The Development of a Multidimensional plotting program

Max Fyall
ID: 180011724
AC40001 Honours Project
BSc (Hons) Computing Science
University of Dundee, 2021
Supervisor: Dr. Iain Martin

Abstract – *The goal of this project was to investigate the possibility of creating a plotting program using OpenGL and C+. The reasoning for this being to find out if the 3D plotting experience could be improved by the use of advanced 3D graphics techniques.*
The researcher adopted the agile framework to provide a flexible and controlled development process throughout. The program was created using a number of different libraries used in both C++ and OpenGL. These libraries contributed to the overall functionality of the application, sometimes providing functionality that could not be found elsewhere. The program adopted a class structure whereby classes we used to create, draw, and destroy objects when necessary. This gave the program a cleaner, more professional look to it.
Due to the findings of the project, the researcher can say overall that the 3D plotting experience has the potential to be enhanced with the implementation of advance 3D graphics techniques such as lighting, shadows etc.

1 Introduction

Plotting programs (Information Graphics Software) are used for visualising data in an intuitive manner. Data visualisation is a critical process in the understanding of large complex data sets and conveying information to people in an intuitive way [1]. There are various software applications that provide tools for data visualisation, a few examples are, GNU-Plot, Tableau, MATLAB etc. Each application has their own style (i.e. GUI interface, Command Line interface), but they all provide some form of data visualisation [2]. The majority of software predominantly uses graphs and charts 2-dimensionally (2-D) . A simple and effect way to convey given data sets, this is the most common style for graph plotting.
Not so commonly used in better-known software is 3-dimensional (3-D) plotting. Similar to 2-D plotting with the only difference being the addition of another dimension (depth). This dimension can be used to great effect when plotting in 3-D. Different perspectives and a better understanding of data (and subsequently much better analytical results) can be gained through correct use of 3-D plots. However, in some cases, they are often overshadowed in favour of their 2-D counterparts. Why is this the case? The short answer being the advantages a 3-D plot gives, mostly, does not warrant using it over a 2-D plot. They are not helped by the fact that they are more complex than their 2-D counterparts as well. If this is the case, what can we do to improve the 3-D plotting experience? One method of developing 3-D figures/models is to use modern 3-D graphics software. This technology has the ability to harness the power of the Graphics Processing Unit (GPU) to draw anything to a window. It is with this in mind that the researcher aims to investigate if a multidimensional plotting program can be created using 3-D graphics technology. These findings will help in proposing if the 3-D plotting experience could be enhanced with advanced graphical techniques.

2 Background

As mentioned previously, there are multiple pieces applications geared towards data visualisation. Each piece of software has their own niche with specific users since each user will have their own set of requirements. For example, Tableau is design towards a Data analyst and Statisticians whereas something like Matplotlib is geared towards Computer programmers since it's a library for the programming language python. The purpose of this section is to look at existing plotting software, examining how they work to understand the best approach when designing a program of this nature.

2.1 Existing Software

2.1.1 MATLAB

Modern graph plotters incorporate both 2-D and 3-D plotting tools. For the purposes of the research, the tools for 3-D plotting will be investigated to understand the features that are present for 3-D plotting. MATLAB in one example that incorporates 3-D plotting very well. MATLAB itself is a programming language used inside the MATLAB software. Using the MATLAB language it is possible to plot functions and data in both 2-D and 3-D [3].

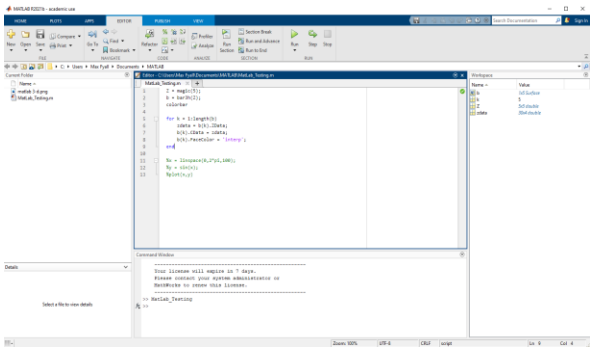

Figure 1: MATLAB GUI

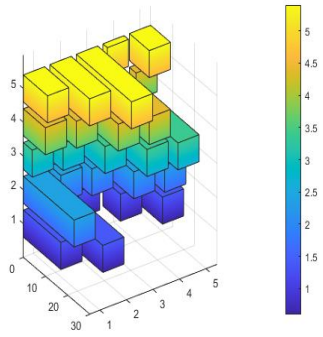

Figure 2: 3-D Plot using MATLAB

Upon clicking "run" with valid plotting code (See Appendix A), a window will appear with the completed plot as shown in Figure 2. From here the user can use the mouse to navigate around the graph to gain a better perspective of all aspects of the graph. This is very intuitive and is necessary when viewing certain parts of the graph as seen in Figure 2. It is something that has been taken on as inspiration when planning the prototype due to the intuitive nature of it and the benefits it brings to viewing graphs. Whilst this is all great, there is a problem. The software relies on the user having some form of coding knowledge. Having little coding knowledge will likely see most users searching online for how to acquire and accomplish tasks [4]. This is fairly easy to do since there are multiple forums regarding MATLAB. However, this could turn some users away since they could be looking for an easy intuitive system that they don't have gradually to get the hang of. With this in mind, one goal of the project is to make the design/functionality intuitive for most users to use. This allows for quicker access to the things end users want the most… results.

2.1.2 GNUPLOT

Another common plotting program mentioned when discussing this particular software is GNU-Plot. Inside the operating system Linux, GNU-Plot incorporates a mixture of command-line and GUI interfaces (See Figure 3 & 4). Meaning you insert commands via a terminal and your resulting graph is displayed within a separate window using "Qt" [5]. It functions very similarly to the MATLAB software in that the software interprets an input and displays a result based on this input. The resulting window also shares the mouse control feature found in the MATLAB software as well as looking somewhat similar (See Figure 4).

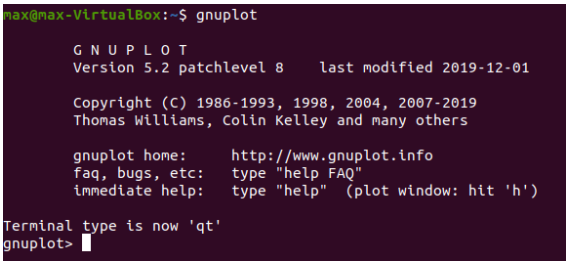

Figure 3: GNU-Plot Terminal Interface

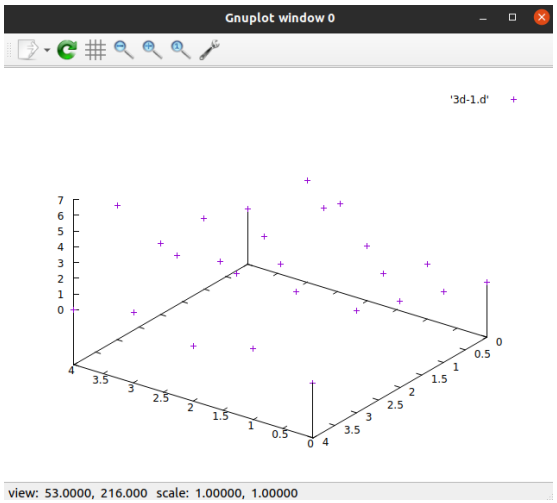

Figure 4: GNU-Plot 3-D Scatter Plot

GNU-Plot is a great example of command interpreted plotting. It would be a good basis if the project was to be script/command based in a terminal. It is fairly simple to plot from data files as long as you know where your file is store since this could cause file-finding issues. The main problem with GNU-Plot is the overall complexity for plots. A lot of time spent using GNU-Plot is usually looking online at the documentation to find a solution. As we saw with MATLAB, this not ideal for most users. This issue is not helped by the fact that GNU-Plot is mainly used in a Unix environment (i.e. Linux distributions). GNU-Plot comes with most Linux distributions by default, this can be handy for any Unix users since there is no install required. However, this does mean if you are on Windows you need to install a Unix environment (e.g. MinGW, WSL) just to get it running which not a lot of novice users would not be able to achieve. The plots themselves are also not the best looking 3-D plots (See Figure 5). Compared to MATLAB, the plots are considerably less in terms of graphical quality. Using the 3-D graphics libraries and the processing power of the GPU, this will be an area that the project aims to show can be improved.

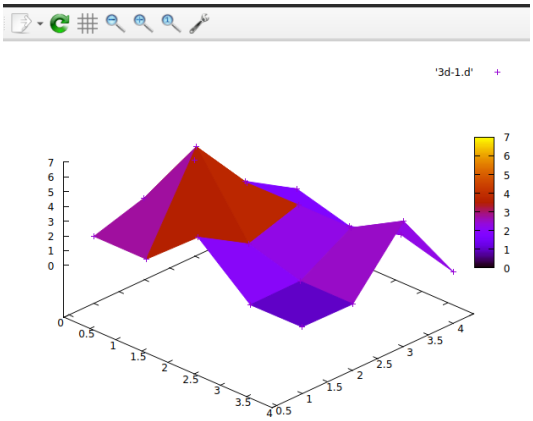

Figure 5: GNU-Plot 3-D Scatter Plot with fancy feature

2.1.3 ONLINE PLOTTERS

When researching similar applications, online website plotters are a frequent occurrence when searching. Many of these applications function in similar manors, for this instance the researcher will be looking at GeoGebra 3D Calculator [6]. The selling point of this is as a mathematical function plotter that has the capabilities to show off 3D mathematical functions.

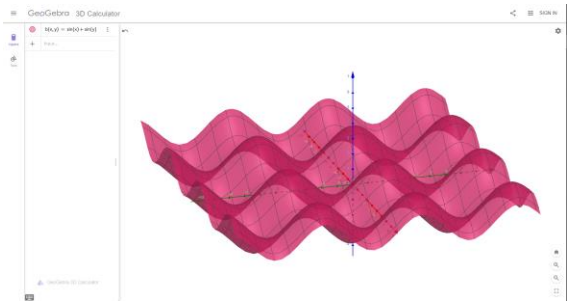

Figure 6: Function f(x) = sin(x) + sin(y)

In Figure 6 we can see this in practice, the function sin(x) plus sin(y). Assuming the user has some mathematical knowledge, this program is very simple to use thanks to its intuitive controls (click and drag mouse control) and graphical user interface (GUI). Plotting is as simple as inserting a function and seeing the results. With this being a key focus of the research project, this design was thought to be a good direction for the design of the project (See Appendix B). A simple and quick way to get a user's data from raw data to the screen. The philosophy is still the same, regardless of what "data" you are trying to display on screen i.e. mathematical function or raw data.

2.2 Summary

To summarize, there are many different applications that provide users with plotting capabilities. By experiencing the strengths and weaknesses of applications, it provided the researcher with a much better understanding of how to approach the creation of a program of this nature. What needed to be considered for the benefit of the user. How the program needed to control and feel for it to replicate a plotting application.

3 Specification

The researcher and project client agreed to an initial meeting in the early stages of the project (See Appendix C). This was where both parties discussed what kind of direction the project could go (See Appendix D). This was the first on many meetings between parties that would occur on a weekly basis (not including holiday periods). The purpose of the meetings being to let the client know how much progress has been achieved since the last meeting, sharing ideas, and answering any questions (See Appendix D). Due to the nature and description of the project, there was a mutual understanding between both parties that the researcher was going to attempt to create a plotting program using 3D graphics programming. The researcher had the most experience programming OpenGL applications using Visual Studio and C++ (and linking the necessary libraries e.g. GLFW, GLEW and glload (See Appendix E)). Therefore, the researcher decided this was how the application would be created when it came to development. The researcher could have chosen other languages such as Python [7] and JavaScript [8] (OpenGL has compatibility for both). However, due to the inexperience the researcher had with both languages, they decided it would be best to stick to what they knew. In order for the researcher to understand the characteristics of a plotting program, it was thought that the researcher should look into the existing software market (As seen in Section 2.1 "Existing Software"). This would be a great chance to gather requirements for the system, understanding how plotting applications work to make the system more intuitive for users. When gathering the requirements for the system, the researcher had to decide which development cycle would suit the project development. The researcher believed that the agile development cycle [9] would be the best approach for the project.

3.1 Agile Approach

The agile development cycle allows for frequent change and minimal planning. The researcher felt this would be useful if a change of thought went into a major part of the development and plans need to change. This could be the case after a sprint has finished. The researcher would create user stories when they were researching the existing software market. The user stories would be created in the form of GitHub issues (See Appendix F) so they could be used later in the agile planning phase. The stories were create using the typical format "As a user I can… so that…". From looking at the existing technologies, the researcher managed to establish a number of user stories that they thought would be necessary for creating a plotting program. One example would be "As a user, I can move around the scene so that I can gain a better understanding of

the scene" (See Appendix F & Figure 7). This was based on a universal feature present in plotting programs, the ability to rotate and move the graphs around.

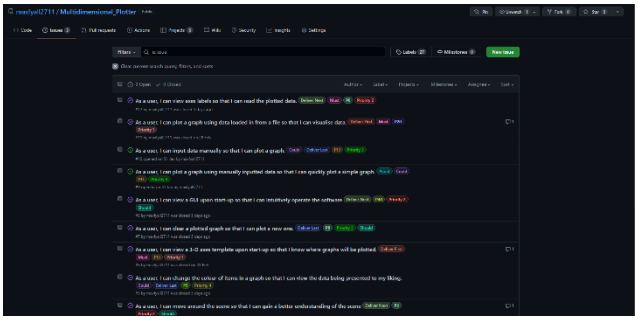

Figure 7: GitHub Issues Page form Researchers Repo

Once the researcher had felt they gathered enough user stories for the system, they moved on to the next stage of agile planning. Prioritising the user stories using 4 methods of prioritisation; MoSCoW, Planning Poker, Risk/Value and Priority Assessment. The GitHub projects page (See Appendix G & Figure 8) was very useful for the researcher to undertake the prioritisation tasks.

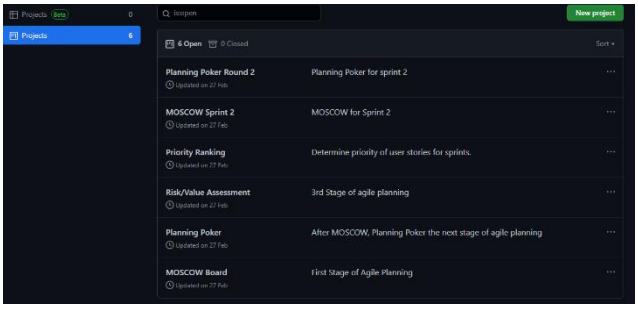

Figure 8: GitHub Projects Page from Researchers Repo

It allowed all the tasks to be carried out in one location and the user stories could be easily transferred over from the issues page. The researcher had created both a sprint and product backlog for tracking the progress of the planned sprints (See Appendix H & I). User stories were subsequently added to both when the researcher was planning each agile sprint (The second sprint being planned after the first one was completed). These would last 3 weeks each as per the Gantt Chart the researcher had create as part of a mid-term report (See Appendix J). After a sprint had finished, a sprint review/retrospective would take place to learn and understand how the next sprint could be improved (See Appendix K). A sprint burndown chart was added to each sprint backlog to help the researcher recognise how much effort was being put into the work. It gave the researcher a good understanding of their sprint velocity.

3.2 Approach Summary

Due to a lack of work in the initial stages, the researcher understood how critical it was to begin planning out the project development (See Appendix D). The client would often reiterate this point to the researcher. The working schedule for the researcher was from ~12pm – 5pm with an hour to 2-hour break and then mild working from 6pm – ~9pm. Sprint and product backlogs (See Appendix K) gave the researcher an understanding of what was going to happen in the up-coming sprint. Allowing the researcher to begin design and development as soon as the backlogs were finished.

4 Design

4.1 Designing an OpenGL program

When creating an OpenGL program, there are a few essentials for creating a program that can run successfully with a simple implementation.

1) Libraries
- In the researcher's case this includes GLFW [10] (Windowing library) and GLEW/"glload" (function loading libraries). These need to be initialise for use. These libraries are also dependant on which language is being used to program with. For example if C++ is being used, the program must used C++ versions of the libraries.

2) Application Code (C++ source code)
- A C++ source file where the function calls with take place (i.e. where the "main" function will be). The researcher choses C++ due to having more experience using this language.

3) Shaders (to load in)
- A vertex shader and a fragment (or pixel) shader. Written using GLSL (OpenGL Shading Language), the shaders are a part of the graphics pipeline and functions required to draw anything on the screen [11]. The shaders can either be loaded in a one long string or as a shader file (.vert for vertex and .frag for fragment) (See Appendix L).

With the correct applications of the above, everything should be present for writing an OpenGL program.

4.2 Creating Classes

It is possible to create an application using a singular source code file. However, this can be messy, and the researcher believed this was not very good program design. When writing a large OpenGL program (with lots of functions and implementations), using a single source file to handle everything could lead the programmer to become confused and lost in the vast lines of source code. This could lead to unforced mistakes and longer debugging times. To avoid this, the researcher used and created classes which could be used inside the application code. This allowed code to be separated, making it easier to understand how certain code chunk's function.

4.2.1 GLFW Wrapper Class

This class was created by Dr Iain Martin for a course in graphics programming. It was further adapted by the researcher with further implementations. The purpose of this class is to initialise the GLFW window, functions and provide the event loop for the program [12]. It is in this class where the functions for loading and building shaders would be used. Shader programs are returned from these functions to be used in the application code.

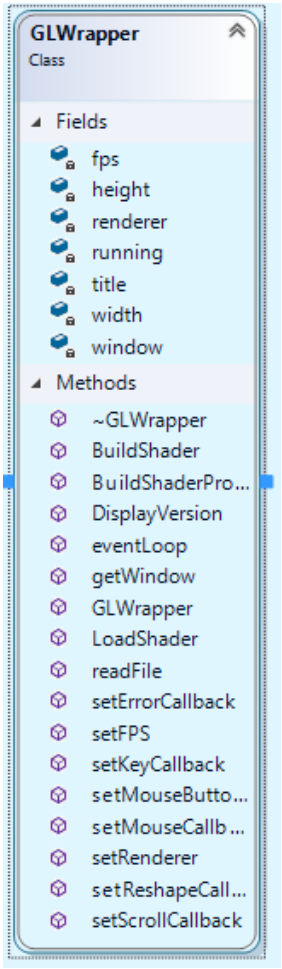

Figure 9: GLFW Wrapper Class (Included in Appendix M)

4.2.2 Object Classes

When creating multiple 3D and even 2D objects in OpenGL, it is often best practice to create the object using a class system rather than defining an object in the application code. Defining methods for creating the object and drawing it. This, as said before, can clear up the application code by moving big code chunks into its own defined class.

The 3D axis object (along with others) was one such object that was created using a class structure.

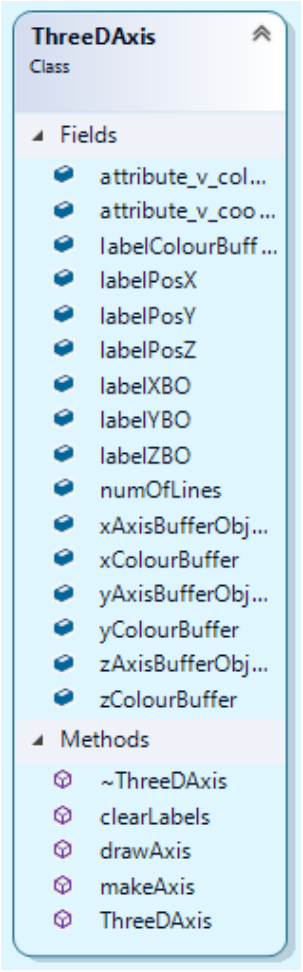

Figure 10: 3D Axis Class (Included in Appendix M)

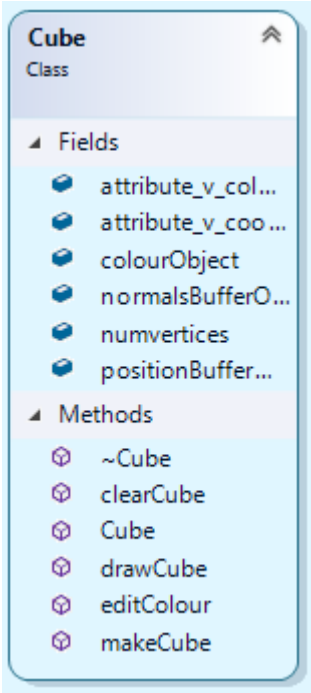

Figure 11: Cube Class (Included in Appendix M)

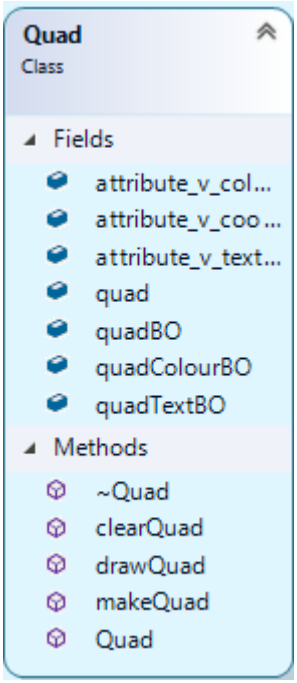

Figure 12: Quad Class (Included in Appendix M)

Using this structure, the researcher could call pre-defined methods and variables to create the object, so long as they create an instance of it inside the application code. This helped clean up the application code and prevented the re-use of certain code chunks. This also kept any object specific functions inside their respective classes. For example, the axis class required creation and deletion of lines (notches) on the axis's lines.

4.3 Shaders

As previously mentioned, shaders (both fragment and vertex) can be loaded in as one string. However, this is never a good practice, and it is easy enough to create a shader file for each shader type. The researcher thought as much also and created shader files for each shader they used. Creating shader files also makes the programming easier when multiple shaders are involved. The researcher believed it would be much simpler to differentiate between the 2 shader programs if two sperate file groups were created. Unbeknownst to the researcher this would later become a reality due to the sudden requirement of more than one shader program. The researcher was also keen on having the shaders be present in their own directory, as appose to being in the same directory as the source code. The researcher did this to keep the shaders all in one place and avoid mixing files around that are in the same directory.

4.4 Program Interface

The researcher had gathered a lot of information regarding plotting applications when gathering requirements for the system. It was during this process that they gained an idea of how they wanted the system to look. The researcher wanted to have a 3D axis be the first thing to user sees when the program launches. The user would then use a GUI to plot/interact with the program. Due to the researchers need to have the program be intuitive for users to use, mouse controls would also need to be considered for moving the scene.

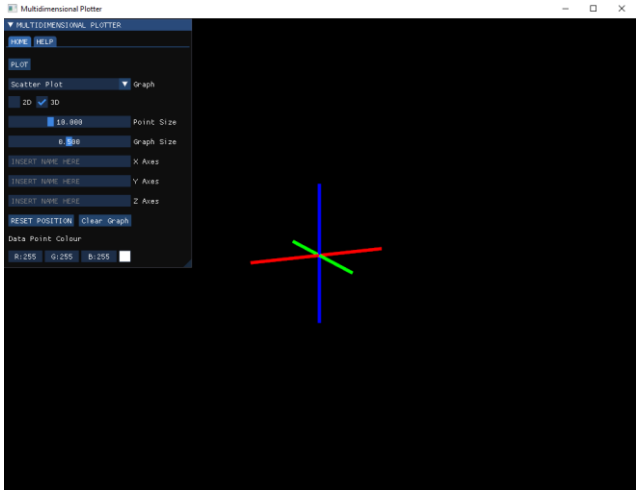

Figure 13: Program Startup with slight mouse movement

Initially, the researcher wanted a feature whereby users could insert data manually via the GUI and use this data to plot a graph. However, it was decided later on that this feature be cancelled as it was felt to be redundant with the system already in place i.e. loading data in from a text file.

5 Implementation and Testing

5.1 Setting up build environment

During the planning phase, the researcher began setting up project inside Visual Studio 2019 that they would use as their development environment for the duration of the project's development. Because the set-up was fairly light, the researcher could manage this whilst they were gathering requirements and planning the first sprint. The set-up was comprised of:

1. Creating the VS Solution
2. Adding/Creating Necessary Files
3. Linking the Libraries Dynamically (For now)
4. Linking Include Directories

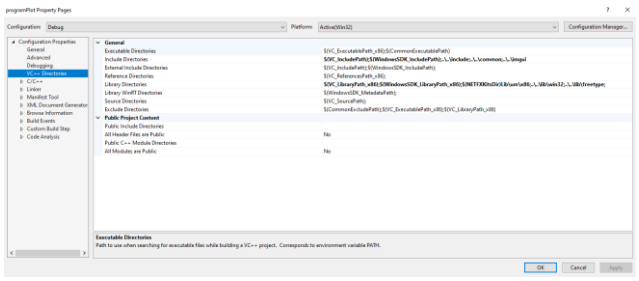

Figure 14: Linking Include and Library Directories (Included as Appendix N)

5.1.1 Setting up GitHub repository

Due to the scale of this project, the researcher wanted to use GitHub as version control for project. This was the first time, for the researcher, setting up a Visual Studio project inside of a GitHub repository. It was because of this lack of experience that would cause some minor setbacks when the researcher was attempting to push commits. Visual Studio can create large files in projects. Most of these files can be removed so that is not an issue. The problems can occur when trying to push a commit with these large files. Git can only push a file no larger than 50 MB [13]. Attempting to push a file over the file size limit will result in a push error. It was this that the researcher was not expecting (See Appendix D), however they did manage to find a fix for this. They would add the larger files to the ".gitignore" file, meaning files inside would be ignored when committing and pushing files.

5.2 Sprint 1

Once the researcher had completed the necessary planning, the first development sprint began on Jan 31st, 2022, and would last for 3 weeks. The researcher's goal for this sprint was to have a basic prototype working by the conclusion of the sprint. To achieve this goal, the researcher had planned the sprint accordingly with the goal in mind (See Appendix I).

5.2.1 Creating the 3D Axis

As said in the "Design" section, the axis is created using a class structure. The functions present in the class are used for creating the axis object using hard-coded vertices and colours. Vertex buffer objects (VBO) [14] would then be created for each axis and colour. The VBO would then be bound when it was time to draw, and each line would be drawn using "glDrawArrays" with GL_LINES as the argument.

```
/* Bind X Axes vertices. Note that this is in attribute index attribute_v_coord */
glBindBuffer(GL_ARRAY_BUFFER, xAxisBufferObject);
glEnableVertexAttribArray(attribute_v_coord);
glVertexAttribPointer(attribute_v_coord, 3, GL_FLOAT, GL_FALSE, 0, 0);

/* Bind X Axes Colours. Note that this is in attribute index attribute_v_colours */
glBindBuffer(GL_ARRAY_BUFFER, xColourBuffer);
glEnableVertexAttribArray(attribute_v_colours);
glVertexAttribPointer(attribute_v_colours, 4, GL_FLOAT, GL_FALSE, 0, 0);

// draw vertex positions
glDrawArrays(GL_LINES, 0, 2);
```

Figure 15: Bind VBO's and drawing the X axis (See Appendix W)

Note that we enable the vertex attribute array (VAA) for both the colour and vertex positions. This is defining the index of the position and colour attribute inside the VAA. In most cases this is an integer value, in this case represent by a constant integer defined in the class constructor. The VAA is used in both shader programs to identify the vertex data (position, colour, normal etc.) at a given vertex or pixel.

Initially, the axis was drawn at origin with axis length of 2, since the line was drawn from -1 to 1 depending on the axis direction e.g. x, y, or z. If data points were to be drawn, they would not line up with the axis because the data point's coordinates would be out of range of the 3D axis. To get around this problem, the researcher made the axis length change dynamically. The axis is created using the highest value from the data set being read in plus 1 (in case highest was 0). For example, if the highest value from the data was 8, the axis would be drawn using -8 to 8. This largest value was passed into the "makeAxis" function during runtime.

```
GLfloat xAxisVertex[]
{
        (boundaries + 1), 0.0f, 0.0f,
        -(boundaries + 1), 0.0f, 0.0f
};
```

Figure 16: Creating vertex positions for x axis (See Appendix W)

This allowed the axis to line up with any data points that are loaded in. Giving the correct position of every data point.

5.2.2 Plotting points

The researcher started by implementing one graph type for the basic prototype (Scatter Plot). That way they could get other essential features implemented.

To draw points in OpenGL, we first need an array like structure to fill with vertex positions to draw. To start, the researcher aimed to use arrays to fulfill the task. Arrays are great for storing static vertex positions i.e. the number of vertex positions never changes. However, in C++, it is not possible to declare an array of an unknown size. The compiler must know at runtime the size of the array. The researcher found that they could use a dynamic array using a global variable to determine the size of the array. This variable would be calculated as the number of vertex positions need to be stored.

```
GLfloat* plotPositions = new GLfloat[x];
```

Figure 17: Creation of dynamic array

This way the array can be used as the researcher intended. Although, the researcher was still not happy with this implementation. The main reason for the researcher using an array was because they knew how to generate VBOs from them. So, after discussing with the client (See Appendix D), the researcher understood that it was possible to use the C++ vector for this implementation. This would eliminate any unnecessary calculations that were previously required when using a dynamic array.

```
#include <vector>

std::vector<float> vertexPos; // stores vertex positions (read in from file)

/* Create Point Buffer for data points */
glGenBuffers(1, &plotBufferObject);
glBindBuffer(GL_ARRAY_BUFFER, plotBufferObject);
glBufferData(GL_ARRAY_BUFFER, vertexPos.size() * sizeof(float), &(vertexPos[0]),
GL_DYNAMIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Figure 18: Creation of VBO with Vector Implementation (See Appendix W)

With this knowledge the researcher decided that any data structure that need to be dynamic (we don't know how many indices the structure will have) they would use a vector to store it and arrays for anything static (we know the size of the structure).

When using the vector implementation, the researcher found an interesting problem. Sometimes, when launching, the program would give an "vector out of range" error.

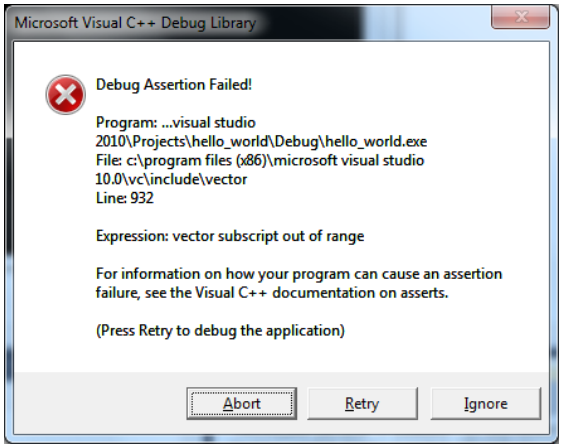

Figure 19: Visual Studio Vector Error Example

Upon further inspection, the researcher found that this error was being triggered during the creation of the VBO (as seen above). The reason for this was because the vector was empty, there was nothing in it even not a 0 value. The researcher then understood that it is not possible to have an empty vector during the creation of a VBO using this vector. To get around this, the researcher would always add one integer number (most likely a 0) to avoid this happening.

```
// create empty vector to pass
std::vector<float> empty;
empty.push_back(0); // insert 0 to prevent memory exceptions
filestream.close(); // close the filestream
return empty; // return the empty vector
```

Figure 20: Example of creating "empty" vector

In the above example, the vector will be empty, so we use the vector function to add a 0 to the end of the vector. When it came to populate the vector, we would clear it in case we had to add an integer value to it. This system of adding a 0 would work if the VBO was recreated once the vector had been fully populated.

```
glDrawArrays(GL_POINTS, 0, size); // draw data as points in 2-D/3-D space
```

Figure 21: Drawing data points (See Appendix W)

As seen above, the GL_POINTS argument was then used to draw the data points to the screen. This makes a nice-looking scatter plot.

5.2.3 Mouse controls

One of the main universal features across all plotting programs is rotation control with the mouse. It was one of the first requirements the researcher had wrote down.

To allow objects within a scene to move, matrix transformations need to occur for this to happen. There are 3 types of transformations that can be applied to a scene. Rotation, Scale and Translate. Transformations require matrix multiplication to work, each transformation has a different. "ruleset" for apply the said transformation.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 22: Rotation Matrices applied during rotate

For instance, the rotation transformation uses Figure 16 in a matrix multiplication to apply the rotation transformation to the scene/object. Transformations are applied to the identity matrix and subsequent transformations are performed on the transformed matrix until all have been complete for the scene/object.

```
/* SCENE ROTATIONS */
model.top() = glm::rotate(model.top(), glm::radians(yaw), glm::vec3(1, 0.0f, 0.0f));
// mouse rotation X
model.top() = glm::rotate(model.top(), glm::radians(pitch), glm::vec3(0.0f, 1.0f,
0.0f)); // mouse rotation Y
```

Figure 23: Rotation Transformations using GLM math library (See Appendix W)

This transformed matrix is then sent to the vertex shader as a uniform variable [15] where it is used to calculate the position on the screen. Other uniform matrix variables, such as view and projection, are also passed to the vertex shader to calculate where the scene will be using the model, view, and projection calculation.

```
// send model uniform matrix to the vertex shader with applied transformations
glUniformMatrix4fv(modelID1, 1, GL_FALSE, &model.top()[0][0]);

// send the uniform matricies to the vertex shader (used in model-view-projection
calculation)
glUniformMatrix4fv(viewID1, 1, GL_FALSE, &view[0][0]);
glUniformMatrix4fv(projectionID1, 1, GL_FALSE, &projection3D[0][0]);
```

Figure 24: Sending uniform matrices to vertex shader (See Appendix W)

The researcher aimed to use transformations (mainly rotations) and find out a way to control them with mouse inputs. Having investigated this further, the researcher found a tutorial [16] for controlling transformations using mouse control and GLFW.

GLFW allows for callback functions to be set during runtime. These functions can be set to any function if the functions fulfil the required arguments. This is also how the researcher sets the renderer for the event loop.

```
// setting glfw callbacks and functions to be used in the wrapper class
glw->setRenderer(display);
glw->setKeyCallback(keyCallback);
glw->setMouseCallback(mouseCallback);
glw->setMouseButtonCallback(mouseButonCallback);
glw->setScrollCallback(scrollCallback);
glw->setReshapeCallback(reshape);
glw->DisplayVersion();
```

Figure 25: Setting function callbacks using GLFW class instance glw (See Appendix W)

By setting a mouse callback function, we can use this function to get the mouse inputs and subsequently edit a variable to rotate the scene.

```
static void mouseCallback(GLFWwindow* window, double xposIn, double yposIn)
{
	// if bool is true (changed by pressing Mouse 3/2)
	if (moveScene)
	{
		// set position variables
		float xpos = static_cast<float>(xposIn);
		float ypos = static_cast<float>(yposIn);

		// if this is first time moving
		if (firstMouse)
		{
			// set position variables
			lastX = xpos;
			lastY = ypos;

			// change bool since first mouse is not needed
			firstMouse = false;
		}

		// set offsets
		double xoffset = xpos - lastX;
		double yoffset = lastY - ypos;

		// store position data
		lastX = xpos;
		lastY = ypos;

		// change pitch and yaw to rotate the camera
		pitch += xoffset * 0.1f;
		yaw += yoffset * 0.1f;

	}

}
```

Figure 26: Mouse input function (See Appendix W)

The above algorithm demonstrates the calculation required to work out the angle to rotate the scene by according to the mouse input. By using the yaw and pitch variable inside the rotation transformation (as seen previously), the user can now rotate the scene on the X and Y axis using the mouse to control the transformation.

5.2.4 Sprint 1 Testing/Review

5.2.4.1 Testing

At the end of the first sprint, the researcher conducted self-testing on the current implementations (See Appendix O). The basis for this was to understand if there were any problems in the current build that need to be fixed.

Test	Input Data	Result
Mouse Control	Move the mouse to rotate the scene.	Rotations correspond with input from mouse. Mouse up rotates on Y axes and Mouse to the side rotates on X axes.

Figure 27: Testing format with example

Each test was recorded in a document with the input data and the result that followed. The researcher was glad the testing occurred since it did manage to bring up some issues that were present within the current build. Most of the issues were very minor and could be fixed during the testing phase.

5.2.4.2 Sprint review

As per the specification (Section 3), the researcher would conduct sprint reviews (See Appendix K) at the conclusion of each development sprint. It was here that the researcher could gain an understanding of what had been completed and what actions should be taken for the next development sprint. The researcher had found that the expected velocity had not been achieved and there were plenty of improvements that could be made for the next sprint. One example of this was to do with the sprint backlog (See Appendix I). The user stories were put into the backlog and were not broken up into smaller user stories. This made it harder for the researcher to understand what was still yet to be done with that specific story. Therefore, it was decided that for the next sprint user stories would be broken up into smaller tasks. This did end up helping the researcher as they found it a lot easier to work out what may require more development time than others.

5.3 Sprint 2

Planning for the second sprint commenced once the first sprint had been wrapped up and the reviews and testing had been completed. Considerations from the last sprint were considered when planning the sprint. These included, re-evaluation of user stories, introduction of new user stories, actions suggested from sprint review etc. The second development sprint began Feb 28th 2022 and was planned to last for 3 weeks. The true finishing date for this sprint turned out to be Apr 6th 2022, which was 2 weeks and 2 days over the estimated duration. The researcher believes this was due to the underestimation of specific user stories and a general lack of effort. However, the researcher still completed, what was thought to be, a very difficult workload. It serves as a lesson to the researcher for what to avoid when developing software. The aim of the second sprint was to make the application suitable for users to be able to test it. Making the application into more of a finished product.

5.3.1 Adding more graph types

At the beginning of the second sprint, there was only one type of graph available to plot (Scatter plot). This was intentional as it was the simplest graph to construct. The researcher had planned to use a couple of different graph types to show of the capabilities of OpenGL. The other plotting considerations were Line graph and Bar Chart.

5.3.1.1 Line Graph

A line graph is a series of points joined together with a line. Since the application already featured a point plotting system in the form of the scatter plot, the quickest and simplest way to change the graph from a scatter to a line is to connect the points. This is very simple to achieve in OpenGL as it only requires a change of drawing argument.

```
// check if graph type is scatter plot
if (graphType == 0)
{
	glDrawArrays(GL_POINTS, 0, size); // draw data as points in 2-D/3-D space
}
else if (graphType == 1) // check if graph type is line graph
{
	if (!drawmode) // drawmode represents how the line graph is drawn
	{
		glDrawArrays(GL_LINE_STRIP, 0, size); // use drawing mode LINE_STRIP
draw the points as a line
	}
	else
	{
		glDrawArrays(GL_LINE_LOOP, 0, size); // use drawing mode LINE_LOOP draw
the points as interconnected line
	}
}
```

Figure 28: Changing drawing argument with condition (See Appendix W)

In Figure 28, a condition is used to check the graph type (i.e. the graph the user selected) which is an integer value. When the user has selected "line graph" the drawing argument will be change to "GL_LINE_STRIP" which connects the data points with a line. Another feature that is present in Figure 28 is Line Loop. This is a basic feature that allows the user to connect the final data point and the first one with a line. It is changed with a check box in the GUI. If the box is checked, then change the drawing argument to "GL_LINE_LOOP" and the feature is enabled.

5.3.1.2 Bar Chart

Bar charts are a bit more complicated in 3D. Each bar is represented by a cuboid with differing height to represent the data. This means that for every data point read in, there must be a cuboid drawn to the screen in the correct place. Meaning differing height and spaced apart like in a normal bar chart.
The researcher used a class to which would be capably of creating and drawing a cuboid. Each cube that is created has similar hard-coded vertices. Since a cuboid is made up of 12 triangle planes, very specific vertices must be modified to give the bar chart effect. Arguments passed into the "makeCube" function change the vertices of the cuboid depending on the argument's value.

```
-0.25f+moveX, (height), -0.25f+moveZ,
-0.25f+moveX, -0.25f, -0.25f+moveZ,
0.25f+moveX, -0.25f, -0.25f+moveZ,
```

Figure 29: Vertices for one of the cuboids triangles

As seen in Figure 29, there are 3 arguments that modify the cuboids vertices. Each argument determines the position the cuboid should be. The height argument is different depending on which triangle plane is being modified. In this case, only one vertex needs to be modified. To create the number of cuboids necessary, the researcher created a vector of cuboids.

```
std::vector<Cube> barChart;
```

Figure 30: Vector of Cuboids being created

This is the data structure the researcher used to store all the cuboids. When creating the cuboids for the bar chart, the researcher had to develop an algorithm to make each cuboid. The number of cuboids was dependent on the number of data points present in the data file that was read in. For example, if there was 6 data points there would be 6 cuboids. This would be calculated during the file reading process using a vector of integers.

```
// increase bar counter - (Number of values in the X direction)
numberofBars++;

// if number of bars is 0 nothing was in that line of the file
if (numberofBars != 0)
{
	// insert bar value into barchart vector
	barsX.insert(barsX.end(), numberofBars);
}
```

Figure 30: Bar chart calculations from read file function

During the reading process, the number of data points from each row would be calculated. This is represented by "numberofBars". Once the loop finishes the current iteration, this value would be added to the bar chart vector, telling the researcher how many data points are in that row. Once all the calculations have been complete, the size of the bar chart vector "barsX" indicates the number of rows i.e. the number of bars in the z direction. With this knowledge, the researcher could create the cuboids necessary to create a bar chart.

```
for (int j = 0; j < barsX[i]; j++)
{
	// insert a cube into the bar chart cube vector
	barChart.insert(barChart.end(), testCube);

	// make the cube inserted passing in correct parameters
	// vertexPos vector - data point read in from file to dictate the height of the
cube
	// moveX float - how much to move the cube in the x direction (i.e. moving the
cube to the next notch on x axes)
	// moveZ float - how much to move the cube in the z direction (i.e. are we on
the first row or the second row?)
	// addby float - specified number, how much to move the cubes by (This is
dependant on the axes configuration)
	// colour float array - the colour to pass to the colour buffer
	// NOTE - x and z positions are multiplied by 4 to give the desired effects in
the scene
	barChart[q].makeCube((vertexPos[q]), ((moveX + 1)) * 4, (-(moveZ + 1) * 4),
color);

	// increment by one to move to the next cube
	q++;

	// increment x indentation by addby variable to move along the x axes
	moveX = moveX + 1;

}
```

Figure 31: Loop to create cuboids (See Appendix W)

Figure 31 shows the main loop to create the cuboids. This loop will run for the value assigned in the bar chart integer vector. Meaning if there were 4 values in that row, the loop will make 4 cuboids in X direction. Each time a new cuboid is made it is inserted into the vector of cuboids and the X argument (moveX) is increased by 1 to move the next cuboid to the next position on the axis.
Outside of this loop is another for loop. This loop will run for the number of values in the Z direction. For example, if there are 6 rows of values (indicated by the size of the integer bar chart vector), the loop will run 6 times. Each time the loop in Figure 31 completes the Z argument (moveZ) is incremented by one to move to the next row of cuboids. To give the bar chart effect, we use the data structure used for holding the plotting data. Passing it to the "makeCube" function to determine the height of each cuboid. When all the cuboids have been made, drawing them all is as simple as looping through the vector of cuboids and calling the draw function built into the cuboid class.

```
// check if graph type is bar chart
if (graphType == 2)
{
// prohibit draw the chart if the cube vector is empty (will cause an exception
otherwise)
	if (!barChart.empty())
	{
	// loop for vector size (vector size is equilvalent to the number of bars in
the z direction)
		for (int i = 0; i < barsX.size(); i++)
		{
			// loop for integer item in barX vector (each item in this vector
is the number of bars in the X direction)
			for (int j = 0; j < barsX[i]; j++)
			{
				barChart[p].drawCube(0); // draw pre-made cube from cube
vector
				p++; // increment by one to move to the next cube
			}
		}
	}

}
```

Figure 32: Drawing created cuboids (See Appendix W)

5.3.2 GUI Implementation

An important feature the researcher wanted to have in the application was a GUI system. The researcher did not what to have a command line operated program since they were too complex and not ideal for beginner users. A GUI system is more user friendly but can also be challenging to incorporate. The researcher had no knowledge regarding GUI libraries for C++ and had never implemented a GUI in a C++ program. The researcher knew that they had to investigate for a solution. A library the researcher had peaked a large interest in was a library by the name of "Intermediate Mode GUI" or "ImGui" for short [17]. A GUI library that had integration with the GLFW windowing system. This made it ideal for the researcher and the project since these were the libraries being used and it was too late to switch windowing libraries. Having no knowledge of the library, the researcher followed a tutorial for installing the library [18]. The installation mostly included transferring the correct files for the project's use case (i.e. project using GLFW) and linking it to the include directories so Visual Studio knew where the files were.

```
// IMGUI Initialisations - initialise ImGUI for GLFW window
IMGUI_CHECKVERSION();
ImGui::CreateContext();
ImGuiIO& io = ImGui::GetIO(); (void)io;
ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 440");

// IMGUI rendering functions (inside render loop)
ImGui_ImplOpenGL3_NewFrame();
ImGui_ImplGlfw_NewFrame();
ImGui::NewFrame();

// IMGUI shutdown functions - close down ImGUI and destroy it.
ImGui_ImplOpenGL3_Shutdown();
ImGui_ImplGlfw_Shutdown();
ImGui::DestroyContext();
```

Figure 33: ImGui initialization inside GLFW class (See Appendix W)

With the GUI initialized and set up as shown in Figure 33, the researcher could make use of the numerous functions to add the functionality the researcher wanted. Some of this functionality included:

1. Open File - With Dialogue Box
2. Graph Selector
3. Colour Picker
4. Clear Data Button

This functionality would allow the program to run based on the actions of the GUI making it operate like more of an application.

5.3.3 Label Generation

Labels are one of the most important aspects of any plotting program. Labels allow the user to read and understand the data being displayed on the screen. If there were no labels, the data would almost be impossible to read and understand. The researcher had a vague understanding of how this could be achieved using OpenGL. Creating an object and texturing a character (whether it's a number or a letter) on top of it. To gain a better understanding of how this could be achieved, the researcher went online to see if anyone else had done something similar. The researcher stumbled across several articles and webpages dedicated to text on screen in OpenGL.

5.3.3.1 Texture Generation

One in particular was from LearnOpenGL [19]. This provided the researcher with a good base what they wanted to achieve. The article provided a great example of generating textures from a font (stored using the Freetype library [20]) and storing these in a map (using a char and Character struct as a key-value pair) that would be accessed later for texturing onto an object. The researcher thought this would be ideal for what they were aiming to achieve. Below (Figure 34) is the algorithm used to generate textures for each character in a font.

```
// load first 128 chracters of ASCII set
for (unsigned char c = 0; c < 128; c++)
{
	// load character glyph
	if (FT_Load_Char(face, c, FT_LOAD_RENDER))
	{
		std::cout << "ERROR::FREETYPE: Failed to load Glyph" << std::endl;
		continue;
	}

	// Generate texture
	unsigned int texture;
	glGenTextures(1, &texture);
	glBindTexture(GL_TEXTURE_2D, texture);
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RED, face->glyph->bitmap.width, face->glyph-
>bitmap.rows, 0, GL_RED, GL_UNSIGNED_BYTE, face->glyph->bitmap.buffer);

	glGenerateMipmap(GL_TEXTURE_2D); // Generate Mipmaps - Limit some aliasing
issues (Unsure if this has any effect due to the sizes)

	// set texture options - using mipmaps
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);


	// store the character for later use
	Character letter = {
		texture,
		glm::ivec2(face->glyph->bitmap.width, face->glyph->bitmap.rows),
		glm::ivec2(face->glyph->bitmap_left, face->glyph->bitmap_top),
		static_cast<unsigned int>(face->glyph->advance.x)

	};

	// add char and character struct to map
		Characters.insert(std::pair<char, Character>(c, letter));
}
```

Figure 34: Generating Textures from font (See Appendix W)

5.3.3.2 Generating Labels

However, can can't display labels if we don't know which labels to display. To accommodate this, the researcher added an algorithm to the Axis class that generates a list of numbers that will be used as labels both positive and negative. This value tends to be the axis boundary value minus 1. This is stored in a vector and the return value of the "makeAxis" function. When the axis is created, the labels we need are return to be used in the application code.

```
labels = newAxes.makeAxis(largest, 0);
```

Figure 35: Setting vector to return value of makeAxis

The object used to texture the labels was a Quad. A quad is a 2D rectangle that can be created using very few vertices. As you may have seen the researcher had create a class for this object. The vertices for the quad class are created in a similar way to the cuboid class.

```
// create vertex positions by inserting quad coordinates (modified by indent variable on the x axes) into vertex position vector
quad.insert(quad.end(), { (0.0f + bump), 0.2f, 0.f });
quad.insert(quad.end(), { (0.09f + bump), 0.2f, 0.f });
quad.insert(quad.end(), { (0.09f + bump), 0.21f, 0.f });
quad.insert(quad.end(), { (0.0f + bump), 0.21f, 0.f });
```

Figure 36: Vertices for X axis quad

Each axis requires different vertices to be modified, therefore each case had to be accounted for. When generating the quads a number is sent as an argument. This number determines which axis to create the quad for. For example, 2 means Z axis. Also, it was found by the researcher that certain characters require smaller quads. If this was not accounted for, the labels would look distorted.

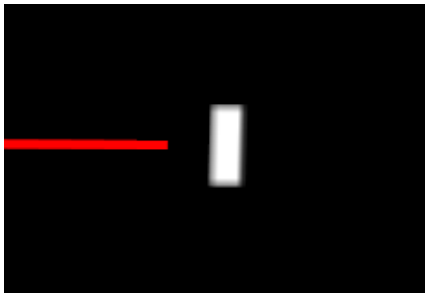

Figure 37: Distorted character from regular quad

The researcher felt that the best work around for this was to make special cases for the most frequently used smaller characters. This mainly boiled down to the characters I and '-'.

```
// create vertex positions by inserting quad coordinates (modified by indent variable on the x axes) into vertex position vector
quad.insert(quad.end(), { (0.0f + bump), 0.1f, 0.f });
quad.insert(quad.end(), { (0.09f + bump), 0.1f, 0.f });
quad.insert(quad.end(), { (0.09f + bump), 0.3f, 0.f });
quad.insert(quad.end(), { (0.0f + bump), 0.3f, 0.f });
```

Figure 38: Quad for "-" character (See Appendix W)

Checks would be added, when creating the quads, to check if the character was an I. However, since we know that the first character of a negative number is a '-'. We can check for this when creating quads with negative numbers.

Before creating the quads necessary for the labels, the researcher thought that splitting the label vector from Figure 35 into 2 vectors (one for positive numbers, one for negative numbers) would be a good idea since it would avoid a lot of confusion trying to determine what numbers are negative. Therefore, the researcher created a function to do such that.

```
/*
* Splits Label Vector into 2 different Vectors
*/
void splitLabelVectors()
{
	// loop through the label vector inserting positive numbers into one vector and negative vectors into another
	for (int i = 0; i < labels.size(); i++)
	{
		// check if vector is empty to prevent memory exceptions
		if (!labels.empty())
		{
			// check for negative number
			if (std::stof(labels[i]) < 0)
			{
				// insert into negative vector (convert to int allows numbers to be to one decimal place i.e. 2 instead of 2.000000)

	negativeLabels.push_back(std::to_string(std::stoi(labels[i])));
			}
			else
			{
				// insert into positive vector (convert to int allows numbers to be to one decimal place i.e. 2 instead of 2.000000)

	positiveLabels.push_back(std::to_string(std::stoi(labels[i])));
			}
		}
	}
	// reverse the negtive vector due to drawing order
	std::reverse(negativeLabels.begin(), negativeLabels.end());
}
```

Figure 39: Splitting label vector function (See Appendix W)

As seen in Figure 39, the labels are split into 2 separate vectors to distinguish their contents. Note that the negative vector's contents are reversed. The reason for this is because the labels are drawn from left to right. This means that the negative labels start at the "smallest" number. For example, if we have labels extending from -10 to 10 the smallest number is -10. This is the point we wish to start from and to do this, we must reverse the negative number vector order.

With that sorted, it is now time to create the quads so the labels can be textured onto them. This algorithm functions similarly to the bar chart algorithm. The similarities being we create a vector of quads to store all the quads, create each quad using dynamic arguments, loop until each quad had been created. This need to be done for each of the 3 axes for both positive and negative labels. For example, if

we need labels for -5 to 5. That would mean we need 5 quads for the positive side, and 10 for the negative side since the negative symbol '-' needs its own quad to be textured on to. This adds up to a total of 15 quads for one axis, then multiply this by 3 which is 45 and then we have the total number of quads we need.

```
// create a quad for each positive number for current axes
for (int i = 0; i < positiveLabels.size(); i++)
{
	// create variable to track size of number i.e. does it require more than 1 quad to draw (10)
	float cycle = 0;

	// check if vector is empty - prevents memory exceptions
	if (!labels.empty())
	{
		// get out first number
		number = positiveLabels[i];

		// set indentation for y axes labels
		yBump = 0;

		// loop through the string using string iterator
		for (test = number.begin(); test != number.end(); test++)
		{
			// check if we are on the Y axes
			if (j == 1)
			{
				// if cycle is higher than one the number is more than one digit
				if (cycle >= 1)
				{
					// increase indent by 0.1
					yBump = yBump + 0.1f;
				}
			}
			else // for x and z axes
			{
				// if cycle is higher than one the number is more than one digit
				if (cycle >= 1 && j != 1)
				{
					// increase indent by 0.1
					bump = bump + 0.1f;
				}
			}

			// insert a quad object into quad vector
			numberLables.insert(numberLables.end(), newQuad);

			// using tracker variable make a quad using variables to decide which quad to make
			// - bump indicates where to draw the quad
			// - 1 indicates what size of quad to draw (i.e. normal or small)
			// - j is which axes to draw for
			// - yBump is used for numbers on the y axes longer than one digit
			numberLables[count].makeQuad(bump, 1, j, yBump);

			// add one the counter to increase quad count
			count++;

			// increase cycle by one since we have completed a cycle
			cycle = cycle + 1;

		}

		// reset indent variable back to original status i.e. status before the loop
		if (cycle > 1 && j != 1)
		{
			// decrease bump by the number of cycles minus one (since an extra cycle is added on when
it didn't really happen)
			bump = bump - (0.1f * (cycle - 1));
		}

		// reset y axes indent variable
		yBump = 0;

		// move along current axes to make quads in correct location i.e. move from 1 to 2 or 2 to 4 in
world space
		// addby can either be 1 or 2 (this is due to the gap between axes notches having variable size
depending on the largest value)
		bump = bump + addby;

	}

} // end of positive label loop
```

Figure 40: Creation of positive label quads algorithm (See Appendix W)

In Figure 40, you can see what is required to generate quads for the positive numbers. This algorithm, along with a similar one for the negative numbers, is repeated 3 times for each axis. There are a few things to check for when looping though this algorithm:

1. Which axis are we generating for?
2. Does the label require more than one quad to texture?

Due to some numbers requiring more than one quad to display them, the program needs to account for multiple quads on in the same spot. In order to do this, we need to check if we have a number with more than one digit. This is denoted by the "cycle" variable. Since the algorithm is looping through the length of the label (number), if this is higher than 1 that must mean the number has more than one digit. If so, move the next quad over slightly. This is shown by the following code:

```
// if cycle is higher than one the number is more than one digit
if (cycle >= 1 && j != 1)
{
	// increase indent by 0.1
	bump = bump + 0.1f;
}
```

Figure 41: Indentation Code

Since the algorithm is used for each axis, we need to check when we want to generate quads for the Y-axis. This means alternating the indentation variable. This is used for multi-digit numbers on the Y-axis. This is due to the main indentation variable not modifying the same vertex position as the one required.

```
// create vertex positions by inserting quad coordinates (modified by indent variables
on the y and x axes, to make multiple characters work on the same line) into vertex
position vector
quad.insert(quad.end(), { (0.09f+yBump), (0.3f+bump), 0.f });
quad.insert(quad.end(), { (0.18f+yBump), (0.3f+bump), 0.f });
quad.insert(quad.end(), { (0.18f+yBump), (0.31f+bump), 0.f });
quad.insert(quad.end(), { (0.09f+yBump), (0.31f+bump), 0.f });
```

Figure 42: Y-axis quad creation (See Appendix W)

To achieve this, the researcher added a check to see what the current iteration of the loop is. If the loop is on the Y-axis and we have multiple digits (implied by variable "cycle"), then we use the "yBump" variable to indent multi-digit numbers.
A similar process to Figure 40 occurs for the negative numbers. To be more efficient, both positive and negative labels are generated for each axis at the same time.

5.3.3.3 Texturing Quads

Now each quad has been generated, the texturing process can begin. This involves looping through each label, both positive and negative, finding the relevant texture for that quad and then drawing the quad with the texture on top. In order to find the relevant texture required we need to use the Character map created earlier.

```
for (q = positiveLabels[i].begin(); q != positiveLabels[i].end(); q++)
```

Figure 43: Setting string iterator to label

By setting a string iterator to the first char of the label, we can find what texture we require for the current quad.

```
Character charTex = Characters[*q];
```

Figure 44: Creating Character struct instance using map

By using the iterator as a key, we can find the texture by creating a new struct instance of the value paired by the iterator key. For example, if the iterator was 'R' the value for this key would be a Character struct with the data for the char 'R' (this incudes the important texture ID). With this we can now use this texture to put on top of our current quad.

```
// make textures active
glActiveTexture(GL_TEXTURE0);

// make our other VAO current
glBindVertexArray(textVertexArrayObj);

// Standard bit of code to enable a uniform sampler for our texture
int loc = glGetUniformLocation(textureShaders, "text");
if (loc >= 0) glUniform1i(loc, 0);

// Bind the texture (character) using the texture ID variable inside the charTEX
struct
glBindTexture(GL_TEXTURE_2D, charTex.TextureID);

// draw the quad to texture the character on to
numberLables[track].drawQuad();
track++; // add one to track var (this moves to the next quad in the vector, which is
the next char in the string or the next notch number)

// unbind the VAO and the texture (unbinding the texture will prevent it from
texturing anything else)
glBindVertexArray(0);
glBindTexture(GL_TEXTURE_2D, 0);
```

Figure 45: Texturing Current Quad (See Appendix W)

Since the quads for the labels were generated in an order of positive, negative, positive… The labels were textured in the same order to match the quads. This process would occur for the size of the quad vector. Once complete, there would now be labels along the axis.

5.3.4 Sprint 2 Review

As this was the final development sprint, the researcher thought is would be good to perform a sprint review for this sprint. This would allow the researcher to use the advice for any future projects/sprints that may happen going forward. Again, the researcher found that the expected velocity had not been reached. The researcher believes this was down to specific user stories not carrying enough story points with them. For example, the user story "As a user, I can view axis labels so that I can read plotted data" was allocated 8 story points. Looking back, the researcher believes this should have been around 20 or 40. This was one of a few actions that the researcher believes should be taken on for a future project/sprint as it would help out significantly with its development.

5.4 Development Challenges

5.4.1 Memory Management

Due the large scall creation of objects, memory usage starts to become an issue if it is ignored. The researcher had initially done nothing about managing the amount of memory allocated during runtime. However, it was after completing the labels that they realized how much memory was being used. This was also the case for the bar chart since it was creating large cuboids, and nothing was being done about it.

```
/*
* Clear the vertex buffers of the cube
*/
void Cube::clearCube()
{
	// create empty vector to copy into the vertex buffer
	std::vector<GLfloat> vertexPositions = { 0 };

	/* Create the empty vertex buffer for the cube */
	glGenBuffers(1, &positionBufferObject);
	glBindBuffer(GL_ARRAY_BUFFER, positionBufferObject);
	glBufferData(GL_ARRAY_BUFFER, vertexPositions.size() * sizeof(GLfloat),
	&(vertexPositions[0]), GL_STATIC_DRAW);
	glBindBuffer(GL_ARRAY_BUFFER, 0);

	// create empty vector to copy into the colour buffer
	std::vector<GLfloat> vertexColours = { 0 };

	/* Create the empty colour buffer for the cube */
	glGenBuffers(1, &colourObject);
	glBindBuffer(GL_ARRAY_BUFFER, colourObject);
	glBufferData(GL_ARRAY_BUFFER, vertexColours.size() * sizeof(GLfloat),
	&(vertexColours[0]), GL_STATIC_DRAW);
	glBindBuffer(GL_ARRAY_BUFFER, 0);
}
```

Figure 46: Function to clear VBO for cube (See Appendix W)

To solve the memory problems, the researcher implemented clearing functions to reduce the memory usage when they were not being used (i.e. when the graph has been cleared). These functions mostly involved clearing the vertex buffer objects for the object as seen in Figure 46. To clear the multiple instances created inside vectors. For example, the cuboids inside the cuboid vector. The researcher made the program loop through the vectors and call the clear function for each object. This was true for all object vectors including the label vectors.

5.4.2 Building executable to run on other machines

Testing was on the agenda after the second sprint had concluded. The researcher wanted to conduct user testing for this version of the application to gain feedback and understand what could be better. However, for this to

happen, the researcher had to build an executable file that could run without any external dependencies. The current build settings meant that this would not be the case without some changes. The project needed to be built as a Static-Release build. This meant switching from debug mode to release mode and configuring the Runtime Library from Multi-Threaded DLL (/MD) to Multi-thread (/MT).

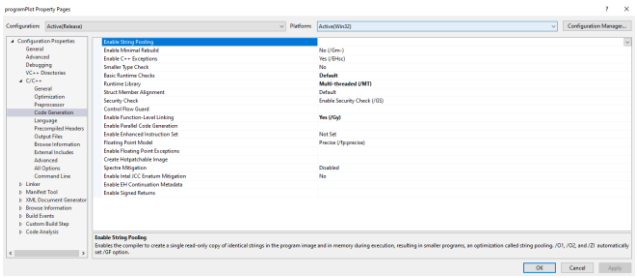

Figure 47: Project Configurations (See Appendix P)

Problems started to occur when trying to configure the runtime library setting. Switching from /MD to /MT create an error that was very hard to diagnose (See Appendix Q). The researcher did a vast amount of research to try and find a solution for this error. However, all the solutions they found were not what they were looking for. It wasn't until the researcher started try their own solutions that some progress was made. Since the error was complaining about "glload", the researcher thought it would be best to update the current version of the libraries they were using. Making sure to download static versions of these libraries. Whilst searching for new libraries, the researcher could not find a version of "glload", so they decided to use GLEW instead. GLEW is another loading library similar to "glload". With the installation of the new libraries, the project was now able to build. However, it would still be depending on specific .dll files.

```
/* Static Library Links - DEBUG/RELEASE MODE */
#ifdef _DEBUG
#pragma comment(lib, "glfw3D.lib")
#pragma comment(lib, "glloadD.lib")
#else
#define GLEW_STATIC // GLEW_Static require for running GLEW in static mode
#pragma comment(lib, "glfw3_mt.lib")
#pragma comment(lib, "glew32s.lib")
#endif
#pragma comment(lib, "opengl32.lib")
#pragma comment(lib, "freetype.lib")
```

Figure 48: New Library Declarations (See Appendix W)

A helpful debugging tool that was pointed out was the "dumpbin" executable [22]. This allowed the researcher to check if the executable was depending on any external .dll files.

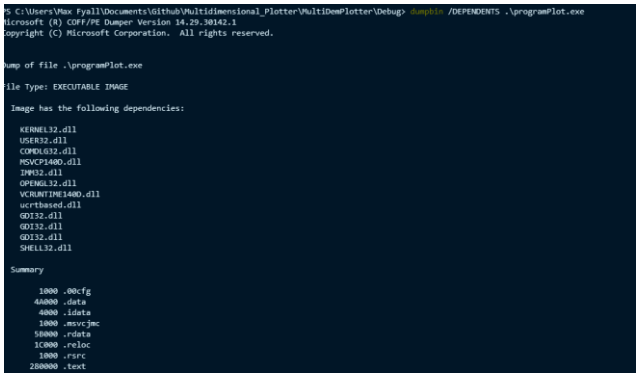

Figure 49: Dumpbin example (See Appendix X)

Thanks to the /DEPENDENTS argument, the researcher could see that the executable was still dependent on a couple .dll files. In order to fix this, the researcher made sure that the executable would not use any of the runtime libraries not defined under /MT [23].

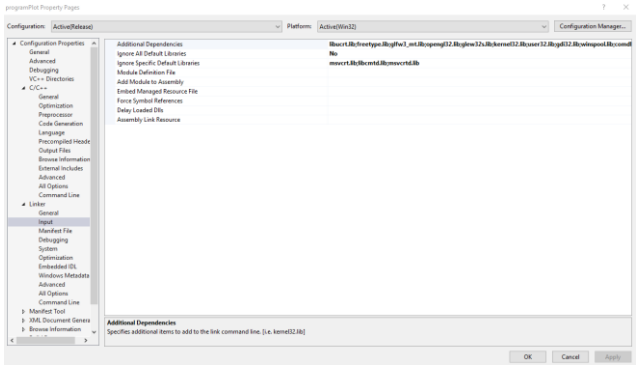

Figure 49: Linker Settings (See Appendix R)

To apply this, the C++ linker need to be told to ignore specified libraries. Once this had been added, the application worked on any windows machine. The researcher breathed a sigh of relief.

6 Evaluation / Testing

6.1 User Testing

Once the researcher had a complete application, they wanted to send it out to people and conduct user testing to evaluate the application. The aim of this test was to gather data on the usability of the application.

6.1.1 Plan

The researcher would send out the completed build along with some pre-made data sets for users to play around with. The researcher had created a questionnaire for the users to complete to provide any feedback to the researcher regarding the application. A message would be sent to a large group of individuals in a social channel to try and gain some participants. In an ideal world, the researcher wanted

to gather users of varying backgrounds such as age, ability, and computing knowledge. However, due to timing constraints, it was hard for the researcher to gather willing participants from differing backgrounds. The researcher subsequently stuck with whatever responses they received. The questionnaire, that the researcher had created, walked the participants through the ethical checks before they took part in the survey. Since the application would be given out to unknown participants, help could not be arranged immediately. The researcher did provide their email address and advised participants to reply to their post if they required any assistance. The application did also include a "HELP" tab as a backup to help any participants during their time using the application.

6.1.2 Ethical Considerations

The project coordinator, supervisor and researcher had submitted an Ethics application to the Ethics Committee to be granted ethical approval. This application covered "Low Risk" projects, which meant the researcher could conduct standard questionnaires with non-vulnerable users over the age of 18 and no personal information is required from the participants. Since the requirements of the researcher user testing did not go beyond the applications remit, no further applications needed to be completed and approved.
The researcher's questionnaire was a combination of the 2 forms required to pass the ethical requirements (See Appendix S). The participant information sheet (See Appendix S) is featured at the start and the users can read it. This is followed by a series of questions from the consent form (See Appendix S). The questionnaire would not let the user participate if they did not agree to the ethical form since each field was required (See Appendix T). This was enough to grant the users consent for taking part in the questionnaire.
All pre-made datasets provided by the researcher (See Appendix V) were auto generated by the researcher or free to use and came under the Data Protection Act.

6.1.3 How it was done

Each participant would be on their own when participating. The researcher was confident that the users would be more than fine operating the software without anyone being alongside due to help being provided in other forms. Mainly, the help tab in the software itself. The researcher had provided clear instructions in the blog post they provided. So, it would be unlikely to have a user unsure of what to do. If there was anything the user did find that wasn't right or needed to be changed, they could always include it in their feedback since the questionnaire allowed for long answers in some of the questions. Results from the questionnaire would give the researcher a good idea of any specific features that needed to be tweaked or fixed.

6.1.4 Feedback

Summarizing the feedback based on the questions asked, the general consensus was:

- The plotted data made sense and could be understood
- The user controls were intuitive to use
- The GUI was easy to use
- The labels were hard to read on some occasions
- More features could be added and/or changed to better the application.

Individual responses can be found in Appendix U. This is where the consensus was derived from and where all the participants consent is.

6.1.5 Interpretation

The researcher was pleased with the consensus of the feedback. They knew that there could be improvements made to the program and its features. However, the fact that users were able to use and understand how the program worked played a significant factor in the conclusions the researcher would make.

6.2 Final Evaluation

The researcher believes the user testing carried out was successful. Participants left good and constructive feedback that would be very useful in any future development of the application. Any and all feedback was aimed at improving the quality of the application for the better. One response in particular suggested the use of colour differentiation across all the different data points (See Appendix U). This would allow the readability of the data to be greatly increased. Subsequently, improving the application as a whole.

6.2.1 Positives

Generally, users found the plotted data being presented to them easy to read. The researcher thought this was true since the data was clear enough on the screen and there was the help of the labels as well.
The users found that the user controls were intuitive to use. This was the answer the researcher was hoping for since they made the user controls as similar to conventional plotting programs as possible.
The users were able to read and navigate the GUI without trouble. This meant users would have had no trouble accessing all the features the application had to offer.
As mentioned previously, even though the general consensus was positive, does not mean that some changes don't need to be made to better the application's experience.

6.2.2 Limitations

It was made clear to the researcher that sometimes the labels could be hard to read in certain instances. For example, when rotating the graph, the labels would get flipped depending on the rotation applied by the user (See Figure 50).

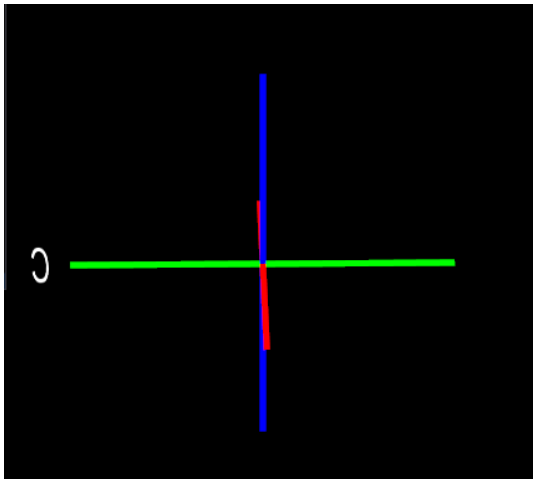

Figure 50: Backwards Label

The researcher believes adding a fix to this would be beneficial to the application. An easy work around for this would be to not have the label show on that side. This is possible by enabling "face culling" in OpenGL.

7 Description of the final product

The final application is an OpenGL program. It was created using 3D graphics technology and C++ (See Appendix W). The following description can be used as a user manual or technical guide.

7.1 Program Start-up

Upon start-up, the program will load alongside a command prompt terminal. The program window will display a 3D axis template drawn at the origin alongside a GUI (See Figure 51).

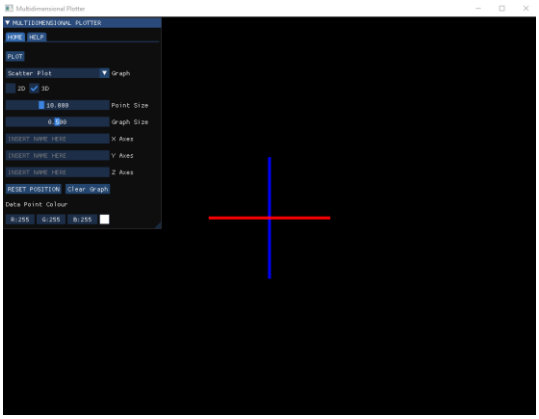

Figure 51: Program start-up

If required, the user can click the "help" tab on the GUI to be given another set of instructions for operating the program.

Figure 52: GUI Help tab

By navigating though the help menu, users can gain a greater understanding of the full workings of the program.

7.2 Graph Selection

To select the graph to plot, chose one of the three options from the drop-down menu. The graph type currently selected will be the graph plotted when data is loaded in.

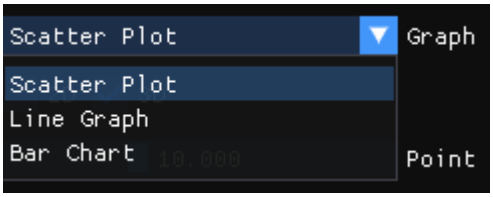

Figure 53: Graph selection drop-down

The drop-down also has the effect of clearing any plotted data. So, if the user is to select another graph the current graph will be cleared. This was based on the assumption that a user selecting a different graph means they will want to plot a different graph.

7.3 Plotting data

Plotting data in the application is down by loading the data from a text file. Once the user is happy with their graph type, they can start plotting by click the "plot" button.

Figure 54: Plot button

Upon clicking the plot button, a file dialog box will appear (See Figure 55).

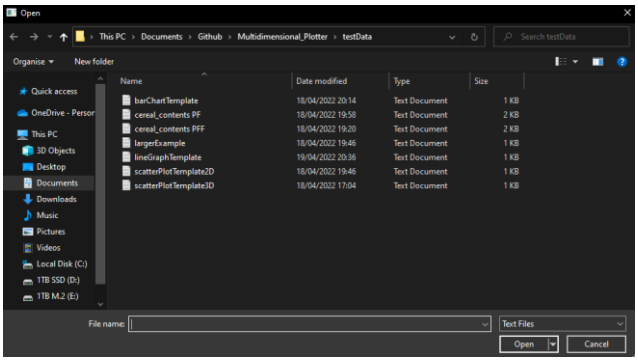

Figure 55: File dialog box after button press

Navigate to find the file you want to plot. Upon opening the file, the dialog box will close, and the data will be read in and plotted using the pre-selected graph type and colour.

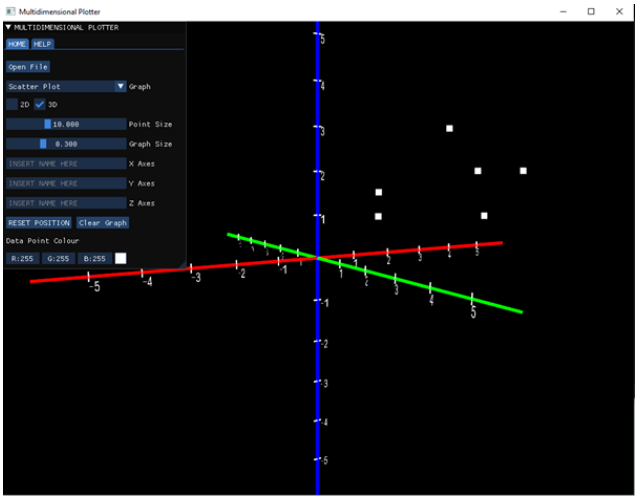

Figure 56: Example scatter plot

7.4 Graph specific features

Each graph (expect the bar chart) has a unique feature the user can use to their liking.

7.4.1 Scatter plot

The scatter plot allows the users to increase/decrease the size of the data points. This is applied using a slider on the GUI (See Figure 57).

Figure 57: Scatter plot features

Also present in Figure 57 is a feature to toggle between dimensions. This allows the user the ability to plot 2D or 3D data if they wish. It must be said that toggling 2D dimension with a 3D data set will result in data not being plotted. The same goes for 3D with 2D data.

7.4.2 Line Graph

As mentioned in Section 5 Implementation & Testing, the line graph has a check box that enables the first and last points to be joined with a line. This is called Line Loop.

Figure 58: Line loop check box

Users can enable this to see their line plots interconnected as one large shape.

7.5 Universal Graph Features

Each graph type has universal features that do not differ from graph to graph.

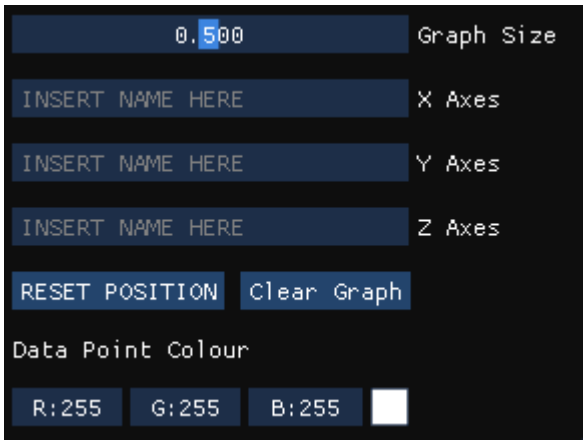

Figure 59: Universal graph features

As shown in Figure 59, this includes:

- Modifying the graph size
- Naming each axis
- Resetting the graph position
- Clearing the graph
- Modifying the colour of data points

Each of these features is described below.

7.5.1 Adjusting graph size

For larger graphs, the size can become overwhelming, and the scroll zoom will only zoom out so far. The same could be said for smaller graphs. To compensate for this, the user can adjust the size of the graph to fit their liking. To do this simple adjust the slider one way or another.

7.5.2 Axis naming

Plotting data requires the user to know what each axis represents. Without a name, the data could mean anything.

To name an axis, type the name/word into the box of the relevant axis. Changing the contents of the box should allow the "Update Label" button to appear (See Figure 60).

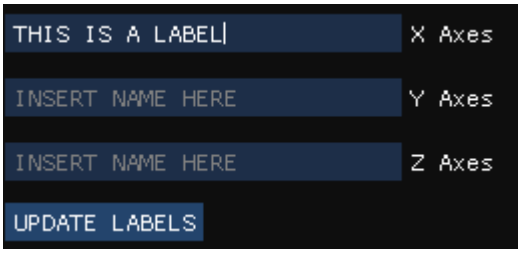

Figure 60: Updating axis name

Clicking this button will add the name to the relevant axis as shown below.

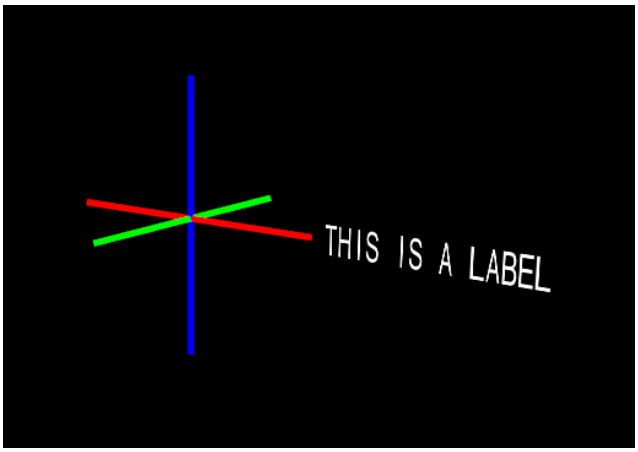

Figure 61: Label after clicking to update

To change the current name, add a new name to the input box and click the "update" button. The old name will be removed and the new one added.

7.5.3 Reset graph position

This button allows the user to reset the position of the graph to the origin. This saves the user time if they have moved the graph from the origin and wish to move it back or go to the opposite side.

7.5.4 Clearing plotted graph

Clearing the graph will clear any plotted data currently on the screen. A message will also appear in the command prompt terminal. If there is no data on the screen, nothing will happen. A message will still appear in the command prompt terminal confirming the button press. Axis names will be kept upon this button press.

7.5.5 Modifying colours

To modify the colour of data points, the user can use the colour picker provided in the GUI.

Figure 62: Colour picker

The user can specify an RGB value by using the mouse to slide the number up and down. Alternatively, clicking the colour sample square will pull up a more detailed colour picker as shown in Figure 63.

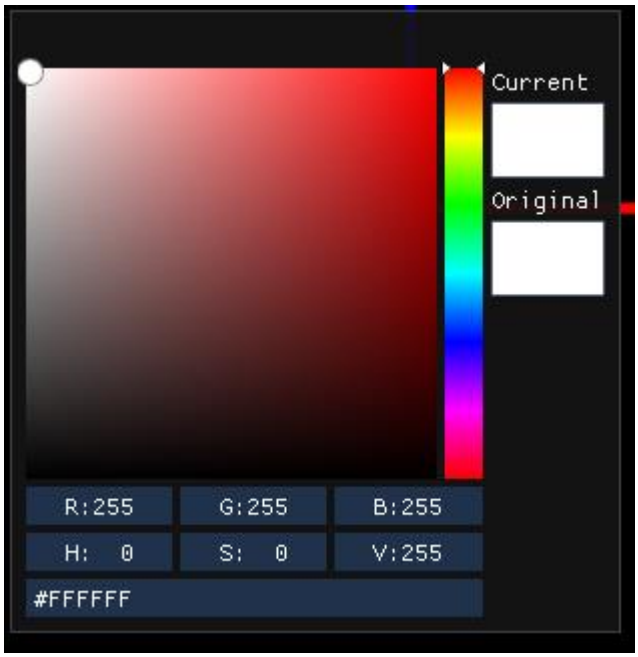

Figure 63: Detailed colour picker

Here the user can specify the colour they want very precisely. This window is draggable so the user can place it where they like. The user can select the colour they want before they plot and after they plot.

7.6 User controls

The user can make use of the pre-defined controls to navigate the application. The "help" tab provides the list of bindings and controls available to the user.

7.6.1 Mouse rotation

Graphs can be rotated on the x and y axis by clicking and dragging with the mouse. The buttons to press to activate this are Mouse 3 or Mouse 2.

7.6.2 Zooming in/out

Use the scroll wheel to zoom in and out. Scrolling up will zoom in and scrolling down will zoom out.

7.6.3 Moving graph position

Moving the graph might be necessary to get a better view of the plotted data. To do this, press the arrow keys in the desired direction to move the graph. As said before, the user can always press the "Reset position" button to move the graph back to the origin.

7.6.4 Exiting the program

A shortcut was added to exit the program. Pressing "ESC" will result in the program being terminated.

8 Summary and Conclusions

To summarise, the researcher has created a data plotting application using OpenGL (C++) and 3D graphics technology. The aim of this was to prove that a program of this nature could be created using the technology present within OpenGL. This subsequently could prove that the 3D plotting experience could be improved through the use of advanced 3D graphics techniques such as lighting etc.
The application create was a GUI driven plotting program. This was to ensure new users would find the application easy an intuitive to use.
Results from the user testing carried out concluded that the application worked as a plotting program but, the application could still be improved with the addition of some feature made apparent from the user testing.
The researcher believes that they have achieved their goal by proving it is possible to create a data plotting program with the technologies used. This, subsequently, can open the door for further additions to the application (See section 8.1.1). Of which can be investigated to see if the bring any improvements to the 3D plotting experience.
The researcher understands that this project could have taken many different approaches across its duration. Some of these would have made the project stand out more according to the researcher. More attention to detail, more commitment were some of the qualities missing from the project at some stages by the researcher's own admission.
Overall, the researcher is very happy with the development they carried out over the course of the project. The issues and problems the researcher encountered over the duration of the project will help shape them to deal with whatever is thrown there way in the future.

8.1 Future Work

This section underlines the details of any future implementations that could be carried out on the application based on the conclusions given in Section 8.

8.1.1 Implementation of advanced 3D graphics techniques

Following up from the conclusions given, the implementation of advanced 3D graphics techniques can be achieved in the application. Features such as, lighting, shadows, point sprites etc. could be implemented in some fashion. The program is definitely able to incorporate such techniques. The question would be, does the implementation of these techniques provided any benefit to the 3D plotting experience? With these added effects, it would be interesting to see what users would say regarding the previous question during a user testing phase.

8.1.2 Fixing apparent issues

It was made clear during user testing that there were some issues that the participants felt could be changed. One in particular was the readability of the labels (both number and names). The issue becomes apparent when plotting larger graphs. The labels become really small. In the future, an algorithm could be created to adjust the size of the labels in accordance with the current zoom level and graph size. This way, the labels could be read clearly at all times.

8.1.3 Addition of new features

Participants also made it clear during user testing that they would like to see more features added to the application. The researcher felt some of these would benefit the system and the users using the system. One feature in particular that was suggested was the addition of a colour differentiation across data points. The idea being that the colour of a data point is determined by its data value. This would add another level of understanding to the plotted data as the colour would inform the user of its data value.

8.1.4 More User Testing

The researcher felt as if they did not get as many user testers are they would have liked. The user that did participant all gave very similar feedback, so the researcher believed this feedback would be fairly universal across many other users. Since this was the case, the researcher believes more user testing should be done in the future to confirm and validate the conclusions made in this report.
Also, if the application were to receive the future implementations featured before (i.e. Sections 8.1.1, 8.1.2 and 8.1.3) then testing should be carried out to investigate how users feel towards them. Are they worthwhile? Do they add anything significant? These questions would be answered with further user testing (after additional development).

9 Appraisal

The researcher looks back on this project with mixed feelings. On the one hand, the researcher is proud of the work they managed to accomplish over the course of the project. On the other, there were many things that could have been done differently that could have potentially benefitted the system.
To start, the researcher could have looked into other potential technologies instead of sticking to what they knew best. Other technologies could have provided better tools to the researcher over the course of the project. An example would be the decision to use the GLFW windowing system as apposed to something like "Qt". The researcher could

have looked into what both sides offer and decide based on what they offer rather than what the researcher knew best.
The researcher also understood towards the end of the project's life, that they could have done a better job planning and understanding what was expected of them. More of a deeper look into potential features and ideas that could have been implemented to the project. This was highlighted by the user testers when potential features were being suggest to the researcher.
When it comes to implementation decisions, the researcher believes they did the best they could to get the system they ended up with. Certain algorithms had never been done before and the researcher felt they did the best job when it came to these algorithms. Using knowledge and other algorithms as a base line to create the end product. The best example of this would be the label generation. A good base line idea (texturing characters onto quads), taking that and apply it to the researchers own problem.
The researcher learned over the course of the project just how much effort is require into the production of a system. Especially one of this size. The inconsistent nature of the researchers working habits played a bad role in the lack of development in certain areas. It was also this that the researcher believes caused the second development sprint to overrun its allocated duration. In hindsight, the researcher would have employed a much stricter working schedule to avoid such habits showing themselves.
Overall, the researcher is pleased with the outcome of this project. They stuck to plan they set out at the start (GUI driven application using OpenGL (C++)), had good communications with the client and worked hard despite any problems that may have occurred.

Acknowledgments

The researcher would like to acknowledge the ImGui [17] and the Freetype [20] libraries for allowing free use within the project. The researcher would also like to appreciate the referenced links in the source code of the project. It was with the help of those referenced that the project is able to function the way it does. Finally, the researcher would like to thank the users who took part in the user testing and the project supervisor Dr. Iain Martin for being an amazing help and providing great support to the researcher during the most challenging times.

References

[1] Why Data Visualization Is Important - Analytiks: 2022. *https://analytiks.co/importance-of-data-visualization/*. Accessed: 2022- 04- 15.

[2] List of information graphics software - Wikipedia: 2022. *https://en.wikipedia.org/wiki/List_of_information_graphics_software*. Accessed: 2022- 01-.

[3] MATLAB Graphics: 2022. *https://uk.mathworks.com/products/matlab/matlab-graphics.html*. Accessed: 2022- 02-.

[4] MATLAB Answers: 2022. *https://uk.mathworks.com/matlabcentral/answers/index*. Accessed: 2022- 01-.

[5] Qt | Cross-platform software development for embedded & desktop: 2022. *https://www.qt.io/*. Accessed: 2022- 02-.

[6] 3D Calculator - GeoGebra: 2022. *https://www.geogebra.org/3d?lang=en*. Accessed: 2022- 02-.

[7] Brief Introduction to OpenGL in Python with PyOpenGL: 2022. *https://stackabuse.com/brief-introduction-to-opengl-in-python-with-pyopengl/*. Accessed: 2022- 04- 22.

[8] Getting started with WebGL - Web APIs | MDN: 2022. *https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Getting_started_with_WebGL*. Accessed: 2022- 04- 22.

[9] The Agile Software Development Life Cycle | Wrike Agile Guide: 2022. *https://www.wrike.com/agile-guide/agile-development-life-cycle/*. Accessed: 2022- 04- 21.

[10] An OpenGL library: 2022. *https://www.glfw.org/*. Accessed: 2022- 04- 23.

[11] GLSL Shaders - Game development | MDN: 2022. *https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/GLSL_Shaders*. Accessed: 2022- 04- 23.

[12] Definition of event loop: 2022. *https://www.pcmag.com/encyclopedia/term/event-loop#:~:text=A%20programming%20structure%20that%20continually,the%20user%20to%20trigger%20something.*. Accessed: 2022- 04- 24.

[13] About size limits on GitHub: 2022. *https://docs.github.com/en/repositories/working-with-files/managing-large-files/about-large-files-on-github*. Accessed: 2022- 04- 24.

[14] Vertex buffer object - Wikipedia: 2022. *https://en.wikipedia.org/wiki/Vertex_buffer_object#:~:text=A%20vertex%20buffer%20object%20(VBO,non%2Dimmediate%2Dmode%20rendering.*. Accessed: 2022- 04- 24.

[15] Uniform (GLSL) - OpenGL Wiki: 2022. *https://www.khronos.org/opengl/wiki/Uniform_(GLSL)#:~:text=A%20uniform%20is%20a%20global,stored%20in%20a%20program%20object.*. Accessed: 2022- 04- 25.

[16] LearnOpenGL - Camera: 2022. *https://learnopengl.com/Getting-started/Camera*. Accessed: 2022- 04- 25.

[17] GitHub - ocornut/imgui: Dear ImGui: Bloat-free Graphical User interface for C++ with minimal dependencies: 2022. *https://github.com/ocornut/imgui*. Accessed: 2022- 04- 24.

[18] ImGui Set Up Guide: 2022. *https://www.youtube.com/watch?v=VRwhNKoxUtk*. Accessed: 2022- 04- 25.

[19] LearnOpenGL - Text Rendering: 2022. *https://learnopengl.com/In-Practice/Text-*

Rendering. Accessed: 2022- 04- 25.
[20] The FreeType Project: 2022. *https://freetype.org/*. Accessed: 2022- 04- 24.
[21] Problems with running EXE file built with Visual Studio on another computer: 2022. *https://stackoverflow.com/questions/15297270/problems-with-running-exe-file-built-with-visual-studio-on-another-computer*. Accessed: 2022- 04- 25.
[22] DUMPBIN Reference: 2022. *https://docs.microsoft.com/en-us/cpp/build/reference/dumpbin-reference?view=msvc-170*. Accessed: 2022- 04- 25.
[23] /MD, -MT, -LD (Use Run-Time Library): 2022. *https://docs.microsoft.com/en-us/cpp/build/reference/md-mt-ld-use-run-time-library?view=msvc-170*. Accessed: 2022- 04- 25.

Appendices

A – MATLAB code used to create figure 2
B – Notes from background research
C – Email exchange between researcher and client
D – Client meeting notes
E – OpenGL Libraries
F – GitHub issues page
G – GitHub projects page
H – Product backlog
I – Sprint backlog
J – Gantt chart from mid-term report
K – Sprint review for sprint 1 and 2
L – Example vertex and fragment shaders
M – Class Diagram
N – Debug version linking (Include and Library)
O – Sprint 1 Self-Testing
P – Configuration Settings for Project
Q – LNK error 2038
R – Linker settings
S – Participant Information Sheet and Consent Form
T – Researcher Questionnaire
U – Questionnaire Responses
V – Pre-made data sets
W – GitHub Repository
X – Dumpbin Example