Scott Gordon

4/12/2021

COMP 5460

Assignment Five Report

The purpose of this paper is to show the issues faced, lessons learned, and any remaining bugs in my assignment five, as well as list any extra effort that was put into the assignment past the requirements.

First, how my program functions from the user’s perspective. The user chooses a function that they want to work with, the DDALine algorithm is chosen as default. For the first set of functions, the user can click and drag and immediately see the results of the algorithm on the canvas. There is rubberbanding implemented as well as using the mouse click to choose points. I chose to take this approach as extra credit and to make the user experience better. For the second set of functions, the user must click four times on the canvas, and each click represents one of the points used for the algorithm. For example, in the Bezier Curve algorithm, your first click location is p1, your second click location is p2, and so on.

Getting started on this assignment was simple. We were given starter code in c and given the task to translate that to JavaScript and instead of output to console, draw the shapes. The DDALine function was simple, essentially just translating the code, only swap the coordinates if it is quadrant two or three. Midpoint circle was the same, a simple translation. Midpoint ellipse was a little confusing in that I was not sure I had the right result. I believe I do, the p1 represents the midpoint of the ellipse, and p2 represents the width and height of the ellipse using the values directly of p2.x and p2.y.

For the curves, the Bezier was straightforward in what each point represented and translating the starter code was simple. For the Hermite and Spline curves, I’m just unsure on what the curves were supposed to look like, but my points are identical to the ones generated by the code given to us, so I assume it is correct.

The largest struggle I had was getting the midpoint line algorithm to work in all slopes and quadrants. I tried flipping the coordinates and other things but struggled for a while with it. What finally worked is treating the dx and dy separate, determining which had a greater impact on the line, then developing the line from there.

For comparing primitive calls to the developed functions, my DDALine algorithm let to the line fading out when the line was close to vertical while the primitive call did not. The midpoint line was basically identical to the primitive call though, while maybe being a little rougher. Each of the circle algorithms produced the same circle, but the primitive calls made the circle more smoothly than mine. The only curve algorithms I was able to find a primitive call for was the Bezier curve, and the result was the exact same, I noticed a small color difference but that is attributed the choice of ‘n’ or number of iterations to draw the curve. I found that my other curves produced the exact same points as the code that was given to us and that was satisfying enough for me to consider that the same as the primitive call.

For extra credit, I implemented the rubberbanding that applies to the appropriate algorithms and allowing the user to click to input the points instead of input into text boxes. That is all the work I did for this assignment, a still image from my program is added below to show program output.

