Benjamin Northrop
5/30/24
CS5310
Summer 24

Introduction

Homework 3 involved consisted of creating graphics primitives, specifically lines, circles, ellipses, and polylines. The majority of the effort is spent getting Bresenham's algorithm to show correctly on the lines and circles, and then the rest is small changes in the remaining algorithms.

SpeedTest:

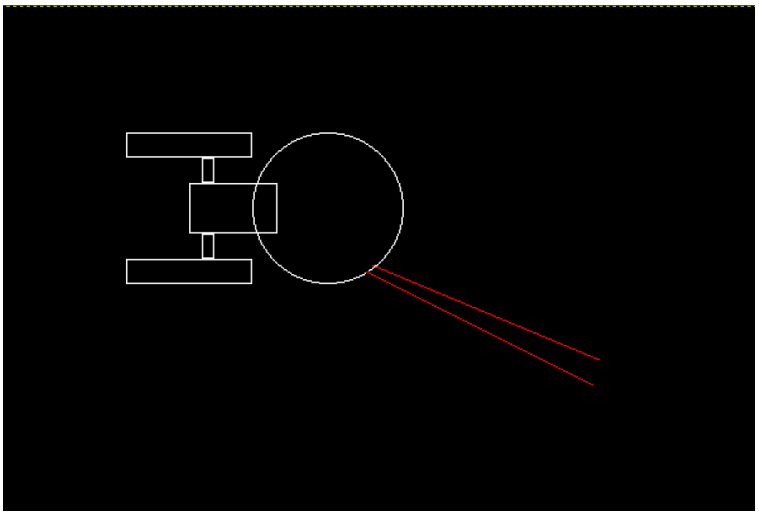
```
bnjnrthrp@BenjisLaptop:/mnt/c/Users/benji/dev/CS5310/graphics/tests$ ../bin/testbench
Average line length = 102.9
3236245.79 lines per second
```

Currently, our algorithm was able to produce 3.2M lines per second. This is without any optimization besides using the integer version of Bresenham's algorithm. Potential for improving efficiency would be to develop a system of memoization, because it seems like as you iterate through each step, a pattern eventually comes out. This could be used to speed up the algorithm.

Projects

Test3a

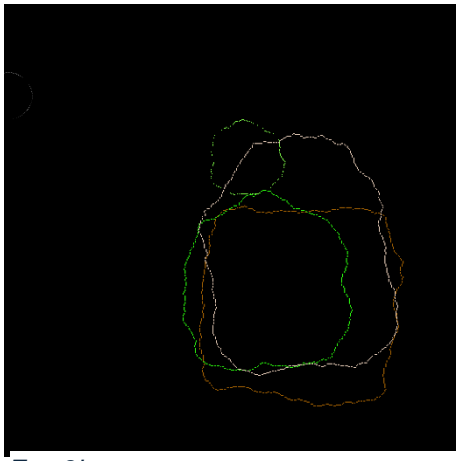
The first image is the USS Enterprise (I think). I started with the line function and commented out everything except the drawing of lines. This helped ensure that the program was actually printing correctly. Then I uncommented the circle portion and implemented the circle. To compare, I saved the original file and use GIMP to compare pixel to pixel to ensure the X and Y coordinates were correct to resolve any off-by-one errors.



Test3a

Test3b

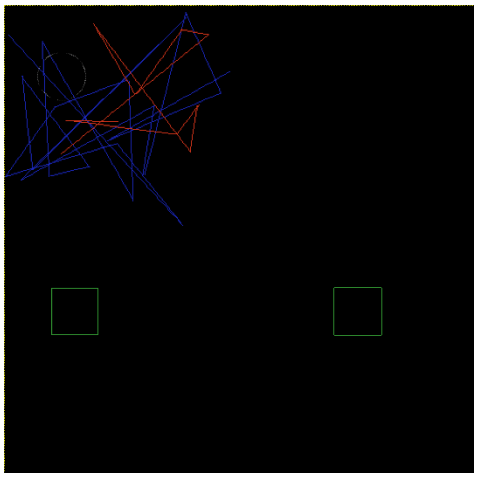
This next test I ran, and I seemed to get results but it seems more dashed. Trying to understand the code, it seems like the test is taking a line and subdividing it into smaller segments so it can include the perturbation factor and give it a random feel. I'm wondering if there is an error in the code in how my lines are drawn that I drop a pixel.



Test3b

Test3c

My primary focus for test3c was to work on the boxes and how they were drawn. I compared the pixel locations for both square drawings to ensure that my output matched the

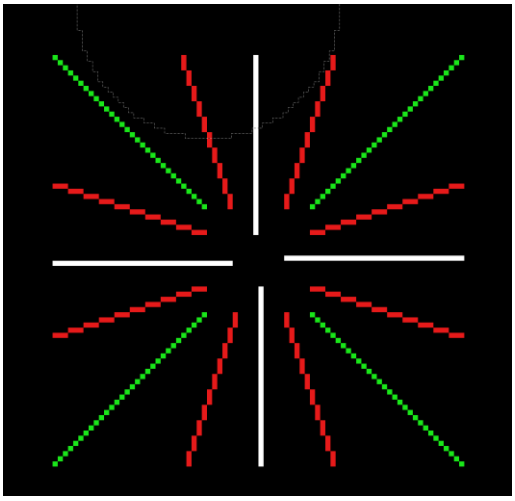


Test3c

actual. This typically involved adding or subtracting 1 in the algorithm to account for the side of the pixel I am drawing.

Test3d

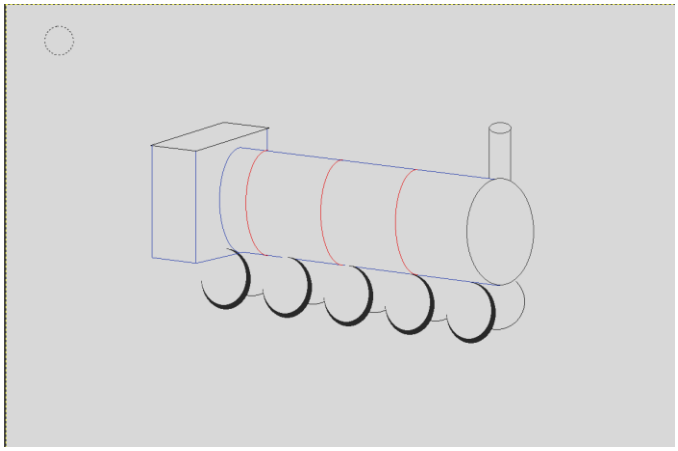
I worked on my line implementation first with this image. I started with the horizontal lines, and adjusted my algorithm so that it would match the original pixel for pixel. This set me off to a great start as I started with the horizontal and vertical lines, then worked through each octant from 1 to 8.



Test3d

For the final personal picture, I decided to draw a train. Initially, I wanted to use more of the filled circle and ellipse functions, but I realized that I hadn't implemented a normal fill function yet, so only the round objects would have been filled and the polylines would remain empty. To start, I drew a template in figma to use as a basis for identifying pixel locations for the objects. Then, I drew one shape at a time, verified the pixel location in GIMP, and continued drawing from there.

For the wheels, I had the opportunity to include a loop to draw each wheel, and incorporate a feature that allows the user to draw partial circles or ellipses, depending on the quadrants/octants they choose.



Train Image

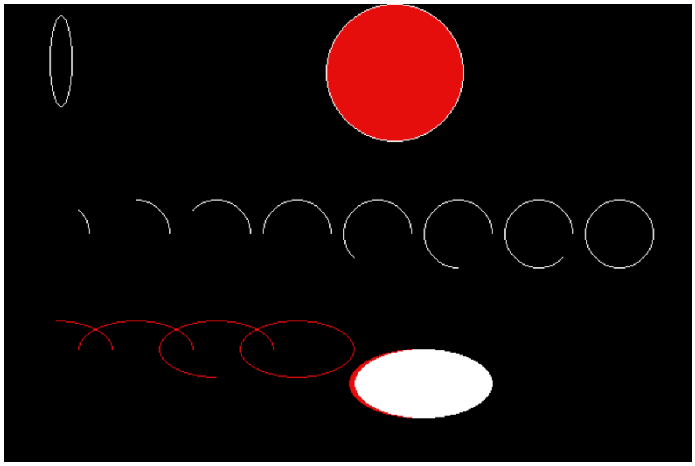
Extensions

While a little clunky, the circle and ellipse functions have the ability to draw partial circles and ellipses. We maximized code reuse and put the algorithm into a helper function, and then both the full circle draw and partial circle will use this algorithm. The downside is we do add some extra overhead to the function run times when drawing these circles.

Currently, we can only fill the full circles.

Reflection

Overall, the test files provided a perfect way to run tests and implement the algorithms incrementally. I was able to draw 1 line at a time and focus on each quadrant/octant individually. On the other hand, I'm very happy with how robust the algorithms have been, once they are working they have been quite reliable despite different inputs.



Circle test results

Acknowledgements

The circle and ellipse algorithms were derived from the lecture notes.