Richard Paredes
UHID: 1492535
COSC4370

Homework 1 Report

The problem entailed developing an algorithm for rasterizing semi-circles with the following constraints:

$$x^2 + y^2 = R^2, where\ x \geq 0\ and\ R = 100$$

$$x^2 + y^2 = R^2, where\ y \geq 0\ and\ R = 150$$

The challenge in this problem involves mapping the coordinate points from the function of a circle to an appropriate pixel on the display such that it minimizes any jaggies on the resulting graphic. The jaggies are a consequence of discretizing the function into approximations that can map into pixel positions that are rendered on a display.

Unfortunately, the brute-force, basic incremental algorithm for scan conversion of lines is very slow and inefficient, so optimizations are needed. Through mathematical insight, slight improvements can be made to the algorithm such as using integer arithmetic and updating the computations to build off previously computed calculations. Instead, an alternative algorithm can be used—the midpoint algorithm. With this algorithm, we determine which pixel is next by taking the intersection of the function on the gridlines. Based on whether the intersection is above or below the midpoint will determine what pixel should be selected. The decision variable is then updated accordingly for use in selecting the next pixel. The advantage of this technique is that only addition is needed for determining the pixel selections.

However, scan conversion of circle graphics is a bit more complicated than simple lines. A technique for optimizing the computations needed to render a circle is to use the symmetry associated with a circle. After finding a 45-degree segment of the circle, we can use symmetry to complete the remainder. Fortunately, an algorithm very similar to the midpoint algorithm for scan conversion of lines can be used, but also takes advantage of the fact that second-order partial differences are being calculated. Furthermore, the arithmetic involved in the algorithm can be simplified to using addition with integers. Simply put, the algorithm does the following:

1. A pixel is selected based on the sign of the decision variable computed from the previous iteration
2. The decision variable by the partial difference accordingly based on which pixel was selected
3. Update the partial differences by accounting for the newly selected pixel
4. Set the selected pixel's value to be rendered

Thus, the implementation of this algorithm was fairly straightforward. The function initializes by using the pixel at the edge of the circle. The partial differences and decision variables are set based on the radius of the circle. A loop is used to iterate until the end of the 45-degree segment is reached, which is denoted when x and y coordinates are equal. As

Richard Paredes
UHID: 1492535
COSC4370
mentioned previously, symmetry is used when updating the selected pixel's value for rendering so that a semi-circle is depicted rather than only circle quadrant.

Traditionally, the center of the circle is situated at the bottom left of the window, so the initialized coordinates needed to account for the x and y offsets so that the semi-circles could be properly centered and entirely rendered in the window. Since two semi-circles with different radii were being rasterized, the algorithm needed to be run twice, once for each semi-circle. Moreover, since the constraints for each semi-circle were different, (i.e. one semi-circle only showed positive y's, the other only positive x's), then the symmetry used to select pixels also differed. For the semi-circle of radius 150 pixels (positive y's), the selected pixels needed to be reflected only along the y-axis so that the top-half of the full circle would be rasterized. In a similar fashion, for the semi-circle of radius 100 pixels (positive x's), the selected pixels needed to be reflected only along the x-axis so that the right-half of the full circle would be rasterized.

Depicted in the figure on the right is the result of the two rasterized semi-circles. The results of the implementation of the algorithm can be further inspected in the circle.ppm file output, which will show two semi-circles adhering to the constraints mentioned in the first paragraph. The source code of the implemented algorithm can be inspected in hw1.cpp file.