An Assertion-Guided Derivation of a Circle Drawing Algorithm

Derrick G. Kourie and Bruce W. Watson
Department of Computer Science, University of Pretoria
Pretoria 0002, South Africa
dkourie@cs.up.ac.za and bwatson@cs.up.ac.za

Abstract

A raster-based algorithm for drawing circles is derived by using the so-called assertion-guided approach to program construction. The initially derived form is slightly different from published ones. It can, however, be transformed into the *second-order midpoint circle-drawing algorithm*. Because of the rigour of the approach, the correctness of the derived algorithm is assured. The approach also naturally brings boundary issues to the fore that might not otherwise be obvious and that appear to have been overlooked in published circle-drawing algorithms.

1 Assertion-guided program development

Assertion-guided program development is the term given by Meyer [1] to a well-known style of designing algorithms. Perhaps the best-known text on the subject is that of Gries [2] (see also [3] — the notation used here will follow that text).

The method centers on the design of loops. It is based on weakest preconditions and loop invariants. In overview, the following steps are recommended when designing a loop.

1. Determine the goal (or post-condition), G, of the loop.

2. Determine a loop invariant, L, and a loop condition, C, such that

   $$L \wedge \neg C \Rightarrow G$$

3. Initialize the loop's variables so that L holds before the loop.

4. Determine the body of the loop which preserves L and which simultaneously drives the loop's variables, during each successive iteration, towards a state such that $\neg C$ holds.

5. Where the body consists of a conditional guarded command, as will be the case below, consider each guarded statement of the form $B \rightarrow S$ separately. In each case, construct S in such a way that L holds after S has been executed. Using the weakest precondition calculus, find $wp(S, L)$. Choose B such that $C \wedge B \wedge L \Rightarrow wp(S, L)$.

Use of this assertion-guided approach to develop raster graphics algorithms is not unique. For example, Gutknecht [4] shows how the approach leads to the well-known Bresenham algorithm for drawing a straight line on a rastered plane. Following reasoning very similar to his, a raster graphics algorithm for drawing a circle is developed below. The algorithm turns out to be slightly different from the one developed by Bresenham [5] and also from modifications of Bresenham's circle algorithm given in standard texts such as Foley et. al. [6]. The resulting algorithm is as efficient as its rivals, and can be transformed into them. The development process focuses attention on subtle boundary conditions that might be violated when using brute force development methods.

2 Problem statement and outline of solution

The problem to be addressed is the following:

> Draw $1/8^{th}$ of a circle which has a positive integer, r, as radius and which is centered at $(0, 0)$. It is to be drawn on a rastered plain, implying that all co-ordinates (x, y) are integer-valued. The first point is to be drawn at $(0, r)$ and the algorithm should end just before $x > y$. This guarantees that no more than $1/8^{th}$ of the circle is drawn.

Assume that $draw(x, y)$ colours the pixel (x, y). If a complete circle was required, $draw(x, y)$ could be designed to colour 7 additional points on the circle's circumference. These points are easily determined from the point (x, y) using symmetry arguments. However, special consideration should be given to ensure that certain symmetry points such as (x, x) and $(0, y)$ are not coloured twice.

The algorithm thus involves a loop which has the following informally-stated post-condition G:

> For each x-coordinate, x_i, in the range $x_0, \ldots, x_{n+1}$, the appropriate y-coordinate, y_i, has been found such that for $i = 0 \ldots n$, $x_i \leq y_i$ and $x_{n+1} > y_{n+1}$. Each of the points in the range $x_0, \ldots, x_n$ have been drawn.

Noting that the x-coordinates will be integers from 0 to some positive integer, the goal suggests the following rough loop invariant (which will later be refined) $L(x)$:

> For each x-coordinate, i, in the range $0, \ldots, x$, the appropriate y-coordinate, y_i, has been found, where $i \leq y_i$ for $i = 0 \ldots x - 1$ and where the points (i, y_i) for $i = 0 \ldots x - 1$ have been drawn.

The condition of the loop, C, must now be such that $\neg C \vee L(x) \Rightarrow G$. Clearly, this will hold if $\neg C \equiv (x > y)$, leading to a loop condition $C \equiv (x \leq y)$.

The general flow of control for the algorithm will be to: colour a pixel; increment x; determine y appropriately; and then repeat the cycle while the loop's condition holds. Furthermore, in each iteration y, will either retain its previous value, or it will be decrement by 1, depending on some yet to be determined conditions. Calling these conditions B_1 and B_2 respectively, the algorithm has the form given in outline below.

Algorithm 2.1:

Initialize variables so that $L(x)$ holds
{ invariant: $L(x)$ }
do $x \leq y \rightarrow$
  $draw(x, y)$;
  if $B_1 \rightarrow$
    $x := x + 1$;
    leave y
  ‖ $B_2 \rightarrow$
    $x := x + 1$;
    decrement y
  fi
od
{ G }

□

3 Refining the loop invariant

Reference to "the appropriate y co-ordinate" in $L(x)$ above is not specific enough to be useful. In order to more precisely specify what an appropriate y co-ordinate is, consider the real-valued y co-ordinate (call it y_r) such that, for a given integer value of x, the point (x, y_r) lies precisely on the circumference of the circle. Thus $y_r^2 = r^2 - x^2$.

The approximated (integer) value of y_r, denoted by y, is the *rounding* of y_r, meaning that when y_r is exactly midway, or more than midway between two integer values, y is its rounded up value; otherwise y is the rounded down value. Formally,

$$y - 1/2 \leq y_r < y + 1/2$$

We can derive a more usable form of this statement (in such a *derivation*, each line is separated by a 'hint' indicating why the subsequent line follows from the precedent one)

$$\begin{aligned} & y - 1/2 \leq y_r < y + 1/2 \\ \equiv & \quad \text{“} y_r^2 = r^2 - x^2 \text{”} \\ & y - 1/2 \leq \sqrt{r^2 - x^2} < y + 1/2 \end{aligned}$$

$$
\begin{array}{ll}
\equiv & \text{“squaring”} \\
& y^2 - y + 1/4 \leq r^2 - x^2 < y^2 + y + 1/4 \\
\equiv & \text{“subtracting } y^2 + 1/4\text{”} \\
& -y \leq r^2 - x^2 - y^2 - 1/4 < y \\
\equiv & \text{“mult. by 4; eliminate fraction”} \\
& -4y \leq 4(r^2 - x^2 - y^2) - 1 < 4y
\end{array}
$$

This can now be used to refine the loop invariant $L(x)$ as follows:

> For each x-coordinate, i, in the range $0, \ldots, x$, a y-coordinate, y_i, has been found that complies with $H(i, y_i, d)$; where $i \leq y_i$ for $i = 0 \ldots x - 1$; where the points (i, y_i) for $i = 0 \ldots x - 1$ have been drawn; and where $H(x, y, d) \equiv (-4y \leq d < 4y) \wedge (d = 4(r^2 - x^2 - y^2) - 1)$

The refined invariant thus introduces a new integer variable, d, into the algorithm, which has to be updated in each iteration to ensure that the invariant continues to hold at the end of the loop.

Since the prime concern will be with the predicate $H(x, y, d)$ in the loop invariant, we shall henceforth refer to this predicate, and regard the informally-stated other parts of $L(x)$ as implicit.

4 Initialization

The obvious way in which to ensure initial compliance with the $L(x)$ is to prepare to draw a pixel at $x = 0$ and $y = r$, and to initialize the variable d so that $H(0, r, d)$ holds. Since $H(0, r, d) \equiv (-r \leq d < r) \wedge (d = 4(r^2 - 0^2 - r^2) - 1)$, d should be initialized to -1. Note, too, that since by assumption r is positive, the first conjunct of $H(0, r, d)$ holds.

The algorithm thus has the following form:

Algorithm 4.1:

$x, y, d := 0, r, -1;$
{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y);$
  if $B_1 \rightarrow$
    $x := x + 1;$
    determine d;
    leave y
  ▯ $B_2 \rightarrow$
    $x := x + 1;$
    determine d;
    decrement y
  fi
od
{ G }

□

5 Determining first guarded command

Let d' be the new value of d after executing the first guarded command. Since $H(x + 1, y, d')$ must hold after this command, it follows that:

$$
\begin{array}{ll}
& d' \\
= & \text{“definition of } H\text{; } x' \text{ is } x + 1\text{”} \\
& 4(r^2 - (x + 1)^2 - y^2) - 1 \\
= & \text{“arithmetic”} \\
& 4(r^2 - (x^2 + 2x + 1) - y^2) - 1 \\
= & \text{“arithmetic”} \\
& 4(r^2 - x^2 - (2x + 1) - y^2) - 1 \\
= & \text{“arithmetic”} \\
& 4(r^2 - x^2 - y^2) - 1 - 4(2x + 1) \\
= & \text{“}H(x, y, d) \Rightarrow d = 4(r^2 - x^2 - y^2) - 1\text{”} \\
& d - 4(2x + 1)
\end{array}
$$

The command thus has the form:

$$B_1 \rightarrow x, d := x + 1, d - 4(2x + 1)$$

Now note that after the first guard, $H(x, y, d)$ must hold. We need to determine the weakest precondition

$$wp(\text{“}x, d := x + 1, d - 4(2x + 1)\text{”}, H(x, y, d))$$

and then find B_1 such that this weakest precondition is implied by $H(x, y, d) \wedge B_1 \wedge (x \leq y)$. Applying the well-known rules (see [2]) of the weakest precondition calculus we obtain:

$$
\begin{array}{ll}
& wp(\text{“}x, d := x + 1, d - 4(2x + 1)\text{”}, H(x, y, d)) \\
\equiv & \text{“substitution according to } wp \text{ rules”} \\
& H((x + 1), y, (d - 4(2x + 1))) \\
\equiv & \text{“definition of } H\text{”} \\
& (-4y \leq (d - 4(2x + 1)) < 4y) \wedge
\end{array}
$$

$$
\begin{array}{ll}
& (d - 4(2x+1) = 4(r^2 - (x+1)^2 - y^2) - 1)) \\
\equiv & \quad \text{“simplification”} \\
& (-4y \leq (d - 4(2x+1)) < 4y) \wedge \\
& (d = 4(r^2 - x^2 - y^2) - 1))
\end{array}
$$

Clearly, the second conjunct of $H(x, y, d)$ implies the second conjunct of this weakest precondition. (They are, if fact, identical.) Also, $H(x, y, d) \Rightarrow (d - 4(2x + 1)) < 4y)$ (since $x \geq 0$ and $d < 4y$ in $H(x, y, d)$).

Consequently, the following choice for B_1 is appropriate:

$$-4y \leq d - 4(2x + 1)$$

equivalently

$$d \geq 4(2x - y + 1)$$

6 Determining the second guarded command

Proceeding as before, let d' be the new value of d after executing the second guarded command. Since $H(x+1, y-1, d')$ should hold after this command, it follows that:

$$
\begin{array}{ll}
& d' \\
= & \quad \text{“definition of } H\text{”} \\
& 4(r^2 - (x+1)^2 - (y-1)^2) - 1 \\
= & \quad \text{“arithmetic with } (x+1)^2\text{”} \\
& 4(r^2 - x^2 - (2x+1) - (y-1)^2) - 1 \\
= & \quad \text{“arithmetic with } (y-1)^2\text{”} \\
& 4(r^2 - x^2 - (2x+1) - y^2 + (2y-1)) - 1 \\
= & \quad \text{“arithmetic”} \\
& 4(r^2 - x^2 - (2x+1) - y^2 + (2y-1)) \\
& -1 - 4(2x + 1 - 2y + 1) \\
= & \quad \text{“}H(x, y, d) \Rightarrow d = 4(r^2 - x^2 - y^2) - 1\text{”} \\
& d - 4(2x + 1 - 2y + 1) \\
= & \quad \text{“simplification”} \\
& d - 8(x - y + 1)
\end{array}
$$

The guarded command thus has the form:

$$B_2 \rightarrow x, y, d := x + 1, y - 1, d - 8(x - y + 1)$$

(Note that $H(x, y, d)$ must hold after the second guarded command.) We need to determine the weakest precondition to this assignment, $x, y, d := \ldots$, and then find B_2 such that this weakest precondition follows from $H(x, y, d) \wedge B_2 \wedge (x \leq y)$.

But

$$
\begin{array}{ll}
& wp(\text{“}x, y, d := x + 1, y - 1, d - 8(x - y + 1)\text{”}, \\
& H(x, y, d)) \\
\equiv & \quad \text{“substitution according to } wp \text{ rules”} \\
& H((x+1), (y+1), (d - 8(x - y + 1))) \\
\equiv & \quad \text{“definition of } H\text{”} \\
& (-4(y-1) \leq (d - 8(x - y + 1)) < 4(y-1)) \\
& \wedge (d - 8(x - y + 1) = \\
& 4(r^2 - (x+1)^2 - (y-1)^2) - 1)) \\
\equiv & \quad \text{“simplification”} \\
& (-4(y-1) \leq (d - 8(x - y + 1)) < 4(y-1)) \\
& \wedge (d = 4(r^2 - x^2 - y^2) - 1))
\end{array}
$$

Again, it trivially follows that $H(x, y, d)$ implies the second conjunct of the weakest precondition.

Following a similar line of reasoning to the derivation of the first guard, symmetry would suggest that

$$H(x, y, d) \wedge (x \leq y) \Rightarrow (-4(y-1) \leq (d-8(x-y+1)) \tag{1}$$

If this were the case, then in order to ensure that $H(x, y, d) \wedge B_2 \wedge (x \leq y) \Rightarrow$ the weakest precondition, B_2 could be chosen as $(d - 8(x - y + 1)) < 4(y - 1))$. We can further simplify this

$$
\begin{array}{ll}
& (d - 8(x - y + 1)) < 4(y - 1)) \\
\equiv & \quad \text{“simplification”} \\
& d < 4y - 4 + 8x - 8y + 8 \\
\equiv & \quad \text{“simplification”} \\
& d < 8x - 4y + 4 \\
\equiv & \quad \text{“simplification”} \\
& d < 4(2x - y + 1)
\end{array}
$$

Although this neatly yields the second guard to be the complement of the first, (1) cannot be proved to hold. Nevertheless, provisionally assume that it does holds. The implications of this assumption will be explored more fully below.

From all of the foregoing, and subject to the assumption that (1) holds, the following circle-drawing algorithm has been derived:

Algorithm 6.1:

$x, y, d := 0, r, -1;$
{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y);$
  if $d \geq 4(2x - y + 1) \rightarrow$
    $x, d := x + 1, d - 4(2x + 1)$
  [] $d < 4(2x - y + 1) \rightarrow$
    $x, y, d := x + 1, y - 1, d - 8(x - y + 1)$
  fi
od
{ G }

□

7 Exploration of assumption

Before considering ways in which the algorithm might be transformed and optimized to bring it closer to classical algorithms, it is instructive to examine the assumption (1) in greater detail. Consider the consequent of (1) more closely: $(-4(y - 1) \leq (d - 8(x - y + 1)) \equiv -4y \leq d + 8(y - x) - 12$.

But $(-4y \leq d) \wedge 8(y - x) - 12 \geq 0 \Rightarrow -4y \leq d + 8(y - x) - 12$. Therefore $H(x, y, d) \wedge (y - x) \geq 1.5 \Rightarrow (-4(y - 1) \leq (d - 8(x - y + 1))$.

Thus, assumption (1) only holds if it is guaranteed that, from an integer arithmetic point of view, x is at least two units smaller than y, i.e. $x = y - 2$. But the loop condition provides for the possibilities $x = y - 1$ or $x = y$ as well, and there is no certainty that the first guard will be chosen when this occurs. Note that if the second guard was to be chosen at this juncture, x would be incremented, y would be decremented, and a point (x, y) would generated, where $x = y + 1$ if $x = y - 1$ held before the command, or where $x = y + 2$ if $x = y$ held before the command. The loop's condition now no longer holds, so no additional point would have be drawn. This complies with the original brief of drawing only $1/8^{th}$ of a circle segment.

Notwithstanding the fact that in both the above boundary conditions the loop would correctly terminate without drawing unrequired points, the theory implies that $H(x, y, d)$ will no longer hold after the second guarded command has been executed. It seems therefore that the loop could terminate in a state in which the loop invariant no longer holds, suggesting that something has gone wrong! The solution to the dilemma lies in seeing that, in retrospect, the invariant was badly phrased to be more strict than is necessary: $H(x, y, d)$ need only hold for points that are actually to be drawn. A more accurate statement of the invariant would thus explicitly require that: $x \leq y \Rightarrow H(x, y, d)$. It can easily be verified that applying the theory to a weakest precondition based on this post-condition would lead to the same guard and to the assurance of the revised invariant holding at the end of the loop.

Of course, this raises the question as to why no similar problem was experienced in developing the first guard. The answer lies in the fact that, even in the boundary case where the guarded command was executed with $x = y$, the subsequent values for x and d are — coincidentally — such that they continue to satisfy $H(x, y, d)$, even though $x > y$ after the command has been executed.

These boundary issues are sometimes ignored in published algorithms. For example, in Foley et. al. [6] the loops have as condition $y > x$ and are designed to first compute the next point, and then to draw that point. Where an iteration in such a loop starts with $x = y - 1$ it will turn out that by decrementing y and incrementing x a point is incorrectly drawn where $x = y + 1$.

8 Transformations to a simpler algorithm

Although the assertion-based approach has derived an algorithm that is fully correct, it is clearly less efficient than others in the literature: it involves relatively complicated guards to be tested, as well as several more operations per guard body than required by its rivals. However, it is intriguing to note in passing that the algorithm can be implemented using shift operations instead of multiplication. A series of elementary transformations will now be presented that results in an algorithm that is as simple as its rivals. A further set of transformations shows that the simplified algorithm is, in fact, a variation of the midpoint circle algorithm derived by using second order differences.

The first transformations are directed at removing multiplication operations (even though it is

recognised that all of them can be implemented as shift operations). This is done at the cost of adding a few more assignments and variables.

As a first step, let t be a new variable which is invariantly related to x and y by the equality

$$t = 4(2x - y + 1)$$

Note that this term appears in both guards. Thus the term in the body of the first guard, $4(2x + 1)$, is $t + 4y$ and the term in the body of the second, $8(x - y + 1)$ is $t + 4(y + 1)$. Furthermore, the initial value of t is $4(1 - r)$. Finally, to ensure that its invariance relation to x and y is retained after each guard's body has been executed, the new value of t, say t', is determined as follows in respect of the first guard:

$$\begin{aligned} & t' \\ = & \quad \text{“Substitution for } x \text{ in } t\text{”} \\ & 4(2(x+1) - y + 1) \\ = & \quad \text{“Arithmetic”} \\ & t + 8 \end{aligned}$$

Similarly, in respect of the second guard:

$$\begin{aligned} & t' \\ = & \quad \text{“Substitution for } x \text{ and } y \text{ in } t\text{”} \\ & 4(2(x+1) - (y-1) + 1) \\ = & \quad \text{“Arithmetic”} \\ & t + 12 \end{aligned}$$

Substitution then leads to a transformed algorithm:

Algorithm 8.1:

$x, y, d, t := 0, r, -1, 4(1 - r);$
{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y);$
  if $d \geq t \rightarrow$
    $x, d, t := x + 1, d - t - 4y, t + 8$
  [] $d < t \rightarrow$
    $x, y, d, t :=$
    $x + 1, y - 1, d - t + 4(y - 1), t + 12$
  fi
od
{ G }

□

This does not quite achieve the goal of eliminating all multiplications, but it is considerably simpler than before. Note that, while on the one hand, all variables are integers, the importance of t and d does not lie in their absolute values, but rather in their values relative to one another: the difference between these two variables controls the flow of logic by indicating which guard is to be selected at each iteration. Suppose, then, that t and d are provisionally replaced by new real-valued variables, where these new variables invariantly have one quarter of the value of the former integer variables in each iteration. The algorithm then changes as follows (retaining the identifier names t and d for the new real variables):

Algorithm 8.2:

$x, y, d, t := 0, r, -0.25, (1 - r);$
{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y);$
  if $d \geq t \rightarrow$
    $x, d, t := x + 1, d - t - y, t + 2$
  [] $d < t \rightarrow$
    $x, y, d, t :=$
    $x + 1, y - 1, d - t + (y - 1), t + 3$
  fi
od
{ G }

□

The advantage of this form is that multiplication operations have been eliminated. This is done at the expense of introducing real-valued variables. However, inspection shows that t starts off as an integer (assuming, of course, an integer radius) and continues to be integer-value after each iteration. It is therefore natural to enquire whether the role played by d can be filled by some integer variable.

Note that d starts off as $(-1 + 0.75)$ and changes from this value by an integer amount in each iteration. Thus at any stage of the algorithm, d always has a value $D + 0.75$ where D is some integer. Consider, then, the effect of replacing the real-valued variable d by a rounded down integer-valued variable d in the algorithm, where d is initialized to -1.

- Whenever the first guard is selected in the real-valued algorithm the corresponding guard is

selected in the integer-valued algorithm. This follows by noting that first guard is selected in the real-valued algorithm when $d = t+k+0.75$ for some non-negative integer, k. At the corresponding point in the integer-valued algorithm $d = t + k$, so that in the most marginal case, where $k = 0$, the guard is selected as an equality.

- By a similar argument, the second guard is selected in the real-valued algorithm when $d = t-k+0.75$ for some positive integer, k. At the corresponding point in the integer-valued algorithm $d = t - k$, so that in the most marginal case, where $k = 1$, the integer d is strictly less than t, and the second guard will also be selected.

This yields the following integer-valued algorithm:

Algorithm 8.3:

$x, y, d, t := 0, r, -1, (1 - r);$
{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y);$
  if $d \geq t \rightarrow$
    $d, t := d - t - y, t + 2$
  ▯ $d < t \rightarrow$
    $y, d, t := y - 1, d - t + y - 1, t + 3$
  fi;
  $x := x + 1$
od
{ G }

□

9 Comparative results

The algorithm now no longer contains multiplications, and involves four addition or subtraction operations per iteration whenever the first guard is selected, and six such operations when the second guard is selected. By implementing the assignments sequentially, the number of operations in the second case can be reduced to five, if y is updated before d. In respect of the number and type of integer operations, the algorithm is therefore identical to the midpoint circle algorithm using second second-order differences presented by Foley et. al. [6, page 87, figure 3.18].

This latter algorithm is given below for comparative purposes. It has been adapted from C code to the syntax used here, and variables have been renamed to effect a resemblance to the above algorithm. However, although already shown to be incorrect, points at which $draw(x, y)$ has been invoked as well as the loop condition have been retained. Furthermore, there is no direct correspondence to the assertion-based algorithm, and meanings attributable to variables in one algorithm should not be construed to mean something similar in the other.

Algorithm 9.1:

$x, y, p, d, t := 0, r, (1 - r), 3, -2 * r + 5;$
$draw(x, y);$
{ invariant: $H(x, y, d)$ }
do $x < y \rightarrow$
  if $p < 0 \rightarrow$
    $p, d, t := p + d, d + 2, t + 2$
  ▯ $p \geq 0 \rightarrow$
    $p, y, d, t := p + t, y - 1, d + 2, t + 4$
  fi;
  $x := x + 1;$
  $draw(x, y)$
od
{ G }

□

Algorithm 8.3 can be transformed to Algorithm 9.1 by introducing a few new variables. Begin by noting that the guards in Algorithm 8.3 can be transformed to guards similar to those in Algorithm 9.1 by introducing a variable, $p = t - d - 1$. Note that initially, $p = r - 2$. Also $p < 0$ whenever $d \geq t$ and $p \geq 0$ whenever $d < t$. In order to compute p for the next iteration, the new values of d and t that apply in each respective guarded command have to be used. Thus, in the first guarded command $p_{new} = (t + 2) - (d - t - y) - 1 = p_{old} + (t+y+2)$. Similarly, in the second guard command, $p_{new} = (t+3)-(d-t+y-1)-1 = p_{old}+(t-y+4)$. This leads to the transformed algorithm:

Algorithm 9.2:

$x, y, p, d, t := 0, r, r - 2, -1, (1 - r);$

{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y)$;
  if $p < 0 \rightarrow$
    $p, d, t :=$
    $p + (t + y + 2), d - t - y, t + 2$
  $[\!]$ $p \geq 0 \rightarrow$
    $p, y, d, t :=$
    $p + (t - y + 4), y - 1, d - t + y - 1,$
    $t + 3$
  fi;
  $x := x + 1$;
od
{ G }

□

Note that d no longer plays any role in Algorithm 9.2 and may be removed. Further comparison of Algorithm 9.2 with Algorithm 9.1 suggests that new variables, say dd and tt, might be introduced where $dd = t + y + 2$ and $tt = t - y + 4$. dd should be initialized to $(1-r)+r+2 = 3$, and tt to $(1-r)-r+4 = -2*r+5$. These values correspond to the initial values of d and t in Algorithm 9.1.

Furthermore, new values for dd and tt to be used in the next iteration can be computed in each guard body as follows. In the first guard body, $dd_{new} = (t + 2) + y + 2 = dd_{old} + 2$ and $tt_{new} = (t + 2) - y + 4 = tt_{old} + 2$. Similarly, in the second guard body, $dd_{new} = (t+3)+(y-1)+2 = dd_{old}+2$ and $tt_{new} = (t + 3) - (y - 1) + 4 = tt_{old} + 4$. Thus t is also not needed to compute any useful value in the new algorithm.

Algorithm 9.3:

$x, y, p, dd, tt := 0, r, r - 2, 3, -2 * r + 5$;
{ invariant: $H(x, y, d)$ }
do $x \leq y \rightarrow$
  $draw(x, y)$;
  if $p < 0 \rightarrow$
    $p, dd, tt := p + dd, dd + 2, tt + 2$
  $[\!]$ $p \geq 0 \rightarrow$
    $p, y, dd, tt := p + tt, y - 1, dd + 2, tt + 4$
  fi;
  $x := x + 1$;
od
{ G }

□

Barring the incorrect flow of logic in Algorithm 9.1, Algorithm 9.3 clearly differs from Algorithm 9.1 in variable names only.

10 Conclusion

The assertion-guided approach to deriving a circle-drawing algorithm naturally highlighted the fact that $draw(x, y)$ should be called at the start of the loop and that the loop condition should be $x \leq y$. While the use of $x < y$ coupled with an invocation of $draw(x, y)$ at the end of the loop might be a small error in the published solutions, it could be a significant one under certain circumstances.

Sproull [7] addressed the line-drawing problem by starting with an "obviously correct" algorithm based on simple geometry, and, by a series of transformations, derived the well-known Bresenham line-drawing algorithm. Citing Sproull's work, Foley et. al. [6] state that: "No equivalent of that derivation for circles or ellipses has yet appeared." Sproull claimed that: "These transformations assure us that the more efficient but more complex algorithms are correct, because they have been derived from correct transformations from a correct algorithm." The claim in this research is that something similar has been achieved in respect of circle-drawing algorithms. Although admittedly not starting by making an appeal to simple geometry, the base-line algorithm is "obviously correct" in the sense of having been derived by a formal procedure. The fact that transforming the base-line algorithm to a version of Algorithm 9.1 throws up inaccuracies in published version, highlights the power and precision of the assertion-guided software development methodology.

References

[1] B. Meyer, Introduction to the Theory of Programming Languages, Addison-Wesley, 1990.

[2] D. Gries, The Science of Computer Programming, 2nd Edition, Springer-Verlag, 1980.

[3] E. W. Dijkstra, A Discipline of Programming, Prentice Hall, 1976.

[4] J. Gutknecht, Programming is teachable or how to leave rabbits in the hat, South African Computer Journal (3) (1990) 1–4.

[5] J. Bresenham, A linear algorithm for incremental digital display of circular arcs, Communications of the ACM 20 (2) (1977) 100–106.

[6] J. Foley, A. van Dam, S. Feiner, J. Hughes, Computer Graphics: Principles and Practice, second, in c Edition, Addison-Wesley, 1995.

[7] R. Sproull, Using programming transformations to derive line-drawing algorithms, ACM Transactions on Graphics 1 (4) (1982) 259–273.

To Derrick:

A very happy birthday to you. I hope you are suitably surprised at being a co-author of this paper in your own 'Festschrift'. This was the first paper which you invited me to co-author — a significant honour for me. It was exactly this sort of correctness-by-construction algorithmics that attracted me to Pretoria, though you were active in this approach long before my first visit to the computer science department in 1994. Your unwavering commitment to such algorithm(ic)s remains a source of inspiration for me, and I wish you many more years of great (and correctly constructed) algorithms. Even more, you have been an outstanding source of inspiration, friend and mentor — and I wish you many more years in those roles too!