Osinor Igwemoh

Report for Homework 1.

Problem Description

The assignment requires us to derive and implement a similar algorithm to that in the reading for rasterizing circles. And it wants us to rasterize specifically half of the circle with the equation $x^2 + y^2 = R^2$ with x being greater or equal to 0. And the positive integer R=100 and another half of a circle with the same equation but R=150 and y will be greater or equal to 0. And the centers will be at the origin point for both circles. I utilized the information from the reading on “Scan Converting Lines” and Scan Converting Circles”.

Algorithm Explanation

I used Linux for compiling my c++ file and graphical viewer GIMP to view my ppm files. The algorithm I used is the Bresenham’s Line drawing algorithm. This algorithm considers only 45 degrees of a circle. It is like the midpoint line algorithm where the strategy is to select which of the 2 pixels is closer to the line of the circle by calculating a function at the midpoint between the 2 pixels in question. The first pixel is drawn from a chosen point and the rest of the points are sampled by the algorithm. The algorithm choses the next pixel from an already chosen pixel based on the shorter distance between the mid pixel and the next two pixels. The algorithm I utilized chooses the next pixel based on the sign of the variable distance (d) and on the next iteration, the variable (d) is updated with the variables deltaE and deltaSE. And these variables are initialized using the start pixel (0,R).

Implementation

For my implementation I utilized the algorithm above in the raterizeArc function. The problem requires two semicircles but because the algorithm only considers the 45degrees of a circle, I took advantage of the Eight-Way Symmetry explained in the reading and how one can render different parts of a circle just from one quadrant without having to repeat the algorithm; this is where the renderPixel function comes in. The render pixel function takes the x and y values as arguments and lights up the corresponding pixel on the image matrix. The algorithm works for half of a quadrant and then interchanges the x and y values to light up its symmetric counterpart. So far just a full semicircle has been rendered to the matrix, I created another renderPixel function to and called it at the same point in the rasterizeArc function using the radius to make differences in the x and y values depending on the quadrant to be rendered and this lights up the symmetric counterpart of the first quadrant in the second quadrant and hence the first semicircle of radius 150 was completed.

For the second semicircle of radius 100 where the x values will be greater or equal to zero, the algorithm is very much similar to the first, the only difference is a change in radius size and because the circle grew I had to make changes to the delta by amplifying them. One of my first render Pixel functions rendered the same pixel as one half of the semicircle for the circle of radius 100 so I called the same one and passed it the new x and y values obtained from the algorithm with radius = 100 and created a third render Pixel function for a quadrant that has not been rendered by any of my previous functions and this idea is from the Eight-Way Symmetry explanation of a circle from the reading.

Result

At first I got a lot uneven circles when I tried to render their symmetric counterparts but after adjusting the values of the deltas in the midpoint algorithm they became more like a circle. The final result was an outer semi-circle with a radius of 150 which occupied the first and second quadrants and a smaller semi-circle on the same origin as the first which occupied the third and fourth quadrants.