Assignment 1 Report

Problem:

For this assignment, we need to create a program that produces a .ppm file that contains two semi-circles: one were the y<=0 and an R of 150, and another with an R of 100 and were the x<=0. The main obstacle that needs to be addressed is the algorithm that draws the semi-circle correctly and being able to it manipulate the radius and only render the parts needed for each.

Algorithm and Methods:

The main algorithm used is the Bresenham's circle drawing algorithm which is derived from the midpoint circle algorithm. Midpoint circle algorithm draws pixels each pixel approximately equidistant from the center point of the circle and is drawn only 45 degrees at a time. The algorithm determines what the next pixel will be plotted. For example, if there exist a pixel (x,y) the next point is either (x-1,y+1) or (x,y+1). In order to choose the correct pixel, the algorithm finds the midpoint in between each point pixel (x-.05,y+1), then determines if this midpoint lies inside or outside the circle. This process repeats until the x is greater than the y. (45 degrees)

The Bresenham's circle drawing algorithm is the midpoint circle algorithm, but exclusively uses integer arithmetic, which results in a faster and less memory consuming algorithm, for float values are bigger than integer values. Therefore, Bresenham's algorithm is seen as an optimized version of the midpoint circle algorithm.

The algorithm utilizes the 8-way symmetry of the circle in order to draw the entire circle if needed, for the algorithm only draws 45 degrees of the circle at a time. In order to draw the entire circle, the algorithm is executed a total of 8 times at different starting points each time. (x,y) (y,x) (x,-y) ….

Implementation:

The code is written in C++. It is a simple code, so I only focus getting the two tasks (both semi-circles drawn) done. There are two functions: rasterizeArc, and renderPixel. rasterizeArc contains the Bresenham's algorithm. We have 4 integer variables r (the radius) x=0, y=r, and d = 1-r, and one Boolean variable type. The variables x,y and r are all based on the equation of a circle, x^2 + y^2 = R^2 and d is the decision parameter for choosing the next pixel to be rendered. Variable type chooses which of the two semi-circles the renderPixel function renders.

The rasterizeArc function follows the Bresenham's algorithm. The function calls renderPixel when the pixel is chosen. The first pixel is chosen before the algorithm enters the while loop. The x is incremented for the next pixel, and then the decision parameter chooses if the next pixel is (x+1,y-1) or (x+1,y). Once chosen, the pixel is rendered, and the loop continues until x>y.

The renderPixel function renders each arc that the rasterizeArc function produces in the correct location. Because the rasterizeArc function only produces 45 degrees of a circle, the renderPixel function utilizes the 8-way symmetry put choosing where each identical arc is placed. Because we are using an array for the pixels we cannot use (x,-y) or (-x,y) from the original concept. Instead, because the origin

(0,0) is at the bottom left of the ppm file, we need to define the origin as the center of the file. Variables cx and cy are the centers of the file. So instead of the original (x,y) (y,x) (x,-y) …. Its (cx+x,cy+y) (cx+y, cy+x) (cx+x, cy-y) …

The type of variable is passed into the renderPixel function as well as the x’s and y’s from the rasterize arc function. The type determines which sections of the circle are rendered based on the radius of the circle.

Results:

The resulting semi-circles produce both semi-circles with the correct conditions in a ppm file that is located where the C++ file is compiled.