There Is No Royal Road to Programs

A Trilogy on Raster Ellipses and Programming Methodology

M. Douglas McIlroy

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Wherein, with some insight, some formality and some scorn, ellipse-drawing algorithms, which had been wont unpredictably to stray by a pixel here and there, are brought to heel. The flawed designs of previous algorithms are attributed to premature ‘‘optimization’’: uncritical reuse of an algorithmic scheme that had been tuned for a special case (circles) beyond the point of no return.

There is no royal road to geometry.

The legendary answer of Euclid to King Ptolomey’s expressed desire for a less arduous introduction to the *Elements* carries equal force in programming. There is no substitute for precise analysis.

The problem of drawing ellipses is simple, and the general outline of a solution is clear. Therein lies a danger. Programs get written by specifying a method of solution without fully specifying an objective; mathematics figures only in deriving arithmetic details. When a few test cases look good enough, the program is declared done. But it is fragile because it lacks well defined mathematical properties. Its output can be looked at but not built upon—a bad state of affairs for a basic subroutine. Better results follow from mathematical study of the program as a whole, not just as a collection of isolated statements.

Published algorithms, which attempt to mimic the most highly optimized algorithms for drawing circles, have failed because the optimization depends on symmetry that ellipses lack. Thus this small example illustrates an often ignored truth of software engineering: to extend the functionality of a program, it is sometimes necessary to back off to a more general starting point and rebuild, not just remodel. Since ‘‘better,’’ i.e. more highly tuned, programs are likely to be less adaptable, it may be wise to preserve earlier and less perfected versions for their evolutionary potential.

Jon Bentley, Brian Kernighan, and Chris Van Wyk gave helpful criticism about presentation.

Contents

Getting Raster Ellipses Right

M. Douglas McIlroy

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

A concise incremental algorithm for raster approximations to ellipses in standard position produces approximations that are good to the last pixel even near octant boundaries or the thin ends of highly eccentric ellipses. The resulting approximations commute with reflection about the diagonal and are mathematically specifiable without reference to details of the algorithm.

1. Introduction

We are concerned with approximating an ellipse by lighting pixels on a bitmap. The ellipse is centered on a point of a square grid, which for simplicity we take to be (0,0). The principal axes are parallel to the grid lines. The lengths of the semiaxes are a and b. When both quantities are positive, the ellipse satisfies the familiar equation,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{1}$$

When the length of a semiaxis is zero, the ellipse degenerates into a line segment.

More particularly, we are concerned with incremental approximation algorithms that involve only integer arithmetic. Accordingly a and b are taken to be integers and the grid is taken to be the plane integer lattice.

Ideally an approximation to a simple curve drawn by lighting points of the integer lattice should be

Metrically accurate. Every point of the approximation should be as close to the curve as possible in some sense.

Connected. The approximation should be connected by chess-king moves.

Topologically accurate. The topology of king-move paths in the approximation should be the same as the topology of the original curve.

Thin. Each lighted point should have exactly two lighted king-move neighbors. Thinness is a corollary of topological accuracy.

Symmetric. Approximation should commute with the symmetry operations of the grid: translations, rotations through multiples of $\pi/2$, and reflections about horizontal, vertical and diagonal axes.

Describable. The approximation should be specifiable mathematically without reference to the approximating algorithm.

These desiderata cannot always be met in full.

Thinness and topological accuracy may not be achievable when the scale of features in the original curve is comparable to or smaller than the grid spacing of the bitmap; then pixels approximating different stretches of the curve may come into adjacency or coincidence. In particular, figures with *tails* may result; see Figure 1 and Appendix 2, Lemma 2. We can save the appearances, however, by understanding coincident or irrelevantly adjacent stretches of the approximation to be traced in separate sheets.

Thinness conflicts with metric accuracy at certain pixels called *square corners.* At a square corner the points at three vertices of a grid square are lighted. Square corners sometimes occur in the approximations adopted in this paper; see Figure 2a. However, there can be at most one square corner per quadrant, near the point where the magnitude of the ellipse's slope is 1. We shall argue that such square corners are inevitable: to exorcise them, one would have to sacrifice other critical properties.

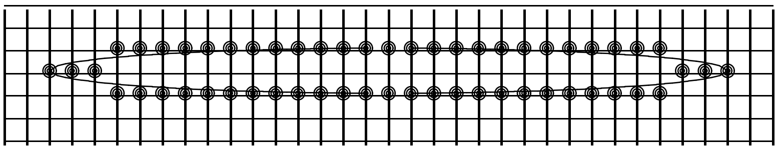


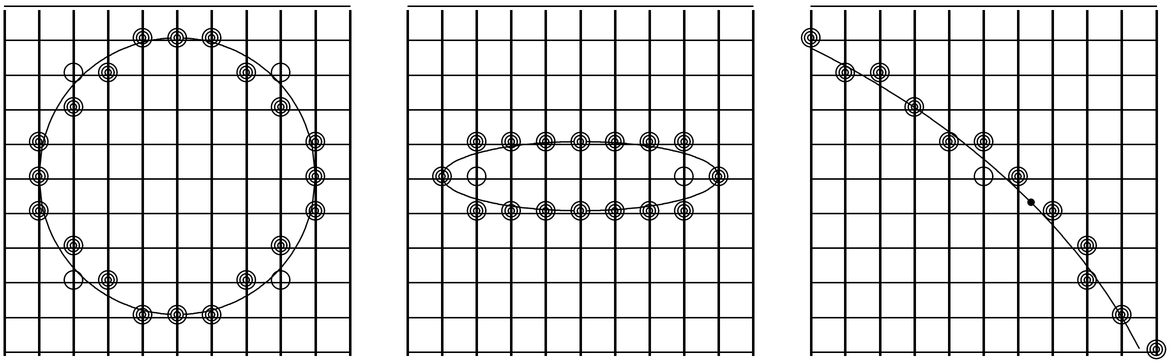
Figure 1. An elongated ellipse with tails, $a = 15,\ b = 1$.

Incremental algorithms trace a connected approximation via chess-king moves, guided by a function that measures goodness of fit. On each of some set of grid lines that intersect the curve a grid point is chosen to minimize one of these criteria:

Displacement of the lighted point from the intersection, measured along the grid line.
Distance of the lighted point from the curve, measured normal to the curve.
Residual of the curve's defining equation evaluated at the lighted point.

The three criteria agree for circles with integer radius,[1, 2] but do not necessarily agree for ellipses; see Figures 2b and 2c.

We shall adopt the minimum-displacement criterion. An approximating point will be classed as a *minimum-horizontal-displacement* point or a *minimum-vertical-displacement* point according to the direction in which the minimized displacement is measured. The two classes are not mutually exclusive.



(a) (b) (c)

Figure 2. (a) Open circles mark square corners in the approximation to the ellipse with $a = b = 4$. (b) The minimum-residual approximation for $a = 4$ and $b = 1$ differs from the minimum-displacement approximation at the points marked by open circles. (c) The minimum-distance approximation for $a = 26$ and $b = 18$ differs from the minimum-displacement approximation at (20,11), marked by an open circle. At (20,11) and (20,12) the vertical displacements from the ellipse are about 0.501 and 0.499; the normal distances are about 0.383 and 0.385. A dot marks the octant juncture where the slope is -1.

A *Freeman approximation* is a minimum-displacement approximation where all grid lines are considered.[3] The handling of ties between two points on one grid line is left open.

Most published incremental algorithms for drawing ellipses split the curve into octants bounded by points where the absolute value of the slope is 0, 1, or ∞. Using compass-point names for a representative outward normal, we speak of the NE quadrant being divided into a NNE and an ENE octant. The *juncture* of the octants is the point with slope -1. Figure 2c shows part of a NE quadrant. A dot marks the juncture. The NNE octant lies to the left of the juncture, the ENE octant to the right. Compass directions are also used to refer to directions between points; with ''northeast'' referring to any bearing properly between north and east, and so forth. A single point lighted to the northeast of the juncture is called an *outside point.* The square corner in the NE quadrant of Figure 2a is an outside point.

The published algorithms that I have seen, by Pitteway,[4, 5] Wirth,[6] Van Aken,[7, 8] Pratt,[9] DaSilva,[10] and Kappel,[11] consider only vertical displacements for the NNE octant and only horizontal displacements for the ENE octant. This convention is certain to yield a thin and connected approximation to each octant because the slope of the ellipse is bounded to the range $[-1,0]$ in the NNE octant and to $[-\infty,-1]$ in the ENE octant. From the slope bounds also follows

Lemma 1. Any minimum-horizontal-displacement point for the NNE *octant is also a minimum-vertical-displacement point for that octant, unless the approximating point is an outside point. A similar statement, with the roles of horizontal and vertical interchanged, holds for the* ENE *octant.*[2]

An outside point may be a minimum-displacement point in both directions, witness configurations **1**, **5**, and **6**, but it need not be, witness configurations **7** and **10**. (Numbered configurations refer to Appendix 1. The reader will find it profitable to detour there and become familiar with the conventions of the diagrams, which illuminate nuances of the problem.)

According to Lemma 1 the one-way approximations that the published algorithms trace are also two-way Freeman approximations—except possibly at the juncture. The thinness of the one-way approximations implies that the Freeman approximation also is thin—again, except possibly at the juncture.

2. Generating the Freeman approximation

We shall develop an analogue of the well known Bresenham algorithm for circles[1] to trace the NE quadrant of an ellipse from north to east. Suppose the approximation has been traced to the point *P* in Figure 3. By monotonicity of the ellipse, the next approximating point will be one of the three neighbors to the east, southeast, or south. Point *E* will be chosen if the ellipse meets either of the unit bars *EV* or *EH* centered there; *S* will be chosen if the ellipse meets *SV* or *SH*; otherwise *SE* will be chosen.

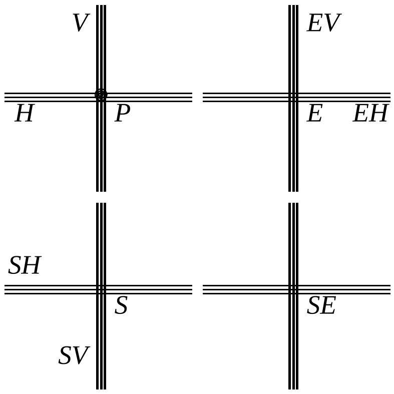


Figure 3.

If the ellipse meets *SV*, then by Lemma 1 it must also meet *SH*, for *S* is a minimum-vertical-displacement point of the ENE octant. The latter fact follows from observing that for the ellipse to meet *SV* with *P* lighted it must have average slope less than -1 in the region south of *H* and north of *SV*; hence the juncture lies north of *SV*.

To check for a south step, then, it suffices to check whether the ellipse meets *SH*. Provided *P* lies above the *x* axis, this is equivalent to checking that the right end of *SH* lies outside the ellipse. We need not worry about the possibility of a tie, where the ellipse meets *SH* and *SE* exactly in their common endpoint, for that cannot happen; see Lemma 3 in Appendix 2. For definiteness in the algorithm, we shall arbitrarily break the tie in favor of the outer point, in this case *SE*.

To check for an east step when *P* lies above the *x* axis, we similarly check whether the ellipse meets *EV* by seeing whether the lower end of *EV* lies inside or on the ellipse. If the ellipse does not meet *EV*, we check whether it meets *EH* by seeing whether the left end of *EH* lies inside or on the ellipse. (For aesthetic consistency, ties are again broken in favor of the outer point.)

By unwarranted analogy with the treatment of *SV*, the cited algorithms ignore *EH*. An algorithm thus simplified will not light the square corner in configurations **7** and **20**, although it will light the corner in the transposed configurations **10** and **18**. Thus the simplification defeats symmetry.

These considerations lead to

Algorithm 0.

```
x := 0
y := b
while y > 0
      mark (x,y)
      if meets EV then step E
      else if meets EH then step E
      else if meets SH then step S
      else step SE
while x ≤ a
      mark (x,y)
      step E
```

The first loop requires $y > 0$ to assure that the *SH* and *EV* tests are made only when *P* lies above the *x* axis. The second loop runs out any remaining steps along the *x* axis.

To express the predicate *meets* analytically, we define an error function *e* as

$$e(x, y) = b^2x^2 + a^2y^2 - a^2b^2$$

The equation of the ellipse is $e(x, y) = 0$ and the meeting conditions are

$$\begin{aligned} &\textit{meets EV}:\ e(x+1, y-½) \le 0 \\ &\textit{meets EH}:\ e(x+½, y) \le 0 \\ &\textit{meets SH}:\ e(x+½, y-1) > 0 \end{aligned}$$

The half-integer quantities can be respected in integer calculations by scaling, or by appropriately rounding the fractional part. At the considered points, which all lie near $e = 0$, the magnitude of *e* is on the order of $\max(a^3, b^3)$. Thirty-two-bit arithmetic with rounding (but not with scaling) is adequate to cope with values of *a* and *b* to just under 900.*

We transform the `if` tests algebraically to use integer arithmetic and the integer common subexpression $e(x+½, y-½) - (a^2 + b^2)/4$.

$$\begin{aligned} &e(x+1, y-½) \le 0 \qquad \{EV\} \\ &e(x+½, y-½) + e(x+1, y-½) - e(x+½, y-½) \le 0 \\ &e(x+½, y-½) + b^2(x+1)^2 - b^2(x+½)^2 \le 0 \\ &e(x+½, y-½) - (a^2 + b^2)/4 + (a^2 + b^2)/4 + b^2(x+1)^2 - b^2(x+½)^2 \le 0 \\ &e(x+½, y-½) - (a^2 + b^2)/4 + b^2x \le -a^2/4 - b^2 \\ &e(x+½, y-½) - (a^2 + b^2)/4 + b^2x \le -\lfloor a^2/4 \rfloor - (a \bmod 2) - b^2 \end{aligned}$$

The rounding in the right side of the last inequality is justified by the integrality of the left hand side. Similarly

$$\begin{aligned} &e(x+½, y) \le 0 \qquad \{EH\} \\ &e(x+½, y-½) - (a^2 + b^2)/4 + (a^2 + b^2)/4 + e(x+½, y) - e(x+½, y-½) \le 0 \\ &e(x+½, y-½) - (a^2 + b^2)/4 + a^2y \le -b^2/4 \\ &e(x+½, y-½) - (a^2 + b^2)/4 + a^2y \le -\lfloor b^2/4 \rfloor - (b \bmod 2) \end{aligned}$$

* Since an approximate point (x,y) may be up to 1/2 unit off the ellipse, a test point, say $(x+½, y-1)$, may be 3/2 unit off. At such a point, with $a = b$, $y \cong a$, and $x \cong 0$, we estimate $|e(x,y)| \cong |(\partial e/\partial y)\Delta y| \cong 3a^3$. Thus $e(x,y)$ is liable to overflow 32-bit registers at $a > (2^{31}/3)^{1/3}$, or $a > 894$.

$$e(x+½,\, y-1) > 0 \qquad \{SH\}$$
$$e(x+½,\, y-½) - (a^2+b^2)/4 + (a^2+b^2)/4 + e(x+½,\, y-1) - e(x+½,\, y-½) > 0$$
$$e(x+½,\, y-½) - (a^2+b^2)/4 - a^2 y > -b^2/4 - a^2$$
$$e(x+½,\, y-½) - (a^2+b^2)/4 - a^2 y > -\lfloor b^2/4 \rfloor - (b \bmod 2) - a^2$$

Installing the transformed tests and arranging to calculate $e(x+½,\, y-½)$ incrementally, we get the following program, for which $t = e(x+½,\, y-½) - (a^2+b^2)/4$ is a loop invariant. Appendix 3 gives an implementation in C.

Algorithm 1.

```
x := 0
y := b
t := b²(x² + x) + a²(y² − y) − a²b²
while y > 0
      mark(x,y)
      if t + b²x ≤ −⌊a²/4⌋ − (a mod 2) − b²          {e(x+1, y−½) ≤ 0; EV}
            x += 1
            t += b²(2x + 2)
      else if t + a²y ≤ −⌊b²/4⌋ − (b mod 2)           {e(x+½, y) ≤ 0; EH}
            x += 1
            t += b²(2x + 2)
      else if t − a²y > −⌊b²/4⌋ − (b mod 2) − a²      {e(x+½, y−1) > 0; SH}
            y −= 1
            t += a²(−2y+2)
      else
            x += 1
            y −= 1
            t += b²(2x + 2) + a²(−2y+2)
while x ≤ a
      mark(x,y)
      x += 1
```

It is a straightforward matter to check that the program works in degenerate cases where a or b is zero.

On the x axis the *EH* test would evaluate to true at points inside the ellipse, or at any point if $b = 0$. By modifying the loop condition so the *EH* test gets performed with $y = 0$ when there is a tail, we may drop the second loop. The program in Appendix 3 incorporates this idea. Appendix 3 also explains how to gain speed by exploiting the fact that not all of the tests are needed in all parts of the quadrant.

To trace an elliptic arc that spans only part of a quadrant, Algorithm 1 can be started at any minimum-displacement point. One possible sticking point, overflow during the initialization of t, is more apparent than real. If t is in range, it can be computed in unsigned arithmetic without regard to overflow to yield a correct twos-complement result.

3. Discussion

Rescuing *EH* from unjustified oblivion, Algorithm 1 generates a genuine Freeman approximation, yet can be implemented as compactly as comparable algorithms that yield ad hoc approximations; witness Appendix 3. Although I have confidence in it, the derivation has been long, informal, and riddled with case analysis. A proof outline exists,[12] but it would be reassuring to have a formal proof.

The Freeman approximation satisfies the six desiderata set forth at the outset, save for the possibility of having four square corners. The corners could be sheared off, although I don't know a way to do so without extra code. Rejecting square corners would entail other difficulties as well. Accuracy may be lost: in configuration **6**, for example, the ‘‘bad’’ corner point is noticeably closer to the ellipse than are either of its ‘‘good’’ neighbors. In visual terms, uniformity of line will be bought at the expense of roundness of shape, as Figure 2a shows. Most tellingly, the algorithm's usefulness for drawing elliptic arcs would be sacrificed, as we shall see shortly. Arcs would have to be drawn differently, with the almost certain result that an arc ending at the juncture would not coincide with the underlying ellipse there.

The cited algorithms differ primarily in their treatment of the juncture. They switch between the NNE and ENE octants according to various heuristic criteria that defeat one or more of the desiderata. As a result none yields an approximation that is mathematically definable without reference to the algorithm. All can, but do not necessarily, shear off square corners. DaSilva's can stray; in configuration **5** DaSilva visits (9,5) rather than (9,6).[10]

Tails bedevil most of the algorithms; only Pratt is careful about them.[9] Pitteway's original paper mentions tails but does not handle them;[4] his later paper tries to cope by backing off to 4-connected (rook-move) approximations.[5] Typically algorithms designed without regard to tails suffer a catastrophic tracking failure when a bar in Figure 3 stretches clear across the ellipse instead of cutting just one branch.[9, 10] The fishy tails of Figure 4, for example, arise from such a tracking failure in Wirth's algorithm.[6] (In partial redemption, Wirth's is the only algorithm that respects symmetry—by virtue of handling the major and minor axes unsymmetrically!)

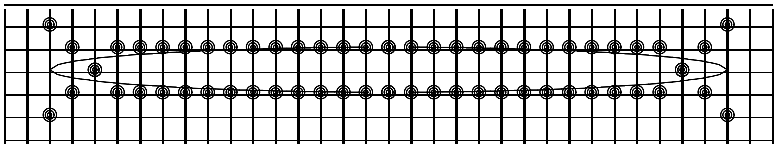


Figure 4. A typical mistake in drawing tails. The ellipse is the same as that in Figure 1: $a = 15$, $b = 1$.

Why have the published algorithms eschewed the Freeman approximation in favor of ad hoc criteria? Optimistic imitation of the best algorithms for circles is doubtless part of the reason. When the Pitteway-like algorithms were seen to produce visually satisfactory ellipses, details such as precise determination of the juncture and respect for symmetry were simply forgotten. Perhaps the Freeman approximation has been overlooked also because of an unspoken (and groundless, in view of Lemma 1) worry about the possibility of excessive square corners. Almost certainly the possibility of configurations such as **7** and **18** has been overlooked.

Most of all, though, a desire for a fast loop[9] has probably upstaged other considerations: the programs have been optimized prematurely. At least for drawing full ellipses, where four points are plotted for each one that is calculated, the price of one extra test to get the Freeman approximation is negligible.

Because their approximations are indescribable, the published algorithms cannot easily be modified to draw elliptic arcs given the endpoints. Even when a proposed endpoint is verified to be a minimum-displacement point, it may not belong to the approximation. It could, for example, be a square corner that the algorithm skips. An infinite loop can result from testing for termination against such a point. In contrast, an arc-tracing program based on the Freeman approximation can be made accurate and safe because the question of whether a point belongs to the approximation can be quickly decided.*

Some open questions: Is the uncertain configuration **13** realizable? Is there a simpler way to find the Freeman approximation? Can general conic sections be handled as easily?

I wish to acknowledge Rob Pike, who requested the program, the reference that stimulated it,[6] fellow members of IFIP Working Group 2.3 on Programming Methodology for their insights into program development, which helped shape it, and conscientious referees, who helped polish it.

* Solve (1) for y at integer x (or vice versa) and round, or check for a sign difference in the the error function e evaluated at the ends of bars V and H in Figure 3.

References

[1] Bresenham, J., ‘‘A linear algorithm for incremental digital display of circular arcs,’’ *Comm. ACM* **20**, pp. 100-106 (1977).

[2] McIlroy, M. D., ‘‘Best approximate circles on integer grids,’’ *ACM Trans. on Graphics* **2**, pp. 237-264 (Oct. 1983).

[3] Freeman, H., ‘‘Computer processing of line-drawing images,’’ *Computing Surveys* **6**, p. 63 (1974).

[4] Pitteway, M. L. V., ‘‘Algorithms for drawing ellipses or hyperbolae with a digital plotter,’’ *Computer J.* **10**, pp. 282-289 (1967).

[5] Pitteway, M. L. V., ‘‘Algorithms of conic generation,’’ pp. 219-237 in *Fundamental Algorithms for Computer Graphics*, Earnshaw, R. A. (Ed.), Springer-Verlag, Heidelberg (1985).

[6] Wirth, N., ‘‘Drawing lines, circles and ellipses in a raster,’’ pp. 427-434 in *Beauty is our Business*, Feijen, W. H. J., van Gasteren, A. J. M., Gries, D., and Misra, J. (Eds.), Springer-Verlag, New York (1990).

[7] Van Aken, J. R., ‘‘An efficient ellipse-drawing algorithm,’’ *IEEE Computer Graphics and Applications* **4**(9), pp. 24-35 (1984).

[8] Van Aken, J. and Novak, M., ‘‘Curve-drawing algorithms for raster displays,’’ *ACM Transactions on Graphics* **4**(2), pp. 147-169 (1985).

[9] Pratt, V., ‘‘Techniques for conic splines,’’ *Computer Graphics* **19**(3), pp. 151-159, Barsky, B. A. (Ed.), ACM (1985). SIGGRAPH ’85 Conference Proceedings.

[10] Foley, J. D., Van Dam, A., Feiner, S. K., and Hughes, J. F., *Computer Graphics Principles and Practice*, Addison-Wesley (1990).

[11] Kappel, M. A., ‘‘An Ellipse-Drawing Algorithm for Raster Displays,’’ pp. 257-280 in *Fundamental Algorithms for Computer Graphics*, Earnshaw, R. A. (Ed.), Springer-Verlag, Heidelberg (1985).

[12] McIlroy, M. D., ‘‘There is no royal road to programs: a trilogy on raster ellipses and programming methodology,’’ Computing Science Technical Report 155, AT&T Bell Laboratories (1990).

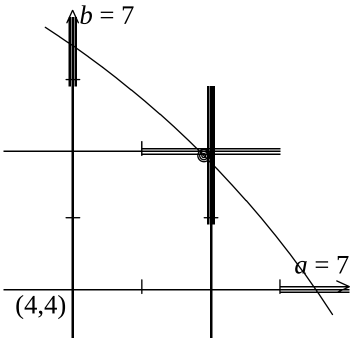
[13] Lakatos, I., *Proofs and Refutations: the Logic of Mathematical Discovery*, Cambridge (1976).

Appendix 1. Inventory of configurations at the juncture

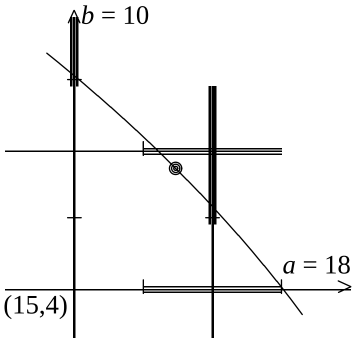
The diagrams below enumerate all distinct ways that the ellipse can cross grid lines in the neighborhood of the juncture, where the slope is -1. Each diagram shows the four grid lines that surround the juncture. Bars indicate intervals of equivalent crossings as in Figure 3. The midpoint of a bar will be lighted if the ellipse intersects the bar. A bar missing from the top (or right) line is understood to be somewhere out of sight to the left (or bottom). A dot marks the juncture.

Numbers show the dimensions and the coordinates of a grid point for an ellipse that realizes the configuration with $a \geq b$. Configurations marked X are impossible; see Lemmas 4 and 5 in Appendix 2. Configuration **13**, marked ?, has no representative with $a \leq 1000$; its possibility is a number-theoretic question. Note, though, that configuration **13** is the transpose of **19**, and thus can be realized with $a < b$.

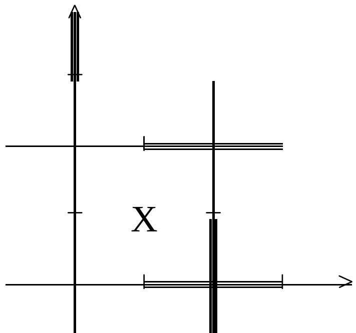
The inventory is ordered lexicographically by decreasing positions of the bar on the left, top, right, and bottom grid lines. Configurations inconsistent with a monotone decreasing curve are not shown. Neither are configurations that would violate slope requirements in grid squares that do not contain the juncture: the average slope of the ellipse across a square must be at least -1 if only the NNE octant enters it, and at most -1 if only the ENE octant.



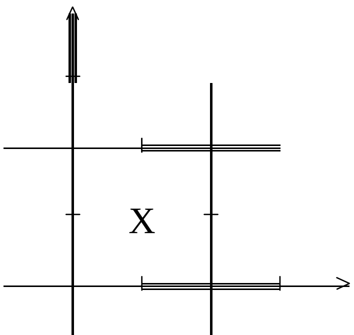
1



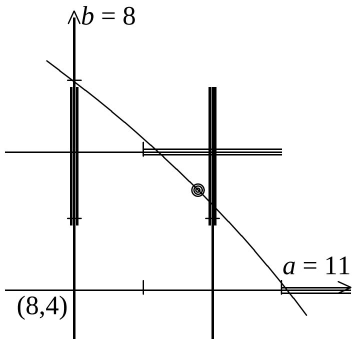
2



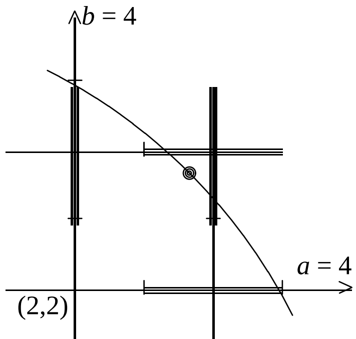
3



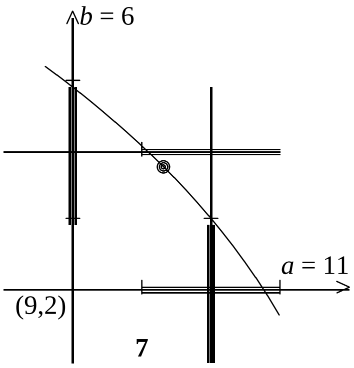
4



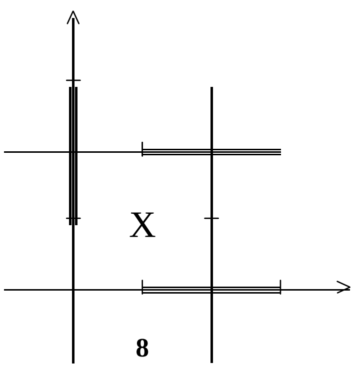
5



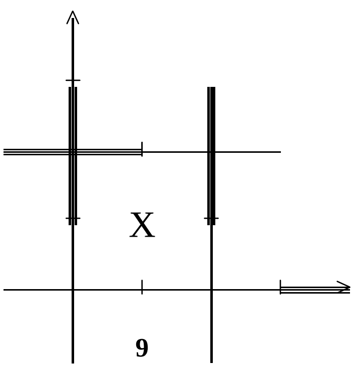
6



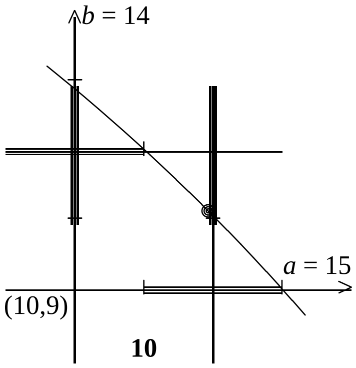
7



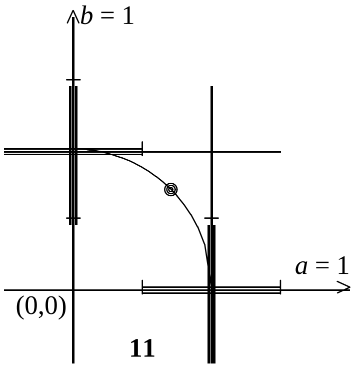
8



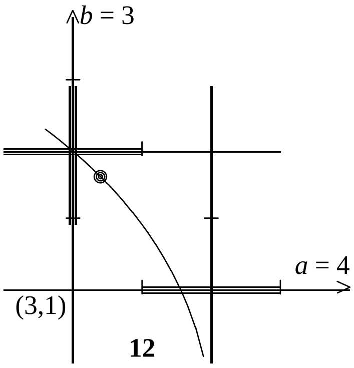
9



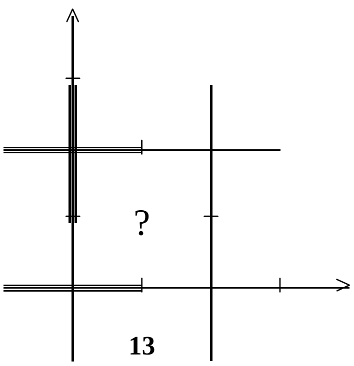
10



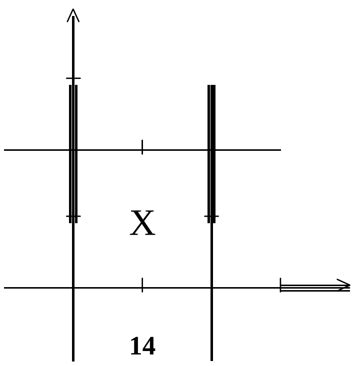
11



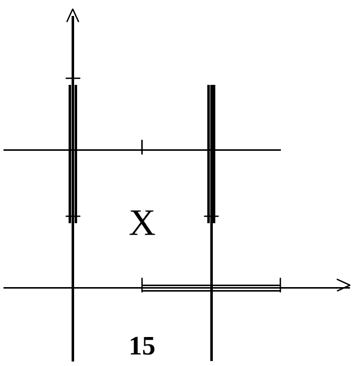
12



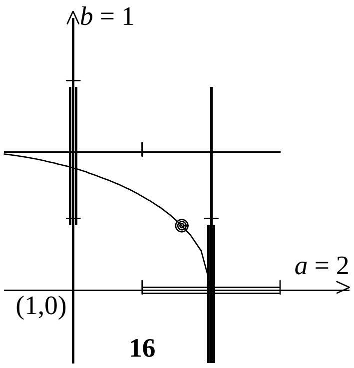
13



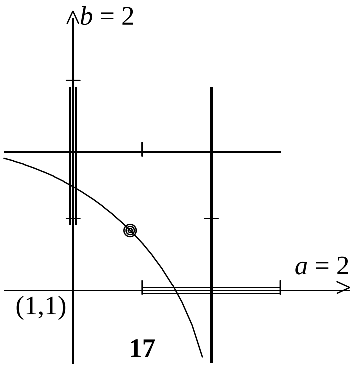
14



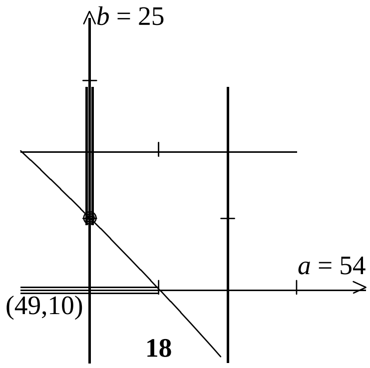
15



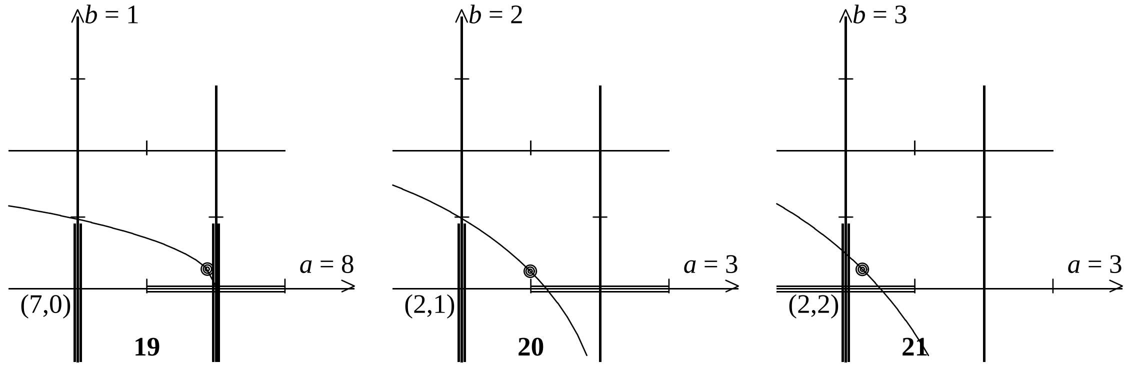
16



17



18



Appendix 2. Supporting lemmas

Lemma 2. An approximate ellipse with $a \geq b$ has tails if and only if $a \geq 8b^2$ and $a > 0$.

A tail occurs if $(a-1,0)$ is lighted, or in other words if and only if the ordinate of the ellipse at $x = a-1$ is less than $1/2$. Thus $a > 0$ and

$$y^2 = b^2(1 - (a-1)^2/a^2) < 1/4$$

Expand and clear of fractions:

$$a^2 - 8ab^2 + 4b^2 > 0$$

Now a must exceed the larger root of the associated quadratic equation. (The smaller root is less than 1.)

$$a > 4b^2 + \sqrt{16b^4 - 4b^2}$$

If $b > 0$, this is equivalent to

$$a > 4b^2 + 4b^2\sqrt{1 - \frac{1}{4b^2}}$$

By Taylor's theorem with remainder

$$a > 8b^2 - 1/2 + R$$

where

$$0 < R \leq \frac{1}{8}\left[\frac{1}{4b^2}\right]^2\left[1 - \frac{1}{4b^2}\right]^{-3/2}$$

Since b is a positive integer, R is surely less than $1/2$. Using the fact that a is an integer, we find $a \geq 8b^2$ for positive b. The result also holds for $b = 0$ and $a > 0$, in which case the approximate ellipse degenerates to a line segment—all tail.

Lemma 3. The ellipse of equation (1) with integral a and b does not pass through any point, one coordinate of which is an integer, and the other half an odd integer.

Suppose that the ellipse does pass through such a point, (x,y). Without loss of generality, let x be an integer and $y = z/2$, where z is an odd integer. We may assume that $gcd(x,a) = gcd(z,2b) = 1$; if it were not, we could reduce the fractions in the defining equation

$$\frac{x^2}{a^2} + \frac{z^2}{4b^2} = 1$$

to get a counterexample of the same form in which the assumption does hold. The sum of two fractions in lowest terms can be 1 only if their denominators are the same. Hence $a = 2b$. Because x/a is in lowest terms, x must be odd. Consequently we have a triple (x,z,a) with parities (odd,odd,even) that satisfies

$$x^2 + z^2 = a^2$$

But as is well known, no such triple exists, for that would imply

$$1 + 1 \equiv x^2 + z^2 \equiv a^2 \equiv 0 \pmod 4.$$

Lemma 4. Configurations **4**, **8**, **14**, *and* **15** *are impossible.*

The coordinates of the juncture, (X, Y), must satisfy both the equation of the ellipse

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{1}$$

and its derivative

$$\frac{2x}{a^2} + \frac{2y}{b^2}\frac{dy}{dx} = 0$$

with $dy/dx = -1$. Solving simultaneously, we find

$$X = \frac{a^2}{\sqrt{a^2 + b^2}}, \quad Y = \frac{b^2}{\sqrt{a^2 + b^2}} \tag{2}$$

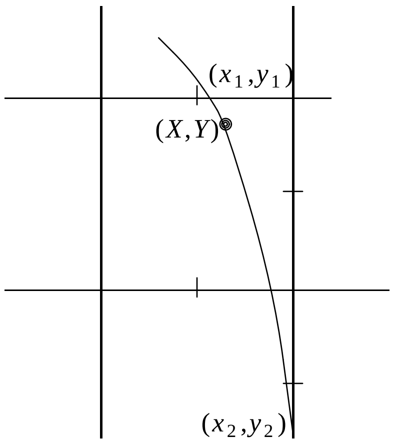


Figure 5.

Configurations **4** and **8** are both exemplified by Figure 5. The ellipse meets the top grid line at (x_1, y_1) and the right grid line at (x_2, y_2). From the figure we see that

$$1/2 \le x_1 \le X < x_2 \le x_1 + 1/2 \tag{3}$$

$$0 \le y_2 < y_1 - 3/2 < Y \le y_1 \tag{4}$$

From the equation of the ellipse (1),

$$(x_1/a)^2 + (y_1/b)^2 = 1$$

$$(x_2/a)^2 + (y_2/b)^2 = 1$$

Subtract and factor.

$$(y_1 - y_2)(y_1 + y_2)/b^2 = (x_2 - x_1)(x_2 + x_1)/a^2$$

From (3) $x_2 - x_1 \le 1/2$. Also from (3), the second factor on the right is at most $2x_1 + 1/2$, which in turn is at most $2X + 1/2$. Hence

$$\frac{a^2}{b^2}(y_1 - y_2)(y_1 + y_2) \le X + \frac{1}{4}$$

From (4), $y_1 - y_2 > 3/2$ and $y_1 \geq Y$:

$$\frac{3}{2}\frac{a^2}{b^2}(Y + y_2) < X + \frac{1}{4}$$

According to (2), $a^2 Y/b^2 = X$. Substituting and simplifying, we find

$$X < \frac{1}{2} - 3\frac{a^2}{b^2}y_2$$

From (4) y_2 is nonnegative. Thus the last inequality implies $X < 1/2$, which contradicts (3). The figure is impossible, as are its instances **4** and **8.** The derivation has not used the assumption $a \geq b$, so a similar argument proves the impossibility of the transposed configurations **14** and **15**.

Lemma 5. Configurations **3** *and* **9** *are impossible.*

Figure 6a illustrates configuration **3**. The ordinates of points A and B differ by more than 1. We shall show this is impossible by considering Figure 6b. There points A and B' are intersections of adjacent grid lines with an ellipse in standard position. The ordinates of A and B' differ by exactly 1. By Rolle's theorem,* the juncture (X', Y'), where the slope is -1, must lie between the grid lines. Let point (x_0, y_0) be the midpoint of AB', and let the lengths of the semiaxes of the ellipse be a' and b'. (None of x_0, y_0, a', or b' is constrained to be integral.) Since both A and B' lie on the ellipse,

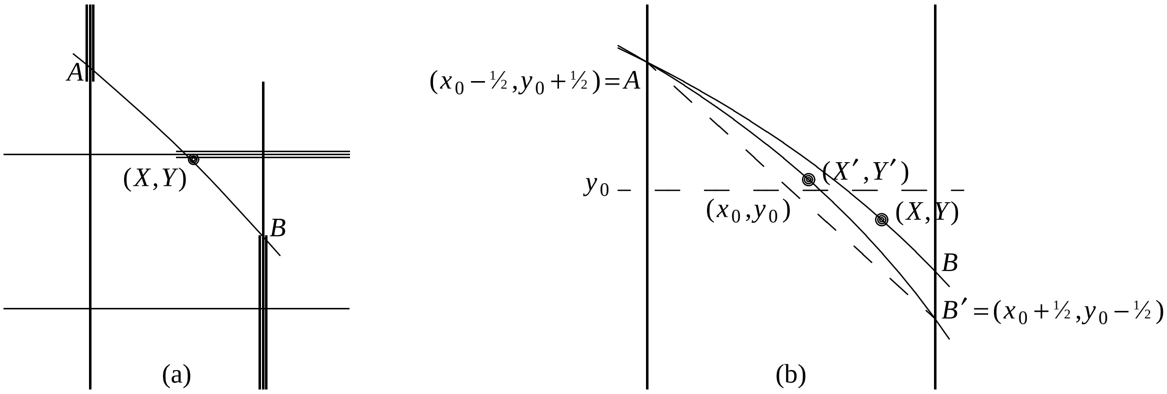


Figure 6.

$$\frac{(x_0 - ½)^2}{a'^2} + \frac{(y_0 + ½)^2}{b'^2} = 1$$

$$\frac{(x_0 + ½)^2}{a'^2} + \frac{(y_0 - ½)^2}{b'^2} = 1$$

Solve simultaneously for a'^2 and b'^2:

$$a'^2 = \frac{(x_0 + y_0)(x_0 y_0 + 1/4)}{y_0}$$

$$b'^2 = \frac{(x_0 + y_0)(x_0 y_0 + 1/4)}{x_0}$$

Substituting in (2), we find

$$Y'^2 = y_0^2 + \frac{1}{4}\frac{y_0}{x_0} \tag{5}$$

* Somewhere between the ends of a chord of a smooth curve, the tangent to the curve must be parallel to the chord.

Consider now the one-parameter family of ellipses that pass through A. One member has arc AB'. A second has arc AB; let a and b be the lengths of its semiaxes. Then

$$\frac{(x_0 - \frac{1}{2})^2}{a^2} + \frac{(y_0 + \frac{1}{2})^2}{b^2} = 1$$

Since x_0 and y_0 are fixed, a and b must vary inversely with each other. Let (X, Y) be the juncture of the second ellipse. From (2) we see that as a increases and b decreases, Y must decrease and vice versa. Suppose the curve in Figure 6a to be the curve AB in Figure 6b. Its juncture must lie at least one half unit below A; thus $Y < y_0$. From (5), $y_0 < Y'$, so $Y < Y'$. From the inverse variation of a and Y it follows that $a > a'$ and arc AB must lie outside arc AB', as shown. Therefore Figure 6b requires the ordinates of A and B to differ by less than 1, while Figure 6a requires them to differ by more than 1. This completes the proof of the impossibility of configuration **3**. Since the proof does not depend on the assumption $a \geq b$, the transposed configuration **9** is also impossible.

Appendix 3. Implementation in C

This program is based on the model in the text, and incorporates simplifications discussed there plus other routine optimizations. The initializer for `t` has been specialized to take into account the known values of `x` and `y`. Constant calculations have been moved out of the loop. The strength of most multiplications in the loop has been reduced. Variables `xc` and `yc` are the coordinates of the center of the ellipse. The arithmetic fits in 32-bit `long` integers for values of `a` and `b` less than 896; the exact value has been confirmed experimentally. Beware, the comma operator denotes serial, not parallel, assignment.

```
extern void point(int, int);

#define incx() x++, dxt += d2xt, t += dxt
#define incy() y--, dyt += d2yt, t += dyt

void ellipse(int xc, int yc, int a, int b)
{                       /* e(x,y) = b^2*x^2 + a^2*y^2 - a^2*b^2 */
        int x = 0, y = b;
        long a2 = (long)a*a, b2 = (long)b*b;
        long crit1 = -(a2/4 + a%2 + b2);
        long crit2 = -(b2/4 + b%2 + a2);
        long crit3 = -(b2/4 + b%2);
        long t = -a2*y; /* t = e(x+1/2,y-1/2) - (a^2+b^2)/4 */
        long dxt = 2*b2*x, dyt = -2*a2*y;
        long d2xt = 2*b2, d2yt = 2*a2;

        while(y>=0 && x<=a) {
                point(xc+x, yc+y);
                if(x!=0 || y!=0)
                        point(xc-x, yc-y);
                if(x!=0 && y!=0) {
                        point(xc+x, yc-y);
                        point(xc-x, yc+y);
                }
                if(t + b2*x <= crit1 ||   /* e(x+1,y-1/2) <= 0 */
                   t + a2*y <= crit3)     /* e(x+1/2,y) <= 0 */
                        incx();
                else if(t - a2*y > crit2) /* e(x+1/2,y-1) > 0 */
                        incy();
                else {
                        incx();
                        incy();
                }
        }
}
```

Optimization

Further modifications for efficiency are possible, but few are justifiable unless point-drawing is unusually fast and ellipses are unusually common in relation to other graphic primitives. Common subexpressions can be eliminated and constants propagated. More multiplications can be reduced to additions. There is no need to test $x \le a$ unless $y = 0$. Further tests can be eliminated by splitting the single loop into four: vertical tail, NNE arc, ENE arc, and horizontal tail. There are only south steps in the vertical tail, which continues while south steps are possible, only east steps in the horizontal tail, which continues while $x \le a$. There are no east steps in the ENE arc, which may begin at the first south step in the NNE arc. Points in the vertical tail are reflected vertically, in the arcs vertically and horizontally, and in the horizontal tail horizontally unless $a = 0$. The rarely effective *EH* test (`crit3`) may be placed last in the loop for the NNE arc, the only place where it remains necessary.

Testing

Mathematical proofs have lives of their own, and evolve as the context of theorems becomes more fully explored.[13] This phenomenon appears also in programming, where the exploration takes the form of testing and use. Programming has the further complication of bridging the gap between proof and implementation: does the code faithfully mirror what was proved? Skeptical testing is never amiss.

Besides various sporadic checks, the C program has been tested

For all values of a and b in the range 0 to 4.

For circles of integer radius up to 20 and of radius divisible by 100 up to 1000.

Over small ranges around the first few critical points for tails, i.e. with one parameter near 8 times the square of the other, in both orientations.

For $a = 1000$ and $b = 1$ and vice versa.

For a and b in the range [890,900], which spans the onset of 32-bit overflow.

For square corners in approximate circles, which happen for only four radii less than 1000, namely 4, 11, 134, and 373.[1]

For the cases pictured in Appendix 1, the parameters for which were independently determined.

Outputs were checked mainly by mathematical, not merely visual, criteria. For each test case some of the following checks were made.

Termination: a quadrant beginning at $(0,b)$ ends at $(a,0)$.

Symmetry: the approximation for (a,b) mirrors that for (b,a); circles have eight-fold symmetry.

Continuity: no coordinate changes by more than one at any step.

Thinness: a quadrant has at most one square corner.

Square corners happen where predicted.

Spot checks against ellipse coordinates calculated in floating point.

The dimensions at onset of tails.

Comparison against output from Wirth’s algorithm, sometimes for agreement and sometimes for predicted disagreement.

Math before Code:
A Soundly Derived Ellipse-drawing Algorithm

M. Douglas McIlroy

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

The problem of drawing an ellipse on a raster is posed mathematically as the problem of constructing a Freeman approximation to a curve. A straightforward program is proved to generate the approximation, and then transformed into an efficient all-integer scheme. Its logic having been shaped by mathematics, the result is free of anomalies that bedevil previously published programs, the outlines of which were designed with only cursory mathematical specification. Mathematics was used mainly to fill in details. The outlines, borrowed from the most efficient algorithms for drawing circles, didn't work because they had been specialized too far to be readily generalizable in a new direction.

This small example highlights a serious pitfall in software evolution: advanced code is often an inappropriate platform from which to launch radical advances in code.

Introduction

This note attempts to go beyond the relatively intuitive development in the accompanying paper, ''Getting Raster Ellipses Right,'' to give a clear outline of a correctness proof of a simple ellipse-drawing algorithm.

The algorithm generates a Freeman approximation,[1] wherein a curve is quantized by plotting on each grid line the nearest grid point to each intersection of that line with the curve. Freeman approximation enjoys the distinction of being mathematically describable, readily computable, and respectful of the symmetries of the grid, in the sense that approximation commutes with the symmetry operations.

A not-so-deeply hidden subtext is that formal analysis often has a place in the practical development even of quite ''obvious'' algorithms. If the objective of a program is not perfectly clear, it will help to spell it out precisely, and to be clear about why the program meets the objective. Ditto for critical invariants. The message is not new, but perhaps the application area is; graphics is one of many redoubts of seat-of-the-pants programming, where mathematical understanding of the application is not matched by mathematical analysis of the code.

At the end of the paper, the algorithm is contrasted with previous ones in the literature, all of which can produce mathematically anomalous results. It is argued that the method of development, from mathematics to program logic and not vice versa, is responsible for the better behavior of the present algorithm.

Terminology

A *point* P is a coordinate pair $(P.x, P.y)$.

Point P is *north* of point Q if $P.y > Q.y$, and *directly north* of Q if $P.x = Q.x$ and $P.y > Q.y$. Similar definitions hold for east, south, and west.

Point P is *northeast* of point Q if P is both north and east of Q, and similarly for southeast, southwest, and northwest.

A *grid point* is a point with integer coordinates. When it can be inferred from context, a grid point may be referred to simply as a point.

The neighbor functions *north*(P) denotes the nearest to P among all grid points directly north of point P; and *northeast*(P) denotes the nearest to grid point P among all grid points northeast of P. Neighbors in the remaining six principal compass directions are designated similarly.

Let function e be defined by $e(a,b,x,y) = b^2x^2 + a^2y^2 - a^2b^2$. If $a > 0$ and $b > 0$, *the ellipse* is the curve in the *x-y* plane defined by $e(a,b,x,y) = 0$, with the traditional canonical form $x^2/a^2 + y^2/b^2 = 1$.

In the first quadrant the ellipse may be equivalently specified by $y = f(a,b,x)$ or by $x = f(b,a,y)$, where

$$f(a,b,x) \;=\; b\sqrt{1 - x^2/a^2}\,, \quad 0 \le x \le a,\ 0 < a,\ 0 < b.$$

If $a = 0$ the ellipse degenerates to a north-south segment; if $b = 0$ it degenerates to an east-west segment.

Associated with each grid point in the first quadrant is a *vertical bar* denoted $P.V$, which is a north-south segment of length 1 centered on P, and a similarly centered *horizontal bar,* $P.H$. The vertical bar is half open to the north; the horizontal bar is half open to the east.

The predicate $V(P)$ means that the ellipse intersects $P.V$ and $H(P)$ means that the ellipse intersects $P.H$. Formally,

$$V(P) \;\equiv\; P.y - ½ \le f(a,b,P.x) < P.y + ½,$$
$$H(P) \;\equiv\; P.x - ½ \le f(b,a,P.y) < P.x + ½.$$

Grid point P is said to be *lighted* if the ellipse intersects either bar, i.e. if the ellipse passes near enough to P to make $V(P)$ or $H(P)$ true.

Basic observations

1. Function f is continuous and one-to-one.
2. The value of $f(a,b,x)$ decreases monotonically as x increases from $x = 0$ to $x = a$.
3. The slope of the curve $y = f(a,b,x)$ decreases monotonically as x increases from $x = 0$ to $x = a$.

The following three lemmas are straightforward consequences of continuity and monotonicity.

Lemma 1. If a first-quadrant point P is lighted, no first-quadrant point northeast or southwest of P is lighted.

Lemma 2. If P is lighted, and P is interior to the first quadrant, then (at least) one of *east*(P), *southeast*(P), or *south*(P) is lighted.

Lemma 3. If grid points P and R, where R is directly east (or south) of P, are lighted, then the ellipse intersects $Q.V$ (or $Q.H$) for every grid point Q in the interior of the line segment from P to R.

Problem statement

Given nonnegative integers a and b, construct the Freeman approximation to the ellipse in the first quadrant, i.e. construct the set S of

lighted grid points in $\{P \mid P.x \ge 0 \;\&\; P.y \ge 0\}$ if $a > 0$ and $b > 0$;
grid points in $\{(0,y) \mid 0 \le y \le b\}$ if $a = 0$;
grid points in $\{(x,0) \mid 0 \le x \le a\}$ if $b = 0$.

The algorithm should use only integer arithmetic.

General plan

We shall walk the grid visiting lighted points and accumulating a set T of lighted points. The walk starts at the north and steps east or south whenever possible, otherwise southeast, and leaves the quadrant only when all lighted points have been visited. Other quadrants may be filled in by symmetry.

Program 0 handles a narrowed problem, which omits the possibility of $a = 0$ or $b = 0$ and uses real, not integer, arithmetic. Transformations and generalizations finally lead to Program 4, an efficient solution to the full problem.

Invariant

When grid point P is visited, P is lighted and T comprises all lighted points north or west of P.

Program 0.

Precondition: $a > 0$ & $b > 0$

```
T := ∅
P := (0,b)
while P.y ≥ 0 & P.x ≤ a
        T ∪= P
        if P.x < a && V(east(P)) → P.x += 1                 {1}
        [] P.x < a & H(east(P)) → P.x += 1                  {2}
        [] P.y > 0 && H(south(P)) → P.y −= 1                {3}
        [] P.y > 0 & V(south(P)) → P.y −= 1                 {4}
        [] P.x = a → P.y −= 1                               {5}
        [] P.y = 0 → P.x += 1                               {6}
        else → P.x += 1, P.y −= 1                           {7}
```

Postcondition: $T = S$

In program 0 some of the guards use &&, the conditional and operator from C, to forestall evaluating *f* outside its range. The set-updating operator $\cup=$ is formed analogously to $+=$ in C.

The pseudoguard `else`, with the obvious meaning of the the negation of the disjunction of all other guards, is used to control the execution of a southeast step. Direct tests of $H(\mathit{southeast}(P))$ and $V(\mathit{southeast}(P))$ would be inadequate because the event of the southeast neighbor being lighted is not necessarily incompatible with the east (or south) neighbor being lighted. In such a case, the other neighbor must be visited first lest it be skipped.

Proof of Program 0

Initialization. The initial point $(0,b)$ is lighted. There are no lighted points to the north of $(0,b)$ and no first quadrant points to the west. Thus the invariant is established.

Termination. At every iteration $P.x$ increases or $P.y$ decreases towards its respective bound. The loop can (and must) terminate only after the extreme point $(a,0)$ is visited. The truth of the postcondition is established at case 5 below.

The invariant will be proved by analyzing the numbered cases in the loop.

Cases 1 and 2. The guards assure that $\mathit{east}(P)$ is lighted. By the invariant, all lighted points north or west of P are in T before the step. By Lemma 1, no point southwest of $\mathit{east}(P)$ nor northeast of P is lighted; adding P to T preserves the invariant.

Cases 3 and 4. Like cases 1 and 2 with south exchanged for east, north for west, and V for H.

Case 5. The current point, $P = (a,y)$, is lighted, as is the extremum of the ellipse, $(a,0)$. Hence (by Lemma 3, if $y > 1$) $\mathit{south}(P)$ is lighted. The invariant is thus preserved when $P.y > 0$. If $P.y = 0$, the invariant and the fact that there are no lighted points east of $(a,0)$, imply that the updating of T establishes the postcondition. The loop terminates by stepping to an unlighted point.

Case 6. Like case 5, with $P = (x,0)$.

Case 7. The other guards assure that neither $\mathit{east}(P)$ nor $\mathit{south}(P)$ is lighted, that $P.y > 0$, and that $P.x < a$. Thus, by Lemma 2, $\mathit{southeast}(P)$ is lighted. Let $Q = \mathit{southeast}(P)$. No points northeast or directly north of Q are lighted (by the guards, Lemma 1, and the invariant). Similarly no points southwest or directly west of Q are lighted. Thus the step preserves the invariant.

Transformation to simpler code

Having proved Program 0, we proceed by transformation. Program 1 follows from weakening some guards and noting that case 4 is superfluous.

Case 1. The guarding term $V(\mathit{east}(P))$ means $P.y - \frac{1}{2} \le f(a,b,P.x+1) < P.y + \frac{1}{2}$. If $f(a,b,P.x) < P.y + \frac{1}{2}$, monotonicity implies $f(a,b,\mathit{east}(P).x) < \mathit{east}(P).y + \frac{1}{2}$, which is the second test in $V(\mathit{east}(P))$. If $f(a,b,P.x) \ge P.y + \frac{1}{2}$, then for P to be lighted as the invariant requires, $H(P)$ must be true. Monotonicity precludes the curve from intersecting $P.H$ and passing above *east*$(P).V$. In either event the second test in $V($*east*$(P))$ is satisfied; it can be dropped from the guard.

Case 2. The guarding term $H($*east*$(P))$ means $P.x + ½ \le f(b,a,P.y) < P.x + ½$, the second inequality of which requires that the ellipse not pass east of *east*$(P).H$. If, however, the ellipse does so, then some point directly east of P must be lighted, so by Lemma 3 the ellipse must intersect *east*$(P).V$ and case 1 is selectable. Because case 1 and case 2 have the same outcome, the second inequality may be dropped from $H($*east*$(P))$ without changing the effect of the program.

Case 3. The guarding term $H($*south*$(P))$ means $P.x - ½ \le f(b,a,y-1) < P.x + ½$. The first inequality can be dropped from the guard for reasons similar to those given for simplifying the guard in case 1.

Case 4. The ellipse intersects *south*$(P).V$. We shall show that it must also intersect *south*$(P).H$, so case 4, having the same outcome as case 3, is subsumed by case 3.

Since the ellipse intersects *south*$(P).V$, it cannot intersect $P.V$. Hence, for P to be lighted, the ellipse must intersect $P.H$. By monotonicity the intersection must lie west of P. By continuity, the ellipse must also intersect the open segment K that joins $(P.x - ½, P.y - ½)$ to $(P.x, P.y - ½)$. Between its intersections with H and K the ellipse has mean slope less than -1, hence by monotonicity of the slope, the ellipse has slope less than -1 at all points south of K. A curve of slope less than -1 and continuous over the north-south range of *south*$(P).V$, which meets *south*$(P).V$, must also meet *south*$(P).H$.

Program 1.

Precondition: $a > 0$ & $b > 0$

```
T := ∅
P := (0,b)
while P.y ≥ 0 & P.x ≤ a
        T ∪= P
        if P.x < a && P.y − ½ ≤ f(a,b,P.x + 1) → P.x += 1          {1}
        [] P.x < a & P.x + ½ ≤ f(b,a,P.y) → P.x += 1                {2}
        [] P.y > 0 && P.x + ½ > f(b,a,P.y − 1) → P.y −= 1           {3}
        [] P.x = a → P.y −= 1                                       {5}
        [] P.y = 0 → P.x += 1                                       {6}
        [] else → P.x += 1, P.y −= 1                                {7}
```

Postcondition: $T = S$

To get rid of the square root we replace formulas in *f* by equivalent formulas in *e*. If *y* can be less than zero, an inequality of the form $y \le f(a,b,x)$ is replaced by $y < 0 \mid e(a,b,x,y) \le 0$; the resulting disjunction is interpreted as a case split (between cases 1 and 1a). The identity $e(b,a,y,x) = e(a,b,x,y)$ allows arguments to be rearranged.

The conditional && operators are no longer necessary because *e*, unlike *f*, has no range restrictions.

Program 2.

Precondition: $a > 0 \ \& \ b > 0$

```
T := ∅
P := (0,b)
while P.y ≥ 0 & P.x ≤ a
        T ∪= P
        if P.x < a & e(a,b,P.x+1,P.y−½) ≤ 0 → P.x += 1          {1}
        [] P.x < a & P.y−½ < 0 → P.x += 1                        {1a}
        [] P.x < a & e(a,b,P.x+½,P.y) ≤ 0 → P.x += 1             {2}
        [] P.y > 0 & e(a,b,P.x+½,P.y−1) > 0 → P.y −= 1           {3}
        [] P.x = a → P.y −= 1                                     {5}
        [] P.y = 0 → P.x += 1                                     {6}
        else → P.x += 1, P.y −= 1                                 {7}
```

Postcondition: $T = S$

For all $x \ge a$, we have $e(a,b,x,y) > 0$ unless $(x,y) = (a,0)$. If $P = (a,0)$, the outcomes of cases 1 and 2 are the same as that of case 6. Otherwise the second tests in cases 1 and 2 will fail. Thus the test $P.x < a$ can be dropped from the guards in cases 1 and 2.

Case 1a is subsumed by case 6.

Because $e(a,b,x+½,0) < 0$ for all integer $x < a$, case 2 will be selectable when $P.y = 0$ and $P.x < a$, thus subsuming case 6 unless $P.x = a$. In the latter situation, $P = (a,0)$ and case 6 has the same outcome (termination) as case 5. Thus case 6 may be dropped.

By trying case 3 only when (the weakened) case 2 fails, we can prevent case 3 from being considered when $P.y = 0$, unless $P.x = a$. If case 3 were executable in the latter situation, it would have the same outcome (termination) as case 5. Thus the test $P.y > 0$ may be dropped from guard 3.

Since $e(a,b,a+½,y) > 0$ for all y, the weakened guard 3 will always be true when $P.x = a$, thus subsuming case 5.

Program 3.

Precondition: $a > 0 \ \& \ b > 0$

```
T := ∅
P := (0,b)
while P.y ≥ 0 & P.x ≤ a
        T ∪= P
        if e(a,b,P.x+1,P.y−½) ≤ 0 → P.x += 1                     {1}
        [] e(a,b,P.x+½,P.y) ≤ 0 → P.x += 1                       {2}
        else →
                if e(a,b,P.x+½,P.y−1) > 0 → P.y −= 1             {3}
                else → P.x += 1, P.y −= 1                         {7}
```

Postcondition: $T = S$

Meeting the full specification

The precondition can be weakened to $a \ge 0 \ \& \ b > 0$ because $e(0,b, \cdots)$ would be positive in the guards of cases 1 and 2. When $a = 0$, only cases 3 and 7 can happen; the program would generate a north-south line segment as required.

The precondition can be further weakened to $a \ge 0 \ \& \ b \ge 0$ because $e(a,0,x,0) = 0$ and at least one of cases 1 and 2 would always be selectable when $b = 0$ and $P.y = 0$. For $b = 0$, the initial value of $P.y$ is 0; hence the program would generate an east-west line segment as required.

The guards in Program 3 involve integer multiples of 1/4. Noninteger values may appear in the terms $b^2(P.x+½)^2$ and $a^2(P.y-½)^2$, which result when the definition of e is substituted into the guards. The requirement for integer arithmetic can be meet by scaling. Scaling, however, hastens the onset of overflow; judicious rounding is better.

The function $h_1(a,b,x,y) = e(a,b,x+1,y-½) - a^2/4$ is a polynomial over the integers. In terms of h_1, the guard in case 1 becomes $h_1(a,b,x,y) \le -a^2/4$, which at integer arguments is equivalent to the all-integer expression $h_1(a,b,x,y) \le -\lfloor a^2/4 \rfloor - a \bmod 2$. Replacing each guard similarly, we obtain an all-integer program.

Program 4.

Precondition: $a \ge 0$ & $b \ge 0$

```
let
        h1(a,b,x,y) = e(a,b,x+1,y−½) − a²/4
        h2(a,b,x,y) = e(a,b,x+½,y) − b²/4
        h3(a,b,x,y) = e(a,b,x+½,y−1) − b²/4
T := ∅
P := (0,b)
while P.y ≥ 0 & P.x ≤ a
        T ∪= P
        if h1(a,b,x,y) ≤ −⌊a²/4⌋ − a mod 2 → P.x += 1            {1}
        [] h2(a,b,x,y) ≤ −⌊b²/4⌋ − b mod 2 → P.x += 1            {2}
        else →
                if h3(a,b,x,y) > −⌊b²/4⌋ − b mod 2 → P.y −= 1 {3}
                else → P.x += 1, P.y −= 1                       {7}
```

Postcondition: $T = S$

Further arithmetic transformations (propagating constants, reducing the strength of multiplications, eliminating common subexpressions, moving invariant expressions out of the loop) can significantly improve efficiency. Such generic optimizations are incorporated in every published ellipse-tracing algorithm, including that in the first paper of this trilogy. However, since they are independent of the details of the ellipse problem, we shall not pursue them further here.

Discussion

Although the topic has been studied for years,[2-6] previous algorithms yield ad hoc approximations that lack a precise mathematical specification independent of the details of the algorithm. The published descriptions tend to concentrate on algebraic manipulation of the program for efficiency, while ignoring the question of just what locus the program traces.

In particular, all but one of the cited algorithms can yield different approximations for the same ellipse depending on the orientation of the major axis. The single exception[6] is an artifact of treating the major and minor axes differently.

All the cited algorithms bear a superficial resemblance to Program 4 with optimizations as suggested above. None, however, incorporates case 2. This lack leads to asymmetry under transposition. For example, without case 2 the lighted point $(1,3)$ would be missed when $(a,b) = (2,3)$, yet its transpose $(3,1)$ would be visited when $(a,b) = (3,2)$.

From the published descriptions, I infer that the logical scheme of the algorithms came first; mathematics was used only to fill in details. The basic intuition was to imitate as closely as possible the octant-plotting technique originated by Pitteway, which had proved so successful with circles. Proper location and treatment of the octant join, which had been a trivial subproblem for circles, became the central, and finally defeating, problem for ellipses. All solutions were heuristic.

The present derivation proceeds in an opposite direction, from the mathematics to the logical structure of Program 4. There is still an initial algorithmic intuition: the plan to walk from north to east. In proving Program 0 it was necessary to confirm the intuition that the walk would visit all lighted points.

From a more abstract viewpoint, the published algorithms represent attempts to generalize from highly specialized code for the circle. The circle codes assume 8-fold symmetry.[5, 7, 8] This paper instead has proceeded by generalizing Bresenham's less refined algorithm for a quadrant,[9] which only assumes four-fold symmetry, all that can be counted on in an ellipse. The latter generalization involves little more than the discovery of case 2, while generalization from octant algorithms requires at least a sound octant test, separate treatment for each octant, and a satisfactory treatment of the octant join. None of the published algorithms meets this multiple challenge fully.

In diagrammatic terms, the two approaches may be understood as proceeding by specialization and generalization in opposite orders, as shown in Figure 1. Unfortunately the two paths do not commute. The stumbling block is generalizing from octant circle algorithms, which have been specialized too far. It is much easier to generalize from the more malleable quadrant circle algorithm.

$$\begin{tikzcd} \begin{array}{c}\text{Quadrant circle}\\ \text{(Bresenham)}\end{array} \arrow[r, "\textit{generalize}"] \arrow[d, "\textit{specialize}"'] & \begin{array}{c}\text{Quadrant ellipse}\\ \text{(Program 0)}\end{array} \arrow[d, "\textit{specialize}"'] \\ \begin{array}{c}\text{Octant circle}\\ \text{(Horn)}\end{array} \arrow[r, "\textit{generalize}"] & \text{Efficient ellipse} \end{tikzcd}$$

Figure 1.

The lesson is broadly applicable. It's easier to reengineer less sophisticated code. ''Industrial strength'' programs are likely to be less amenable to major changes than are prototypes. For this reason, major steps in software evolution may necessitate stepping back in the phylogenetic tree in order to surge forward in an altered direction. Perhaps it would be well to preserve primitive versions of evolving programs as a kind of health insurance against software sclerosis.

Moral: You can't add raisins after the bread is baked.

References

[1] Freeman, H., ''Computer processing of line-drawing images,'' *Computing Surveys* **6**, p. 63 (1974).

[2] Pitteway, M. L. V., ''Algorithms for drawing ellipses or hyperbolae with a digital plotter,'' *Computer J.* **10**, pp. 282-289 (1967).

[3] Pratt, V., ''Techniques for conic splines,'' *Computer Graphics* **19**(3), pp. 151-159, Barsky, B. A. (Ed.), ACM (1985). SIGGRAPH '85 Conference Proceedings.

[4] Van Aken, J. R., ''An efficient ellipse-drawing algorithm,'' *IEEE Computer Graphics and Applications* **4**(9), pp. 24-35 (1984).

[5] Foley, J. D., Van Dam, A., Feiner, S. K., and Hughes, J. F., *Computer Graphics Principles and Practice*, Addison-Wesley (1990).

[6] Wirth, N., ''Drawing lines, circles and ellipses in a raster,'' pp. 427-434 in *Beauty is our Business*, Feijen, W. H. J., van Gasteren, A. J. M., Gries, D., and Misra, J. (Eds.), Springer-Verlag, New York (1990).

[7] Horn, B. K. P., ''Circle generators for display devices,'' *Computer Graphics and Image Processing* **5** (1976).

[8] McIlroy, M. D., ''Best approximate circles on integer grids,'' *ACM Trans. on Graphics* **2**, pp. 237-264 (Oct. 1983).

[9] Bresenham, J., ''A linear algorithm for incremental digital display of circular arcs,'' *Comm. ACM* **20**, pp. 100-106 (1977).

Ellipses Not Yet Made Easy

M. D. McIlroy

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

A paper by N. Wirth, ‘‘Drawing Lines, Circles and Ellipses in a Raster,’’[1] excerpted here by permission, illustrates methodological issues that arise in the simple problem of drawing ellipses. The paper, like others on the subject, attempts to generalize from a highly optimized algorithm for drawing circles. The result is foredoomed because the model has been specialized beyond the point of no return. More engagingly and crisply written than much practical literature in computer science, the paper affords an attractive and instructive addition to that branch of the literature which Wirth himself has acknowledged as having ‘‘taught how not to do it,’’[2] in matters of both style and substance.

Wirth’s words appear in full-size type, my annotations in small size.

Abstract. In a tutorial style, Bresenham’s algorithms for drawing straight lines and circles are developed using Dijkstra’s notation and discipline. The circle algorithm is then generalized for drawing ellipses.

The ‘‘tutorial’’ exposition is admirably suited as a case study, for it reveals just how the design of the ellipse-drawing algorithm went astray, as had many similar algorithms published previously. It does not, however, well illustrate Dijkstra’s rigor, as it later admits: ‘‘We adopt his notation but deviate from his discipline by specifying the task algorithmically rather than by a result predicate.’’ Concentrating on method without careful regard for purpose, the development fails to consider the problem and the program as a connected whole. No precise objective is stated, and single statements are analyzed in isolation, without reference to boundary conditions imposed by context. The result is a program with unknown properties, which cannot be trusted for general use.

$\boxed{\text{Beware of programs with imprecise specifications.}}$

Introduction. Recently, I needed to incorporate a raster drawing algorithm into one of my programs. The Bresenham algorithm is known to be efficient and therefore was the target of my search. Literature quickly revealed descriptions in several sources [1,3]; all I needed to do was to translate them into my favourite notation. However, I wished—in contrast to the computer—not to interpret the algorithms but to *understand* them. I had to discover that the sources picked were, albeit typical, quite inadequate for this purpose. They reflected the widespread view that programming courses are to teach the use of a (specific) programming language, whereas the algorithms are simply given.

How frequently technical papers utter the word ‘‘recent’’ in the first sentence to suggest labor at the scientific frontier! The present topic, though admittedly not at the frontier, has more than passing interest. Everybody (including me) who conscientiously studies algorithms of the Pitteway-Bresenham type seems impelled to improve the never quite complete analysis.[3-5] The analysis is delicate—more delicate than the paper recognizes.

If, as Wirth charges, graphics texts intend to help teach programming languages, then so does the present paper. More analysis is aimed at getting efficient code for languages like Pascal than at critical understanding of the problem. The avowed concern for efficiency upstages considerations of purpose.

The unusual diction of the last phrase, ‘‘whereas the algorithms are simply given,’’ inspires a complementary interpretation: a good tutorial will strive to give algorithms simply, but it will also strive to justify them adequately. Wirth succeeds on presentation, but not justification; one can’t justify the unjustifiable. A more thorough attempt might have uncovered the impasse.

Dijkstra was an early and outspoken critic of this view, and he correctly pointed out that the difficulties of programming are primarily inherent in the subject, namely in constructive reasoning. In order to emphasize this central theme, he compressed the notational issue to a bare minimum by postulating his own notation that is concisely defined within a few formulas [2].

> The capsule description misses the genius of Dijkstra's notation: its suppression of spurious detail about sequencing. Had mere ''compression of the notational issue'' been the central purpose, the notation would not stand out among others.
>
> The present treatment illustrates Dijkstra's notation little more than it does his discipline. Guarded commands appear only as trivial equivalents for everyday `while` and `if-then-else` constructs. The deployment of synonyms is window-dressing, not a methodological advance. The notation that is mainly—and productively—used in the paper is elementary algebra; no computer scientist should be without it.

[Further introduction, a section on lines, and a section on circles are omitted.]

Ellipses. Similarly to the circle algorithm, we wish to design an algorithm for plotting ellipses by proceeding in steps to find raster points to be marked.

> Grammatically the adverb ''similarly'' has to modify the main verb, but that gives, ''We wish similarly.'' However algorithms wish, it is probably not as we do. Perhaps it is as computers do; see Wirth's introductory paragraph. The slapdash English, even though well above threshold for most computing journals, symptomizes a less than careful approach to the whole work. In writing inexactly one hides inexact reasoning, even from oneself.

$\boxed{\text{What's worth telling is worth telling well.}}$

We concentrate on the first quadrant; the other three quadrants can be covered by symmetry arguments and require no additional computation.

> ''No additional computation'' really means ''no more code to be displayed in this paper.''

Let the ellipse be defined by the following equation. Again without loss of generality, we assume $0 < a \le b$.

$$E\colon\ (x/a)^2 + (y/b)^2 = 1$$

> The customary meaning of ''without loss of generality'' is that in some obvious way the general problem can be mapped into a special case. Here, however, generality has certainly been lost. The possibility of $a = 0$, an ellipse of zero width, and a perfectly reasonable limiting case, should be restored in any real implementation. Imagine a time-lapse animation of Saturn dying at the moment the rings appear edge on.

$\boxed{\text{Handle limiting cases.}}$

> More seriously, the highly technical restriction $a \le b$ is ''simply given''—never explained and never appealed to in the development. Yet the program can fail without it.

We start with the point $P(0,b)$ and proceed by incrementing x in each step, and decrementing y if necessary.

> Here, as throughout the paper, the reader is left to infer that a and b are integers. The assumption is central to the correctness of the algorithm.
>
> The extra identifier P, like E in the previous equation, serves no purpose whatever.

The exact ordinate of the next point follows from the defining equation:

$$Y = b\sqrt{(1 - ((x+1)/a)^2)}$$

> The notation here depends on the omitted part of the paper. Y is an ordinate on the true ellipse; y is a raster approximation. ''Next point'' was defined informally by usage to mean the point on the true ellipse at the next integer abscissa.

The raster point coordinate must satisfy

$$y - 1/2 < b\sqrt{(1 - ((x+1)/a)^2)}$$

$$y^2 - y + 1/4 < b^2 - b^2(x+1)^2/a^2$$

$$a^2y^2 - a^2y + a^2/4 < a^2b^2 - b^2x^2 - 2b^2x - b^2$$

$$b^2x^2 + 2b^2x + a^2y^2 - a^2y + a^2/4 - a^2b^2 + b^2 < 0$$

The second line of the derivation is unjustified if $y < 1/2$ or if $x + 1 > a$. The former event can happen and cause trouble, as we shall see later. The latter cannot, but that fact is not foreseeable at this stage of the derivation.

Attend to boundary conditions.

The necessary and sufficient condition for decrementing *y* is therefore $h \geq 0$ with the auxiliary variable *h* being defined as

$$h = b^2x^2 + 2b^2x + a^2y^2 - a^2y + a^2/4 - a^2b^2 + b^2$$

Although it appears suddenly and unexplainedly here, the discussion about decrementing *y* parallels that in the omitted discussion of circles.

As in the case of the circle, the termination condition is met as soon as *y* might have to be decreased by more than 1 after an increase of *x* by 1, i.e. when the tangent to the curve is greater than 45°. Unlike in the case of the circle, however, this condition is not obviously given by $x = \mathrm{y}$. We reject the obvious solution of computing the ordinate for which the curve's derivative is -1, [sic] because this computation alone would involve at least the square root function.

The English of the paragraph, and especially of the sentence beginning, ''Unlike in the case of,'' won't stand up to scrutiny.

The notion of decreasing *y* after increasing *x* is excessively sequential. To simplify the maintenance of the loop invariant, and to avoid needlessly overspecifying the code, one would prefer to say that at each step either *x* alone is modified, or *x* and *y* are modified simultaneously. The presentation here, which decides which to do first, bears on Pascal more than on the problem.

The last sentence betrays a lack of analysis. The ordinate in question is $y = b^2(a^2 + b^2)^{-1/2}$. The square root can be removed by squaring to get a polynomial discriminator function, as Wirth has just done to obtain *h*. The resulting fourth powers, however, threaten to overflow small registers. Unless unusually wide arithmetic is at hand, it is well to seek a discriminator of lower degree, which the paper proceeds to do in a novel way.

Instead we compute a function *g*, similar to *h*, incrementally. Its origin stems [sic] from the inequality

$$y - 3/2 < b\sqrt{(1 - ((x + 1)/a)^2)}$$

implying that the ordinate of the next point be at least 3/2 units below the current raster point. Therefore, a decrease of *y* by 2 would be necessary for an increase of *x* by 1 only. A similar development as for *h* yields the function *g* as

$$g = b^2x^2 + 2b^2x + a^2y^2 - 3a^2y + 9a^2/4 - a^2b^2 + b^2$$

and *x* can be incremented as long as $g < 0$.

Beware, the explanation is backward. Violation, not satisfaction, of the inequality would imply the undesired outcome. Furthermore, if the ellipse is sufficiently narrow, *y* can decrease by any integer amount, not just 1 or 2. More significantly, what if *y* should never decrease by more than 1? This happens when $a = b = 1$. In this case the *g* test turns out to work by luck of a compensating error: the derivation of *g* is flawed by the same inattention to range restrictions as was the derivation of *h*.

The first quadrant of the ellipse is then completed by the same process, starting at the point $P(a, 0)$, of [sic] incrementing *y* and conditionally decrementing *x*. The auxiliary function here is obtained from the previous case of *h* by systematically substituting x, y, a, b for y, x, b, a.

The wording is imprecise. If one understands ''the same process'' to test for termination the same way, then it will not necessarily work for drawing the long branch of a skinny ellipse. Here the asymmetry imposed by the unexplained precondition $a \leq b$ comes into play.

The derivation of the incrementing values for *h* and *g* follow [sic] from the application of the axiom of assignment: on incrementing *x* the incrementation of *h* is obtained from

$\{h = b^2x^2 + 2b^2x + k\}$
$h := h + b^2(2x + 3)$
$\{h = b^2x^2 + 2b^2x + b^2 + 2b^2x + 2b^2 + k\}$
$x := x + 1$
$\{h = b^2x^2 + 2b^2x + k\}$

on incrementing *y*, the incrementation of *h* is obtained from

$\{h = a^2y^2 - a^2y + k\}$
$h := h - 2a^2(y - 1)$
$\{h = a^2y^2 - 2a^2y + a^2 - (a^2y - a^2) + k\}$
$y := y - 1$
$\{h = a^2y^2 + a^2y + k\}$

and the incrementation of *g* is obtained from

$\{g = a^2y^2 - 3a^2y + k\}$
$g := g - 2a^2(y - 2)$
$\{g = a^2y^2 - 2a^2y + a^2 - 3(a^2y - a^2) + k\}$
$y := y - 1$
$\{g = a^2y^2 - 3a^2y + k\}$

In each stretch of the preceding derivation, *k* represents nonchanging terms, as was explained in the omitted part of the paper. All this formalism, however, is misplaced methodology. It simply says that the update step is

$$x, y, h, g := x + 1, y + \Delta y, h + \Delta h, g + \Delta g$$

where $\Delta h = h(x+1, y+\Delta y) - h(x,y)$ and Δg is defined similarly. Most of the development is concerned with inconsistent intermediate states. Their necessity in Pascal is no reason to inflict them on an exposition of an algorithmic idea.

Use formalism for function, not fashion.

This completes the design considerations for the following algorithm.

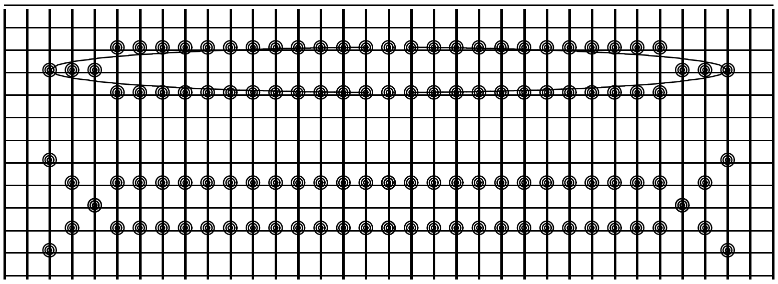
$x := 0$; $y := 0$;
$h := (a^2\ DIV\ 4) - ba^2 + b^2$; $g := (9/4)a^2 - 3ba^2 + b^2$;
do $g < 0 \rightarrow Mark(x,y)$;
  if $h < 0 \rightarrow d := (2x+3)b^2$; $g := g + d$
  $[\,]\ h \geq 0 \rightarrow d := (2x + 3)b^2 - 2(y - 1)a^2$;
    $g := g + d + 2a^2$;
    $y := y - 1$
  fi;
  $h := h + d$; $x := x + 1$
od;
$x := a$; $y1 := y$; $y := 0$;
$h := (b^2\ DIV\ 4) - ab^2 + 2a^2$;
do $y \leq y1 \rightarrow Mark(x,y)$;
  if $h < 0 \rightarrow h := h + (2y + 3)a^2$
  $[\,]\ h \geq 0 \rightarrow h := h + (2y + 3)a^2 - 2(x - 1)b^2$; $x := x - 1$
  fi;
  $y := y + 1$
od

The reader is left to puzzle out the inconsistent division operators in the second line. The *h* in the program is not the same as the *h* in the development. It is rounded down to the nearest integer. As the omitted part of the paper explained, rounding does not change the outcome of any test in the algorithm. Similarly, *g* may be rounded down and the first term of its initializer may be replaced by $(9a^2)\ DIV\ 4$.

The second initialization of h should be the same as the first with a and b interchanged.

The second loop is fatally flawed, because the unstated side condition for the validity of the h test can be violated. That condition, $x \geq 1/2$, is violated whenever an ellipse is so narrow as to be rendered with tails one pixel wide at either end. See the accompanying figure for the result. Apparently the program was never tested against such obviously stressful cases.

A subtler trouble is that the endpoint of the second loop does not necessarily coincide with the last point calculated (but not plotted) in the first loop. For example, with $a = 2$ and $b = 3$, the first loop ends at $(1,3)$, while the second loop ends at $(0,3)$. I infer that it was simply assumed that the two endpoints would coincide. If the possibility of mismatch had been recognized, there should have been some analysis of how bad it can be.



Proper tails and fishy tails, $a = 1$, $b = 15$. Figure rotated 90° to save space.

We close this essay with the remark that values of h may become quite large and that therefore overflow may occur when the algorithm is interpreted by computers with insufficient word size. Unfortunately, most computer systems do not indicate integer overflow! Using 32-bit arithmetic, ellipses with values of a and b up to 1000 can be drawn without failure.

The exclamation directs attention away from software to hardware. All computer hardware that I can think of indicates integer overflow, although not by trapping. Compiled code for languages such as Pascal almost universally ignores the indication, however.

The claim of a range up to 1000 is too rosy. Where the slope of the ellipse is near zero, the discriminator g may be evaluated at points up to 2 units away from $g = 0$. (The algorithm visits points as much as 1/2 unit off the ellipse, and $g = 0$ is displaced 3/2 units from the ellipse.) At such a point with $a = b$, $x \sim 0$ and $y \sim a$, the magnitude of g, estimated as $|(\partial g/\partial y)\Delta y|$, is approximately $4a^3$. Thus overflow is liable to occur at parameter values around $(2^{31}/4)^{1/3}$, not much more than 800. Testing confirms this estimate.

Big-oh estimates are not quantitative.

There is more to say. It is bad practice to draw points twice. In particular, double plotting is self-nullifying when drawing by exclusive or into a bitmap. Double plotting at the beginnings of the arcs points can be averted by proper coding of the *Mark* procedure. However, double plotting can also occur where the two branches meet. For example, in the poorly closing example mentioned above ($a = 2$ and $b = 3$), $Mark(0,3)$ will be called in the second loop as well as in the first.

Other important properties of the algorithm are left to be taken on faith. Will the two branches always meet without a gap? If not, color would leak out on attempting to shade the inside of an ellipse. (This is not an idle question. The omitted algorithm for circles *can* produce gaps.) Will circles drawn by the algorithm be symmetric about the diagonal, $y = x$? The answer is not immediately obvious, because in all but the smallest circles, the juncture of the two branches lies off the diagonal.

Formulate and confirm proper behavior.

The ellipse-drawing algorithm works like two ships setting out from fixed points on the shores of the first quadrant to rendezvous near the octant juncture. The most difficult sailing will be experienced in leaving the harbors, where the sharpest and most confined turns must be navigated, and at the meeting point, where precision docking is required. Just as at sea, where the steering of a ship may be trusted to an apprentice seaman in open water, but needs an experienced pilot for close navigation, so ellipse-drawing can be entrusted to simple homework-assignment code only in the open and needs more attention in the critical stretches. The present algorithm has not earned a pilot's license.

REFERENCES [for Wirth]

[1] N. Cossitt. *Line Drawing with the NS32CG16 and Drawing Circles with the NS32CG16.* Technical Report AN-522 and AN-523, National Semiconductor Corp., 1988

[2] E. W. Dijkstra. Guarded commands, non-determinacy, and the formal derivation of programs. *Comm. ACM*, 18(8):453-457, August 1975.

[3] J. D. Foley and A Van Dam. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, 1982.

References [for McIlroy]

[1] Wirth, N., ''Drawing lines, circles and ellipses in a raster,'' pp. 427-434 in *Beauty is our Business*, Feijen, W. H. J., van Gasteren, A. J. M., Gries, D., and Misra, J. (Eds.), Springer-Verlag, New York (1990).

[2] Wirth, N., ''From Modula to Oberon,'' *Software—Practice and Experience* **18**, pp. 661-670 (1988). Acknowledgements.

[3] Pitteway, M. L. V., ''Algorithms for drawing ellipses or hyperbolae with a digital plotter,'' *Computer J.* **10**, pp. 282-289 (1967).

[4] Bresenham, J., ''A linear algorithm for incremental digital display of circular arcs,'' *Comm. ACM* **20**, pp. 100-106 (1977).

[5] McIlroy, M. D., ''Best approximate circles on integer grids,'' *ACM Trans. on Graphics* **2**, pp. 237-264 (Oct. 1983).