# Arcs

# An Efficient Three-Point Arc Algorithm

**Ian Galton**
**Caltech**

For many applications a 2D circular arc can be conveniently specified by three points that lie on the arc. Since the radius of curvature grows without bound as the three points become collinear, any practical algorithm must avoid calculating the arc's radius or functions of the radius. The algorithm presented here achieves this objective. It is very efficient because it uses exclusively integer arithmetic and requires only addition, subtraction, comparison, and branch operations in the inner loop.

Many graphics standards such as the Computer Graphics Interface (CGI), the Computer Graphics Metafile (CGM), and Videotex specify arcs with three points. The method provides for a convenient user interface, since the user can specify the two endpoints of the arc and then "rubberband" the curve by manipulating the third point.

Although there are several integer circle algorithms in the literature, most maintain variables that are unbounded functions of the radius.[1-4] Such algorithms invariably impose a minimum curvature limit on the arcs they draw.

Since the minimum curvature associated with a particular implementation of an algorithm depends on the resolution of the display device, it is difficult to design resolution-independent software systems using the algorithm.[5] If the same implementation of the algorithm were used to draw equivalently scaled arcs on two display devices with different resolutions, it might be possible to draw the arc on one of the devices and not on the other.

The algorithm presented here has no minimum curvature limit and is similar in efficiency to the previous integer algorithms.

## The fundamental equation

Assume that the three defining points are at integer locations specified by the coordinates $(j, k)$, $(l, m)$, and $(n, o)$, and are distinct. The algorithm draws each arc from the point $(j, k)$ through the point $(l, m)$ and to the point $(n, o)$. Hence, $(j, k)$ is referred to as the *starting point*, $(l, m)$ as the *intermediate point*, and $(n, o)$ as the *ending point*.

The algorithm is based on the following theorem.

**Theorem 1.** Let $(j, k)$ be a starting point and let $(h, i)$ be any point not equal to $(j, k)$. For each pair of positive, nonzero constants $\alpha$ and $\beta$ there exists a circle or line passing through the starting point which is the only solution to

$$\alpha v^2(x, y) - \beta u^2(x, y) = \gamma \qquad (1)$$

where $\gamma$ is a constant and

$$u(x, y) = \sqrt{(j - x)^2 + (k - y)^2}$$


0272-17-16/89/1100-0044$01.00 ©1989 IEEE

$$v(x, y) = \sqrt{(h - x)^2 + (i - y)^2} \quad (2)$$

**Proof.** Evaluating Equation 1 at the point $(j, k)$ gives

$$\gamma = \alpha v^2(j, k) \quad (3)$$

Combining Equations 1, 2, and 3 gives

$$(\alpha - \beta)x^2 + 2(\beta j - \alpha h)x - (\alpha + \beta)j^2 +$$
$$(\alpha - \beta)y^2 + 2(\beta k - \alpha i)y - (\alpha + \beta)k^2 = 0$$

If $\alpha \neq \beta$, the equation can be written as

$$(x - x_0)^2 + (y - y_0)^2 = R_0^2$$

where $x_0$, $y_0$, and $R_0$ are constants that depend only on the three points defining the arc.

If $\alpha = \beta$, the equation can be written as

$$y = mx + b$$

where $m$ and $b$ are constants that depend only on the three points defining the arc.

In the first case, the equation defines a circle. In the second case, it defines a line. □

The relationships among point $(h, i)$, the arc, and the functions $u(x, y)$ and $v(x, y)$ are shown in Figure 1.

Any point $(h, i)$ that can be used with Equation 1 to generate a given circle will be referred to as an *anchor point* for that circle. The constants $\alpha$ and $\beta$ will be referred to as *curvature factors.*

Equation 1 provides the means by which the algorithm generates the arc. To use the equation, the algorithm must first calculate the curvature factors and the location of an anchor point.

Since by Theorem 1 the only solution to Equation 1 is a circle or line, it follows that if three distinct points satisfy the equation, so must the circle or line passing through the points. Therefore, an anchor point can be found by solving the set of three equations generated by evaluating Equation 1 at the points $(j, k)$, $(l, m)$, and $(n, o)$ respectively:

$$\begin{aligned} \alpha v^2(j, k) &= \gamma \\ \alpha v^2(l, m) - \beta u^2(l, m) &= \gamma \\ \alpha v^2(n, o) - \beta u^2(n, o) &= \gamma \end{aligned} \quad (4)$$

Solving these equations for the anchor point gives

$$h = j - \frac{C_0}{\tau}$$

$$i = k + \frac{C_1}{\tau} \quad (5)$$

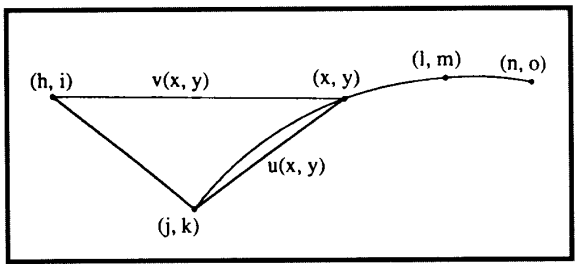


**Figure 1. The relationship between the arc and the anchor point.**

where

$$C_0 = (k - m)u^2(n, o) - (k - o)u^2(l, m)$$
$$C_1 = (j - l)u^2(n, o) - (j - n)u^2(l, m)$$

and $\tau$ is any nonzero constant. Because $\tau$ is not specified, an infinite number of anchor points exist for a given circle or line. Note that Equations 5 are a parametric representation (in $\tau$) of a line passing through the starting point. Any point on this line other than the starting point (because of the constraint in Theorem 1) can be used as an anchor point.

Once an anchor point is chosen, the first and third of Equations 4 can be solved to find the curvature factors. This gives

$$\begin{aligned} \alpha &= u^2(n, o) \\ \beta &= v^2(n, o) - v^2(j, k) \end{aligned} \quad (6)$$

Note that the anchor point and the curvature factors are all bounded functions of the three points that define the arc. Therefore, Equation 1 can be used to generate any arc (including the special case of a line segment) specified by three distinct points.

## Difference equations

The equations derived so far are exact continuous-variable equations. The problem at hand, however, is to generate quantized discrete-variable (i.e., rasterized) arcs. Although we could quantize the points generated using the exact equations, such an algorithm would be inefficient; unnecessary information (i.e., the fractional part of each point) would be generated, and each point would be calculated from scratch (instead of benefiting from the calculations performed to find the previous point).

A more efficient approach uses information about the current point (the point just drawn) to choose the next point.[1-8] The next point is assumed to be one of

the current point's eight nearest neighbors. The slope of the arc's tangent and the drawing direction are used to rule out all but two of these points. The errors associated with the two remaining points are calculated, and the point with the smallest error is chosen as the next point. This process continues until the curve is complete.

The error function is obtained by quantizing the variables in Equation 1. It is the difference between the left-hand side and right-hand side of the equation. Specifically,

$$\varepsilon(\hat{x}, \hat{y}) = \alpha v^2(\hat{x}, \hat{y}) - \beta u^2(\hat{x}, \hat{y}) - \gamma \quad (7)$$

where $(\hat{x}, \hat{y})$ is the quantized version of the point $(x, y)$. The magnitude of $\varepsilon(\hat{x}, \hat{y})$ provides a measure of how close the point $(\hat{x}, \hat{y})$ is to the exact arc.

Although Equation 7 could be used to calculate the error at each point, it is more efficient to use a set of difference equations. The difference equations provide an incremental implementation of Equation 7. Given the error associated with the current point, the difference equations provide the error associated with the neighboring points.

Each difference equation can be derived by subtracting Equation 7 evaluated at the point $(\hat{x}, \hat{y})$ from that evaluated at the appropriate neighboring point. The difference equations are

$$\begin{aligned}
\varepsilon(\hat{x}+1, \hat{y}) &= \varepsilon(\hat{x}, \hat{y}) + \Delta\varepsilon_{x+}(\hat{x}) \\
\varepsilon(\hat{x}-1, \hat{y}) &= \varepsilon(\hat{x}, \hat{y}) + \Delta\varepsilon_{x-}(\hat{x}) \\
\varepsilon(\hat{x}, \hat{y}+1) &= \varepsilon(\hat{x}, \hat{y}) + \Delta\varepsilon_{y+}(\hat{y}) \\
\varepsilon(\hat{x}, \hat{y}-1) &= \varepsilon(\hat{x}, \hat{y}) + \Delta\varepsilon_{y-}(\hat{y})
\end{aligned} \quad (8)$$

where

$$\begin{aligned}
\Delta\varepsilon_{x+}(\hat{x}) &= \alpha[1 - 2(h - \hat{x})] - \beta[1 - 2(j - \hat{x})] \\
\Delta\varepsilon_{x-}(\hat{x}) &= \alpha[1 + 2(h - \hat{x})] - \beta[1 + 2(j - \hat{x})] \\
\Delta\varepsilon_{y+}(\hat{y}) &= \alpha[1 - 2(i - \hat{y})] - \beta[1 - 2(k - \hat{y})] \\
\Delta\varepsilon_{y-}(\hat{y}) &= \alpha[1 + 2(i - \hat{y})] - \beta[1 + 2(k - \hat{y})]
\end{aligned}$$

The functions $\Delta\varepsilon_{x+}(\hat{x})$, $\Delta\varepsilon_{x-}(\hat{x})$, $\Delta\varepsilon_{y+}(\hat{y})$, and $\Delta\varepsilon_{y-}(\hat{y})$ are called the *error difference functions*.

In practice, the error difference functions need to be calculated only for the first point. For the remaining points, a second set of difference equations can be used to calculate their values. These difference equations are analogous to Equations 8 and are derived in the same fashion. They are

$$\begin{aligned}
\Delta\varepsilon_{x+}(\hat{x} \pm 1) &= \Delta\varepsilon_{x+}(\hat{x}) \pm 2(\alpha - \beta) \\
\Delta\varepsilon_{x-}(\hat{x} \pm 1) &= \Delta\varepsilon_{x-}(\hat{x}) \mp 2(\alpha - \beta) \\
\Delta\varepsilon_{y+}(\hat{y} \pm 1) &= \Delta\varepsilon_{y+}(\hat{y}) \pm 2(\alpha - \beta) \\
\Delta\varepsilon_{y-}(\hat{y} \pm 1) &= \Delta\varepsilon_{y-}(\hat{y}) \mp 2(\alpha - \beta)
\end{aligned}$$

## Octant change detection

A circle consists of eight octants. Within each of these octants, the slope of the tangent to the circle is such that, for a given drawing direction, all but two of the eight points surrounding the current point can be ruled out as appropriate next points. For example, in the first octant, $dx/dy < -1$. Therefore, for a counterclockwise drawing direction, $y$ increases faster than $x$ decreases at all points within the octant. In this case, if the current point is $(\hat{x}, \hat{y})$, the next point might be either $(\hat{x}, \hat{y}+1)$ or $(\hat{x}-1, \hat{y}+1)$.

Within each octant, to find the next point one of the variables is always incremented (or decremented) while the other is left unchanged or incremented (or decremented) so as to minimize the error. The variable always incremented (or decremented) in a given octant is referred to as the *step variable,* the other variable as the *test variable.*

To use this scheme, the algorithm must keep track of which octant it is currently drawing. The error difference functions can provide this information. Note that at a given arc point, the next arc point is located farther along the step-variable axis than it is along the test-variable axis. Therefore, the error difference functions increase less along the step-variable axis than they do along the test-variable axis. Furthermore, it can be shown that the sign of the error at points inside the circle is opposite to that at points outside the circle. Therefore, the signs of the error difference functions provide information regarding the direction in which to change $\hat{x}$ and $\hat{y}$.

The relationships between the error difference functions and the eight octants for arcs drawn in the counterclockwise direction are listed in the table.

## Anchor point approximation

To ensure that the algorithm requires only integer variables, it is necessary to use an integer anchor point. There are two choices. The first is to set $\tau$ equal to 1 in Equations 5. This gives rise to the desired arc, but requires that the algorithm maintain some storage

registers with six times more bits than are required to hold the largest of the parameters $j$, $k$, $l$, $m$, $n$, and $o$.

The second choice is to use a $\tau$ greater than 1 and quantize the resulting noninteger anchor point. As shown in the appendix, if

$$\tau = \frac{\max(|C_o|, |C_1|)}{2(|j-l| + |k-m|)} \quad (9)$$

in Equations 5 and the resulting values of $h$ and $i$ are rounded to the nearest integer, then the rasterized arc will contain the three defining points, even though the corresponding exact arc misses one of the points slightly. The scheme reduces the register bit length requirements of the algorithm by a factor of 2.

## Implementation example

The following code segments illustrate a possible implementation of the algorithm. For brevity, only the portion of the algorithm associated with the first octant is treated.

Two code segments are shown: the initialization routine and the inner-loop routine. Once the initialization routine has been executed, the inner loop is executed. The inner-loop routine returns when either the arc reaches the ending point or another octant is about to be entered.

```
initialize_arc()
{
   error = 0;
   x = j;
   y = k;
   c0 = (k-m) * SQR(u(n, o)) -
        (k-o) * SQR(u(l, m));
   c1 = (j-l) * SQR(u(n, o)) -
        (j-n) * SQR(u(l, m));
   tau = (MAX(ABS(c1), ABS(c2)) /
        (ABS(j-l) + ABS(k-m));
   h = j - c0/tau;
   i = k + c1/tau;
   alpha = SQR(u(n, o));
   beta = SQR(v(n, o)) - SQR(v(j, k));
   diff_err_x = alpha*(1+h-j) - beta;
   diff_err_y = alpha*(1-i+k) - beta;
}

inner_loop()
{
   do
   {
     plot(x, y);
```

**Table. The error difference function conditions associated with each octant for the counterclockwise drawing direction.**

| Octant | Error Difference Function Conditions |
|---|---|
| $[0, \frac{\pi}{4})$ | $\Delta\varepsilon_{x-}(\hat{x}) > 0 \quad \Delta\varepsilon_{y+}(\hat{y}) < 0 \quad \lvert\Delta\varepsilon_{x-}(\hat{x})\rvert > \lvert\Delta\varepsilon_{y+}(\hat{y})\rvert$ |
| $[\frac{\pi}{4}, \frac{\pi}{2})$ | $\Delta\varepsilon_{x-}(\hat{x}) \geq 0 \quad \Delta\varepsilon_{y+}(\hat{y}) < 0 \quad \lvert\Delta\varepsilon_{x-}(\hat{x})\rvert \leq \lvert\Delta\varepsilon_{y+}(\hat{y})\rvert$ |
| $[\frac{\pi}{2}, \frac{3\pi}{4})$ | $\Delta\varepsilon_{x-}(\hat{x}) < 0 \quad \Delta\varepsilon_{y-}(\hat{y}) > 0 \quad \lvert\Delta\varepsilon_{x-}(\hat{x})\rvert < \lvert\Delta\varepsilon_{y-}(\hat{y})\rvert$ |
| $[\frac{3\pi}{4}, \pi)$ | $\Delta\varepsilon_{x-}(\hat{x}) < 0 \quad \Delta\varepsilon_{y-}(\hat{y}) \geq 0 \quad \lvert\Delta\varepsilon_{x-}(\hat{x})\rvert \geq \lvert\Delta\varepsilon_{y-}(\hat{y})\rvert$ |
| $[\pi, \frac{5\pi}{4})$ | $\Delta\varepsilon_{x+}(\hat{x}) > 0 \quad \Delta\varepsilon_{y-}(\hat{y}) < 0 \quad \lvert\Delta\varepsilon_{x+}(\hat{x})\rvert > \lvert\Delta\varepsilon_{y-}(\hat{y})\rvert$ |
| $[\frac{5\pi}{4}, \frac{3\pi}{2})$ | $\Delta\varepsilon_{x+}(\hat{x}) \geq 0 \quad \Delta\varepsilon_{y-}(\hat{y}) < 0 \quad \lvert\Delta\varepsilon_{x+}(\hat{x})\rvert \leq \lvert\Delta\varepsilon_{y-}(\hat{y})\rvert$ |
| $[\frac{3\pi}{2}, \frac{7\pi}{4})$ | $\Delta\varepsilon_{x+}(\hat{x}) < 0 \quad \Delta\varepsilon_{y+}(\hat{y}) > 0 \quad \lvert\Delta\varepsilon_{x+}(\hat{x})\rvert < \lvert\Delta\varepsilon_{y+}(\hat{y})\rvert$ |
| $[\frac{7\pi}{4}, 2\pi)$ | $\Delta\varepsilon_{x+}(\hat{x}) < 0 \quad \Delta\varepsilon_{y+}(\hat{y}) \geq 0 \quad \lvert\Delta\varepsilon_{x+}(\hat{x})\rvert \geq \lvert\Delta\varepsilon_{y+}(\hat{y})\rvert$ |

```
     y++;
     error += diff_err_y;
     diff_err_y += 2*(alpha - beta);
     test_error = error + diff_err_x;
     if (ABS(test_error) < ABS(error))
     {
       x--;
       error = test_error;
       diff_err_x += 2*(alpha - beta);
       if (-diff_err_y > diff_err_x)
         return(OCTANT_CHANGE);
     }
   } while (x != n || y != n);
   return(FINISHED);
}
```

The variables in the code segments are related to the variables in the equations of the previous sections as follows:

$$\begin{aligned} \texttt{c0} &= C_0 \\ \texttt{c1} &= C_1 \\ \texttt{alpha} &= \alpha \\ \texttt{beta} &= \beta \\ \texttt{error} &= \varepsilon(\hat{x}, \hat{y}) \\ \texttt{test\_error} &= \varepsilon(\hat{x}-1, \hat{y}+1) \\ \texttt{diff\_err\_x} &= \Delta\varepsilon_{x-}(\hat{x}) \\ \texttt{diff\_err\_y} &= \Delta\varepsilon_{y+}(\hat{y}) \end{aligned}$$

## Performance

As the code segments show, the algorithm performs very few operations for each arc point it calculates. No floating-point arithmetic is required, and in the inner loop only addition, subtraction, compare, and branch operations are required. The complexity of the inner loop is similar to that of the fastest circle algorithms in the literature.

The algorithm always draws the best rasterized version of an exact arc. If an exact integer anchor point is used ($\tau = 1$ in Equations 5), both the rasterized arc and

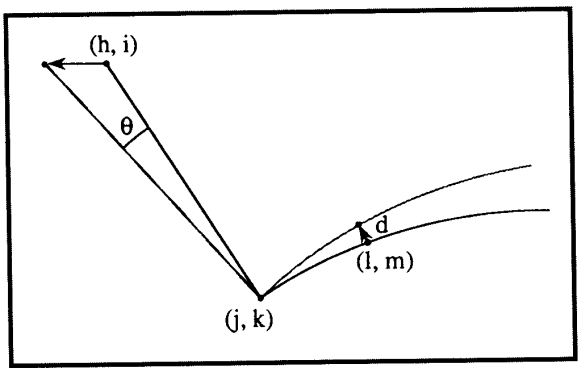


**Figure 2. Rotation about the starting point caused by anchor-point displacement.**

the corresponding exact arc pass through the three defining points. If the approximated anchor point of Theorem 2 is used, the rasterized arc passes through the three defining points but the corresponding exact arc misses the intermediate point slightly.

All of the variables maintained by the algorithm are bounded functions of the three defining points. Suppose $N$ bits are required to represent the largest of the parameters $j$, $k$, $l$, $m$, $n$, and $o$. If an exact anchor point is chosen using $\tau = 1$ in Equations 5, then $6N$ bit registers are required for the error variables. If the approximated anchor point is used, then $3N + 1$ bit registers are required for the error variables.

## Conclusion

An original algorithm designed specifically for drawing three-point circular arcs has been presented and analyzed. Its complexity is similar to that of previous algorithms but, unlike the previous algorithms, it imposes no minimum curvature limit on the arcs it can draw. The arcs produced by the algorithm always contain the three defining points. ■

## Appendix

This appendix verifies Equation 9. The following two lemmas are required to prove the theorem.

**Lemma 1.** The line on which the anchor points for a given circle lie is perpendicular to the tangent of the circle at the starting point.

**Proof.** From Equations 5 the slope of the line on which the anchor points lie is

$$m = \frac{C_1}{C_0}$$

To find the slope of the tangent of the circle at the starting point, differentiate Equation 1 with respect to $x$ and solve for $dy/dx$. This gives

$$\frac{dy}{dx} = -\frac{C_0}{C_1} \quad \square$$

**Lemma 2.** Let $C_1$ be the rasterized arc specified by the three integer points $(j, k)$, $(l, m)$, and $(n, o)$. Choose the anchor point by setting

$$\tau < \frac{\sqrt{C_o^2 + C_1^2}}{2u(l,m)} \tag{10}$$

in Equations 5. Let $C_2$ be the rasterized arc generated by using the same starting point and curvature factors as $C_1$ and an anchor point displaced from the anchor point of $C_1$ by less than 0.5. Then $C_2$ contains the point $(l, m)$.

**Proof.** From Equations 5 with $\tau$ as above the distance between the starting point and anchor point is related to $u(l, m)$ as

$$\sqrt{(h-j)^2 + (i-k)^2} > 2u(l, m)$$

Since Equation 1 depends only on the starting point, the anchor point, and the curvature factors, displacing the anchor point causes the arc to rotate about the starting point (see Figure 2).

Since the anchor point is displaced less than 0.5, the angle of rotation is

$$\theta < \arcsin\left(\frac{0.5}{\sqrt{0.25 + 4u^2(l, m)}}\right)$$

The same angle of rotation applies to the line joining the starting point and intermediate point. Therefore, the displacement $d$ of the intermediate point is related to theta by

$$d < \sqrt{u^2(l, m) + d^2}\ \sin\theta$$

$$< \frac{\sqrt{u^2(l, m) + d^2}}{2\sqrt{0.25 + 4u^2(l, m)}}$$

Solving for $d$ gives

$$d < \frac{1}{4}$$

It can be shown that for any arc specified by three integer points the incremental algorithm will choose each point that is within 1/4 of the exact arc. □

**Theorem 2.** Let the three defining points $(j, k)$, $(l, m)$ and $(n, o)$ be integer points. Let $(\hat{h}, \hat{i})$ be the quantized (to the nearest integer) version of the point $(h, i)$ obtained using

$$\tau = \frac{\max(|C_o|, |C_1|)}{2(|j - l| + |k - m|)}$$

in Equations 5. Let $\hat{\alpha}$ and $\hat{\beta}$ be the curvature factors obtained by using $(\hat{h}, \hat{i})$ as the anchor point in Equations 6. Then the rasterized arc obtained by using $(\hat{h}, \hat{i})$ as the anchor point and $\hat{\alpha}$ and $\hat{\beta}$ as the curvature factors contains the three defining points.

**Proof.** Let $C$ be the exact arc passing through the three defining points. Let $C_a$ be the exact arc obtained using $(\hat{h}, \hat{i})$ as the anchor point and $\hat{\alpha}$ and $\hat{\beta}$ as the curvature factors. Let $C_b$ be the exact arc obtained using $(\hat{h}, \hat{i})$ as the anchor point and the $\alpha$ and $\beta$ associated with $C$ as the curvature factors (see Figure 3).

From Equations 5 and the definition of $\tau$ in the statement of the theorem, it follows that $(\hat{h}, \hat{i})$ must be at no greater distance than 0.5 from the unquantized anchor point $(h, i)$. Since $\tau$ satisfies the inequality in Lemma 2, the rasterized version of $C_b$ contains the intermediate point.

From Lemma 1 it follows that the tangents of $C_a$ and $C_b$ at the starting point must be equal. Therefore, $C_a$ and $C_b$ touch each other only at the starting point if they are not equal.

From the derivation of Equations 6, it follows that $C_a$ passes through the starting and ending points. But $C$ also passes through the starting and ending points. Therefore, $C$ and $C_a$ intersect at two points. If they intersect at a third point, they are the same arc and the theorem is proved. Otherwise, all points on $C_a$ must be between $C$ and $C_b$ such that the shortest path from any point on $C$ to $C_b$ must intersect $C_a$.

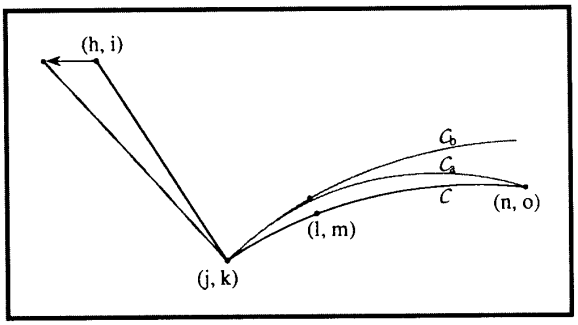


**Figure 3. The arcs $C$, $C_a$, and $C_b$.**

Therefore, $C_a$ passes at least as close to the intermediate point as does $C_b$. Since the rasterized version of $C_b$ contains the intermediate point, so must that of $C_a$. □

## References


1. J. Bresenham, "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *CACM*, Vol. 20, No. 2, Feb. 1977, pp. 100-106.
2. M. Lawrence, "Simple Algorithms for Lines and Circles," *Computer Language*, Vol. 3, No. 11, Nov. 1986, pp. 83-90.
3. J.F. Blinn, "How Many Ways Can You Draw a Circle?" *CG&A*, Vol. 7, No. 8, Aug. 1987, pp. 39-44.
4. J. Van Aken and M. Novak, "Curve-Drawing Algorithms for Raster Displays," *ACM Trans. Graphics*, Vol. 4, No. 2, Apr. 1985, pp. 147-169.
5. J.D. Foley, and A. van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Mass, 1982.
6. P.E. Danielson, "Incremental Curve Generation," *IEEE Trans. Computers*, Vol. 19, No. 9, Sept. 1970, pp. 783-793.
7. B.W. Jordan, W.J. Lennon, and B.D. Holm, "An Improved Algorithm for the Generation of Nonparametric Curves," *IEEE Trans. Computers*, Vol. 22, No. 12, Dec. 1973, pp. 1052-1060.
8. G. Hegron, *Image Synthesis: Elementary Algorithms*, MIT Press, Cambridge, Mass, 1988.



**Ian Galton** is a doctoral student in the Electrical Engineering Department of the California Institute of Technology. His research interests include machine vision and pattern classification algorithms. Before attending Caltech, Galton developed acoustic-beam-formation software for use with a medical ultrasound imaging system at Acuson Corporation in Mountain View, Calif. He also worked as a consultant in the field of computer graphics algorithms.

Galton received his BS in electrical engineering from Brown University in 1984 and his MS in electrical engineering from Caltech in 1989.

The author can be reached at Caltech 116-81, Pasadena, CA 91125, and at galton@electra.caltech.edu.