COMPUTER GRAPHICS AND IMAGE PROCESSING 5, 280–288 (1976)

Circle Generators for Display Devices

BERTHOLD K. P. HORN

Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

Communicated by A. Rosenfeld

Received July 28, 1975

Simple methods of generating digital approximations to circles and other types of curves are described.

The short notes [1, 2] published in this Journal discussiong methods for producing circles on graphic output devices do not address the problem of efficiently generating contiguous sets of dots lying on the discrete grid characteristic of the display device. Methods for producing such discrete approximations of circles efficiently are of importance both for the traditional C.R.T. point-displays, driven by a display processor, and the increasingly popular raster-scanned devices, using a frame-buffer memory.

No claim is made for the originality of the scheme presented here—it has no doubt been reinvented innumerable times, and the author would not be surprised if it was, in fact, embodied in display hardware somewhere. The hope is that presenting the method here may save somebody some effort. It is simple to develop the circle generator by analogy with an ordinary vector generator. A brief exposition of a line-generating scheme follows for this reason.

1. A LINE GENERATOR

As a discrete approximation to the straight line $bx = ay$, one would like a set of contiguous grid points as near as possible to the line. If the line lies in the first octant, one can clearly pick one point in each vertical column that will be no more than one-half of the grid interval removed vertically from the line. If the point just below the line is as far from the line as the nearest one above it, we will arbitrarily pick the lower one. The points so defined lie on or above the line $bx = a(y + \frac{1}{2})$ and below the line $bx = a(y - \frac{1}{2})$. Evidently only one point in each vertical column lies between these limits, so all the grid points falling in the band defined by the two lines will in fact be used.

One way to generate these points is to calculate $y = \lceil bx/a - \frac{1}{2} \rceil$ for each vertical column. (Here $\lceil s \rceil$ signifies the smallest integer not smaller than s.) This method requires multiplication and division. Generating the points sequentially and performing incremental calculations is more efficient.

Advancing from one column to the next is achieved by incrementing x. If the point defined by this new value of x and the previous value of y falls below the lower edge of the acceptable band, then the grid point directly above it is the desired next point. That is, if $s = bx - a(y + \frac{1}{2}) > 0$, then y should be incremented, too. It is simple to keep track of the value of s. It is initially set to $-\frac{1}{2}a$ and b is added to it whenever x is incremented and a is subtracted from it whenever y is incremented. The function vector (a, b) performs these computations. (This is essentially Bresenham's algorithm [5].)

Simply swapping x and y will produce lines in the second octant. Negating x or y or both takes care of the other possible directions. Finally, adding an offset allows vectors to start at arbitrary grid points. The functions line (x_0, y_0, x_1, y_1) and plot (x, y) take care of the required bookkeeping.

Note that only fixed-point additions and subtractions are required. (Since $bx - ay$ can take on only integer values, the potential round-off in calculating $-\frac{1}{2}a$, for a odd, can be ignored.) Note also that $-a \leq s \leq b$, so only as many bits (plus a sign bit) are needed to store s as are required for a, b, x, and y.

```
line ($x_0, y_0, x_1, y_1$):  $a \leftarrow x_1 - x_0; b \leftarrow y_1 - y_0$
                              negate_x $\leftarrow f$; negate_y $\leftarrow f$; swap_xy $\leftarrow f$
                              if $a < 0$, then $a \leftarrow -a$; negate_x $\leftarrow t$
                              if $b < 0$, then $b \leftarrow -b$; negate_y $\leftarrow t$
                              if $a < b$, then $a \leftrightarrow b$; swap_xy $\leftarrow t$
                              vector $(a, b)$
end
vector $(a, b)$:              $x \leftarrow 0; y \leftarrow 0; s \leftarrow -a/2$
                              do until $x > a$
                                 plot $(x, y)$; $s \leftarrow s + b$; $x \leftarrow x + 1$
                                 if $s > 0$, then $s \leftarrow s - a$; $y \leftarrow y + 1$
                              end
end
plot $(x, y)$:                if swap_xy, then $x \leftrightarrow y$
                              if negate_y, then $y \leftarrow -y$
                              if negate_x, then $x \leftarrow -x$
                              point $(x + x_0, y + y_0)$
end.
```

2. A CIRCLE GENERATOR

For the discrete approximation, one would like to pick a set of contiguous grid points as near as possible to the circle $x^2 + y^2 = r^2$. In the first octant one can clearly pick one point in each horizontal row that will be no more than half the grid interval removed horizontally from the circle. If the point just to the left of the circle happens to be as far from the circle as the nearest point on the right, we will arbitrarily pick the right one. The points so defined lie to the right of the circle $(x + \frac{1}{2})^2 + y^2 = r^2$ and on or to the left of the circle $(x - \frac{1}{2})^2 + y^2 = r^2$.

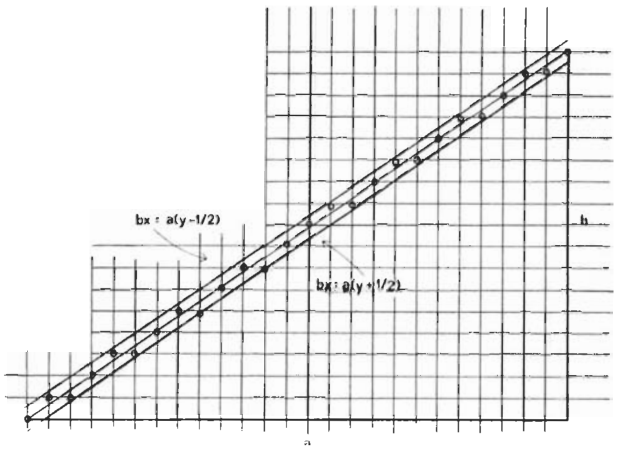

Fig. 1. Points on the discrete display grid picked by the line generator.

Again only one point per horizontal row will lie between these limits, so all grid points falling in the band defined by the two circles will be used.

One could proceed now by calculating $x = \lfloor (r^2 - y^2)^{\frac{1}{2}} + \frac{1}{2} \rfloor$ for each of the

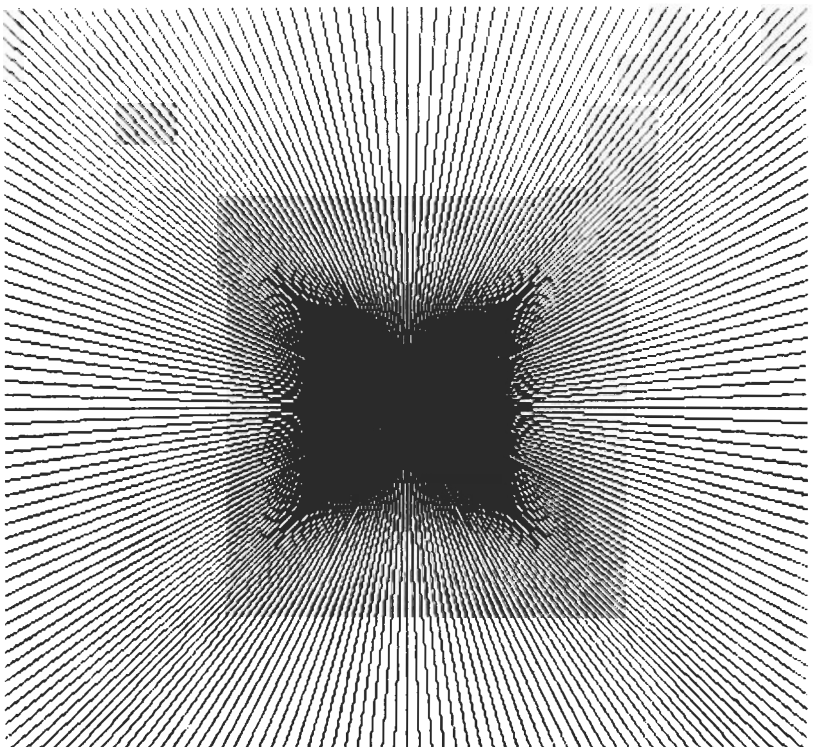

Fig. 2. Sample output of the line generator.

horizontal rows. (Here $[s]$ signifies the largest integer not larger than s.) This is obviously computationally inefficient and an incremental scheme is to be preferred.

Advancing from one row to the next is achieved by incrementing y. If the point defined by the previous value of x and this new value of y falls to the right of the right-hand edge of the acceptable band, then the grid point directly to its left is chosen. That is, if $s = (x - \frac{1}{2})^2 + y^2 - r^2 > 0$, then x should be decremented. It is again straightforward to update the value of s. It is initially set to $(-r + \frac{1}{4})$ and $(2y + 1)$ is added whenever y is incremented and $(2x - 2)$ is subtracted whenever x is decremented. The function sector (r) performs these operations.

Simply swapping x and y will produce the arc of the circle falling in the second octant. Negating x or y or both generates the rest. Once again, an offset can be added to place the center of the circle in an arbitrary position. The functions circle (x_0, y_0, r) and plot (x, y) perform the required bookkeeping and iteration.

Note that only fixed-point additions and subtractions are needed (again, since $x^2 + y^2 - r^2$ can take on only integer values, the $\frac{1}{4}$ in the initialization of s can be ignored). Note also that $-2x \leq s \leq 2y$ and so one more bit (plus a sign bit) is required to store s as are needed for r, x and y.

```
circle ($x_0$, $y_0$, $r$):   do for negate_x = $f, t$
                                do for negate_y = $f, t$
                                  do for swap_xy = $f, t$
                                    sector ($r$)
                                  end
                                end
                              end
end
sector ($r$):                 $x \leftarrow r; y \leftarrow 0; s \leftarrow -r$
                              do until $y > x$
                                plot ($x, y$); $s \leftarrow s + 2y + 1; y \leftarrow y + 1$
                                if $s > 0$, then $s \leftarrow s - 2x + 2; x \leftarrow x - 1$
                              end
end
plot ($x, y$):                if swap_xy, then $x \leftrightarrow y$
                              if negate_y, then $y \leftarrow -y$
                              if negate_x, then $x \leftarrow -x$
                              point ($x + x_0, y + y_0$)
end.
```

3. GENERALIZATIONS

Discrete approximations to lines connecting points not on the display grid can be developed simply by using ks instead of s in the incremental calculations. This is on the assumption that the lines *do* end on grid points of a grid k times finer. For completely arbitrary end-points the value of s can be carried as a floating point quantity. The same idea can be used for circles with noninteger radii and centers not lying on the display grid.

The method also can be generalized to curves defined by arbitrary polynomials

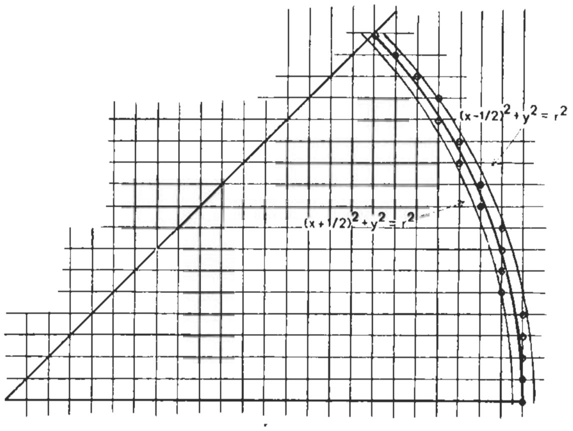

FIG. 3. Points on the discrete display grid picked by the circle generator.

in x and y, including the conic sections. Conic sections can be generated much in the same way as circles, but higher-order curves require multiplications for the incremental calculation of s, as well as dynamic decisions about what octant

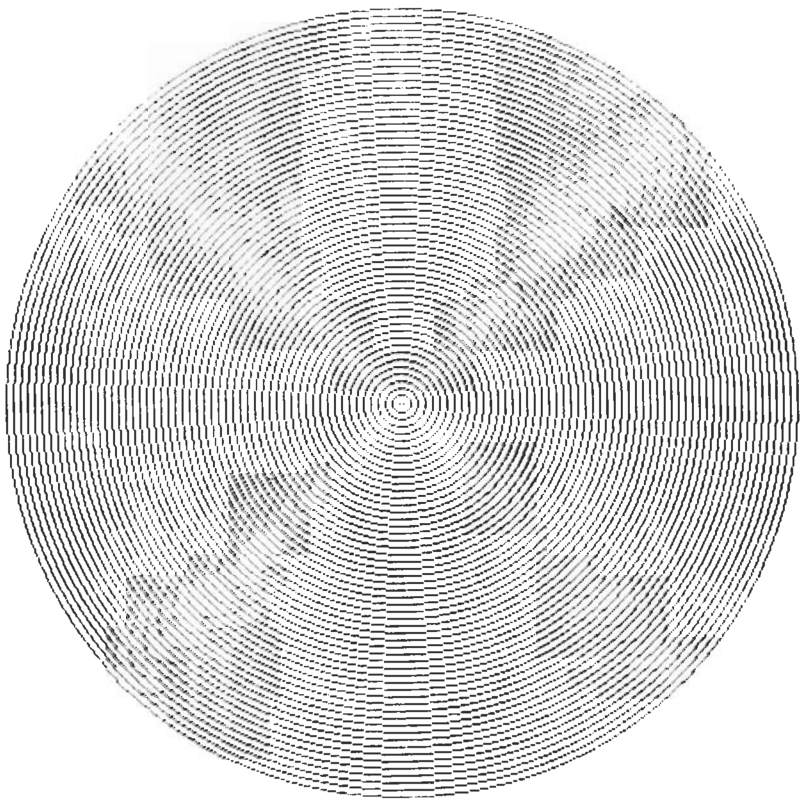

FIG. 4. Sample output of the circle generator.

the curve falls in, and more complicated terminating conditions. (See also Pitteway's algorithm [6].)

The method described here for generating discrete approximations to circles for display purposes has the advantage that it produces contiguous sets of points, unlike the schemes of Denert [1] and Lappalainen [2]. It also requires only fixed point addition and subtraction.

4. A CONNECT-THE-DOTS SCHEME

In many situations one is faced with an existing display device that has point- and vector-generators, but no circle-generators. For efficiency reasons, one might accept an approximation produced by drawing vectors between points lying on or near the circle. Generating the required points by repeatedly evaluating $r \cos(i\pi/n)$ and $r \sin(i\pi/n)$ is computationally inefficient, as pointed out by Denert [1]. An alternative would be to precalculate $\cos(\pi/n)$ and $\sin(\pi/n)$ and perform successive incremental rotations by matrix multiplication:

$$\begin{vmatrix} x_n + 1 \\ y_n + 1 \end{vmatrix} = \begin{vmatrix} \cos(\pi/n) & -\sin(\pi/n) \\ \sin(\pi/n) & \cos(\pi/n) \end{vmatrix} \begin{vmatrix} x_n \\ y_n \end{vmatrix}$$

While this can be done without recourse to trigonometric functions, floating point arithmetic is still needed.

Fortunately, methods exist for producing more or less evenly spaced points lying very near the circumference of a circle, by algorithms that require only fixed-point arithmetic. Ivan Sutherland used such a scheme in his SKETCHPAD system [4].

The basic idea is to use a difference equation approximation to the differential equations defining a circle,

$$dx/d\theta = -y \qquad \text{and} \qquad dy/d\theta = x.$$

A suitable set of difference equations might be

$$x_{n+1} = x_n - y_n/k \qquad y_{n+1} = x_n/k + y_n.$$

This set of difference equations has a coefficient matrix with determinant greater than one and, in fact, $(x_{n+1}^2 + y_{n+1}^2) = (1 + 1/k^2)(x_n^2 + y_n^2)$. So the points generated lie on a spiral, not a circle. When implementing this scheme a common programming bug is the use of the new value, x_{n+1}, instead of x_n, when calculating y_{n+1}. The corresponding equations are:

$$x_{n+1} = x_n - y_n/k \qquad y_{n+1} = x_{n+1}/k + y_n.$$

Curiously this set of equations produces points in a bounded region, and it is shown in the appendix that these points lie on an ellipse. This ellipse tends to a circle as k is made larger and larger.

We are still faced with the use of floating-point arithmetic operations, however. What happens if we commit a second programming blunder and use fixed-point arithmetic, simply truncating the result of the division? Surprisingly, the points so generated still lie in a bounded region. For k in a certain range, the points even lie close to the expected ellipse. Clearly, if all points lie in a bounded region on the display grid, there must be some duplication. Eventually the algorithm will produce some point a second time. Since the next point is determined uniquely be the current point, the algorithm will retrace its steps from there on.

Notice also that the operation of the algorithm is reversible since

$$y_n = y_{n+1} - x_{n+1}/k \qquad x_n = x_{n+1} + y_n/k.$$

It follows that the first point retraced must be the initial point. This does not mean, however, that the output will cycle after one revolution. The starting point may, in fact, be reached only after several circuits around the origin, depending on the value of k and the position of the initial point. The function circle_dots (x_0, y_0, r) performs the required operations.

```
circle_dots ($x_0$, $y_0$, $r$):  $x \rightarrow r$; $y \rightarrow 0$; step_point
                               do until $x = r \, V \, y = 0$, step_point
end
step_point:                    point ($x_0 + x$, $y_0 + y$)
                               $x \leftarrow x - y/k$; $y \leftarrow x/k + y$
end.
```

It remains to discuss the choice of k. For small values of k, the approximate ellipse will be quite eccentric, while for large values the steps become so small that the figure degenerates into an octagon or square, because of truncation in the arithmetic. This happens when k is near max $(|x_0|, |y_0|)$, where (x_0, y_0) is the initial point. In between, there is a wide range of values for k that will produce satisfactory approximations to circles. Powers of two are particularly useful since the division can then be replaced by a simple right-shift operation. The function circle_line (x_0, y_0, r) generates the polygonal approximation to the circle we have been developing.

```
circle_line ($x_0$, $y_0$, $r$):  $x_n \leftarrow r$; $y_n \leftarrow 0$; step_line
                               do while $y_p \geq 0 \wedge y_n < 0$, step_line
                               line ($x_n + x_0$, $y_n + y_0$, $r + x_0$, $y_0$)
end
step_line:                     $x_p \leftarrow x_n$; $y_p \leftarrow y_n$
                               $x_n \leftarrow x_p - y_p/k$; $y_n \leftarrow x_n/k + y_p$
                               line ($x_p + x_0$, $y_p + y_0$, $x_n + x_0$, $y_n + y_0$)
end.
```

The points generated by this method lie quite close to the desired circle for reasonable values of k. If for some reason higher accuracy is desired one can resort to a variety of methods. First, it is simple to store multiples of x and y—in effect, working internally with a finer grid. Because of the larger values of k possible on this grid, a closer approximation to a circle is possible. The values are truncated on output.

Secondly, one can make use of the observation that x_n in effect lags a half a step behind y_n. Using $\frac{1}{2}(x_n + x_{n+1})$ improves matters considerably. The ellipse now generated has its major axis along the x-axis and an eccentricity $e = 1/2k$. This becomes smaller more rapidly with increases in k than our previous value. (It varied inversely with the square root of k.)

Finally, one can approximate the transformation to x' and y' given in the appendix. A good approximation is

$$x' \approx (x + y)\left(1 - \frac{1}{4k}\right), \qquad y' \approx (x - y)\left(1 + \frac{1}{4k}\right).$$

If k is a power of 2, this transformation again requires only addition, subtraction, and shifting. This generates very accurate, evenly spaced sets of points.

The advantage of this algorithm over the one presented by Denert [1] is that the points produced are evenly spaced and do not require prior commitment to a fixed number of approximating vectors.

APPENDIX: SOLUTION OF THE PAIR OF DIFFERENCE EQUATIONS

The set of difference equations

$$x_{n+1} = x_n - y_n/k \qquad y_{n+1} = x_{n+1}/k + y_n$$

also can be written as

$$x_{n+1} = x_n - y_n/k \qquad x_{n+1} = x_n/k + (1 - 1/k^2)y_n.$$

The determinant of the corresponding coefficient matrix is 1. A solution can be found by assuming that it has the form $x_n = as^n$ and $y_n = bs^n$ (where a, b, and s may be complex). For $|k| > \frac{1}{2}$, the solutions are oscillatory and of equal amplitude in x and y. The form of this solution (which is not presented here) suggests the following simplifying substitution. Let

$$x' = \frac{x + y}{[1 + (1/2k)]^{\frac{1}{2}}} \quad \text{and} \quad y' = \frac{x - y}{[1 - (1/2k)]^{\frac{1}{2}}}.$$

Then

$$x = \frac{1}{2}\left[\left(1 + \frac{1}{2k}\right)^{\frac{1}{2}} x' + \left(1 - \frac{1}{2k}\right)^{\frac{1}{2}} y'\right]$$

$$y = \frac{1}{2}\left[\left(1 + \frac{1}{2k}\right)^{\frac{1}{2}} x' - \left(1 - \frac{1}{2k}\right)^{\frac{1}{2}} y'\right].$$

Substituting into the set of difference equations, one can show that

$$x_{n+1}' = \left(1 - \frac{1}{2k^2}\right)x_n' + \frac{[1 - (1/4k^2)]^{\frac{1}{2}}}{k} y_n'$$

$$y_{n+1}' = -\frac{[1 - (1/4k^2)]^{\frac{1}{2}}}{k} x_n' + \left(1 - \frac{1}{2k^2}\right)y_n'.$$

Clearly now

$$(x_{n+1}')^2 + (y_{n+1}')^2 = (x_n')^2 + (y_n')^2.$$

So the transformed coordinates all fall on a circle and in fact each step produces a new point at an angle θ ahead of the previous one, where $\cos(\theta) = 1 - 1/2k^2$ or $\sin(\theta/2) = 1/2k$ (for large k, we have $\theta \approx 1/k$ and, hence, there are about $2\pi k$ steps per revolution).

Returning to the original variables we find

$$\frac{(x+y)^2}{1+(1/2k)} + \frac{(x-y)^2}{1-(1/2k)} = 2\frac{r^2}{1-(1/4k^2)}.$$

This is the equation of an ellipse. Here r is the distance from the origin at which the ellipse crosses the x- and y-axes. The major axis lies along the line $x = y$ and has a half-length of $r/(1 - 1/2k)^{\frac{1}{2}}$, while the minor axis, at right angles, has a half-length $r/(1 + 1/2k)^{\frac{1}{2}}$. The eccentricity of the ellipse is $e = 1/(1 + k/2)^{\frac{1}{2}}$. When k becomes large, the ellipse tends to become nearly circular.

One way of understanding why these equations generate an ellipse rather than a circle is to note that in effect x_n lags a half a step $(\theta/2)$ behind y_n in its oscillation. This also explains why, when the steps become small, the ellipse rounds out into a near-circle.

REFERENCES

1. E. Denert, A method for computing points of a circle using only integers, *Computer Graphics Image Processing* 2, 1973, 83.
2. C. V. Kameswara Rao, Comment on a method for computing points of a circle using only integers, *Computer Graphics Image Processing* 4, 1975, 79.
3. H. Freeman, A review of relevant problems in the processing of line-drawing data, in *Automatic Interpretation and Classification of Images*, pp. 168–172, Academic Press, New York, 1969.
4. I. E. Sutherland, SKETCHPAD—A man-machine graphical communication system, in *Proceedings of the Joint Computer Conference*, Detroit, Michigan, May 1963, p. 335.
5. J. E. Bresenham, Algorithm for computer control of a digital plotter, *IBM Systems J.* **4**, 1965, 25–30.
6. M. L. V. Pitteway, Algorithm for drawing ellipses and hyperbolae with a digital plotter, *Computer J.* **10**, 1967, 282–289.