Chapter 3

Circles

See the website for details on how the programming assignment will be graded. Each lab specifies a complete list of filenames to be used in the lab; use these names so that the grader can find your work.

It is important that you complete each step before going on to the next, as the exercises build upon one another. You will find it helpful to diagram the action of each method or function as you go along. If you have difficulty in some step, DO NOT proceed before resolving it. You will not be able to fully appreciate the remaining content of the lab and you are likely to compound the problem.

3.1 Introduction

This lab will acquaint the student with Michener's integer-based circle drawing algorithm and direct the creation of a visual program which will implement this algorithm.

3.2 Circles and Michener's Algorithm

Consider the generation of pixels along the circumference of a circle of radius R and centered at the origin as depicted in Figure 3.1. Note that the symmetry of the circle is such that when the pixel labeled P in Figure 3.2 has been selected to represent part of the circle, seven other pixels are also known; reflections of P about the x-axis, y-axis, and line $y = x$ yield three more pixels immediately, and additional reflections yield

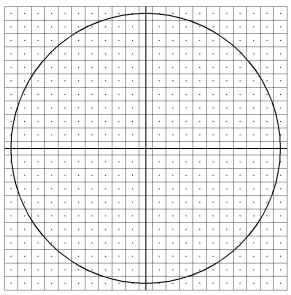

Figure 3.1: Circle of Radius R Centered at the Origin

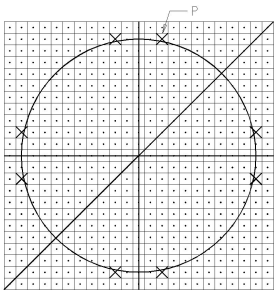

Figure 3.2: Pixel Symmetry of the Circle

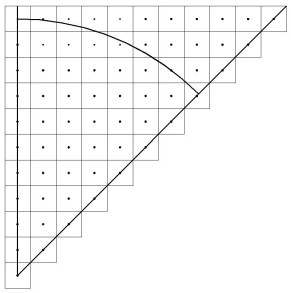

Figure 3.3: Pixels in One Octant of the Circle

four more. From this it can be seen that generation of pixels in only one octant of the circle is necessary; symmetry will provide those in the other seven.

In the development of the algorithm for a drawing a line, the slope was initially limited to the range between 0 and 1 so that as x increased by 1, y would either have to remain constant or increase by precisely 1. This same reasoning can be applied to a circle. Obviously, the pixel whose center is $(0, R)$ will be part of the circle; at that point, the line tangent to the circle has a slope of 0. As x increases from that point, the slope of the tangent slowly decreases. When the point is reached where x equals y on the circle, the slope of the tangent has become -1, and past this point y will begin to decrease more rapidly than x increases. Pixel generation will be derived for this octant of the circle, as illustrated in Figure 3.3.

In this octant then, the subsequent pixel from any given pixel is either on the same row or on the row below it. As with the line, let pixel S_i be the upper pixel of the two, in this case the pixel whose y-value does not change; also, let T_i be the pixel for which y does change.

$$S_i = (x_i, y_{i-1}) \tag{3.1}$$

$$T_i = (x_i, y_{i-1} - 1) \tag{3.2}$$

The error associated with the choices for the line's pixels was calculated as a vertical distance; for the circle, this approach would lead to a costly calculation. Instead, consider a measure of error to be determined radially.

$$e'(S_i) = \sqrt{x_i^2 + y_{i-1}^2} - R \tag{3.3}$$

$$e'(T_i) = R - \sqrt{x_i^2 + (y_{i-1} - 1)^2} \tag{3.4}$$

Since it is the difference in error values and not the error values themselves that is of interest, the error terms may be defined in a less costly fashion in terms of the difference in the *squares* of radial distances to

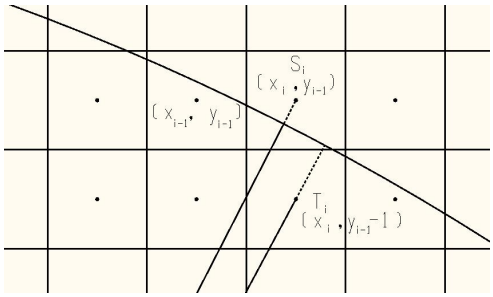

Figure 3.4: Passing Between Pixel Centers S_i and T_i

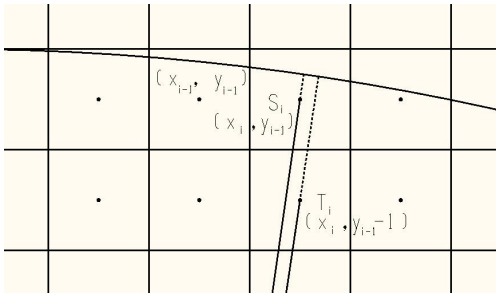

Figure 3.5: Passing Above Pixel Centers S_i and T_i

the circle and pixel centers as follows:

$$e(S_i) = x_i^2 + y_{i-1}^2 - R^2 \tag{3.5}$$

$$e(T_i) = R^2 - (x_i^2 + (y_{i-1} - 1)^2) \tag{3.6}$$

Figure 3.4 illustrates the situation for which both $e(S_i)$ and $e(T_i)$ are positive; clearly, the smaller of these two values indicates the pixel closest to the actual circle.

if $e(S_i) < e(T_i)$
    choose S_i
else // $e(S_i) > e(T_i)$
    choose T_i

Figure 3.5 depicts a region in which the circle passes above both S_i and T_i; in this case, $e(S_i)$ becomes negative and is, of course, smaller than $e(T_i)$, which has remained positive.

if $e(S_i) < 0 < e(T_i)$
    choose S_i

Figure 3.6 depicts another region in which the circles passes below both S_i and T_i; in this final case, $e(T_i)$ becomes negative and is thus smaller than the positive $e(S_i)$.

if $e(T_i) < 0 < e(S_i)$
    choose T_i

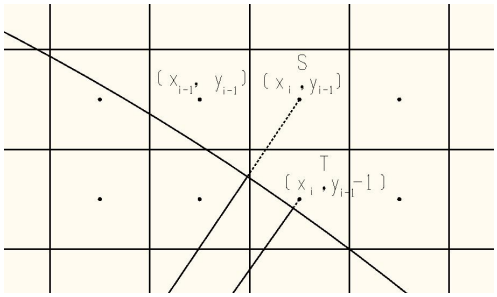

Figure 3.6: Passing Below Pixel Centers S_i and T_i

In all three cases, the smaller of the two values dictates which of the two pixels should be chosen. Consider then the comparison to be made between them.

$$\begin{aligned} e(S_i) &< e(T_i) && (3.7)\\ e(S_i) - e(T_i) &< 0 && (3.8)\\ x_i^2 + y_{i-1}^2 - R^2 - (R^2 - (x_i^2 + (y_{i-1} - 1)^2)) &< 0 && (3.9)\\ x_i^2 + y_{i-1}^2 - R^2 - R^2 + x_i^2 + (y_{i-1}^2 - 2y_{i-1} + 1) &< 0 && (3.10)\\ 2x_i^2 + 2y_{i-1}^2 - 2y_{i-1} - 2R^2 + 1 &< 0 && (3.11) \end{aligned}$$

For computational purposes, let e_i assume the value of the left hand side of the inequality; further, determine e_{i+1} for use in determining the change in e from one pixel column to the next.

$$\begin{aligned} e_i &= 2x_i^2 + 2y_{i-1}^2 - 2y_{i-1} - 2R^2 + 1 && (3.12)\\ e_{i+1} &= 2(x_i + 1)^2 + 2y_i^2 - 2y_i - 2R^2 + 1 && (3.13)\\ &= 2(x_i^2 + 2x_i + 1) + 2y_i^2 - 2y_i - 2R^2 + 1 && (3.14)\\ &= 2x_i^2 + 4x_i + 2y_i^2 - 2y_i - 2R^2 + 3 && (3.15) \end{aligned}$$

The difference between successive e_i values is labeled Δe.

$$\begin{aligned} \Delta e &= e_{i+1} - e_i && (3.16)\\ &= 2x_i^2 + 4x_i + 2y_i^2 - 2y_i - 2R^2 + 3 - (2x_i^2 + 2y_{i-1}^2 - 2y_{i-1} - 2R^2 + 1) && (3.17)\\ &= 2x_i^2 + 4x_i + 2y_i^2 - 2y_i - 2R^2 + 3 - 2x_i^2 - 2y_{i-1}^2 + 2y_{i-1} + 2R^2 - 1 && (3.18)\\ &= 4x_i + 2(y_i^2 - y_{i-1}^2) - (2y_i - 2y_{i-1}) + 2 && (3.19) \end{aligned}$$

In moving from one pixel column to the next, x increases by one; if $e_i < 0$, there is no decrement to y.

$$\begin{aligned} x_i &= x_{i-1} + 1 && (3.20)\\ y_i &= y_{i-1} && (3.21)\\ \Delta e_{noDec} &= 4(x_{i-1} + 1) + 2(0) - (0) + 2 && (3.22)\\ &= 4x_{i-1} + 6 && (3.23) \end{aligned}$$

If $e_i > 0$, then y decrements to the next pixel row as x increases by one.

$$
\begin{aligned}
x_i &= x_{i-1} + 1 && (3.24)\\
y_i &= y_{i-1} - 1 && (3.25)\\
\Delta e_{Dec} &= 4(x_{i-1}+1) + 2((y_{i-1}-1)^2 - y_{i-1}^2) - (2(y_{i-1}-1) - 2y_{i-1}) + 2 && (3.26)\\
&= 4x_{i-1} + 4 + 2(y_{i-1}^2 - 2y_{i-1} + 1 - y_{i-1}^2) - (2y_{i-1} - 2 - 2y_{i-1}) + 2 && (3.27)\\
&= 4x_{i-1} + 2(-2y_{i-1} + 1) - (-2)) + 6 && (3.28)\\
&= 4x_{i-1} - 4y_{i-1} + 10 && (3.29)
\end{aligned}
$$

Initially,

$$
\begin{aligned}
x_0 &= 0 && (3.30)\\
y_0 &= R && (3.31)\\
e_1 &= 2(1)^2 + 2(R)^2 - 2(R) - 2R^2 + 1 && (3.32)\\
&= 3 - 2R && (3.33)
\end{aligned}
$$

Repetitive calculations will now generate the remainder of the points in the octant, and the symmetries described earlier may be used to complete the other seven octants.

A final point remains to be made. While the preceding derivation generates pixel coordinates about a center at origin (0, 0), it is the circle about arbitrary origin (h, k) is typically desired. To obtain this set of pixels, one need simply add offsets h and k to each pixel coordinate generated above; the result is a *translated* circle.

3.3 Example

Consider any circle to be drawn with radius 12. The initial conditions are as follows.

$$
\begin{aligned}
x_0 &= 0 && (3.34)\\
y_0 &= R && (3.35)\\
&= 12 && (3.36)\\
e_1 &= 3 - 2R && (3.37)\\
&= 3 - 2(12) && (3.38)\\
&= -21 && (3.39)
\end{aligned}
$$

The mathematical circle passes through the center of the pixel at (0, 12). Since e is initially negative, the mathematical circle passes more closely to the pixel at (1, 12) than that at (1, 11); the fact that y is not decremented in turn dictates the Δe change in value e_1 to e_2 ($4x_{i-1} + 6 = 4(0) + 6 = 6$ rather than $4x_{i-1} - 4y_{i-1} + 10 = 4(0) - 4(12) + 10 = -38$), as shown in the third column of the Table 3.1. Subsequent values are arrived at by applying the same logic. The iteration stops at the point at which x is no longer less than y; past this point, the assumption made about the rate of change of x being greater than that of y is no longer valid. This poses no problem, however: as described earlier, symmetry may be used to reflect the points generated in this first octant to complete the circle in the other seven.

i:	0	1	2	3	4	5	6	7	8
Δe:			6	10	14	-26	22	-14	30
e_i:		-21	-15	-5	9	-17	5	-9	21
x_i:	0	1	2	3	4	5	6	7	8
y_i:	12	12	12	12	11	11	10	10	9

Table 3.1: Example Michener Pixels

3.4 Assignments

3.4.1 Numeric Results

Write a console-oriented program which will begin by prompting the user for the name of a disk file, e. g. *circles.txt*. The first line of the file will contain an integer n indicating how many subsequent lines the file contains; each of the remaining n lines of the file will correspond to one mathematical circle for which pixels are to be calculated. Following is an illustration of the file format:

```
3
h1 k1 r1
h2 k2 r2
h3 k3 r3
```

Your program should calculate and print the coordinates of the pixels to be drawn to represent each of the circles in the file in a format similar to that for lines in Table 3.1 (although perhaps with columns instead of rows); include the error calculation for each loop iteration.

3.4.2 Graphical Results

Write a program which will create a 500 x 500 frame window and then present the user with a dialog box requesting the name of a disk file identical in contact to that of the preceding program. Render each of the circles in this file in the frame window in color *Qt::green*.

3.4.3 Program Submission Details

When the assignment has been completed and tested thorougly, submit one zip file containing all files associated with the assignment by using the course submission utility. (An evaluation copy of WinZip may be downloaded at *http://www.winzip.com/ddchomea.htm.*) Zipping the files will preserve the project and prevent corruption of the source code. (For example, Hotmail inserts carriage returns after about 80 characters of text.)

See the course website *Grading Criteria* page for details on how programming assignments will be graded.

If twenty or thirty minutes have been spent working on the same problem with no discernible progress, an impasse may have been reached for which assistance should be sought. In this case, contact the instructor or teaching assistant during office hours or by e-mail for assistance. When requesting assistance via e-mail, the following steps will reduce the time required to resolve the problem:

- Include the word "help" and the course number in the e-mail subject line.

- Describe the problem two ways: first, by describing the offensive behavior of the program and second, by identifying what appears to be the troubled area of program code.

The nature of the problem may be such that an office call will be more effective. For an office call, in addition to the program itself bring all associated paperwork detailing algorithm development and data structure design. For more on assistance, see the *Assistance* link on the course website.