE, T., AND KOBAYASHI, T. A high-speed algorithm for the generation of straight lines and circular arcs. *IEEE Trans. Comput.* C-28, 10 (Oct. 1979), 728-736.
11. VAN AKEN, J. An efficient ellipse-drawing algorithm. *IEEE Comput. Graph. & Appl.* 4, 9 (Sept. 1984), 24-35.

Received December 1983; accepted November 1984