Larkin Wisdom

Assignment 1 Report

**Problem Statement**

I drew the letters WAKE as well as an XYZ coordinate system using both OpenGL functions with GL\_LINES (red color) and the Bresenham line algorithm with GL\_POINTS (green color). I made a system which draws lines on left-click (dark blue), and keeps them on the screen when a new line is being drawn. The current line being drawn is actively updated on the screen. The lines for this system are drawn using the midpoint algorithm. For extra credit, I made a system which places a circle on the screen when ‘R’ is pressed (light blue), an ellipse when ‘E’ is pressed, and clears the circles and ellipses when ‘W’ is pressed. The circles and ellipses are drawn using the midpoint ellipse algorithm.

**Algorithm Design**

There were four different types of constructions used in this program, each of which is a different color (red, green, dark blue, light blue).

The red was drawn using the basic GL\_LINES function, so there is little to comment on.

The green was drawn using the Bresenham line algorithm. The Bresenham algorithm works by determining whether or not the next value should be (x+1,y) or (x+1,y+1) based on a decision variable. If the variable is greater than 0, then we know that y should be incremented. This method allows us to avoid non-integer arithmetic, so it is much faster than a brute force approach with rounding values to the nearest y. The algorithm generates a set of points, which I then put in a vector called bren. Each point in bren is displayed when display() is called. The Bresenham algorithm is only designed to work in the first octant of the coordinate plane, however, so I modified the algorithm to work in every octant. I do this by first flipping the first second coordinate, if (x\_1 > x\_2), to ensure that I am working in the “positive x quadrants”. Then, I realign the points, making x\_1, y\_1 the origin, and putting x\_2,y\_2 in either the first, second, seventh, or eighth octants. From here, if x\_2, y\_2 is below the x-axis, I flip it over, and then I flip the coordinates to get it into the first octant, if necessary. I have two different functions written, brensenhamneg for values below the x-axis, and brensenhamabove1, for values where the slope is >1. The bresenham function itself simply determines when they should be used, based on the location of the points. After this, I perform the Bresenham algorithm, and then do the same process in reverse, undoing all of the geometric changes made before to each of the points generated, so that when they are put into the bren vector, they are correct and not in the first octant (unless they were supposed to be there).

The dark blue, drawn using the mouse, used the midpoint algorithm for line drawing. The midpoint algorithm works by considering the midpoint between (x+1, y) and (x+1, y+1). If the line at x+1 is above the midpoint, then we plot the point at (x+1, y+1). If it is below, then we plot the point at (x+1, y). This is a similar process as the Bresenham algorithm, using a decision parameter which requires only integer computation to determine the relationship. Furthermore, the midpoint algorithm also only works in the first octant, so I applied the same modifications as described in the above paragraph to the midpoint algorithm. The only difference here is that I did not make separate functions for each of the different octants, but packed them all into one function. The midpoint\_current function is the same as the midpoint function, but it does the values for the line currently being drawn.

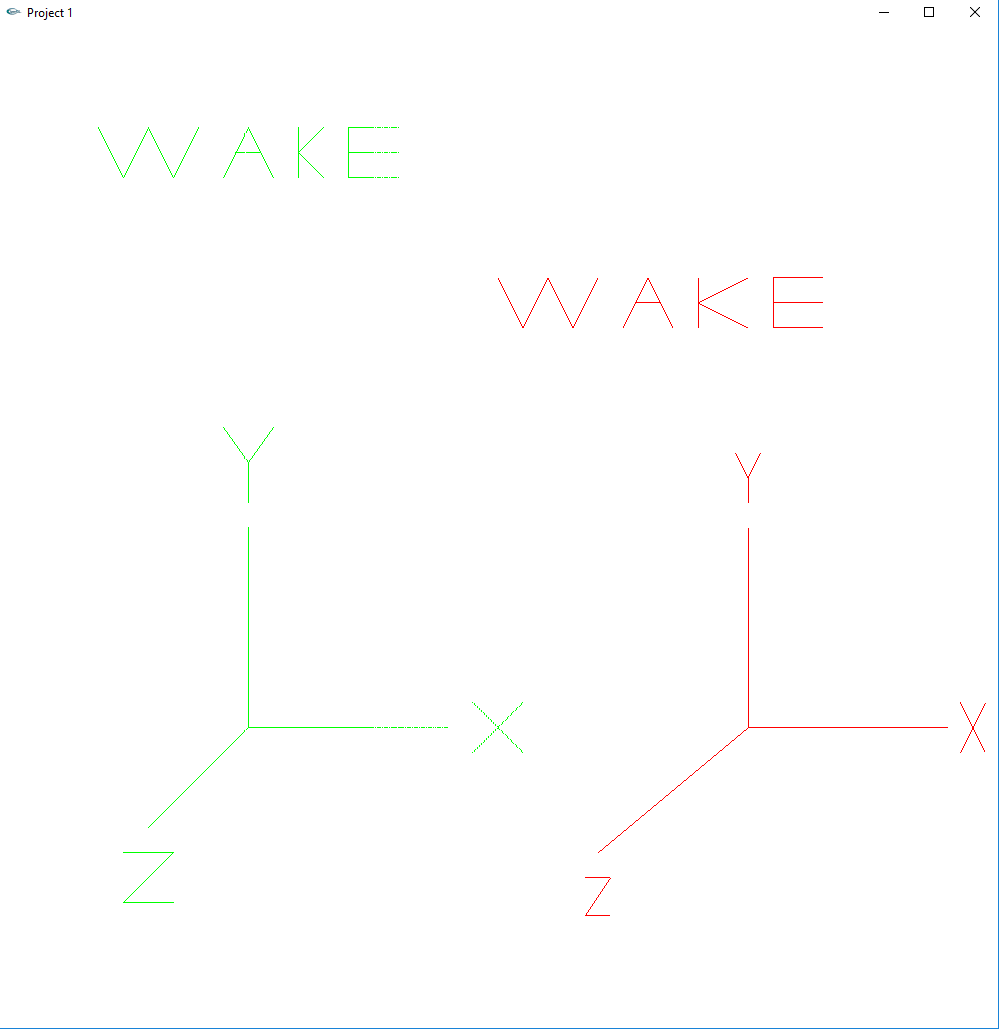
The light blue ellipses and circles were drawn using the midpoint ellipse algorithm. Similar to the midpoint line algorithm, there are decision parameters which help to determine which point should be selected. We increment by whether or not this decision parameter is positive or negative. In the second quadrant, we increment y if the decision variable is positive, while in the first quadrant we increment x if the decision variable is negative. Two different bodies of code each determine points for the first and second octants, and because ellipses and circles practice a symmetry through their origin, simply negating x, y, or both will yield the other 3 quadrants of the circle/ellipse.

**How to Run the Code**

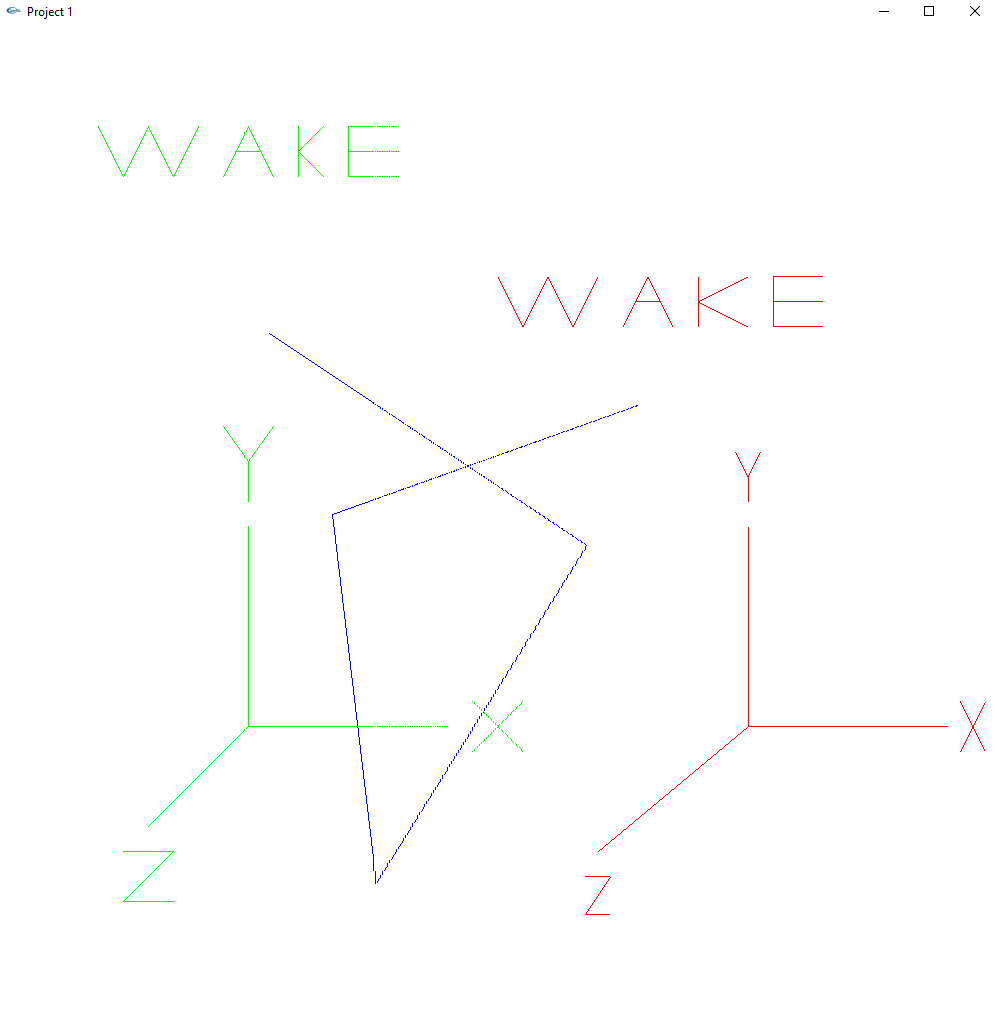
The program requires an installation of glut (I used the original v3.7.6, not freeglut). I have attached the entire Visual Studio project, but all you need to do is click on the .exe in the “RUN PROGRAM HERE & PROJECT REPORT” folder to run. The project report is there too. Email me at [lwisdom1@binghamton.edu](mailto:lwisdom1@binghamton.edu) if there is a problem.

**Screenshots**

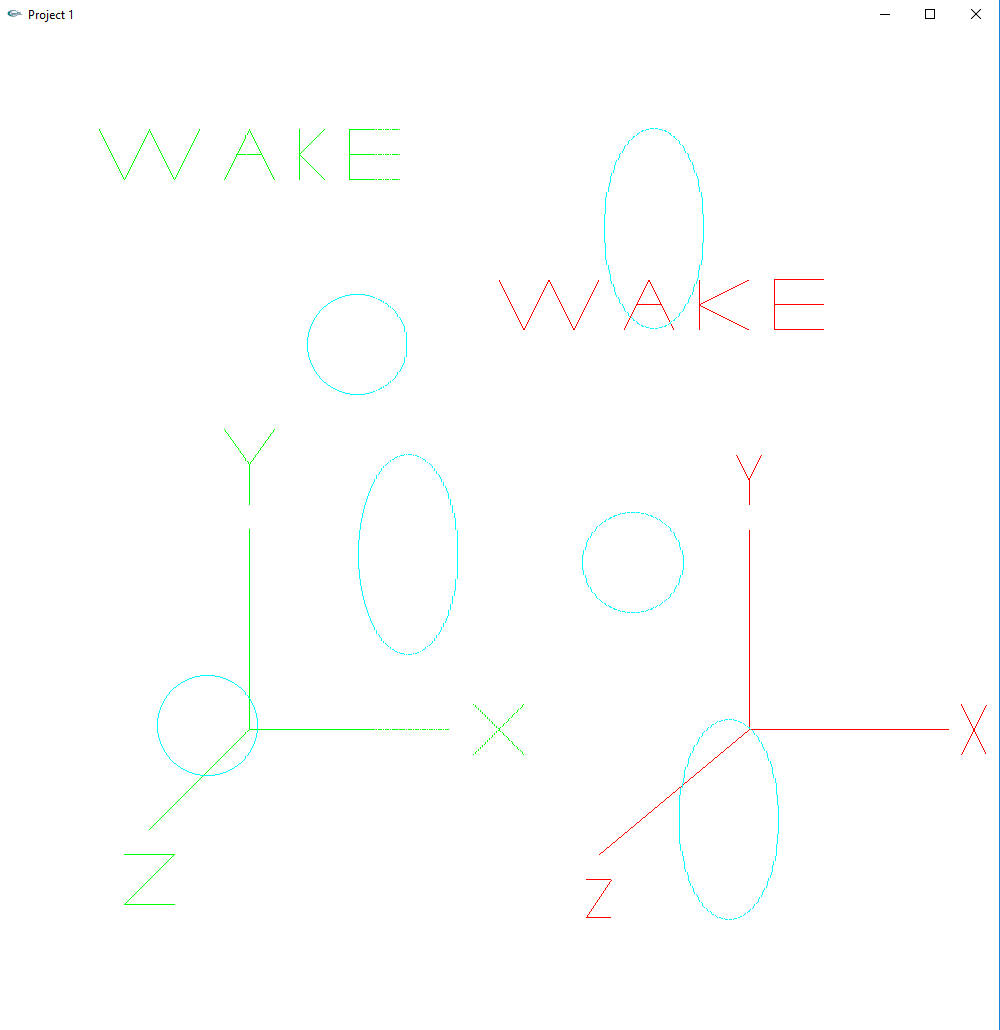
Basic window when the program first runs



Program running with several lines drawn using left click



Program running with several circles and ellipses drawn



Lines, circles, and ellipses

