The Line Game: Write Up

My final project is a game that lets the user “drive” the cursor around, using keyboard controls. The cursor is moved in the X direction by the up and down arrows, the Y direction by using the left and right arrows, and the Z direction using hyphen and equals key (- and +). Using the space bar plots a point at the cursor’s current location, and after the first point it connects that point with that last point. The program also uses these two points to calculate the equation of the line and displays it next to the point. Pressing the R key will reset the graph, P will pause the cursor movement, and Q will quit the graph. Pressing the A key will also calculate the integral over the entire plotted area.

This project was inspired by a game I played as a child on old Apple II computers. In this game you controlled a turtle and he drew lines and gave you the equations of the lines. However, I have expanded on this, providing both and integral function as well as the ability to map in three dimensions.

Creating the keyboard interface in MATLAB was surprisingly easy. The figure object has a predefined property for keyboard interrupt callback. MATLAB passes an object when you press a key to the specified callback function and allows you to specify actions based on the key pressed. This ability can be disabled by changing the interrupt property of the figure or axes.

Movement is controlled by the keyboard callback feature and a struct called “current”, which is a struct of different status effects of the game, such as the current direction and whether to continue moving the cursor or to pause it. “Current” essential
acts as the master flagging system. This variable retains the original value during function callbacks from plotting a point or finding the area under the plot.

Finding the integral of the drawn lines uses the mathematical powers of MATLAB. When a line is drawn the equation for the line is calculated and converted to a string for display on the figure is accomplished by taking the points at both ends and creating a 3d vector. MATLAB allows the conversion from a string to a symbolic equation, so by taking this string and converting it into a symbolic equation and substituting \( mx+b \) for \( y \) and since the equation is in terms of \( z \), there is no need to substitute for \( z \), the integral equation can ignore it.

The major difficulties in writing this program were optimization and display. The main program is executed through a simple while loop, while, very inefficient proved more efficient than recursion or other common alternatives. The program still begins to hang after about 20 or so plots, but by putting in artificial pauses of one tenth of a second the visible speed of the program is appears unchanged, and it helps prevent the appearance of hang time. Making the display correct was another big issue. The figure object doesn’t allow for basic text formatting like the console does. The equations were all very verbose outputs, at least 4 decimals, the mantissa, and its power in scientific notation. MATLAB lacks an inherent way to change this output format, so manual formatting had to be done. This problem however is minor compared to losing the path trail after moving along the z-axis. I was unable to find a solution for this as it seemed to be regraphing the lines over the points and the path would no longer be displayed. Even though this problem should be within MATLAB’s ability, a solution could not be found.
When I undertook this project, I expected there to be a lot more backend work manipulating MATLAB, but it turned out that MATLAB had bigger difficulties with proper display then actually doing the math and scripting required. I enjoyed learning and using some of the more mathematical powers of MATLAB, however it is very limited in term of power. MATLAB can only handle single variable calculus, which luckily was enough for this project, but I could see it underperforming in this area.