Programming Project 1 Report

Luke Brandon
010817623

Problem Statement:

The goals of this programming assignment are to write 2 programs. The first one generates graphics commands that will draw the graph based on the user input and also the integer inputs. The second program will read these commands and draw the 4 different types of graphs in a window using OpenGL (dot plot, bar graph, line graph, and a filled in line graph).

The inputs of the program are both user input to select which type of graph to display and also given input from the project definition which are the values that will be displayed. The given values are {2 3 5 8 13 21 34 42 42 42}. The input could also be random integer values between 1 and 50. The output should display a graph which is either a dot plot, bar graph, line graph, or a filled in line graph depending on which graph the user wants to view. There was not any error handling that was required for this project.

Design:

The 2 programs were divided into 2 separate .cpp files where the first one generated the commands to draw the 4 types of graphs from the data and the second program read the commands and used them to draw the graphs using OpenGL. The data structure I used to store commands was a pair of a string and a vector of integers, which looks like pair<string, vector<int>>. The only algorithm that was used was the linear interpolation algorithm which was used to draw the points on the graph relative to the size of the graph's draw-able area. There is a 50 pixel border on the left and bottom sides of the window for the x and y axis, which left the draw-able area to 450 pixels x 450 pixels which is taken into account in the linear interpolation algorithm.

The pro of the data structure that I used is that it makes handling the information much simpler and easy to parse, the linear interpolation algorithm makes the graph drawn the scale which makes the graph actually make sense as opposed to not being drawn to scale. The con of using the data structure above is that is has to be converted to a string to print to the commands.txt file and also has to be recreated from a string when reading from the commands.txt file which makes the code longer and more complicated but the use of the data structure is more robust and better for cleaner code overall.

Implementation:

When implementing both of the programs, I use a simple process that looked like writing a bit of code, testing it to make sure nothing is broken, writing more code, and repeating. I did not implement more than 1 feature or method between tests to keep debugging simple and to make the development process easier. I started with the sample code for Square.cpp from Dr. Gauch's website, which helped me by implementing all of the boiler plate OpenGL initialization that has to be done in order to draw to the screen. I adapted this code by adding all of the relevant features such as parsing and generating the

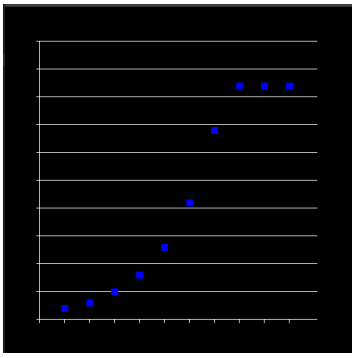
commands, displaying the graphs, drawing the shapes on the screen, etc. The starting code did not do any of the function that was necessary for this project, simple getting OpenGL set up which left all of the development for me to do. The entire development process took about 2-3 hours including developing, testing, fixing, and revising the code that was written at each step of the way.

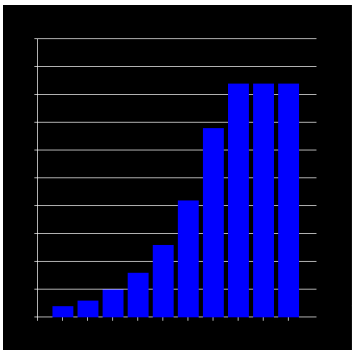
Testing:

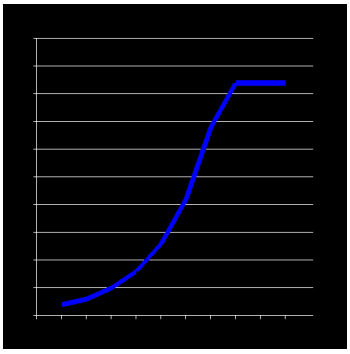
I tested the programs by running them with the example input provided {2 3 5 8 13 21 34 42 42 42}. The input used during testing was a number of integer values which needed to be graphed in the graphs that we were tasked with creating. No specific special cases were tested as that was unnecessary since we were tasked with graphing the given input and not any further, user-defined input meaning that the input can always be expected to be valid

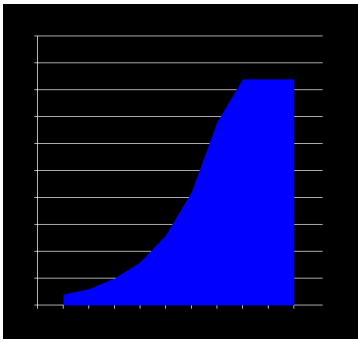
The program works as expected and does all 4 of the graph types correctly, those being dot plot, bar graph, line graph and, a filled in line graph.

These 4 graphs were made from the example input.









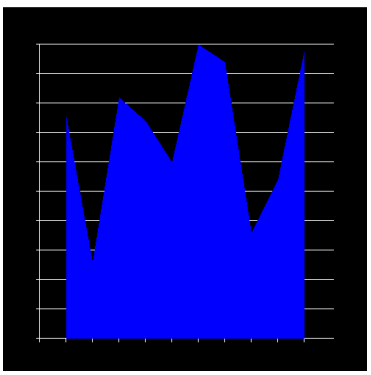
```
set_color 0 0 1
draw_point 10 86 66
draw_point 10 122 74
draw_point 10 158 90
draw_point 10 194 114
draw_point 10 230 154
draw_point 10 266 218
draw_point 10 302 322
draw_point 10 338 386
draw_point 10 374 386
draw_point 10 410 386
```

```
set_color 0 0 1
draw_polygon 71 66 101 66 101 50 71 50
draw_polygon 107 74 137 74 137 50 107 50
draw_polygon 143 90 173 90 173 50 143 50
draw_polygon 179 114 209 114 209 50 179 50
draw_polygon 215 154 245 154 245 50 215 50
draw_polygon 251 218 281 218 281 50 251 50
draw_polygon 287 322 317 322 317 50 287 50
draw_polygon 323 386 353 386 353 50 323 50
draw_polygon 359 386 389 386 389 50 359 50
draw_polygon 395 386 425 386 425 50 395 50
```

commands.txt

```
set_color 0 0 1
draw_line 10 86 66 122 74
draw_line 10 122 74 158 90
draw_line 10 158 90 194 114
draw_line 10 194 114 230 154
draw_line 10 230 154 266 218
draw_line 10 266 218 302 322
draw_line 10 302 322 338 386
draw_line 10 338 386 374 386
draw_line 10 374 386 410 386
```

```
set_color 0 0 1
draw_polygon 86 50 86 66 122 74 122 50
draw_polygon 122 50 122 74 158 90 158 50
draw_polygon 158 50 158 90 194 114 194 50
draw_polygon 194 50 194 114 230 154 230 50
draw_polygon 230 50 230 154 266 218 266 50
draw_polygon 266 50 266 218 302 322 302 50
draw_polygon 302 50 302 322 338 386 338 50
draw_polygon 338 50 338 386 374 386 374 50
draw_polygon 374 50 374 386 410 386 410 50
```



The above graph was made from the following input: 38 13 41 37 30 50 47 18 27 49

Conclusions:

The overall result of the assignment was a fully functional graphing program that graphs the 4 type of graphs that were described before and also a separate program that reads in the input and generates all of the necessary commands in order to graph those 4 types of the graphs. The type of graph is up to the user to decide. The programming project was a total success. If I were to do this project differently, I would not print out the commands to a separate file and then read them in from that file in the graphing program as this is not a very robust way of generating and reading commands and is very error-prone and unnecessarily complicated. Despite changing that, I would keep everything else the same including the data structure to store the commands, the linear interpolation algorithm, etc. In total this project took 2-3 hours to complete from start to finish.