The Development of a

Multidimensional plotting program

**Max Fyall**

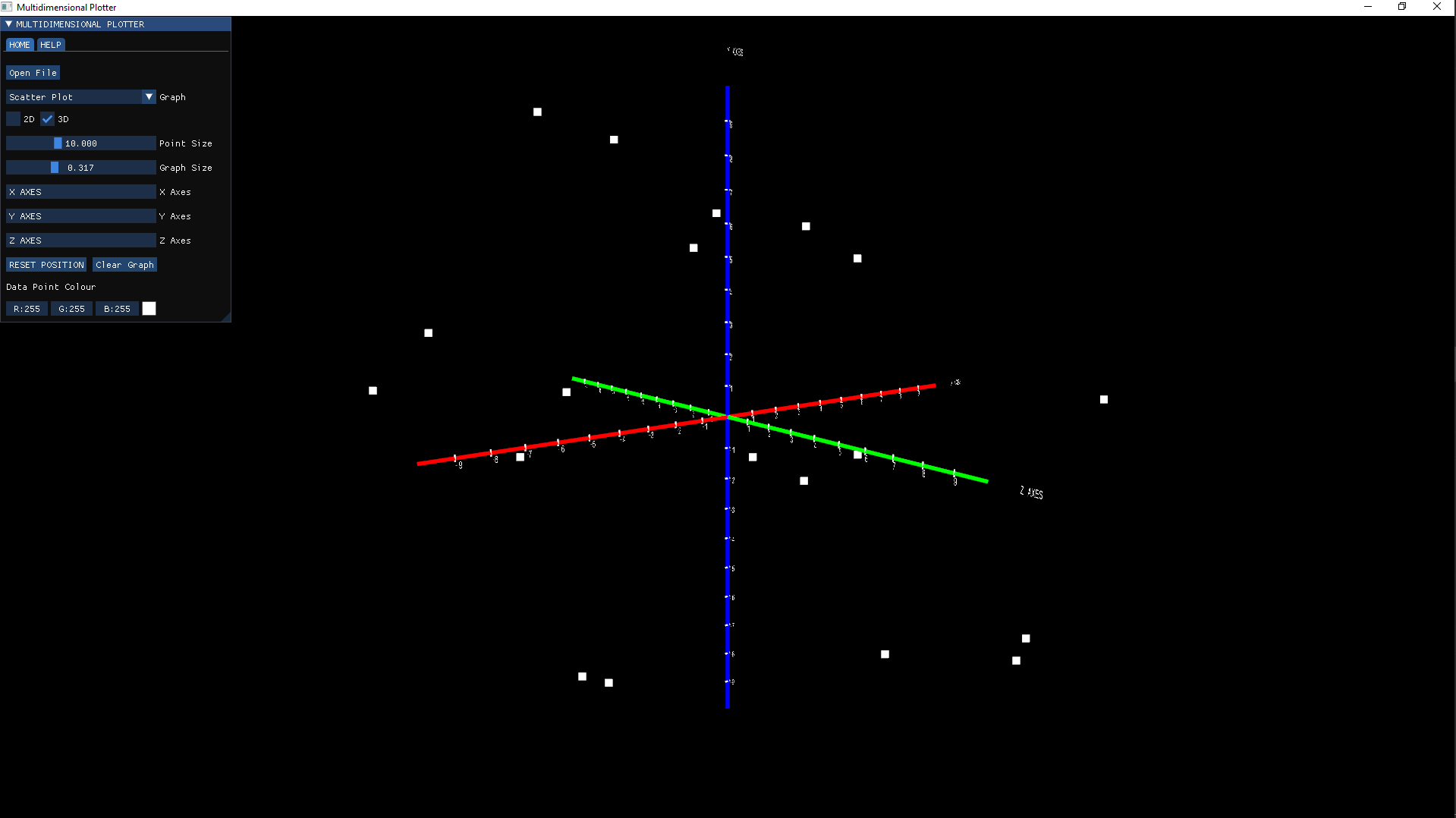
**ID: 180011724**

**AC40001 Honours Project**

**BSc (Hons) Computing Science**

**University of Dundee, 2021**

**Supervisor: Dr. Iain Martin**

****

***Abstract* -** *The abstract (or ‘executive summary’) is an important part of your report. In essence, it is a summary of the purpose, methods, findings, and conclusion of your project. It should be no more than 200 words. It should be clearly and concisely written. Provide only the most pertinent information, avoid citing references and include a brief statement of your main conclusions.*

# 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.

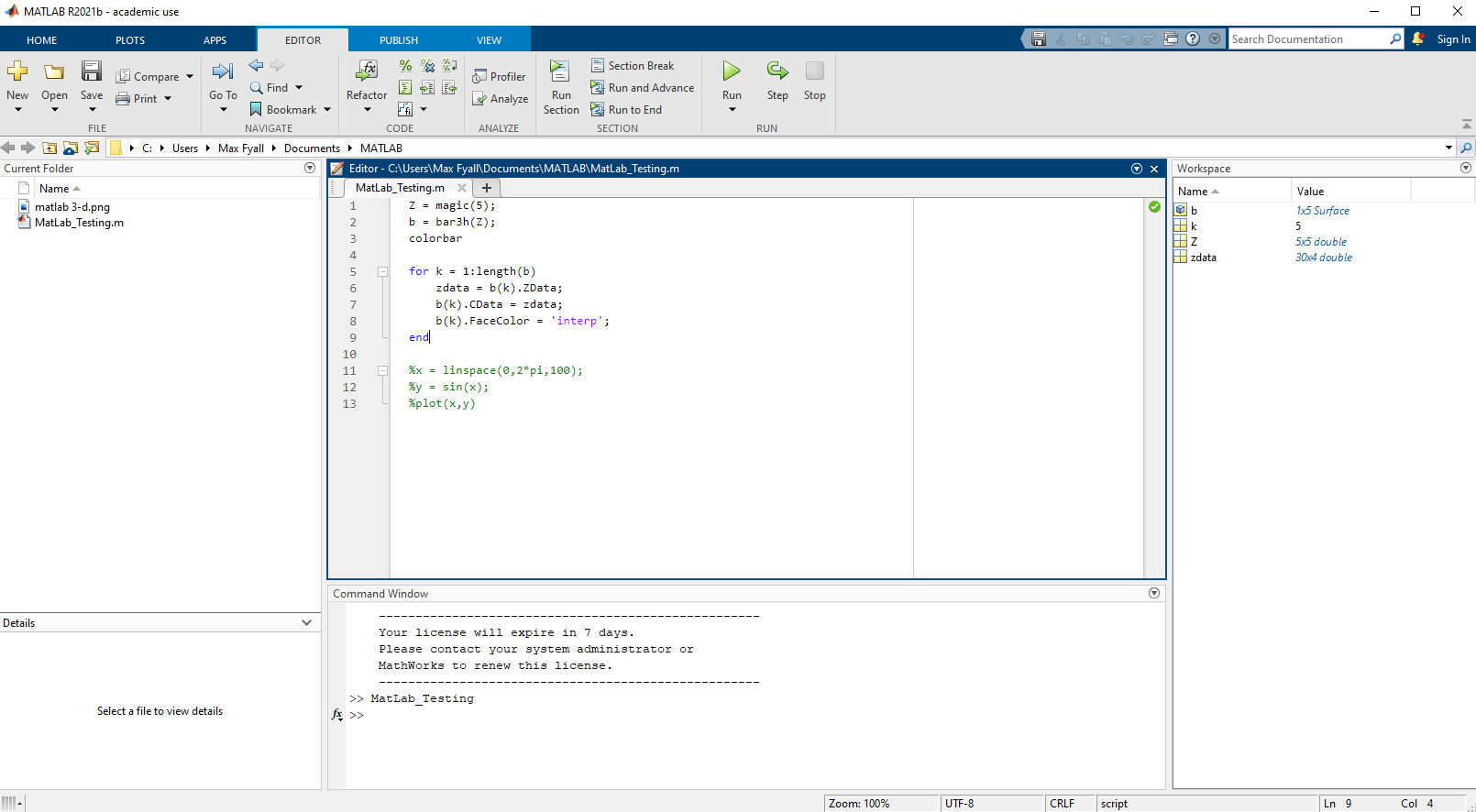
# Background

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.

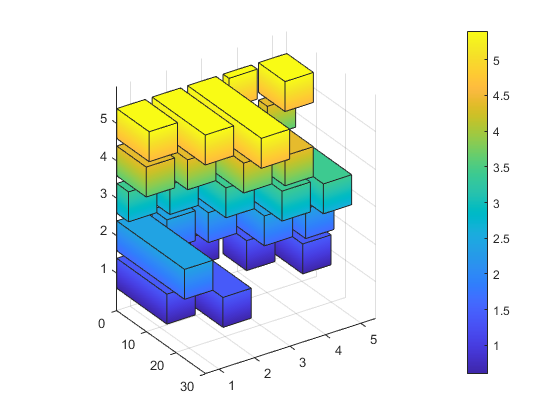
## Existing Software

### 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*

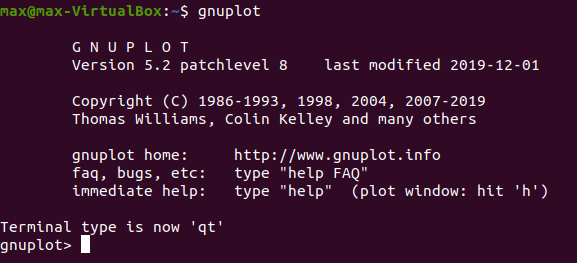


*Figure 2: 3-D Plot using MATLAB*

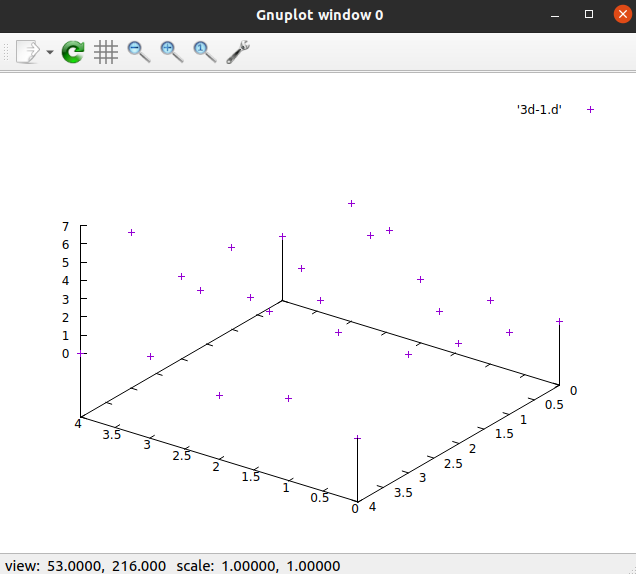
Upon clicking “run” with valid plotting code (See Appendix A), a window will appear with the completed plot as shown in Figure 2. From here the user can use the mouse to navigate around the graph to gain a better perspective of all aspects of the graph. This is very intuitive and is necessary when viewing 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.

### 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).

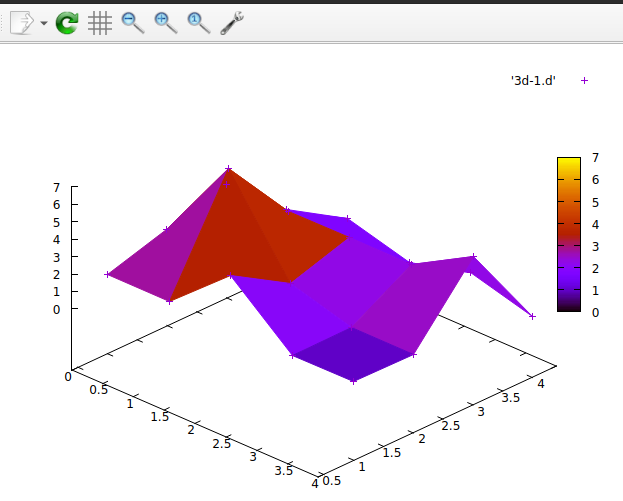


*Figure 3: GNU-Plot Terminal Interface*



*Figure 4: GNU-Plot 3-D Scatter Plot*

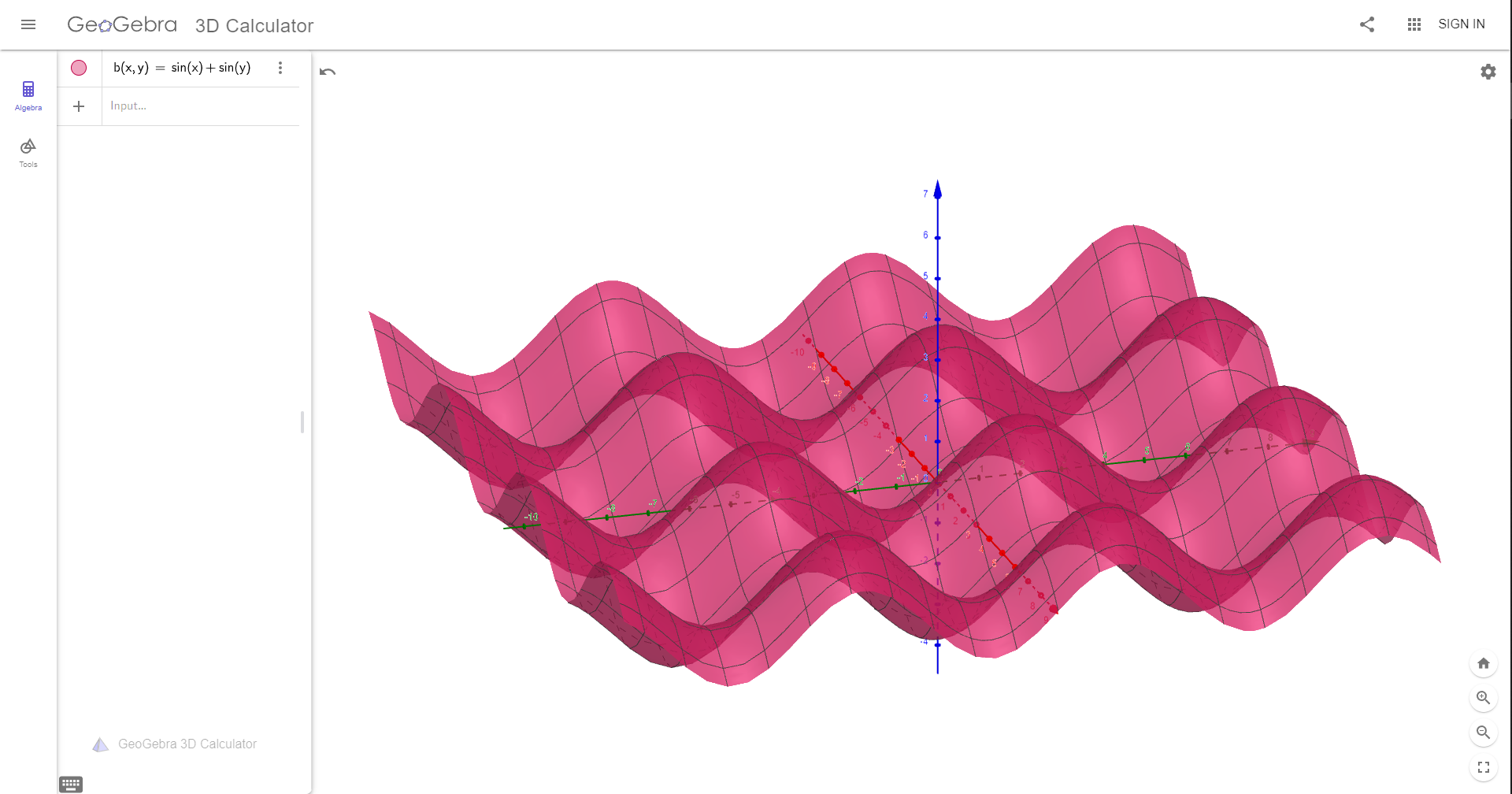
GNU-Plot is a great example of command interpreted plotting. It would be a good basis if the project was to be script/command based in a terminal. It is fairly simple to plot from data files as long as you know where your file is store since this could cause file-finding issues. The main problem with 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*

### 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.

## Summary

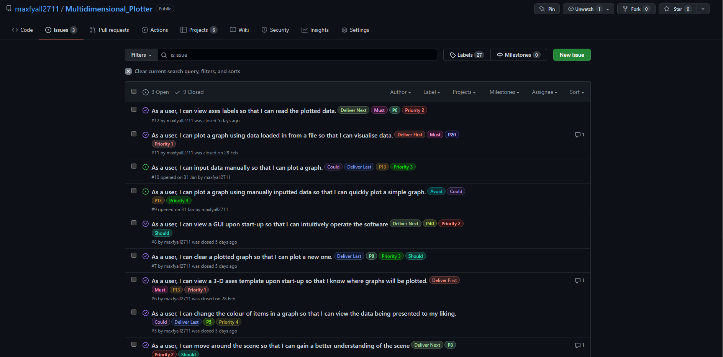
To summarize, there are many different applications that provide users with plotting capabilities. By experiencing the strengths and weaknesses of applications, it provided the researcher with a much better understanding of how to approach the creation of a program of this nature. What needed to be considered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

# 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.

## 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.

## 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.

# Design

## 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.

1. 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.

1. 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).

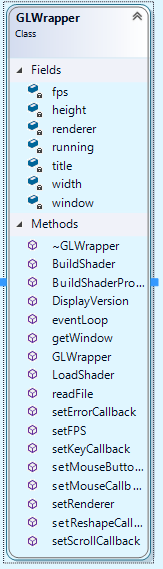
With the correct applications of the above, everything should be present for writing an OpenGL program.

## 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.

### 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.

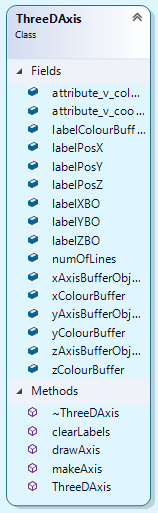


*Figure 9: GLFW Wrapper Class (Included in Appendix M)*

### 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)*

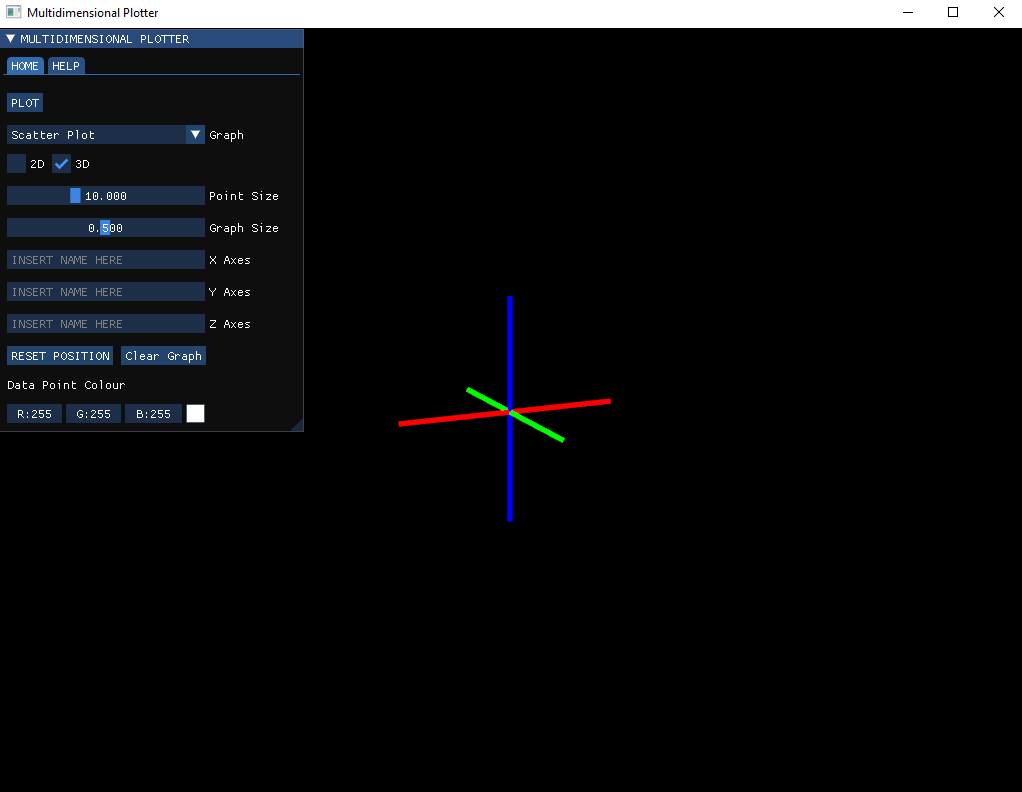
Using this structure, the researcher could call pre-defined methods and variables to create the object, so long as they create an instance of it inside the application code. This helped clean up the application code and prevented the re-use of certain code chunks. This also kept any object specific functions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

## 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.

## 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 11: Program Startup with slight mouse movement*

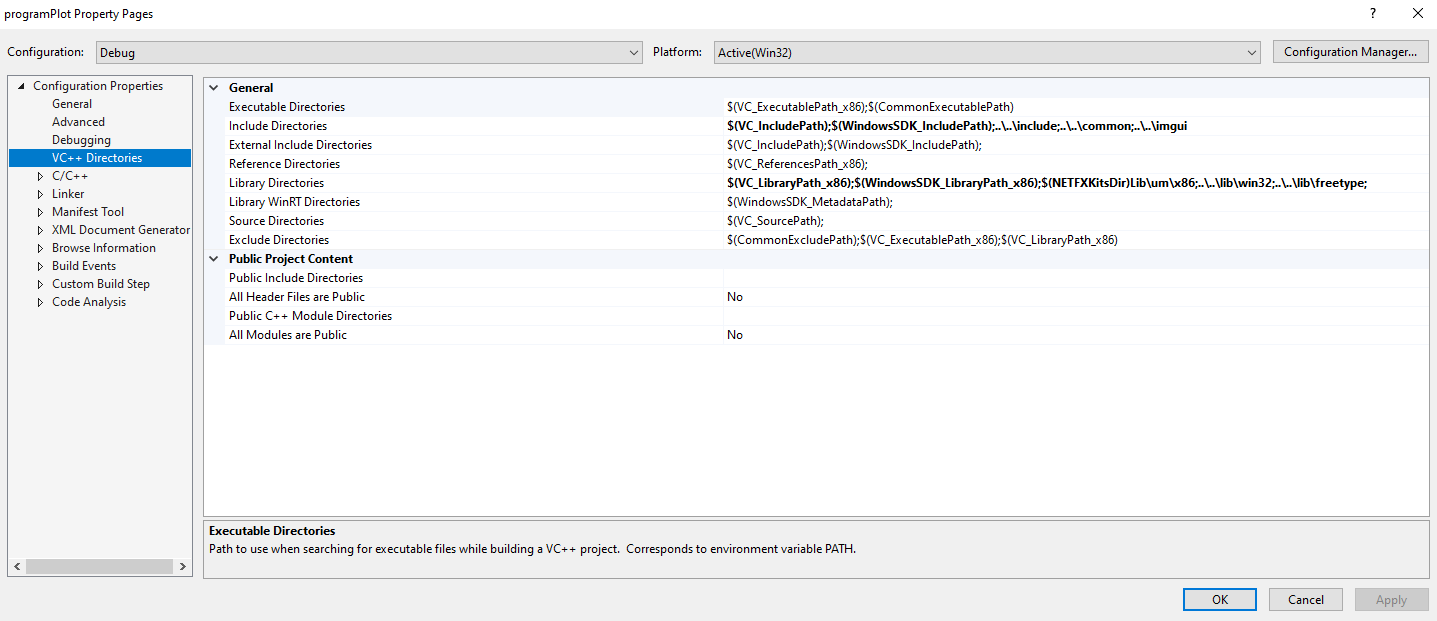
Initially, the researcher wanted a feature whereby users could insert data manually via the GUI and use this data to plot a graph. However, it was decided later on that this feature be cancelled as it was felt to be redundant with the system already in place i.e. loading data in from a text file.

# Implementation and Testing

## 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 12: Linking Include and Library Directories*

*(Included as Appendix N)*

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.

## Sprint 1

## Sprint 2

## Development Setbacks

You should describe the important aspects of implementation, testing, and debugging that you went through to produce your system. You can structure this in different ways, depending on the development methodology adopted and the needs of your project. You may wish to start with a review and overview of the main features to be implemented and a general, architectural overview of the system. You may then wish to walk through the major features, components, or sub-systems that were created, one after another. These could be sub-sections in your report, e.g., Feature X, Feature Y, etc. Or you may wish to present a time-based review of the implementation process, according to the stages you went through in your project plan. Indeed, if you have adopted an Agile approach, you may wish to structure your discussion around the various Sprints that were undertaken. In your discussion, highlight any important features that were implemented, any major problems that were encountered, and the workarounds that you produced. Your aim is to convince the reader that you are technically competent and that you are capable of problem solving and adapting to needs of the project. The amount / extent of technical contribution is also being assessed and the extent to which you have been able offer original ideas of your own. Regarding the amount of technical contribution. For example, a basic website, with a few, static pages is likely to be rated somewhat poorly. Instead, one would expect dynamic content, a database, more complex code and problems being solved, additional considerations for accessibility, usability, security, etc.

Regarding the implementation section. You may wish to illustrate your discussion with diagrams, or code snippets, that offer additional insights into your work or achievements. You may wish to emphasize user-centred processes, where applicable, and how the system evolved during implementation. For technically oriented projects, it is understood that you may wish to focus more on the performance, accuracy, reliability, or precision in your outcomes, including benchmarking against the work of others. For an additional layer of sophistication, any project can consider additional non-functional aspects of the system which are applicable, e.g., security, scalability, performance, usability, accessibility.

Later in your report, there is a related section: Description of Final Product. This later section is focused around providing a summary overview of your finished product. In contrast, the implementation section focuses on the stages that you went through to achieve and deliver it. There may be some areas of overlap, e.g., when you discuss the implementation of a particular user interface component, and you wish to use a screenshot to highlight the implementation choices made. Meanwhile, it turns out that a similar screenshot is necessary later in the Description of Final Product section, where you are simply presenting what the key aspect of the interface looks like. That is OK. There is just a difference of emphasis here.

For additional sophistication in your implementation, you should consider the use of software testing techniques, e.g., unit testing or similar. If so, the markers would need to see evidence of their use, e.g., in your source code or similar. In addition, you could consider traceability back to your original requirements, and verification or validation that they have been achieved.

As noted above, you may wish to include snippets of code in your report, to accompany your discussion of the implementation. Commonly, these may be included as screenshots of the relevant portions of code. It is best to keep these focused on specific areas of the code, e.g., it may be a specific method or a section of a method. For example, we are developing a web-based system which has a sequence of code for iterating through groups of product items. There is perhaps some reason why this code is noteworthy, e.g., it illustrates a novel approach or solves a tricky problem or is just something you are pleased with. Having discussed the feature, we wish to show a code snippet too. An example of this is below, e.g., please see Figure 2. Code Snippet. Iterating product options. below which illustrates the routine that was implemented to address this challenge. In the code snippet, you will see how the product items are iterated to complete the relevant basket page for the user.

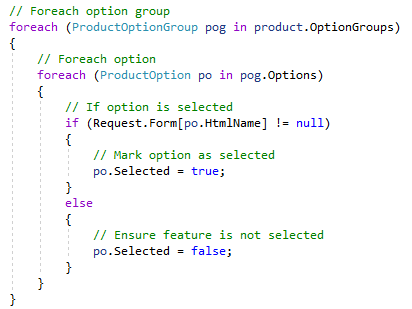


Figure . Code Snippet. Iterating product options.

Sometimes, you find that you need to include a larger image in your report that doesn’t fit easily into the two-column layout. In that case, you may be able to use Section Breaks in the document, to temporarily switch to a single-column layout and back again. For example, at this point in the report I have inserted a Continuous Section break immediately below this paragraph. Inside the new section, I have modified the layout to be single-column instead of double-column. I have included an example of a figure that spans the entire page. In this case, the figure is shown on the next page, because it doesn’t fit here. This has left a little but of a gap here on this page. This can be a common problem. Sometimes you can position things in such a way that the gaps are minimised. It is also worth noting that you don’t always need to place a figure immediately beside the text that it refers to. You could place a figure on the next page instead for example, whilst you continue to refer to it here, and afterwards. That way, you can fill up as much of the text here as you can too, without the need to have the figure placed in between. It is also possible to have several figures placed together on one page, and these can be referred to from relevant locations in your text. If you have large images that are impossible to fit into the report without being legible, you can include images in your appendices too.

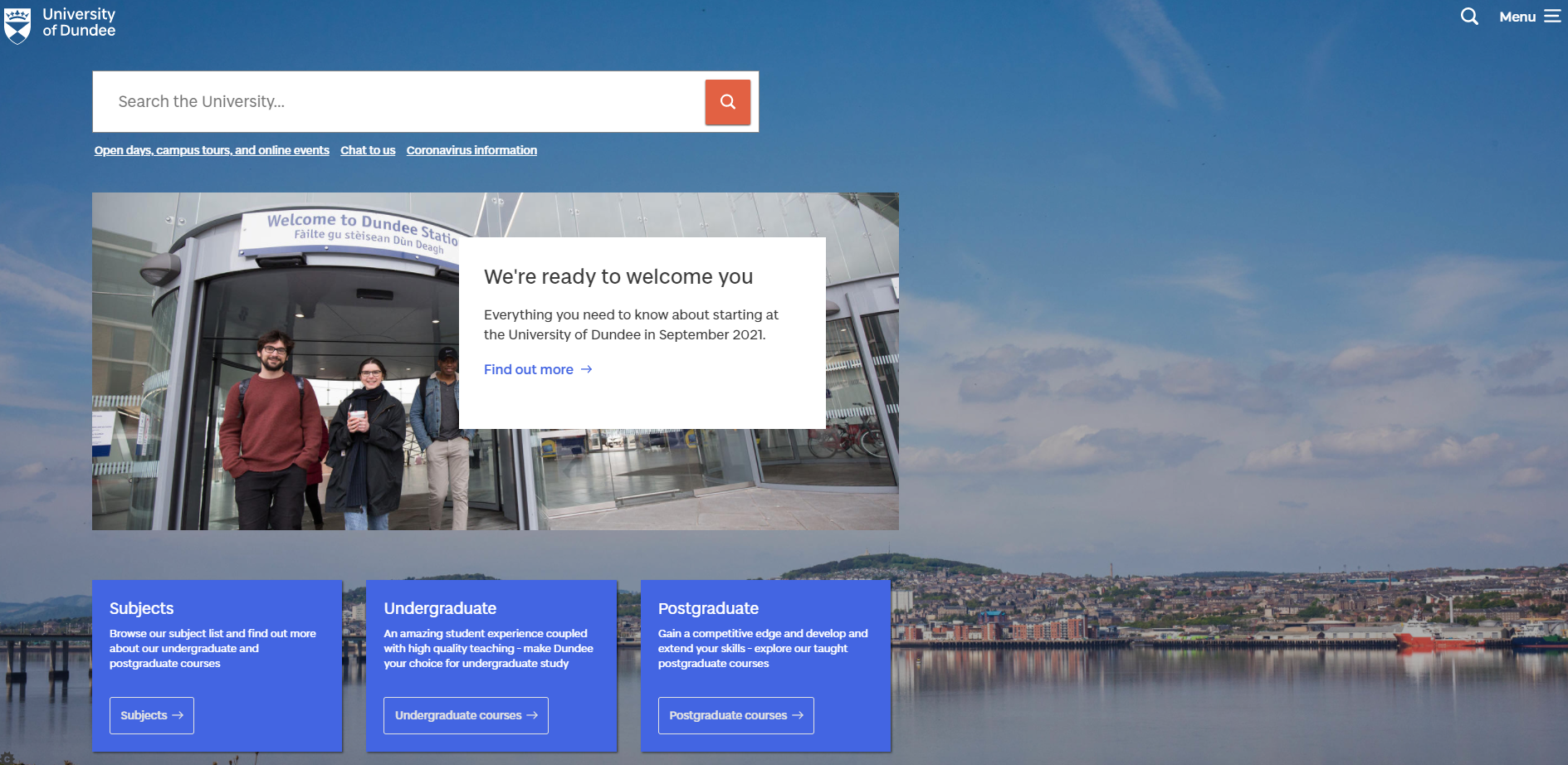


Figure . Random Image. In this case, a screenshot of the UoD website.

Now that I have finished what I want to do in this section, I have placed another continuous section break immediately after this sentence, which ends this single-column section and takes me back to the double-column layout afterwards.

# Evaluation / Testing

You must evaluate your system. This will be done in different ways depending on the project. For example, if you are developing a web application or app, it is common to do user testing and you may wish to seek feedback and comments from end users through interviews or questionnaires. If your project has a technical, non-user-based focus, your testing may focus more on benchmarking, comparing different algorithms or parameters, measuring performance or precision, etc. For any type of project, you can consider additional criteria where applicable, e.g., security, performance, accessibility, and computational efficiency. In the case of Cloud-based applications or services, one could also consider the cost implications (e.g., 'x' pence per query) and whether this has influenced the design and testing of the application.

Regardless of the project, you must describe the evaluation or testing of your system in your report, and this must include the following: a presentation of any relevant data; a discussion and analysis of the data; a discussion of the significant results and outcomes you have found. Ideally, you should consider any limitations in your evaluation and the extent to which your outcomes can be generalized to a wider ‘population’, or not.

Consider what you want to evaluate or test, and how you will achieve it. Develop the necessary evaluation plans / materials / methods, and make sure these are described in your report. Be mindful of ETHICS where required and make sure that the relevant Ethics documents are utilised, and it is clear where and how ethics has been adopted in your evaluation. Describe how your tests or evaluations were conducted. You can include the materials you have used in your appendices, e.g., test plans, evaluation checklists or tasks, copies of questionnaires used. Present and discuss the data in your report. You can include copies of the data in the Appendices too. Discuss the main outcome or findings from your evaluation / testing.

## Using subsections in your report

Remember, you can use subsections throughout your report to structure the content. This is often desirable to break up large expanses of text and to aid the reader too. Examples of subsections here, in the Evaluation section could be general such as Methodology or Results, and/or specific such as Usability, Performance, etc. Please remember to use the built-in styles for this that Word provides (Heading 1, Heading 2, etc.). This is necessary to ensure that your document is accessible.

### Here is a sub-subsection

You can use additional layers of hierarchy to progressively structure the content. In this case, if there was a subsection named Methodology, perhaps it could contain sub-subsections such as Participants, Tasks, Ethics, etc.

#### Be mindful of taking the structure too far

Whilst you can use as many hierarchies as you wish in structuring your content, there is usually a limit to what is useful in terms of readability. Aim to go no more than 3 layers deep in the hierarchy, if possible.

# Description of the final product

You should provide a clear description of what the final product looks like and what it does. You do not have to explore every minute detail of the system, you should attempt to convey the key, major areas of functionality. In some ways, you could consider this section to be a cut-down version of a user manual. Even in systems where there is no user interface, there may still be some general aspects that you can mention. However, if it is the case that this section of the report is just not relevant to your project, please just state that or omit this section.

When you are writing your report, you may find that the content of this section could overlap with earlier content in the report too, such as the implementation section. We want to avoid repetition in the report. At the same time, a degree of overlap is OK, bearing in mind that it is other people who are reading your report and they may benefit from a reminder, and a focused overview of what the final product looks like. As noted earlier, this section provides an overview of your finished product whereas earlier sections such as the implementation focus more on how you got to that point, i.e., the stages you went through, the decisions you made, and the problems you had to solve along the way.

# Appraisal

Provide a critical appraisal of the project. The question that I would pose to you here is as follows: if you were doing the whole project again, what would you do differently, what would you do the same, what advice would you give to others if they were doing the same project? Here you should reflect on the entirety of your project including your choice of technologies, your implementation decisions, and the project plan. With the benefit of hindsight, what are the lessons learned during the project and the evaluation of the final product and the process of its production (including a review of the plan and any deviations from it). Also consider what have been the most useful learning aspects for you.

NOTE: the appraisal section could potentially occur after the Summary and Conclusions below, or even as a sub-section within the Summary and Conclusions. See what works best for you and your advisor.

# Summary and Conclusions

Summarise the main points of what your project was and what the report has provided. Provide a summary. Describe the conclusions and outcomes that you have found.

# Future Work

What recommendations do you have for future work? Are there more features that need to be included? More testing? More evaluations? Are there follow-on projects or ideas that could be explored? Do you plan to do any more with the project yourself? Please discuss this here.

NOTE: this section could possibly appear as a sub-section within the preceding Summary and Conclusions.

## Acknowledgments

You can provide acknowledgements here to anyone who has been helpful in your project, or beyond. In some cases, the licensing of certain software products you have used may require you to acknowledge them here, e.g., in return for free use.

# 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.

# Appendices

A – MATLAB code used to create figure 2

B – Notes from background research

C – Email exchange

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 Linking (Include and Library)

The appendices to your report will not appear here, they are submitted separately. However, you can provide a summary / bulleted list of what the appendices are here if you wish.