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.

$\boxed{\text{Attend to boundary conditions.}}$

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 $x = y$. 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

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

on incrementing y, the incrementation of h is obtained from

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

and the incrementation of g is obtained from

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

> 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.

$\boxed{\text{Use formalism for function, not fashion.}}$

This completes the design considerations for the following algorithm.

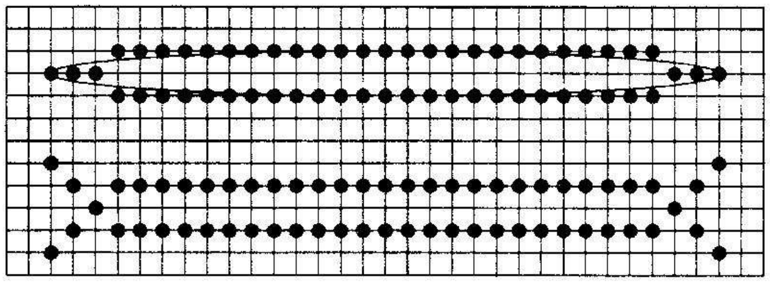
```
x := 0; y := 0;
h := (a² DIV 4) − ba² + b²; g := (9/4)a² − 3ba² + b²;
do g < 0 → Mark(x,y);
      if h < 0 → d := (2x +3)b²; g := g + d
      [] h ≥ 0 → d := (2x + 3)b² − 2(y − 1)a²;
                 g := g + d + 2a²;
                 y := y − 1
      fi;
      h := h + d; x := x + 1
od;
x := a; y1 := y; y := 0;
h := (b² DIV 4) − ab² + 2a²;
do y ≤ y1 → Mark(x,y);
      if h < 0 → h := h + (2y + 3)a²
      [] h ≥ 0 → h := h + (2y + 3)a² − 2(x − 1)b²; 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.

$\boxed{\text{Big-oh estimates are not quantitative.}}$

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.

$\boxed{\text{Formulate and confirm proper behavior.}}$

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).