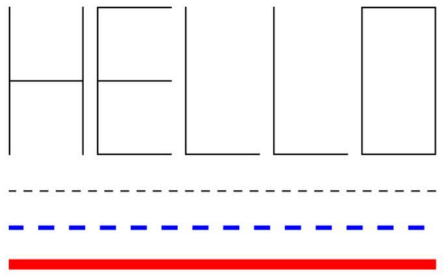
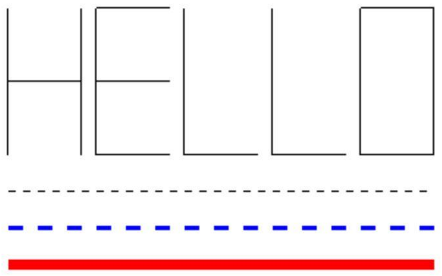
Assignment 1 Writeup Ryan Shendler

The problem for this assignment was twofold. First, I had to implement the Bresenham line drawing algorithm to draw the letters "HELLO" and several horizontal lines with varying color, width, and stippling. I also had to recreate this same drawing using the OpenGL library functions for drawing lines. The second part of this assignment was to implement the midpoint line drawing algorithm such that users could draw a series of interconnected line segments by clicking control points in the display window.

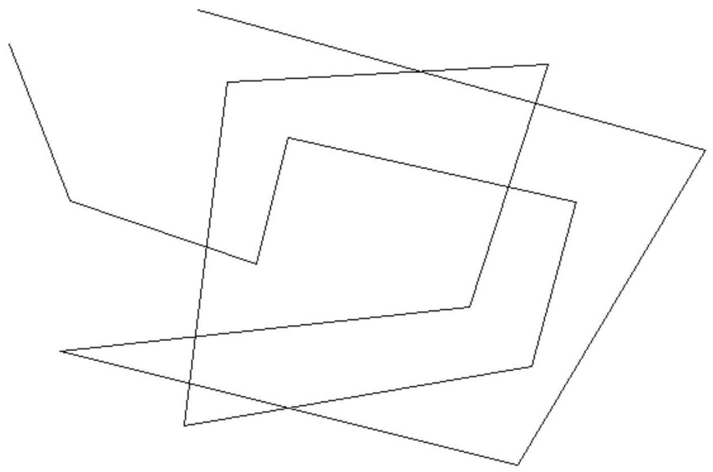
My Bresenham algorithm is implemented in bresenham.cpp, which is located in the BresenhamLine project. For my Bresenham line drawing algorithm, I split the line drawing function into three main functions: drawLowSlope(), drawHighSlope(), and drawLine(). I did this because the standard Bresenham algorithm we discussed in class only works for lines in the first quadrant, and therefore requires you to perform some sort of transformation on your endpoints in order to draw lines in different quadrants. I designed drawLowSlope() to draw lines with a slope between -1 and 1, while drawHighSlope() draws lines with slopes greater than or less than 1. All drawLine() does is determine which quadrant the specified line falls in and then calls one of the two previous functions. I also defined the helper function setPixel() which simply draws a point at the specified pixel. When you run bresenham.cpp in Visual Studio, it will automatically display the below drawing.





The drawing on the left side of the window uses my implementation of the Bresenham algorithm, while the drawing on the right uses OpenGL's implementation of the line drawing function.

My midpoint algorithm is implemented in midpoint.cpp in the MidpointLine project. Since the midpoint line drawing algorithm is almost identical to the Bresenham algorithm, my implementation for drawing lines with the midpoint algorithm is essentially the same as my Bresenham implementation. The main difference with my midpoint implementation are the functions processMotion() and processMouse(), which process mouse motion and mouse clicks respectively. Upon the first left mouse click, processMouse() clears the screen and then adds the clicked point to the vector vertices, which stores all of the current control points. Upon subsequent left mouse clicks, processMouse() will clear the screen, add the clicked point to the vertices vector and then draw lines between all of the control points in the vertices vector. Upon a right-click, processMouse() clears the screen, draws lines between all the control points (including the right-clicked point), and then clears the vertices vector. Therefore, upon the next left mouse click, all of the drawn line segments will be cleared and the clicked point will be added to the vertices vector as the first of a new set of control points. Every time the mouse is moved, the processMotion() function checks to see if the vertices vector is empty. If it isn't empty, it clears the screen and then draws lines between all of the control points in the vectors. It then connects the final point in the vector to the point where the mouse moved to. Since this drawing is refreshed every time the mouse is moved, this makes it seem like the line is following the mouse. The following drawing is an example of a line segment that I drew with my midpoint algorithm implementation.



Both of my implementations can be run by opening Assignment1.sln in Visual Studio and then building and running the respective project.