Scatter3D: A 3D Graphing Application Using Unity

Moaz Adnan

MSc in Advanced Computer Science,
School of Computing Science, Newcastle University.
m.m.adnan2@newcastle.ac.uk

Abstract. Data visualization plays an important part in deriving meaningful conclusions from large datasets. With data production on the rise [13], graphing applications have become commonplace as a means of using colour, shape, size, and other attributes of a diagram to communicate quantitative content. This paper discusses the development of a 3D graphing application using Unity. The aim is to provide a curated set of features and graphs that work best when translated to 3D for data analysis.

Declaration: I declare that this dissertation represents my own work except where otherwise explicitly stated.

1 Introduction

Data are units of information that are gathered for analysing some type of object/behaviour. In science and engineering, these are a set of qualitative or quantitative variables that represent different properties of the subject matter. The end goal of any experiment is to derive meaningful conclusions from its findings, as a result, this may involve rigorous testing and a high volume of data to sift through.

When data is numerous, a series of rows and columns on a spreadsheet become counter-productive towards gaining useful insights. A much more effective way of presenting larger datasets is to employ visualization techniques. Data visualization entails gathering information and placing it into a diagrammatic format, such as a graph or a picture. With diagrams, these observations can be described using colour, shape, and size. Each property is more effective at conveying a ‘sense’ of change through visual cues than just plain text. They can also be combined to provide additional details about how data points relate to each other and the dataset as a whole (outliers). For graphical media, there are two ways of displaying data: 2D and 3D. Two Dimensional plots are concerned with the XY-axes, whereas three-dimensional plots have an extra depth component in the Z-axis. The aim of this project is to develop a 3D graphing application using the Unity game engine. It will allow users to construct multiple 3D plots from CSV files with an emphasis on interactivity through a mouse and keyboard.

1.1 Objectives

1. **Explore what current programs do and don’t offer when it comes to interactive tools for data visualization.** This provides a good starting point for curating a set of useful features that can be built upon if needed.

2. **Narrow down the types of graphs accommodated.** To keep the scope of this project within reason, more focus is placed on the navigation controls rather than having a wide range of figures.

3. **Determine the UI format for data acquisition and presentation.** As the data source is a CSV file, there needs to be a form-like interface which allows the user to load from it.

4. **Construct the 3D graphing application using Unity.** Once the requirements have been identified, the next step is to implement the program in the Unity game engine.

5. **Test the application.** The testing covers input validation, text overflow, and runtime performance under heavy loads (large files).

1.2 Document Structure

2. **Background** – Investigate some common services provided by contemporary graphing tools for displaying information in 3D. The purpose is to find a list of utilities that will guide development.

3. **Implementation –** Cover the design and incorporation of selected features. This includes how a component may or may not be feasible due to the limitations in Unity’s API.

4. **Evaluation –** Test the application for bugs. Undesirable behaviour can stem from a variety of actions, be it from the user or the program itself. By thoroughly assessing the software, proper safeguards can be introduced to prevent it from crashing.

5. **Conclusion –** Summarize work done. This section delineates all that was achieved over the course of this project. It examines what the application is capable of and what areas can be improved.

2 Background

Before discussing how different graphing libraries support 3D projection, it is important to understand *when* it should be considered. Case and Tullis [1] conducted an online study to determine if there was a measurable difference in perceiving 2D and 3D plots. Both versions of bar, column, donut, and pie charts were tested by asking participants to answer a series of questions using either format. After completion, each participant rated the charts in terms of ease/difficulty of reading, ease of learning, effectiveness, confidence, visual appeal, familiarity with all other versions, and overall appearance. Their findings showed 2D charts having a slight edge over their 3D counterparts [1].

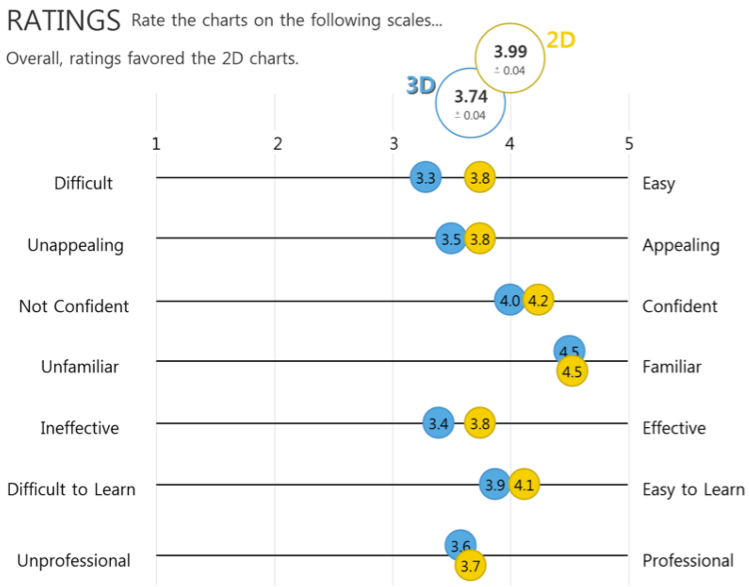


Figure 1. Shows the average ratings for the 2D and 3D charts [1].

The biggest gaps were in ease of reading and effectiveness. This may be attributed to the viewing angles of 3D charts and how they can make it harder to see where the data sits on an axis. So if 2D's the favourite, what reason is there to choose otherwise?

2.1 Why 3D?

The study employs 3D purely for style. There are only two scales and the depth component shows no information. A 3D plot works best when at least three variables are considered for the same diagram. Each axis has a list of values that make up a position in 3D space according to a set scale. These positions can be viewed from multiple angles in case two or more data points become muddled due to overlapping.

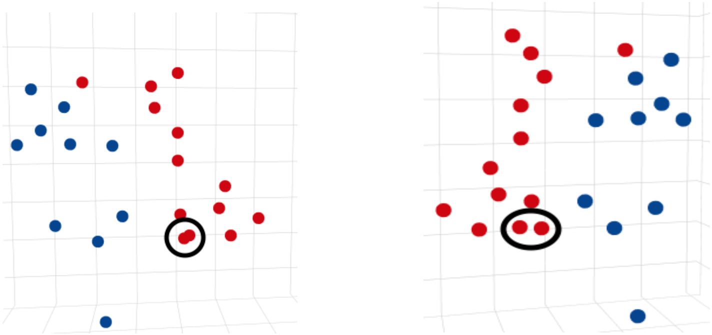


Figure 2. Shows the same scatter plot from different angles: the two data points aren't obscured in the second one [2].

Furthermore, any viewing problems become less of a concern when the plot is responsive, i.e., can rotate/translate. The human brain is quite adept at constructing 3D scenes out of a series of images taken from different angles (films) [14]. A moving graph is able to provide such images, whereas a static graph is not.

2.2 What works with 3D?

It is evident that navigation controls are crucial towards effectively using the depth component in 3D, and that the format should be avoided unless three position scales are available. However, not all graphs lend themselves to 3D even with movement; whether interpretation is hindered or not is entirely determined by what type of data a graph accommodates. Take the bar and scatter plots for example; a bar graph compares the measure of a categorical dimension without any regard to how one may affect the other, so adding another axis isn't insightful, whereas the scatterplot focuses on the correlation between two or more attributes over a continuum, something that can be analysed from different perspectives [7, 8]. Consequently, correlation plots such as the scatter and heatmap work well as 3D models, which is why they're the target graphics for this project.

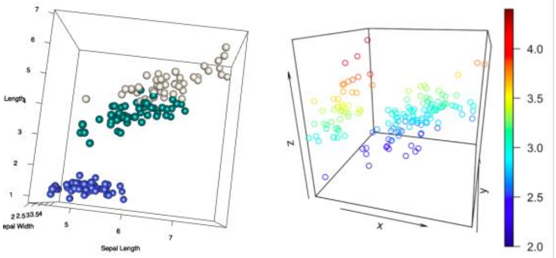


Figure 3. Shows a 3D scatterplot [7] and heatmap [9], respectively.

2.3 Graphing Programs

A detailed assessment of plots across multiple instances is not helpful. As the diagrams are standardized, it doesn't matter which tool is used as long as the plot is available. Instead, a better approach is to focus on features that use 3D to enhance data analysis and readability. For this purpose, two of the most popular programs are RGL and Plotly [3]. They have a comprehensive line up of utilities that target 3D visualization with interactivity in mind.

1. **RGL** is a 3D, real-time rendering system for the R programming language. It offers medium to high level functions for creating interactive graphics [4].

   - **Data Analysis -** The plots can be rotated, scaled, and viewed side by side in their own windows. A full range of motion ensures that data can be analysed from whatever perspective the user wants. Additionally, having multiple windows makes it easier to compare two datasets.

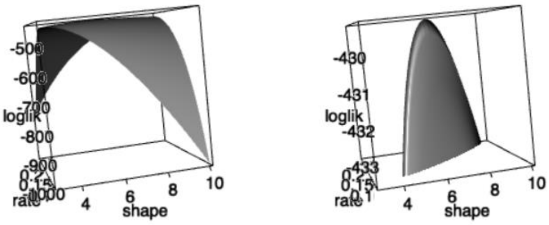


Figure 4. Shows two surface plots with boundaries marked by an enclosure [4].

   - **Data readability -** High resolution 3D models with lit textures keep the images well-defined even under crowded conditions. However, there are no hover text options to view information about individual points.

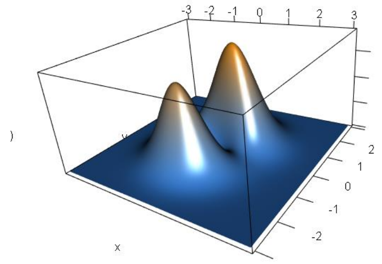


Figure 5. Shows a 3D surface plot with lit textures [5].

2. **Plotly** is a visualization platform with API support for different programming languages, R included. It has a suite of interactive graphs for both 2D and 3D formats.

   - **Data Analysis -** The plots can be moved, scaled, and displayed on separate windows. With multi-plot arrangements, each graph is treated as a distinct canvas with its own navigation controls.

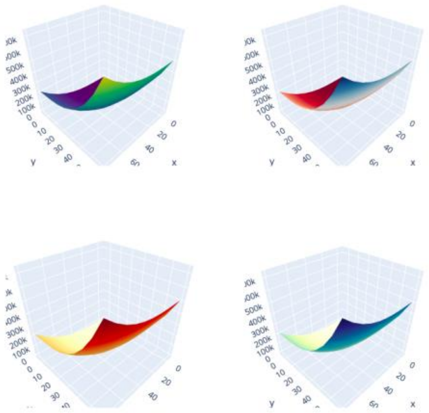


Figure 6. Shows four different surface plots. They can move independently while sharing a screen [6].

   - **Data readability –** Produces Pseudo 3D models that end up losing more definition than RGL when overcrowded. On the other hand, hover text is available with visual feedback when a point is examined.

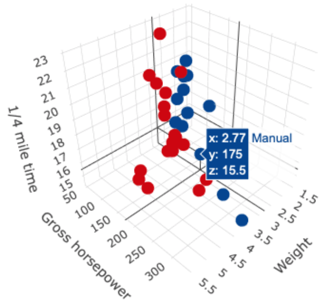


Figure 7. Shows the hover text feature in Plotly; a panel displays information about the data point [2].

2.4 Feature Set

The following table comprises all the selected features that are expected in the program. Nevertheless, the final version is subject to change based on Unity’s limitations and what the time frame allows.

Table 1. Proposed Features

Reference No.	Feature	Description
F1	Navigation Controls	To rotate and scale the graphs from different orientations.
F2	Hover Text	To view data values in 3D space.
F3	Multiple Plots	To compare multiple datasets
F4	Scatter/Heatmap	Graphs which benefit the most from a depth component.
F5	Screenshot	To save an image of the plot
F6	CSV Loader	To load the dataset
F7	Colour Picker	To colour code data points.

The reference numbers act as placeholders to provide context in later sections.

3 Implementation

There are three parts to the application: acquisition, preparation, and presentation. Each step has a series of UI elements that manipulate the data in either format or appearance before it reaches the screen. These elements are driven by a combination of Unity’s APIs and custom functions written in C#. To keep the description centred, their design and underlying logic are kept together while discussing the feature set (see Table 1).

3.1 UI Design Rules

Since there is no coding involved to use the application, the UI acts as the intermediary between the user and the program. As such, it is important that its design does not obstruct the intended experience. The measure of how “good” an interface is requires user participation and feedback. However, when availability is an issue, the next best thing is a set of guidelines that detail what to note during development.

Table 2. UI design principles comprised from recommendations mentioned in the following source [10].

Reference No.	Rule	Rationale
R1	Create an easy-to-navigate interface	Allow users to navigate easily by providing points of reference as they move through the interface. These are visual cues, such as a highlighted button, which inform the user of what action they're currently doing.
R2	Give informative feedback	For every user action, the UI should show a meaningful, clean reaction. A system with varied feedback helps users achieve their goals without friction.
R3	Remove the clutter	Examine elements based on how valuable they are to the user. Strive to design UI in a way that all information presented on the screen will be useful and relevant.
R4	KISS (Keep It Simple and Short)	Reduce the number of steps taken to perform a task if possible.
R5	Functional consistency	The behaviour of buttons and other UI elements should not change within the program. Users don't want surprises or variations in familiar behaviour.

The rules listed in Table 2 are good design practices for emphasizing a clean and immersive interface. They will ensure that the system strikes a balance between the navigation and feedback options offered to the user.

3.2 Data Acquisition

Data acquisition occurs when the source file, a CSV in this case, is loaded from the computer. This file format stores plain text as a series of rows with each entry delimited by a comma. The information is converted to a set of key-value pairs for structuring the data as a table where the attributes are the column names and their values are the items. The UI element for this task is a form (F6) which accepts the absolute file path as an input argument from the user. Once provided, it responds with a message telling if the procedure was successful (R2); if it was, then the user can proceed, and if not, then attempts are made to identify the issue (R1).

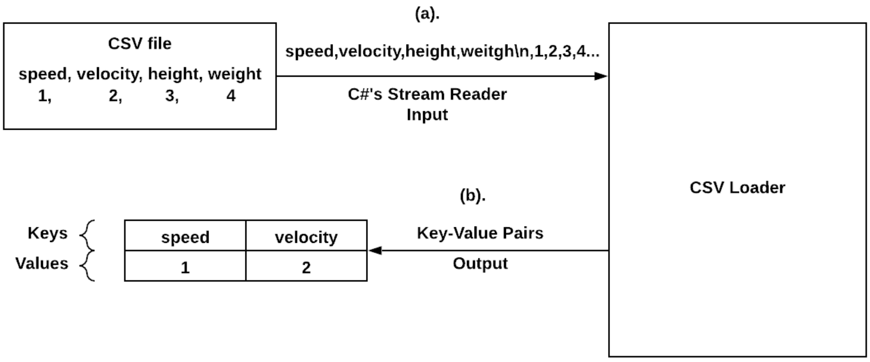
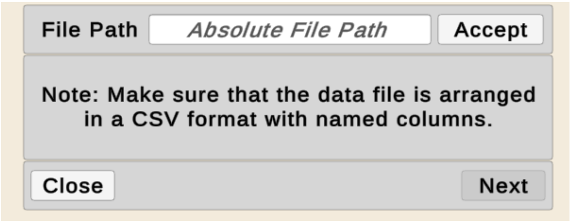


Figure 8. Shows the input and output for the CSV Loader. The data is read as a single string from the CSV file (a). The output is a dictionary of key-value pairs (b).

The reason for using the *absolute path* is because a pre-built file browser is not available in Unity. Every interactive UI element on screen is attached to some function that gets triggered by user activity. They can be merged to make more sophisticated components, like a file browser, but the end result requires more time and resources. The current approach is much simpler with only two steps for the user to follow: copy and paste (R4). However, both actions *can only be performed via hotkeys* because their respective mouse prompts would have to be built from scratch to work.

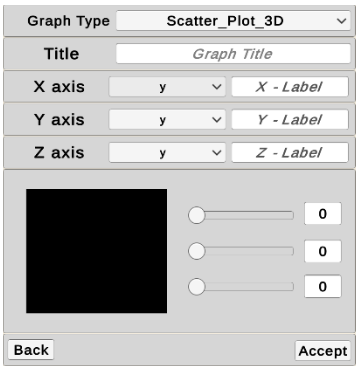


Note: Make sure that the data file is arranged in a CSV format with named columns.

Figure 9. Shows the CSV Loader form with a message and input field. Notice how colour codes are used to indicate different button states: "Next" is inactive while the others are not.

3.3 Data Processing

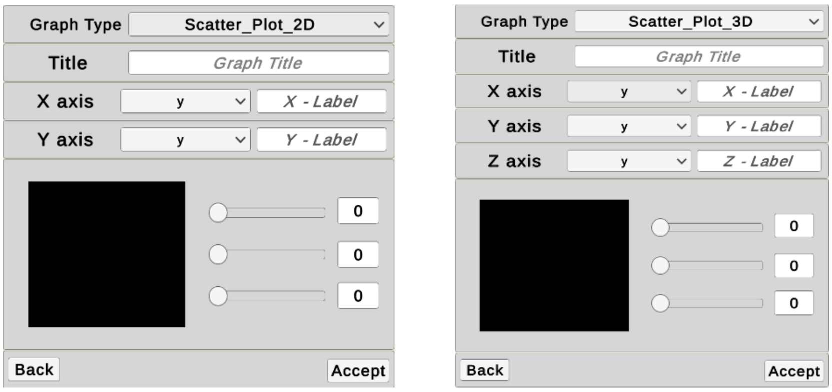
After a CSV file is successfully loaded, a new form appears with options for setting the plot's type, fields, colour, axis labels, and title. The field names are presented as dropdown lists where each entry corresponds to a column from the source file. The title and axis labels are input elements that act as tags when multiple plots are created. Colour selection is done through a picker interface (F7) which has a preview window and slider controls for configuring RGB values.



Graph Type: Scatter_Plot_3D
Title: Graph Title
X axis: y, X - Label
Y axis: y, Y - Label
Z axis: y, Z - Label
0
0
0

Figure 10. Shows the "Options" menu with the colour picker.

The featured graph types are the scatter plot, both 2D and 3D, and the heatmap (F4). Their underlying assets are the same with only slight variations in either their axis or the colour scheme. The UI form changes what options are accessible to the user based on the selected version: if 2D is chosen, then the z-axis disappears (R1).



Graph Type: Scatter_Plot_2D
Title: Graph Title
X axis: y, X - Label
Y axis: y, Y - Label
0
0
0

Graph Type: Scatter_Plot_3D
Title: Graph Title
X axis: y, X - Label
Y axis: y, Y - Label
Z axis: y, Z - Label
0
0
0

Figure 11. Shows the form menu for different graph types.

In order to streamline the support for multiple graphs (F3), the factory design pattern is employed to supplement the coding logic. Each type has a different set of parameters for instantiation (see Figure 11), but they all share the same assets and navigation controls. The factory pattern [15] works by grouping the creation methods of all objects from the same parent into a single class. The parent serves as the foundation for each child and as the access point for common functions. This class can then be used to generate different graphs according to their respective parameters. The factory design affords a plug-and-play upgrade path for new additions without fragmenting the code base.

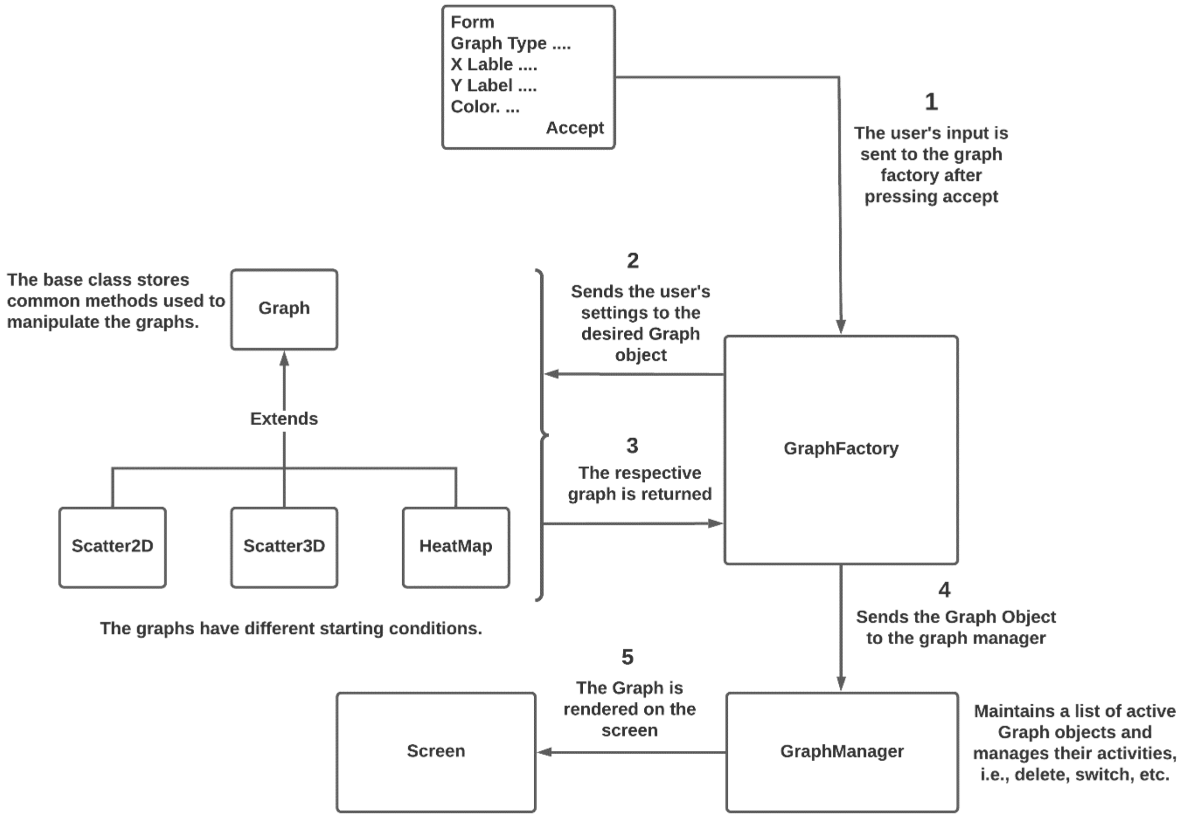

Form
Graph Type ....
X Lable ....
Y Label ....
Color. ...
Accept
1
The user's input is sent to the graph factory after pressing accept
The base class stores common methods used to manipulate the graphs.
Graph
2
Sends the user's settings to the desired Graph object
Extends
3
The respective graph is returned
GraphFactory
Scatter2D
Scatter3D
HeatMap
4
Sends the Graph Object to the graph manager
The graphs have different starting conditions.
5
The Graph is rendered on the screen
Screen
GraphManager
Maintains a list of active Graph objects and manages their activities, i.e., delete, switch, etc.

Figure 12. Shows the data-flow diagram from the user's input to the screen.

3.4 Data Presentation

The final step in the pipeline is to generate the graph on-screen. After the factory has returned the relevant object, it is handed over to the graph manager for further processing (see Figure 12). Any changes to the graph(s) beyond this point are done through the manager, rendering included. The graph manager employs three methods to create the 3D models: *getResources*, *findIncrements*, and *generateAxes*. All graph objects have their own definitions for these functions where needed.

1. getResources – This method retrieves the assets for the 3D model. When executed, it requests the resource manager to send over the objects that represent the data points and the surrounding graph. As there can be many similar 3D models on the screen at once (all data points circular), the resource manager stores a reference to the original asset and returns a copy when asked. By doing so, any accidental changes made to a copy can be reverted by producing another.

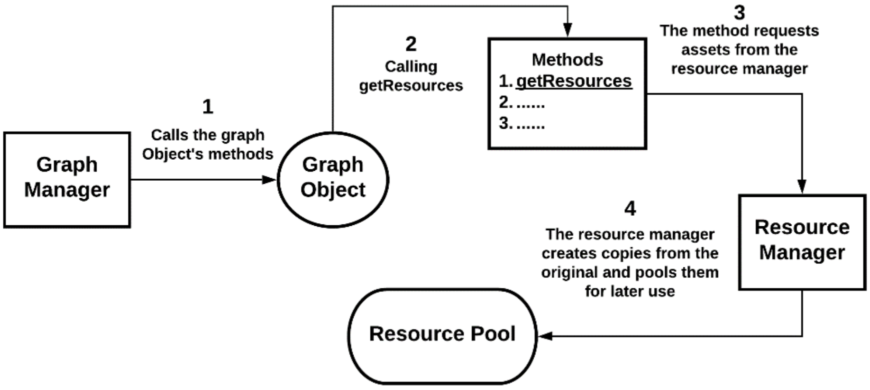

Graph Manager
1
Calls the graph Object's methods
Graph Object
2
Calling getResources
Methods
1. getResources
2. ......
3. ......
3
The method requests assets from the resource manager
Resource Manager
4
The resource manager creates copies from the original and pools them for later use
Resource Pool

Figure 13. Shows the execution order for getting resources.

2. findIncrements – This method normalizes the data values along the axes. Relative scaling for each axis is important to fit the data inside the graph. It's common for the dataset to have a wide variety of ranges that exceed the number of graduations present. The following formula ensures that the data point models stay within the bounds of the graph while maintaining their local distances from each other.

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

3. generateAxes – This method fills in the axis' divisions, such as the x, y, and z ranges. These are the normalized values calculated in the previous function. The graph is now ready for the screen.

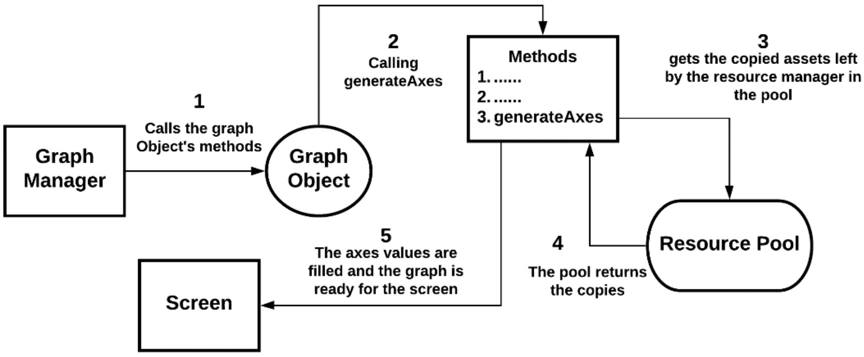
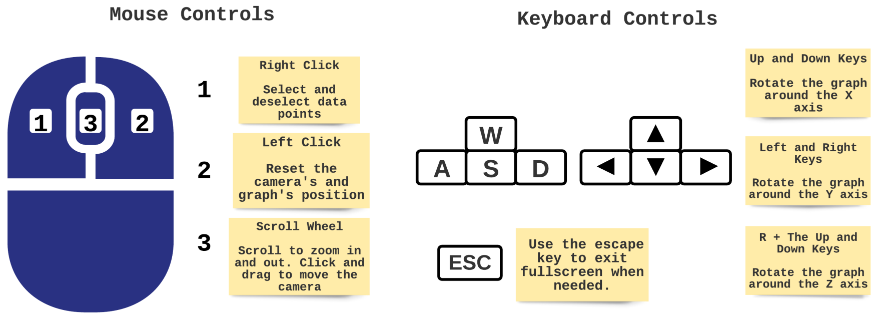

Graph Manager
1
Calls the graph Object's methods
Graph Object
2
Calling generateAxes
Methods
1. ......
2. ......
3. generateAxes
3
gets the copied assets left by the resource manager in the pool
Resource Pool
4
The pool returns the copies
5
The axes values are filled and the graph is ready for the screen
Screen

Figure 14. Shows the final steps before the graph is displayed.

3.4.1 Navigation Controls

The presence of navigation controls (F1) for 3D graphs has been emphasized throughout this document. Being able to view the diagram from different angles is what makes 3D useful. With that in mind, the user is afforded complete freedom to rotate and scale the graph via a keyboard and mouse, respectively. Unity provides a comprehensive input system that allows for seamless integration of peripherals.



Mouse Controls
1
3
2
1
Right Click
Select and deselect data points
2
Left Click
Reset the camera's and graph's position
3
Scroll Wheel
Scroll to zoom in and out. Click and drag to move the camera
Keyboard Controls
W
A
S
D
▲
◀
▼
▶
Up and Down Keys
Rotate the graph around the X axis
Left and Right Keys
Rotate the graph around the Y axis
ESC
Use the escape key to exit fullscreen when needed.
R + The Up and Down Keys
Rotate the graph around the Z axis

Figure 15. Shows the navigation controls for traversing the graph.

3.4.2 Multi-graph Arrangement

The advantage of having several graphs on-screen is that multiple datasets can be compared side by side. Both RGL and Plotly (refer to Section 2.3) support this feature with every plot having its own window and controls. However, a multi-graph arrangement finds its limitations in the screen real estate and peripherals.

- **Fighting for space –** The concept of sharing entails distributing resources amongst two or more entities. When dealing with multiple diagrams, a graphing program, such as Plotly, divides its canvas into chunks specified by the user. This often means that plots are shrunk to stay within the bounds of the drawing space while trying to keep the scale legible. A side effect of this resizing is that some parts of the diagram become obscured in the process.

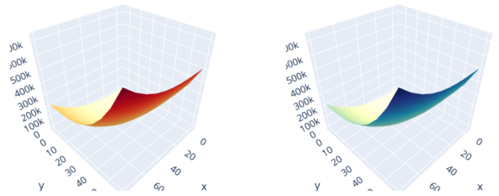


Figure 16. Shows a multi-plot arrangement in Plotly. Notice how the x and y axes are cut off to keep the surface plots on-screen [6].

- **One mouse cursor –** Analysing datasets from afar by observing their colour scheme and shape works well with multiple graphs. The user can simply rotate the figures to visualize change without having to zoom in on a specific point. However, a more granular approach for comparing individual values across different plots is seldom considered. Plotly's hover text feature doesn't "hold" the highlighted point when the cursor is moved onto the next diagram. The user would have to continually alternate between the points of interest as there's only one cursor to go around.

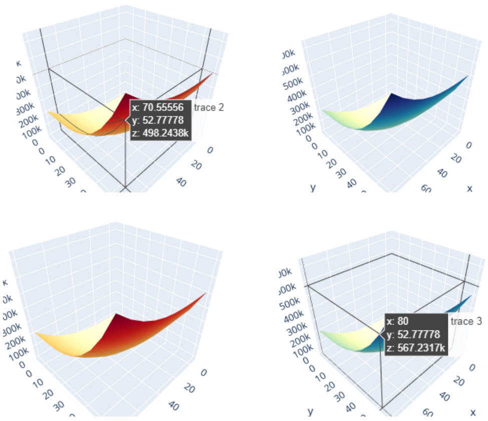


Figure 17. The hover text is not maintained across multiple graphs [6].

An alternative arrangement that circumvents both issues is to switch between the diagrams instead of keeping them all together. No scaling problems occur as only one graph is displayed at a time, and any persistence logic for the hover text can be incorporated in the switching mechanism. There is also the added benefit of resource pooling because the underlying 3D assets are the same: all datasets appear as dots regardless of which scatter plot they belong to.

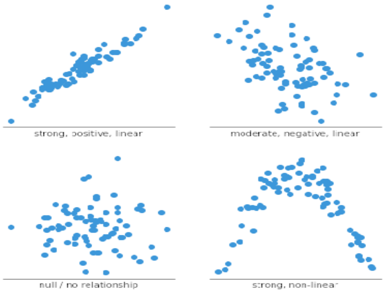


Figure 18. All graphs use the same asset but with different quantities [11].

This application adopts the switching arrangement with resource pooling to accommodate multiple plots. The switching mechanism is comprised of a dropdown list to access a graph by name, and two navigation buttons for moving back and forth from the current plot (R4).

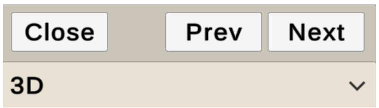


Figure 19. Shows the navigation panel for switching between graphs.

Resource pooling is very useful for optimising CPU workloads when a lot of asset-sharing occurs [16]. Take a graph with 700 data points as an example: when the plot is first rendered, Unity instantiates 700 spheres to represent each value. Instantiation calls can compromise performance if made too often, but this is unavoidable when the pool size is smaller than the number of data points, which is always the case at the beginning. Afterwards, let's say the user decides to upload another plot of the same size. Any further instantiation calls would be wasteful as only one diagram can be viewed at a time. The better way forward is to reuse all 700 points from the previous graph when the user switches to a different one. This method avoids unnecessary creation of objects and ends up saving memory.

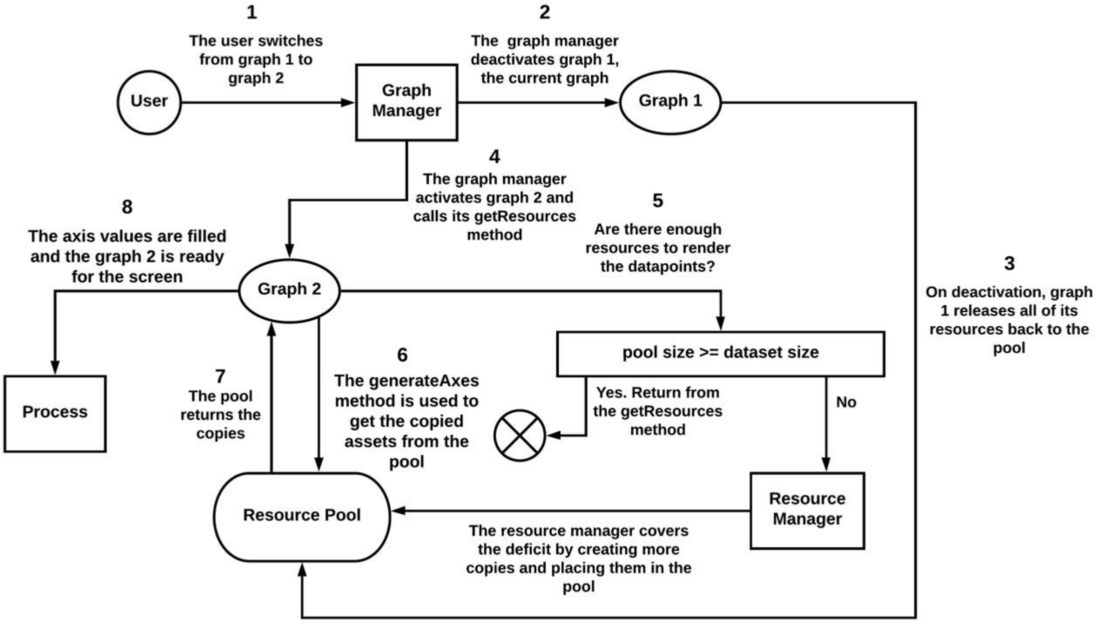
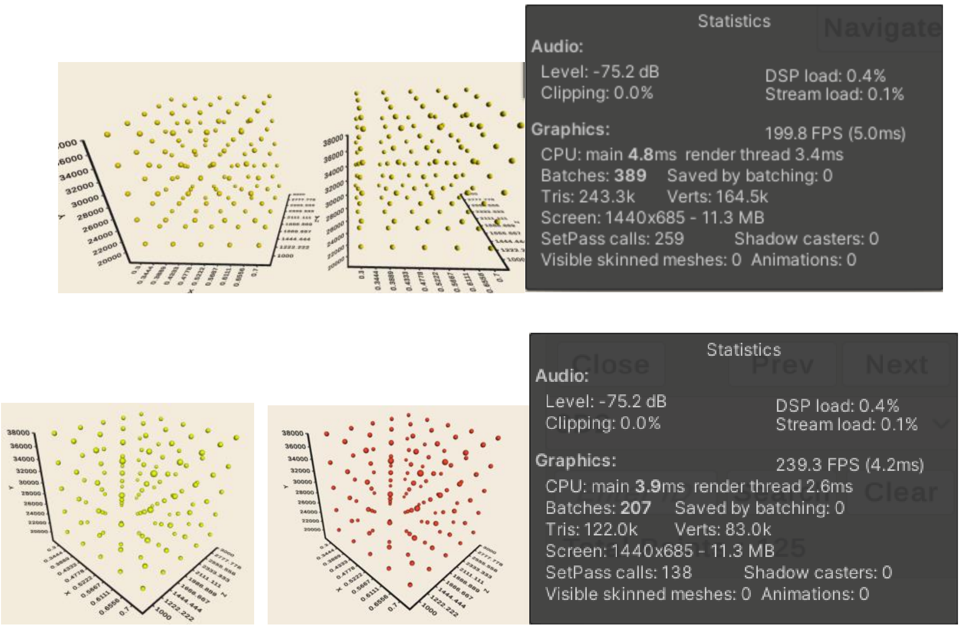

1
The user switches from graph 1 to graph 2
2
The graph manager deactivates graph 1, the current graph
User
Graph Manager
Graph 1
4
The graph manager activates graph 2 and calls its getResources method
5
Are there enough resources to render the datapoints?
8
The axis values are filled and the graph 2 is ready for the screen
3
On deactivation, graph 1 releases all of its resources back to the pool
Graph 2
pool size >= dataset size
6
The generateAxes method is used to get the copied assets from the pool
7
The pool returns the copies
Yes. Return from the getResources method
No
Process
Resource Pool
Resource Manager
The resource manager covers the deficit by creating more copies and placing them in the pool

Figure 20. Shows how resource pooling is used to switch between plots.



```
Statistics
Audio:
  Level: -75.2 dB              DSP load: 0.4%
  Clipping: 0.0%               Stream load: 0.1%
Graphics:                      199.8 FPS (5.0ms)
  CPU: main 4.8ms  render thread 3.4ms
  Batches: 389    Saved by batching: 0
  Tris: 243.3k     Verts: 164.5k
  Screen: 1440x685 - 11.3 MB
  SetPass calls: 259        Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
```

```
Statistics
Audio:
  Level: -75.2 dB              DSP load: 0.4%
  Clipping: 0.0%               Stream load: 0.1%
Graphics:                      239.3 FPS (4.2ms)
  CPU: main 3.9ms  render thread 2.6ms
  Batches: 207    Saved by batching: 0
  Tris: 122.0k     Verts: 83.0k
  Screen: 1440x685 - 11.3 MB
  SetPass calls: 138        Shadow casters: 0
  Visible skinned meshes: 0  Animations: 0
```

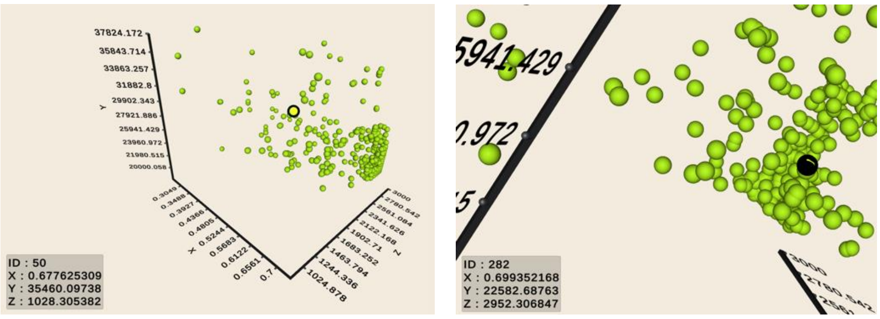
Figure 21. Screen sharing vs. switching: the top image shows two graphs on the same window. The bottom image shows two graphs after switching from yellow to red. Note how the FPS increases and the number of polygons (Tris) are cut in half (lower memory consumption).

3.4.2 Hover Text

Hover text (F2) is what brings focus to a specific value in a cluster of data points. It presents the numerical information on either one side of the screen or directly on top of the selected object (Plotly). This is typically followed by a visual indicator in the form of an outline (see Figure 7). The program implements hover text via shaders that govern how a 3D object is displayed in Unity. Shaders are a set of instructions which are executed at once for every pixel on the screen [12]. By using them, a great degree of customization can be achieved while accentuating a data point's position.

The highlighted effect is composed of a black line surrounding the 3D model with the colour getting brighter for better contrast. The model's silhouette can be seen even when obstructed by other objects. The information panel is locked to the lower left corner of the screen so that it doesn't get in the way of the cursor. Its size changes in accordance with the amount text to display as the user navigates the plot.

Unlike Plotly, it is possible to maintain the hover text by right-clicking the data point. Only one object is selectable at a time, but the user can still cycle through different values on the same information panel; it's just that the panel reverts to whatever element is currently in focus. When switching, the chosen point, if there is one, is saved inside the graph object and automatically emphasized on the way back.

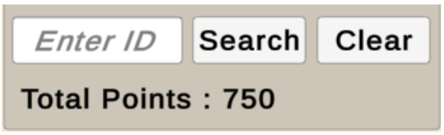


ID : 50
X : 0.677625309
Y : 35460.09738
Z : 1028.305382

ID : 282
X : 0.699352168
Y : 22582.68763
Z : 2952.306847

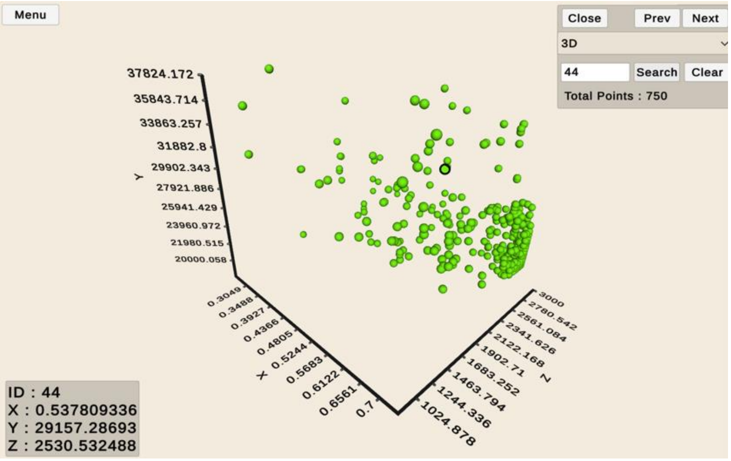
Figure 22. The first image shows a highlighted data point. The second image shows the silhouette of an obstructed data point.

There is also a search feature in place for locating data points by their IDs. The ID is the row number of a value in the CSV file. The ability to query the graph is handy when points are so closely packed together that some become unreachable via the mouse pointer.



Total Points : 750

Figure 23. Shows the search panel with a tally of all the points in the current graph



44

Total Points : 750

ID : 44
X : 0.537809336
Y : 29157.28693
Z : 2530.532488

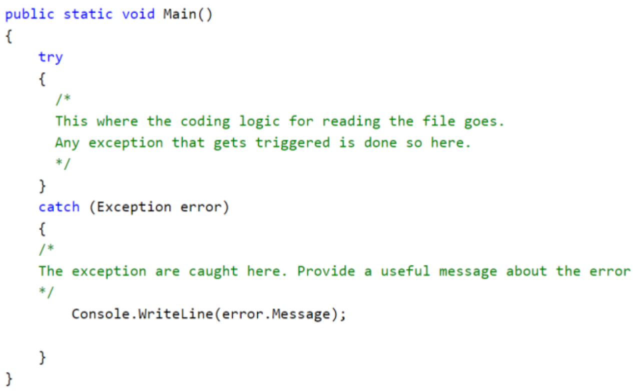
Figure 24. Shows a highlighted point after it was searched using its ID from the CSV file.

4 Evaluation

Assessing the application helps weed out inconsistencies which could at best be a nuisance and at worst brick the program. Planning for a wide variety of errors becomes increasingly difficult when a lot of things are happening. Fortunately, there's only one source of irregular behaviour that could potentially crash the application if left untreated: the user's input. The graphing software adheres to a strict CSV format with named columns followed by a list of numerical values. As such, input validation is conducted to ensure that the data file complies before any rendering occurs. Other areas such as text overflow and the application's runtime performance are also explored.

4.1 Input Validation

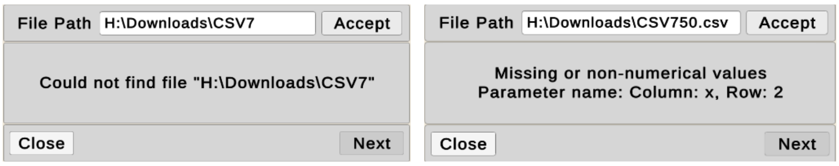
Input validation prevents improperly structured data from entering the graphing tool. The CSV Loader is stacked on top of C#'s built-in file reader that does the error handling. It has its own set of exceptions with configurable messages that cover an extensive list of fault states. Where needed, the exceptions class is extended to provide bespoke error prompts in an attempt to pinpoint the cause of the problem (R2).



```
public static void Main()
{
    try
    {
      /*
      This where the coding logic for reading the file goes.
      Any exception that gets triggered is done so here.
      */
    }
    catch (Exception error)
    {
    /*
    The exception are caught here. Provide a useful message about the error
    */
        Console.WriteLine(error.Message);

    }
}
```

Figure 25. Shows a try-catch block used for executing code that can throw an exception.



File Path H:\Downloads\CSV7

Could not find file "H:\Downloads\CSV7"

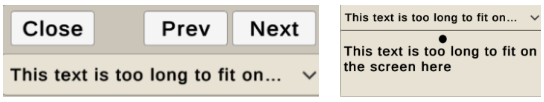
File Path H:\Downloads\CSV750.csv

Missing or non-numerical values
Parameter name: Column: x, Row: 2

Figure 26. Shows two error messages: the first one is caused from providing an incorrect file path, while the second is triggered due to a missing value in the CSV file (all columns must be of the same length).

4.2 Text Overflow

A consequence of having limited screen real estate is that the UI forms and their input elements have a fixed size to work with. In some cases, this may not be enough to accommodate longer strings of text. Legibility is directly affected when the text overflows and starts to obfuscate other interfaces. Reducing the font size alone becomes counterproductive if the text is no longer visible (too small). Instead, truncating the text on the panel and wrapping it when expanded avoids reducing the font size while keeping it within bounds. All text elements in Unity come with overflow settings which are exposed on the editor.



This text is too long to fit on...

This text is too long to fit on...

This text is too long to fit on the screen here

Figure 27. The first image shows truncated text on the panel. The second image shows its wrapped version after the dropdown is expanded.

4.3 Runtime Performance

The decision of switching between graphs with resource pooling means that the largest dataset determines the highest memory and CPU usage. As the pool is shared by all the plots, no extra memory is required until a new maximum size appears. When switching, each graph deallocates its resources for the next one. If the upcoming plot is larger than the pool, only the deficit, the difference between the pool's size and the dataset, is added from the resource manager before rendering it (see Figure 20). The deallocation, and the subsequent reallocation, makes switching one of the more expensive functions in the application. However, it is needed to ensure consistency across multiple graphs when the user interacts with them: choosing a data point, rotating the plot, etc.

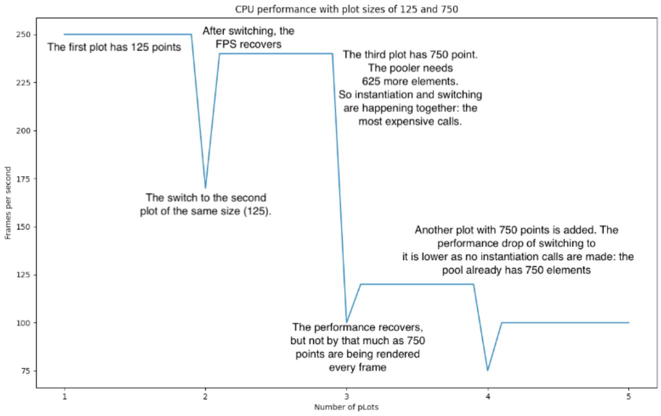


Figure 28. Shows the CPU performance in FPS. Notice how the FPS recovers after switching, indicating that the active graph utilizes more CPU resources than those in the background (allocation, deallocation).

5 Conclusion

The project's goal was to develop a 3D graphing application in Unity with an emphasis on controls that would assist data visualization. To that end, background research was conducted to ascertain a set of features common amongst contemporary plotting tools that supported the format. Afterwards, the types of graphs to accommodate were narrowed down based on what the surrounding literature deemed appropriate. Finally, the curated feature set (see Table 1) was implemented using Unity's APIs and the C# programming language.

F1 - Navigation Controls (Implemented) – The graph can be scaled, dragged, and rotated via a mouse and keyboard, respectively. The user is afforded full range of motion in that the plots can be viewed from any perspective regardless of the starting position.

F2 - Hover Text (Implemented) – The numerical information of a point is displayed on a panel by either hovering the mouse cursor on top it, or by selecting it with a right-click. Any point can be searched for through its ID (row number).

F3 - Multiple Plots (Implemented) – The application supports a multi-plot arrangement where the user can switch between graphs. All plots have the same control scheme but move independently of one another: their orientations are maintained.

F4 - Scatter/Heatmap (Implemented) - The application provides three types of graphs: a 2D scatter plot, a 3D scatter plot, and a 3D heatmap.

F5 - Screenshot (Not Implemented) – The screenshot utility wasn't included due to time constraints.

F6 - CSV Loader (Implemented) - A file reader for loading the data source file. It expects a CSV layout which it processes further into a list of key-value pairs for the user to choose from.

F7 - Colour Picker (Implemented) – A UI element for setting the colour scheme of the graph, barring the heatmap. The heatmap has a separate colour gamut.

In its current state, the application is serviceable for offering interactive 3D graphs, although in a very limited capacity, with freedom of movement at its forefront. By opting for switching, as opposed to screen sharing, multiple plots can be rendered with careful resource management and no scaling issues. Some future considerations would be to expand its repertoire of diagrams with other graphs that benefit from 3D, such as a surface or a contour plot, and to incorporate the missing screenshot feature as a means of exporting the graphs for later use.

Acknowledgments. I would like to thank my dissertation supervisor, Ken Pierce, for guiding me through this project.

References

1. Tullis, T. and Case, M., 2017. 2D vs. 3D Charts: Does 3D Representation Help or Hurt?. In: *User Experience Professionals Association 2017*. [online] Toronto: User Experience Professionals Association. Available at: <http://uxmetricsgeek.com/pubtype/cpaper/> [Accessed 11 July 2021].

2. Plotly.com. 2021. *3D Scatter Plots*. [online] Available at: <https://plotly.com/python/3d-scatter-plots/> [Accessed 11 July 2021].

3. Medium. 2021. *Advantages and Best Uses of Four Popular Data Visualization Tools*. [online] Available at: <https://medium.com/@ODSC/advantages-and-best-uses-of-four-popular-data-visualization-tools-63df88619662> [Accessed 11 July 2021].

4. Murdoch, D., 2021. *rgl Overview*. [online] Cran.r-project.org. Available at: <https://cran.r-project.org/web/packages/rgl/vignettes/rgl.html> [Accessed 11 July 2021].

5. Nyffenegger, R., 2021. *R package: rgl*. [online] Renenyffenegger.ch. Available at: <https://renenyffenegger.ch/notes/development/languages/R/packages/rgl/index> [Accessed 11 July 2021].

6. Plotly.com. 2021. *3D Subplots*. [online] Available at: <https://plotly.com/python/3d-subplots/> [Accessed 11 July 2021].

7. Healy, Y., 2021. *The issue with 3D in data visualization*. [online] Data-to-viz.com. Available at: <https://www.data-to-viz.com/caveat/3d.html> [Accessed 11 July 2021].

8. Visual-design.net. 2021. *How to Choose the Most Appropriate Chart?*. [online] Available at: <https://www.visual-design.net/post/choose-the-right-chart> [Accessed 12 July 2021].

9. Sthda.com. 2021. *Impressive package for 3D and 4D graph - R software and data visualization - Easy Guides - Wiki - STHDA*. [online] Available at: <http://www.sthda.com/english/wiki/impressive-package-for-3d-and-4d-graph-r-software-and-data-visualization> [Accessed 12 July 2021].

10. Babich, N., 2021. *The 4 Golden Rules of UI Design | Adobe XD Ideas*. [online] Ideas. Available at: <https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/> [Accessed 13 July 2021].

11. Yi, M., 2021. *A Complete Guide to Scatter Plots*. [online] Chartio. Available at: <https://chartio.com/learn/charts/what-is-a-scatter-plot/> [Accessed 17 July 2021].

12. Gonzalez Vivo, P. and Lowe, J., 2021. *The Book of Shaders*. [online] The Book of Shaders. Available at: <https://thebookofshaders.com/01/> [Accessed 18 July 2021].

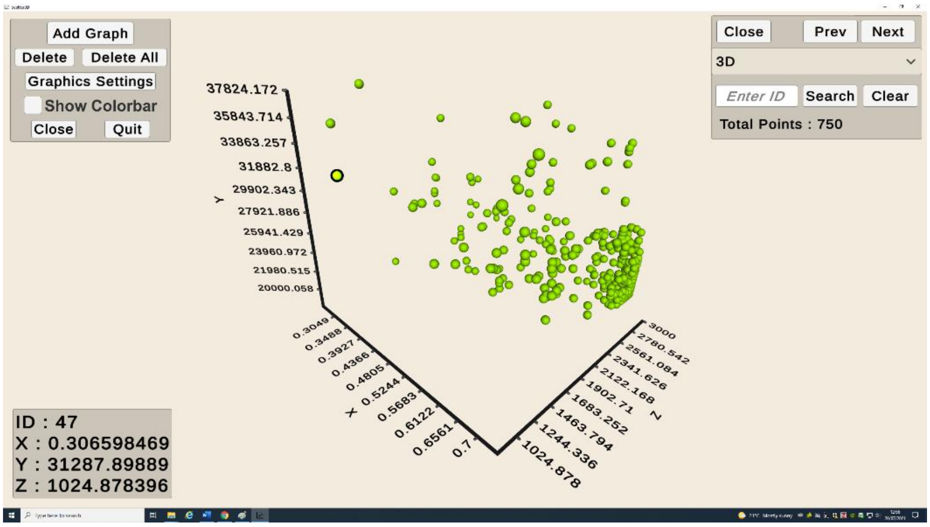
13. Holst, A., 2021. *Total data volume worldwide 2010-2025 | Statista*. [online] Statista. Available at: <https://www.statista.com/statistics/871513/worldwide-data-created/> [Accessed 20 July 2021].

14. Welchman, A., 2016. The Human Brain in Depth: How We See in 3D. AnnualReview of Vision Science, [online] 2(1), pp.345-376. Available at: <http://vision.annualreviews.org> [Accessed 30 July 2021].

15. Gamma, E., Helm, R., Johnson, R. E.,, Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley. ISBN: 978-0-201-63361-0

16. Placzek, M., 2021. *Object Pooling in Unity*. [online] raywenderlich.com. Available at: <https://www.raywenderlich.com/847-object-pooling-in-unity> [Accessed 31 July 2021].

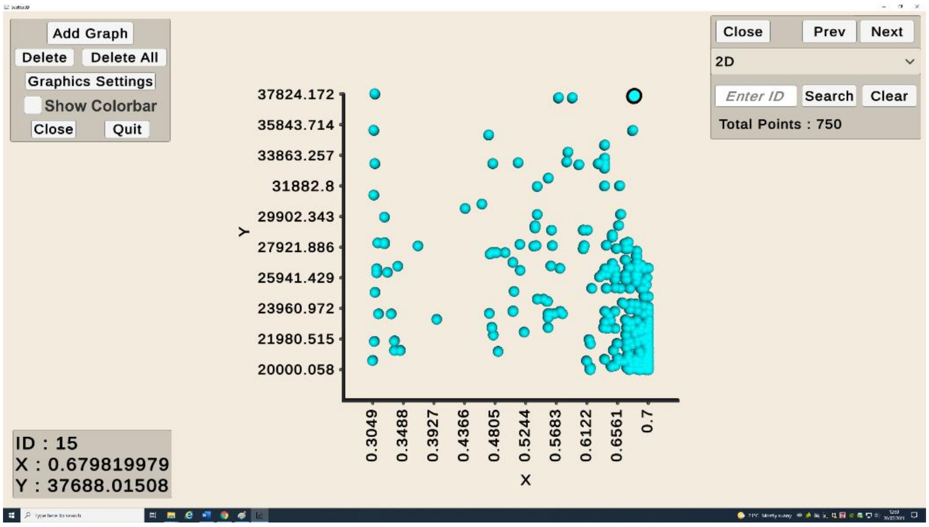
Appendix A – Graph Photos and The Navigation Controls



Total Points : 750

ID : 47
X : 0.306598469
Y : 31287.89889
Z : 1024.878396

Figure 29. Shows a 3D scatter plot.

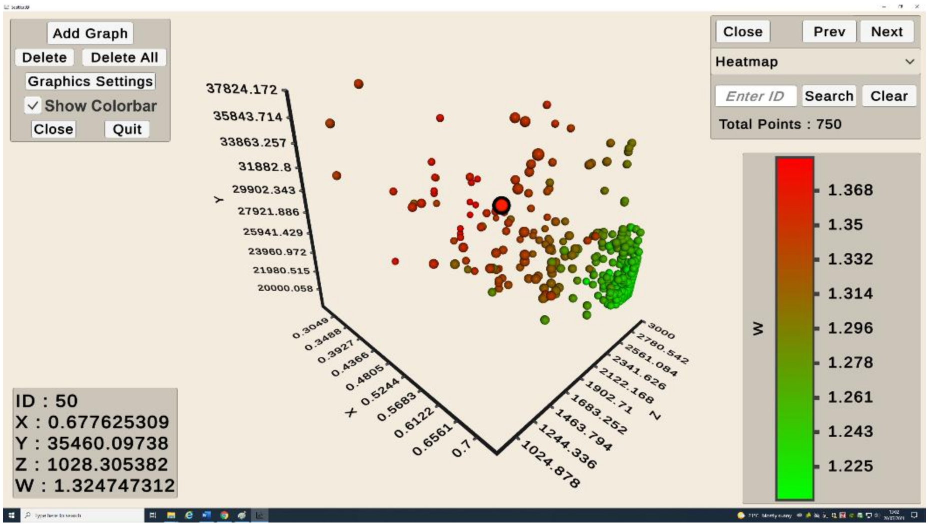


2D

Total Points : 750

ID : 15
X : 0.679819979
Y : 37688.01508

Figure 30. Shows a 2D scatter plot.



Heatmap

Total Points : 750

ID : 50
X : 0.677625309
Y : 35460.09738
Z : 1028.305382
W : 1.324747312

Figure 31. Shows a 3D heatmap.

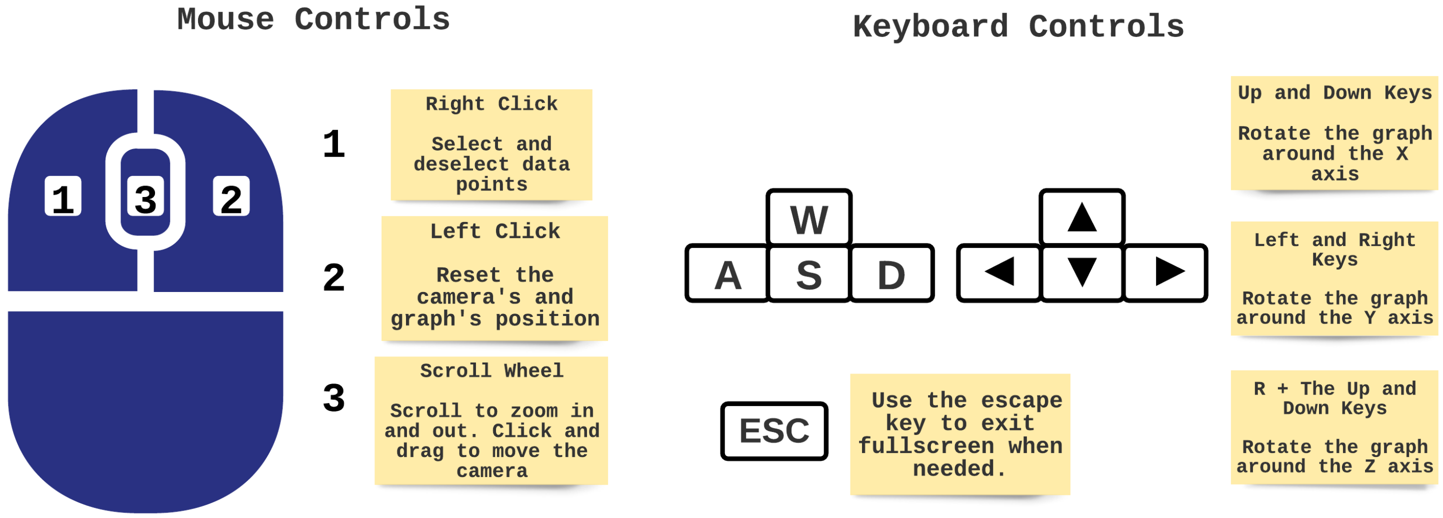

Mouse Controls
1
3
2
1
Right Click
Select and deselect data points
2
Left Click
Reset the camera's and graph's position
3
Scroll Wheel
Scroll to zoom in and out. Click and drag to move the camera
Keyboard Controls
W
A
S
D
▲
◀
▼
▶
ESC
Use the escape key to exit fullscreen when needed.
Up and Down Keys
Rotate the graph around the X axis
Left and Right Keys
Rotate the graph around the Y axis
R + The Up and Down Keys
Rotate the graph around the Z axis

Figure 32. Shows the mouse and keyboard controls.