The Development of a Web-based Multidimensional Data Plotter

AC40001 Honours Project

Sameer Al Harbi
ID: 190007048/1
BSc (Hons) Computing Science
Supervisor: Dr. Iain Martin
University of Dundee
Dundee, UK

ABSTRACT

This **should** be a *summary* of 250 words that summarises your project, including your aims and contributions that will arise from it. Think of it as a summary of your introduction.

1 INTRODUCTION

Data visualisation is an important step in the data analysis process. An importance which is only further compounded by the rise of Big Data and the subsequent need for extracting insight from exponentially larger and more complex data sets. This insight, whether it's used to highlight results in an easy to digest manner or to identify patterns otherwise hidden by the complexity of a dataset- can bring value across a project's team, with each member being able to extract value specific to their own tasks.

For all members working on a project (With the word project being used in the context of any work done that uses data within an industrial or academic sector) visualisation is the main "window" through which insight and thus value is accessed. Analysis is done towards some end- and that end must result in the return of something- which often must be visually consumed (See Big Data above). In many cases visualisation in itself can act as a form of analysis, both creating insight and communicating it when applied to a dataset.

To be more specific, it has been found that there is a definite market among both technical and non-technical professionals for tools that can unlock the results of analysis.

The aim of this project was to then create an accessible, web-based application that can be used by anyone to quickly plot data in multiple dimensions, regardless of programming skill. This paper highlights the decisions that lead to this final

2 BACKGROUND

The result of background research done for this project can be subdivided into two distinct sections being Related Work and Application.

2.1 Related Work

This section highlights the wider context that affected the structure and what the application was planned to do. The research identified here acted to directly influence what kind of application was developed, for what exact purpose and for what users.

2.1.1 Project Context. One of the first steps for this project was to place and understand the needs of this project through the lens of some wider context. By what metric should potential requirements be prioritised? and what those requirements should even be? Those are some of the questions whose answers will depend on the context that they are looked at. After some consideration, it was decided by the student to focus on the current business industry, and it's needs as this context.

> This context (the business market) sets the aim of this project to then be an application that would be considered a valuable market entry compared to its competitors. It should also preferably, address some specific pain points of the market that are not fully covered by some other solution on the market.

The alternative would also have been to look at this project from a more academic, literature driven view- where a project's success would depend on more research oriented exploratory work. This alternative though, had some drawbacks that ultimately resulted in the selection of a Business Context. Mainly, a focus on existing market solutions allowed a more logical selection of features to develop for a practical development project. Although it should be stressed that this doesn't mean that the academic background for this project was disregarded (See Section 2.3)- It's just that the focus was on the business-based requirements first and foremost.

2.1.2 Market Research. With the context set- It was then considered that an analysis of the current options on the market would be a prudent next step to start understanding what a successful project should incorporate. This analysis was open-ended and focused on identifying key points that differentiated each product apart.

Detailed notes on each competitor can be found in the Appendix [A], which are although not exhaustive, nevertheless highlight some important aspect of the overall market- But the main important trends and highlights identified are the following:

Firstly, it was found that most solutions are highly technical and need at least some degree of programming knowledge to use. A rough relationship would be that the complexity of using the solution and its capability is inversely proportional. Feature rich solutions need programming experience while the ones that don't, have more limited features, and are more likely to be not free to use. Obviously, there were outliers such as MATLAB which has both a UI interface and a programming interface, but this relationship still stands.

Another interesting point identified is that The Stack Overflow 2021 Developer Survey [1] shows that a large majority of the most used libraires and frameworks were for data analysis or data-based projects, specifically Python. Which is also the third most used

language identified by the Survey [2]. Although these statistics don't directly relate to visualization solutions, they help form an important background to the context in which visualization may be needed in. Which is that Python and its libraries are a highly prevalent option for undertaking data analysis projects, meaning that any visualization solution is likely to be part of a workflow that contains these tools.

But Solutions focusing on Visualization first and foremost seemed to be much fewer when compared to analysis solutions with visualization features. Those that were, were also much more likely to be highly technical. This point was found particularly important because data analysis and by extension visualization has been found to be needed in a wide field of industries, where programming expertise is not as widely common. And this need is expected to rise in the future with the advent of Industry 4.0- Which is a widely believed idea of increasing business productivity fueled by disruptive emergent technology in the near future [3].

Taking all this into consideration, several key points were identified on how to structure a new development project to create a valuable solution from the market's point of view:

- Focus on accessibility first and foremost, including an easy-to-use interface that does not need programming experience.
- Focus on the visualisation aspect, but either ensure that adequate analysis tools are available or importing or exporting data is easy and straight forward. Visualisation is often only one step of a multi-step analysis project.

On the other hand, some thing's to avoid are:

- Focusing on the features but not accessibility. It is unreasonable to try to beat the current market players on features alone You can't get more feature-rich than a programming language such as Python, which is arguably one of the most used solutions on the market, See Appendix A.3 for more information. It is important instead to create an alternative that is powerful enough but makes the process of data visualisation much easier.

2.1.3 Academic Background. A General study of research in the field was conducted with a focus on identifying any other similar projects such as this one undertaken by the student, and more importantly the wider context from an academic point of view. Which was followed by a comparison to previous market research in section 2.2. This was done to give the student a better understanding of how visualization works in different settings and if there are any trends and patterns that can help prioritize functionality or uncover new opportunities.

This was done by analyzing a small subset of important research papers talking about data analytics and visualization. Some of these papers focus on market research which were particularly important to expand upon the market research done by the student themselves. The most important points identified are mentioned below (Consider this a continuation of the market research / section 2.2 points above):

- Current visualization techniques are not capable of handling big data. New solutions are being created and investigated by both industry and academia []. This is a big point that this project could contribute to fixing.
- No matter how good a visualization is- if it cannot be understood its value is nullified. There currently exists a gap between what computation is capable of creating and what can be easily grasped by a user. Naturally, there exists opportunity in creating solutions to close this gap.

2.1.4 Other Research. Another important aspect of research that was done included a more technical look into general design points that could help drive the design of the application itself and how features are developed.

One specific aspect researched has been software planning methodologies. The student was aware of agile and waterfall techniques from their study but those techniques were usually applied in and made for multi-person teams, which naturally conflicted with the single-person structure of this project. Through this research, a technique called PSP [] (Personal Software Process) was discovered. This is a methodology that allows a singular developer to apply a structured, continuously improving process to how they develop software. A further inspired process was also identified called PXP [], which was a fusion of Agile XP principles adapted to a single person team inspired by PSP.

Another notable finding in this phase, was an analysis of graphing and visualization techniques. In the 2.2 Market Research phase a lot of different solutions with a large variety of graphing types were seen. An analysis of the most common ones was undertaken with the hopes of identifying the most suitable graphs in terms of flexibility at higher dimensions (3+). The full analysis document can be seen here [] but in summary- It was found that Scatterplots fit these needs the most.

2.2 Application

With the Application Context and Background set, the next step for this section is to highlight what technologies and design paradigms would be available for the development of this project followed by a comparison between them. No decisions were made during this stage of the project- The research and analysis recorded here act to justify and shortlist the final design in Section 4.

It should also be noted that obviously not all combinations of technologies mentioned below are viable. Usually deciding on one aspect would limit what can be made for another. But nevertheless, each section below was looked at separately to fully understand the opportunity cost for each decision when going one route as opposed to another.

2.2.1 Application Infastructure. This section specifically focuses on the structural decisions in the design of an application that directly affect how it is created, run and in some cases what features are possible to implement. In general, the design options for this application in particular could be split into two groups.

Client-Side Run Application. This is the simplest design. All code that makes up the application is run on the user's device locally. No need for any server resources (other than for serving the initial code which even then may not need a server, Disc etc..) thus can

be ran offline. But resources are limited to only what the user has and no way to synchronize data among multiple users- such as for user accounts, etc.

Server-Side or Full Stack Application. This is a solution to some of the limitations mentioned for Client-side. Either have the client application be able to connect to a central (or server-less functions based) server and offload some tasks to it, or have the application be fully computed on-server with only static content responses being sent to the user. This design is naturally more complicated and often results in a split codebase among backend and frontend components. There is a need to provision computing resources, and more security considerations need to be made.

2.2.2 Run Environment. This can be defined as the environment where the application will run. The main options identified were the following:

.NET Environment. .NET, which is an open-source developer platform for building Software mainly developed by Microsoft [4]. It is cross platform among Windows, MacOS and Linux (With support for iOS and Android using frameworks such .NET Multi-platform App UI / MAUI [5]). It provides a great rich set of libraries and tools for any kind of projects including both frontend and backend webapp needs and client ran applications (As long as .NET Framework is installed). .NET supports only 3 languages those being C#, F# and Visual Basic.

Node.js. Node.js is aptly described on its website as an "an open-source, cross-platform JavaScript runtime environment." [6] It is the most popular web technology among the Stack Overflow 2022 Survey respondents [7]. Although most commonly used for Server-side processing as a webserver, it can also be used for running local applications as long as it is installed on the client's device (although it usually makes more sense to serve the app over a network). For package managers, npm is tightly linked with node.js and provides a huge repository of packages. With the addition of Web Assembly, which is a new standard assembly like language that acts as a compilation target from a wide range of other languages, it is technically possible to use almost any language with node.js that has a compiler to web assembly made.

Browser Engine. Browser Engine's- Modern browsers all have some JavaScript runtime engine, with V8 being the most widely used engine [8] Although all browsers are supposed to be cross-compatible, in practice this can vary, and some incompatibilities can arise. Browsers naturally only run client-side code but communicating with backend solutions is a common practice. Only JavaScript is natively supported and Web Assembly in "4 major browser engines".

JRE (Java Runtime Environment). JRE, is a runtime environment that allows an application to be cross compatible between Operating systems by acting as a compatibility layer. It runs Java bytecode which can be compiled from a large variety of languages. A standard library is also available with the runtime environment. Commonly used for backend development but also capable of Frontend.

Compiled Program (OS only environment). Compiled Program, One of the most flexible options on the list. Languages such as C++ and C compile to machine code which are run directly on a user's device. No inherit cross platform support and a need to recompile for each OS. Usually, higher performance due to lower abstraction. But a lower abstraction means more functionality needs to be managed by the programmer.

2.2.3 Rendering Solution. Being a project with visual requirements naturally meant that some rendering technology would be needed. Like with all other technologies mentioned thus far- there is a wide range of options that each have their own distinct design and ability. This section then aims to segment the available options by two factors at the minimum- One, the rendering solution must support 3D graphics in some capacity, and two, the solution must run and integrate with an application targeting one of the above-mentioned run environments. With those key requirements in mind, these were the identified options:

OpenGL. OpenGL is a low-level cross-platform rendering API which can be traced back to a release date in 1992. Although no longer in active development as of 2017, OpenGL still remains highly supported across both newly releasing GPUs and older. This also includes mobile devices. In terms of language bindings, OpenGL is cross-language and can be called from almost any language and environment that has had bindings made for it.

Vulkan. Vulkan is a cross-platform low level open standard rendering API that has superseded OpenGL in active development. It's main driving improvement over OpenGL is lower overhead and more control over how code is run on the GPU. This greater control though does mean that development is more time consuming over OpenGL. Being much newer, with the initial release date being 2017, it is much less supported among older devices than OpenGL.

DirectX3D. DirectX3D is a subset API of the proprietary DirectX family of multimedia APIs developed by Microsoft. It is a low-level API similar to OpenGL but created exclusively for Microsoft Windows, Xbox, and some Embedded Windows versions. In terms of languages, only C++ is supported targeting an executable on one of the above-mentioned systems.

WebGL. A web integrated, JavaScript only low-level API version of OpenGL. It's developed as an open web standard and is implemented in most widely used browsers, without the need to install it in any way for the client or developer. It's a low-level API just like OpenGL and comes in two main versions based on OpenGL, 1.0 is based on the OpenGL ES 2.0 standard while 2.0 is based on OpenGL ES 3.0 which is less supported among older devices. As of writing, WebGL remains the lowest level of abstraction for running GPU code on the web.

Metal. Metal is a low-level rendering API designed by Apple for their devices. It has a very limited set of supported hardware and is further limited to only iOS and MacOS for the Operating System. It was created by apple to replace OpenGL on their hardware which was depreciated in 2018. In terms of language bindings, Objective-C and C++ are the only supported options.

2.2.4 Optional Abstractions for Rendering Solutions. The underlying rendering technologies mentioned in the last section, although

usable on their own, can be offloaded to be managed by a higher-level framework or engine. Although it must be stressed that the inherit properties of the underlying rendering solution still has an effect on the application being developed. The main reason to bring in an abstraction would usually be to lower development complexity, and in turn time. But the drawback often results in less flexibility regarding what can be made and lower performance.

Three.js. Three.js is a JavaScript library that abstracts WebGL to simplify the creation of 3D graphics. It is highly extensive and offers a wide array of useful functions and constructs to that end. It is available as a npm package and can be easily integrated into a web-based project. If comparing to WebGL though, which is already a part of the browser, three.js contributes to a much bigger application download footprint. It also suffers from the same general relationship mentioned at the start of this section, development is greatly simplified but a lot of the flexibility and performance is lost.

Unity. Unity, although first and foremost a game engine, can be applied to any development project where high performance graphics are needed. Unity supports DirectX, OpenGL, Vulkan and Metal- with only a few settings changes. A wide range of target devices can also then be built for. It is also possible to target WebGL as a platform in a similar fashion for web applications. Beyond this great cross-device and API support- Unity has an extensive collection of tools and APIs to streamline development. Although it should be mentioned that web applications are not as well supported, especially on mobile devices. The engine also only supports C# as the only option.

2.2.5 Languages. There are a large variety of potential languages that could be used for writing this program. The following were selected based on how extensively the languages are supported in previously mentioned Rendering solutions and Run Environment Sections. And some key exceptions that were considered to be important for consideration by the student.

C++. A well know general-purpose compiled programming language most often used in performance constraint applications. It is Object-Oriented but can be used to also write functional code. It is arguably the de-facto standard for systems programming.

Rust. Rust is a relatively new general-purpose compiled programming language often used in systems programming. It is a modern alternative to C++ that combines its high performance with a multitude of modern features such as a default package manager, and numerous safety features especially for memory (While still keeping a low performance impact).

Java. Java is a general-purpose, object-oriented focused programming language that compiles to Java Bytecode that can be run on any device running a JVM (Java Virtual Machine).

C#. C# is a general-purpose, multi-paradigm language developed by Microsoft. Although it is technically possible to compile to machine code, it is most often compiled to run on a .NET environment.

JavaScript. JavaScript is the natively supported programming language on the web. It is multi-paradigm and is just-in-time compiled. It requires a runtime which is often a browser or dedicated runtime environments such as Node.js

TypeScript. Typescript is a superset version of JavaScript developed by Microsoft that a number of features such as stricter syntax with the ability to have types. It transpiles to JavaScript which is then run as described above in 2.2.5.5.

3 SPECIFICATION

This section aims to inform the reader on the formal project plan that was created for the development of this projects, and the decisions on how it was structured.

3.1 Development Methodology

PXP as introduced in the research phase was selected for its well defined (extensively documented in paper) agile based methodology that was well suited to a single-person development team. Agile in general though, was selected over a waterfall technique to allow user feedback to drive design and development which was often collected. The details of the methodology are mainly based on this paper[] As is a fundamental principle of agile, this methodology was adapted to the project at hand with some main changes being:

- No automated testing setup at the start of the project. Graphics development is fairly tricky to automate the testing of as the large majority of testing is completely visual. It is still possible and was considered in section X but was chosen to set aside to rather focus on development and proving that the technology stack was suitable before any more work was committed on it.
- Allow refactoring to be raised at any point, also allow grouping of similar user stories to be refactored together. Allowed fixing problems before they became too big and interlinked.
- MoSCoW and Cost factors agile ceremonies were considered for the project's user stories. Allowed better planning and time management.
- Instead of tasks created from user stories, user stories themselves are treated like tasks with any non-user story work labelled as tasks instead and treated the same as stories. Done to lower duplication of information.
- Reworked development process to better fit the students workflow. See Section S

For a more in-depth look into how these changes were implemented and the overall development flow, see section X below

3.2 Requirements Gathering

With the goals that the developed application needs to achieve set as per the research phase- The next step was to create a concrete plan on what exact requirements / features would contribute towards achieving those goals. To do that, two main ideas were identified by the student.

The first idea was to conduct Interviews with individuals who commonly use visualization tools or do general data science work. This would provide key insight into what real users of visualization technology think is key for a successful application to have. These requirements would constitute the first phase of the project and set a strong start to either a waterfall type methodology or an agile project. But there were a couple of downsides that cancelled out this idea at that time:

- With no initial product to focus insight into actionable requirements- the interview process is more likely to return conflicting or infeasible requirements.
- It might be difficult to offer insight that is not generic for the same reasons. Application should plot data vs Application should do it like this instead of like this. With the latter insight being much more valued.
- Time and access to experts is very valuable and needs extensive preparation. It is important to make the most of it, which the student didn't feel like they could do at that time.

The next idea to identify requirements was a two-step process, and one which was inspired by the development methodology chosen to be followed by the student in section X. The student took on the role of the client to create a client brief using the insight gained during the research phase. This brief can be read in full in Appendix A but in short, creates a written source document describing the minimum simplest application that would need to be created. Further improvements would then come from user testing and analysis by the student. The main reason to do this instead of just skipping to creating user stories directly was to have a consistent main source from where user stories were pulled from and were focused on completing the brief's informal requirements.

With a brief set, the next step was to extract user stories (formal requirements) from the brief. Those user stories were all bundled up into an MVP (minimum viable product) feature set which was scheduled as further covered in section X. The specific user stories are all attached in Appendix A.

3.3 Project Management Tools

A number of tools and services were employed in the process to allow the organization and techniques mentioned to be applied. Those consisted of the following:

3.3.1 Git, Version Control. A version control tool is an important tool for controlling and recording changes to a codebase. Git specifically though was chosen due to the student having previous experience, and to allow GitHub to be used within the project, See subsubsection X.

3.3.2 GitHub, Cloud Code Repositary. Github is a cloud host for git repositories with additional extensive tooling to support all aspects of a software development project. The student has had extensive experience with this platform thus it was chosen for this project to minimize learning downtime. Alternatives such as Bitbucket and GitLab also exist.

3.3.3 GitHub Issues and Projects, Organizing Backlog and managing stories. GitHub also has integrated issues and project views for organizing and tracking stories. An alternative would not have been as tightly integrated with the code repository.

3.4 Scheduling

With a backlog of user stories ready the next step was to combine similar stories into feature sets. Feature sets were then given a deadline on when all of their user stories should be completed. This would allow then the student to pick the most urgent Feature Set to focus on.

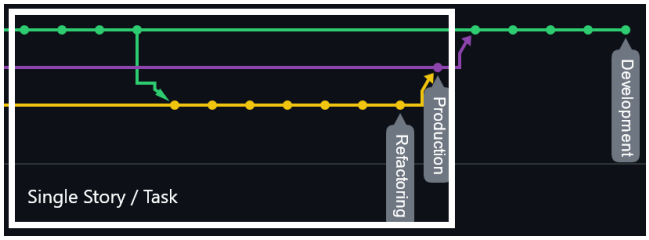

Figure 1: Development Timeline of a Single Story

Once a Feature set was picked, each user story and task within was analyzed and given a development time cost and importance to the project. With this, a prioritized backlog was created that was split among the maximum iterations/sprints that would fit into the Feature set's time frame with a consideration for the students iteration velocity. On the off chance that there wasn't enough time to complete all stories and tasks, the student would re-analyze what stories and tasks could be dropped due to time constraints.

Having feature sets allowed the student to put into perspective what are the major additions to the application and if there was enough time to fully develop those additions.

3.5 Development Flow

Development followed a predefined process (See Figure 1) where a user story or task was picked to be developed. Once the student has finished developing the one or more tasks picked in the simplest way possible, the student would then push the code to the refactor branch where it would be organized and possibly reworked to follow better practices. This process allowed the student to move fast and encounter obstacles in implementation much more quickly, which minimized the risks of large roadblocks knocking the schedule out of balance. But once the feature was reworked to a sufficient standard with bugs fixed, it was only then pushed to the final Production branch, which was publicly hosted. This process helped ensure that any code committed to production was vetted and would be unlikely to cause unexpected issues. It also allowed the student to always have a safe version of the application for demos and user testing without having to worry about what work was done before hand. The production version is then pushed to the development branch and the cycle repeats.

4 DESIGN

An initial design phase was undergone by the student before development started. Although it should be noted that the work done there did not constitute the final design as a waterfall methodology would require but was more akin to setting the scene for the start of development. The majority of design on how something should be coded, look and such was done on a story by story basis during sprints. The initial design phase itself consisted of the following key parts:

4.1 Technology Stack

The student decided to settle on a technology stack during this phase. Which although would still be open to changes should they be needed, nevertheless set what the stack should be otherwise.

This decision was done by comparing valid options built up from those identified during the research phase mentioned here X. The stack then, which was initially chosen and ended up remaining unchanged throughout the project is as follows

Client-Side run application. This project was not expected to require any server resources such as central database access. As such, client-side only was chosen to allow the student to focus on a single code base.

Browser Run Enviroment. Chosen for its greater accessibility over downloaded options and extensive support among devices without the need for any extra environment download by the user. Only a browser is needed which is usually preinstalled on most internet capable Operating Systems.

Rendering Solution. WebGL, no other rendering API is as widely supported by major browsers that is capable of 3D graphics. WebGL1 is further picked as the target version to further increase compatibility.

Optional Abstractions for Rendering Solutions. None, The student wanted to gain experience in how underlying low-level APIs worked, and did not want to sacrifice performance and flexibility of what was possible to create.

Languages. With the browser set as the environment, the choice was limited to JavaScript, TypeScript or another language through Web Assembly. To ensure the best support among existent web tooling though, TypeScript was chosen which has the wide support of JavaScript while providing useful language features not available in JavaScript. One of those features, types, were found to be particularly important as OpenGL and by extension WebGL is very heavily dependant on correct types being used at all times, which would have been very hard to do with only JavaScript.

4.2 Tooling

After the Technology Stack was identified, the next step was to identify the development tools for that stack. The following were selected for the project:

IDE or Code Editor. Visual Studio Code (VScode) was selected for all writing tasks (Both for code and written information). This decision came down to what the student was comfortable with using through previous experience but also had a practical factor for it's selection. With that mainly being a great selection of packages to help with development and great integration with GitHub.

Development Server. Node.js was used as a development and build server due to it's straightforward ability to download and manage npm packages while also running development tools such as parcel, which in addition to it's usage below, also acted as a development server with hot reloading.

Compiler and Build Tool. As a compiler, Parcel was used to compile and optimize TypeScript into runnable by the browser JavaScript. As a build tool, parcel build all npm packages and the usage of some node.js only features into a handful of static client-side only files that could be hosted. This was crucial and allowed some code designs to be implemented that otherwise would not have been possible (See somewhere).

Hosting Provider. Digital Ocean's App Platform was used to host and build the application from the production branch of the code repository. It was chosen due to the student having previous experience hosting resources on the platform, simple setup with minimal networking on the students part, support for building on a node.js environment (The same as the development environment) and the availability of free credits to do all this. The site hosted here allowed user testers to access the application from anywhere at their own time.

4.2.1 Additional Libraries. Some of the mentioned npm packages used are recorded here and the reason for their use.

Pico.css. Simple CSS framework to do the heavy lifting for styling the application UI. This allowed the student to focus more on rendering work.

gl-matrix. Pre-made functions for the math very commonly used in graphics development. Allowed the student to bypass creating these basic functions from scratch in turn saving time.

jquery-csv. A jquery syntax compliant .csv parser. No particular reason to select this particularly from other alternatives. Covers a lot of edge cases in parsing .csv files that would have taken a while to implement and test from scratch.

4.3 Prototyping

An important part of the initial design stage was creating simple prototype applications using the technology stack to give the student experience with using the stack and to identify any critical issues that may need the stack to be redesigned or tools to be changed. These were in a sense mock sprint runs to smooth out the process and prepare for when development would officially start. The prototypes created also turned out general enough that they were able to be reused for the development of the actual application, in particular prototype 2.

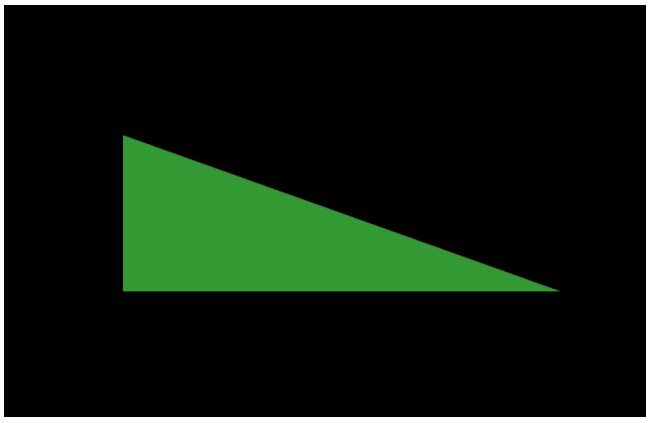

Figure 2: Prototype 1 - A simple 2D triangle drawn using the defined Technology Stack

4.3.1 Prototype 1. This was an important milestone in proving the validity of the development process identified. All of the tools and technologies mentioned thus far were setup and used to create a valid WebGL context and render a triangle. It was found that this process was well defined and no changes were done.

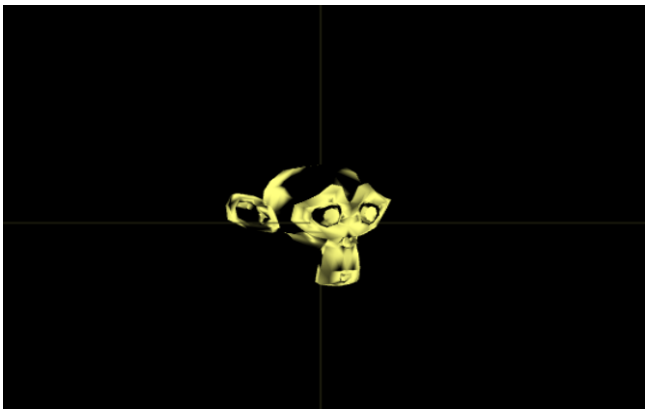

Figure 3: Prototype 2 - A 3D rendering engine

4.3.2 Prototype 2. WebGL is a low level API and considerable work needs to be done just to render a 3D model. With this prototype, the aim was both to further test the tech stacks suitability by doing more involved work, but also to see if a general purpose rendering engine could be created that implemented all of the critical functions for the application to be developed. It was considered important to create this as quickly as possible to identify any shortcomings as soon as possible that may bottleneck the project when development starts. Creating this prototype took approximately a month and yielded in the creation of a successful rendering engine with a charting focused design.

To be more specific, the features implemented by this engine were the following:

- 3D Mathematics and View, Projection and model system for rendering
- Multiple Models Rendered at the same time
- Model Object Abstraction for storing and rendering a 3D model. Implements a prototype pattern where one object can be rendered multiple times. This is particularly useful as a memory saving measure for rendering lot's of models. Something that was considered a very likely scenario for a plotting application with likely many data points.
- Simple Flat Shading based on a preset light direction.
- Loader Object for loading and organizing data that is passed to Model. Allows .OBJ models to be loaded into the renderer to be displayed.
- Rendering labels with HTML and positioning them approximately at some relative to WebGL scene coordinates.

Like previously mentioned, This prototype then set the basis from which development was started.

4.4 UX Design

As mentioned in the research phase- Ease of use and accessibility for non-programmers was a priority for this project. One that had to also dictate design decisions around how a user would interact with the application and how that application would communicate with the user. Those key decisions are mentioned here.

4.4.1 Visualisation Design. "How should data be plotted and shown to the user?" as a question encapsulates a range of design problems that apply to this section. Specifically, any chart component for this application must embody the following properties to be suitable for completing the application goals:

- Must be a common chart type that doesn't require specialist knowledge to understand nor is limited to very specific types of data.
- Must be suitable for 3D data and preferably extendable to beyond three dimensions. The more the better.
- Chart can be rendered with 3D graphics and preferably that provides some value to the visualizations with the chart type.

The chart type that in the end embodied these requirements best was a scatterplot chart. It is a chart that is commonly implemented in 3D by other plotting applications looked at. A scatterplot also naturally fits very well in a 3D rendered environment as it's relatively straight forward to encode plotted data into world coordinates that can be extensively navigated in a 3D environment. Further dimensions can also be encoded by modifying each data point further through colour, saturation, shape of data points, opacity and more.

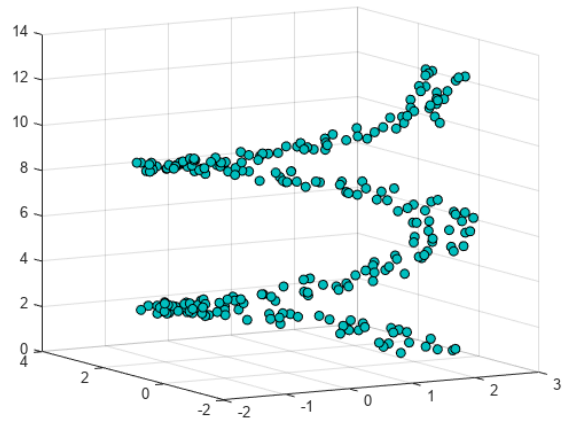

Figure 4: A 3D scatterplot generated through MATLAB

To render the scatterplot, two main ideas were identified.

- Unlimited Axis Values, World View. The scatterplot would be rendered into a 3D world by directly translating data values into world coordinates. The resultant graph could then be navigated by the user akin to 3D modelling software or a video game (first-person camera). Axis lines would be part of that world and would extend to infinity from

Original	Modified
Write chart properties programmatically	change properties through UI controls
Static chart image rendered once code written is run	Dynamically update rendered image as properties are changed
Exploration to explore	Exploration possible

Table 1: Comparison of design changes to how figures are created in the application developed compared to programmatic market alternatives

origin. This is a very easy to implement design but might be difficult to navigate for an inexperienced user.
- Classic cut-off figure. This is directly inspired by the way the majority of applications seen implement scatterplot. As a user of those systems, you would load in some data and set aspects such as the position, view angle and zoom programmatically which would generate an image as seen in Figure 4. This kind of view is not much harder to render but is much more concise as all data can be made to fit in a predefined visual space. This also allows greater control on how the graph is rendered allowing for a more consistent user experience to be tailored. But, this option has a severe disadvantage- where unlike the previous option, it is not commonly navigated but generated for specific parameters- making exploration not an obvious feature to implement.

In the end, the student decided to go with a modified option two. That modification was to replace the programmatic controls that have been seen to be used to compose / generate the chart in applications such as MATLAB with instead similar, although limited UI controls. In a sense, the design that the student came up with abstracts the programming into user controls instead, which are much easier to use (which is the guiding goal for this project) even if they are more limited in their capability.

4.4.2 User Controls. The design created by the student in the last section requires a robust design of user controls. With an agile approach, those controls were looked at as the rendering solution improved and opened up possibility for modification.

5 IMPLEMENTATION

This project was developed over 8 sprints of between 7-10 days each with the first one started on the 26th of December, 2022 and the last one stared on March 19, 2023. What follows is a detailed summary of the work done during those sprints divided into three subsections:

- Plan - Highlights what user stories were chosen to be implemented during the sprint and how long that sprint was.
- Implementation - How the features were implemented, any design decisions and any issues
- Summary - Have all user stories been completed? What went wrong? Anything learned and moved to next sprint

5.1 Sprint 1 - Start 26th December

5.1.1 Plan. This was the first sprint undertaken for the project and so the focus was to test everything out and ensure that the process was right for the student. The initial MVP feature set was chosen for development and two user stories were planned for. Those being user stories 3 and 11 with a total workload estimate of 6 over a 7 day sprint. The main goal for this sprint was to create a labelled scatterplot graph with some pre-set test data.

5.1.2 Implementation. Prototype 2 was chosen to be extended into the application being developed. It was copied into the development branch and work started on implementing the user stories mentioned. A 3D Cube with axis lines was created as the chart and an origin position was set from which data points would be rendered. This was not as difficult as expected as the model matrix used could be copied and modified by each data point to ensure that they were always at a correct position relative to the cube and axis lines.

The next step was to label the axis lines (also relative to the chart using it's model matrix). This caused some difficulty as test labels would not align properly even if they were supposed to based on their coordinates. This was particularly troublesome with perspective lines that had a considerable z change in position. This was found to be a result of inaccurate placement by the browser (browsers favors flexibility over screen sizes instead of rigid pixel-perfect placement) and not fully correct world to screen coordinate calculation. Instead, it was decided that text should be rendered within the WebGL scene to ensure accuracy- To that end, a bitmap font technique was adopted.

Figure 5: Arial font bitmap glyph image

This is where a texture atlas with pre-rendered glyphs is downloaded and used to apply individual glyphs to flat surfaces (usually two triangles of geometry). This technique was mainly adopted due to it's simpler implementation and higher performance over the main alternative of using a geometric approach for rendering text (Where geometry is used to shape each glyph, which uses much more triangles). This alternative is highly inefficient as geometry is computationally expensive to render. Although it should be noted that bitmap font's do also carry their own limitations, mainly it is difficult to have large character sets as each character set would need to be saved into an image and downloaded, which if changing font sizes is required further requires that different sized copies of the character sets are available to avoid blurriness and pixelation at large sizes. For this project though, those were considered to be tolerable limitations. Blurriness could be avoided by keeping glyphs the same size and the limited glyph set wouldn't matter as much with only one supported application language. With bitmap fonts

decided upon, the student started work on adapting the application to support rendering text in this way. This required a couple of key changes and additions within the renderer part of the application:

- It should be possible to apply textures to models- This was done by adapting the Model class to store and apply texture data on render
- Textures had to rendered and stored- This was done by expanding the Loader class to allow .png images to be loaded
- There had to be some way to abstract which letter was rendered, manually slicing the bitmap texture to get each glyph would quickly become too tedious and time consuming for anything beyond a few glyphs- A State-Machine based Font Class was created that generated font texture data for an input letter that could be fed into a Model Object.

Have some class diagram here

5.1.3 Summary. The new technique for rendering text ended up fixing the accuracy issues caused by the previous label system and all user stories managed to be completed as expected and the application was pushed to production on january 1st, 2023. This was a slow start but the student expected that more user stories would be able to be tackled as they progressed with the project.

5.2 Sprint 2 - Start 4th January

5.2.1 Plan. This was the second sprint and the plan was to start adding user controls to allow a user to modify what was being rendered. This included not only the ability to upload custom data but also rotate the resultant graph. The MVP feature set was again the developed feature set from which 4 user stories were selected, with a total work load estimate of 16 over a longer 11 day sprint. This was originally a 7 day sprint but an extension was deemed necessary to have an atomic conclusion to the sprint. In total, 16 Units of work were completed for an average of 8 per week.

5.2.2 Implementation. Loader was once again expanded to handle .csv loading which was done using jquery-csv, which sped up development considerably and covered a lot of edge cases in possible file uploads. For user controls, the use of modern UI frameworks was considered but at this moment in time- there wasn't much UI requirement for the application. Instead, as per PXP, the student focused on hitting user stories as fast as possible. Thus, an observer like structure was implemented using Event listeners connected to overlaid HTML Elements on the page (In a similar way to how the old label system worked, See Sprint 1) which ran functions that modified the state of the application. With this design, 6 buttons were made to rotate each axis of the chart individually. Another button was added to upload a file that was handed to Loader.

During this time, when implementing rotation- which consisted of applying a rotating model matrix component to the entire scene. An expected issue cropped up of having to rotate glyphs to always face the camera even when the scene rotates. This consisted a task that needed a billboarding shader to be written that was used when text is rendered in the application.

5.2.3 Summary. This was almost a double sprint. This was mainly due to lower work hours done by the student during the winter vacation. Nevertheless, All user stories were completed and the application was pushed to production on January 29th. This delay was due to the previously mentioned lower hours and semester start travel. At this point, a technically functional plotter was created, though still very limited in functionality.

5.3 Sprint 3 - Start 29th January

5.4 Plan

This was the third sprint and first sprint undertaken during semester 2. The plan was to tackle some of the biggest requirements from the MVP feature set- in turn making the application more capable. A total of 6 User stories were selected with one of them, story 6 alone being ranked at a workload of 10. In total, the sprint consisted of 25 work load points over a period of 12 days. This was the biggest sprint undertaken to date but the student was confident in rising up to the challenge. Some of the sprints assigned to this sprint were tackled in previous sprints but were not developed to a suitable standard of quality to qualify to be committed.

5.4.1 Implementation. One of the user stories was to implement the ability to change what slice the graph showed / what the axis values are. This is the beginning of adding dynamic navigation to the data that was mentioned in the design. During this sprint, this idea of navigation was distilled down into 2 essential parts. With that being:

- Moving the chart "slice" in each axis. An example would be if a 1-10 on each axis chart is moved +1 in the y direction the chart x, z axis would still show 1-10 but the y axis will now show 2-11.
- Zooming. This can be considered as increasing the size of the slice shown by a chart. If a 1-10 on each axis slice chart is scaled to a factor of 2x it would now show 1-20 on each axis.

Before any of these could be implemented though, glyph rendering had to be reworked to allow values consisting of more than one digit to be shown at each labelled line of the chart. This was a relatively straight-forward addition where instead of a single glyph being rendered at each line, an array of glyphs was rendered instead. With each digit of a number being a single glyph in order stored in the array.

Moving the chart was then started to be implemented. This was done by adding new movement buttons (2 for each axis that either moved the axis ahead by 1 by back by 1) then using event listeners getting the modification value and applying it calculate the range of axis vales that needed to be rendered (This was at most 10 values). These values were then applied in order for each labelled line on the chart. This was a relatively straight forward addition.

Zooming was one of the most challenging features of this project to implement. The implementation at this time attempted to create a visually pleasing effect where axis lines would move out to give the illusion that the camera was moving closer without actually changing camera position nor size of the viewing graph cube. This was an easy enough effect to achieve and one that could correctly sync up with the position of data cubes too.

```
glmath.mat4.translate(
Axismodel, // Model Matrix in var
Axismodel, // Resultant Matrix var
    [0.5, 0, (i / (10 * zoom))]);
```

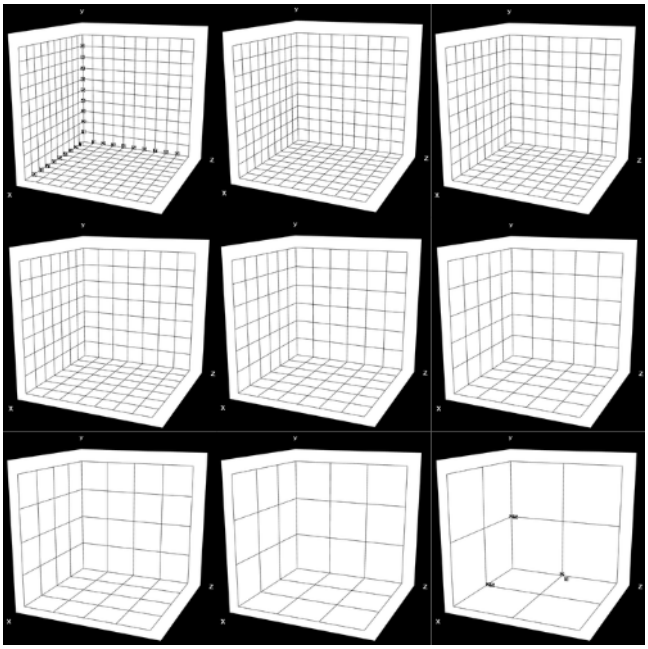

Figure 6: Zooming effect formed by Axis Lines (Zooming Out)

For cubes the effect could be applied simply as:

```
glmath.mat4.translate(
point_model, // Model Matrix in var
point_model, // Resultant Matrix var
    [x * zoom, y * zoom, z * zoom]);
```

The problem then, was correctly labelling axis lines at every possible state of zoom level. In other words, Even though positionally the cubes and axis lines were correctly placed at all zoom levels-figuring out what the exact values of each line were was very challenging., this was further exacerbated by the ability "move" the graphed slice to show different values. After extensive tinkering, labels were turned off when the application was in an uncertain state- As a temporary solution, a hardcoded zoom modifier was set for maximum and minimum zoom values (These were the only levels at which the zoom mod values could be confirmed as correct) allowing accurate labels to be shown at those two levels. With this, the application could now zoom up to a maximum of 5x showing values from 0 to 50 and a minimum of 1 to 10 at 1x. In between those two maximum and minimum levels (including those levels) there were 10 pre-set zoom values that could be scrolled through.

5.4.2 Summary. This was one of the most important sprints undertaken in terms of extending chart capabilities. All mentioned user stories were completed and the application was pushed to production on February 9th. At the end of this sprint the application was starting to become ready for user testing and could technically handle most data- Further information on user testing is mentioned in section X.

5.5 Sprint 4 - Start February 10th

5.5.1 Plan. This was the fourth sprint. The main driving factor during this sprint was to prepare the application for user testers. To that end, 4 user stories were selected with a total workload of 10. These all focused on critical UI accessibility problems that would likely otherwise dominate user feedback. The sprint was planned to run for 6 days.

5.5.2 Implementation. This was a relatively straight forward sprint. No major systems or additions were added and most work used the UI Observer pattern using Event listeners as previously defined for UI needs. Although it was starting to become noticeable that adding more controls was starting to affect code quality and maintainability. A refactoring task was added to the backlog to look at opportunities to restructure the app.

5.5.3 Summary. The sprint was completed in record time. All user stories were implemented and the app was pushed to production on Feb 12, two days after the sprint started. The remaining time was instead reallocated to plan for user testing. At this point, the application could not only render most data but had reasonable ability to display that data correctly and provide basic user controls to that end. It would now be important to get user feedback to drive the application forward beyond it's most basic state.

5.6 Sprint 5 - Start February 19th

5.6.1 Plan. This was the fifth sprint. User testing 1 has already started at this point in time and the student was waiting results to finish come in before analysis could start. In this relative downtime. A 7 day sprint was planned with 2 user stories for a workload of 9 points. These were task based stories that arose during development in previous sprints, and the student thought they would be important additions to have.

5.6.2 Implementation. The first story was to display names for each axis taken from the first row of the dataset. This was pretty straight forward thanks to the Font + Model Abstraction and was promptly implemented.

The next user story though was to look at new ways to structure the app to increase maintainability. The main areas of opportunity thus found were as follow:

- Restructure application to be fully Object-Oriented - At the moment the application is a script that extensively uses Objects (From classes such as Model and Loader) to implement the chart in a high level as a result of the abstractions provided by those objects. This is fine and probably the best design for a single chart to be rendered but there may be potential for further abstraction to allow multiple charts per page that can interact with each other.
- Find a way to better manage UI state. This is currently not a problem with the relatively simple user controls but may become cumbersome when more controls are added and if some controls might start being dependant on other controls.
- Find an overarching architecture to manage UI and Rendering code.At the moment user controls are highly coupled to rendering which is not ideal. It would be best to decouple

them in some way to make adding more UI components easier.

Each of these points were researched in turn and prototyped with in turn. Although time was a very limited factor as the student considered it important to provide value to the user first and foremost. Which meant refactoring would likely not be able to continue beyond this 7 day sprint as requirements extracted from user testing would then be higher prioritized. First, some attempts were made to restructure the render loop into an object but it was more work than expected to port everything correctly so it was abandoned.

As for state management, there were a number of techniques and pre-made solutions for managing state. React + React Context though, has been found to be a great fit for the project. React is a UI library for creating user interfaces through components and now has support for state management by sharing states between these components. Parcel also has great support for react so integration would not have been too problematic. Unfortunately, the unique circumstances of this project made the student reject integrating react into the tool stack at the current time for the following key reasons:

- Performance cost. Although most likely negligible, there is still none the less a performance cost in using an abstraction such as react. This is particularly important as the application already was performance heavy due to 3D rendering.
- Lack of time and experience. In a perfect world the library could be easily implemented with no downtime and no future issues. Unfortunately, the reality is that adding react would have heavily changed the tech stack that could cause unknown problems. This is further exemplified by the student not having any considerable experience in using react meaning any issues that did pop-up would be much more difficult to solve. There was also no time to have a prototyping phase to mitigate these concerns as there was with the current stack at the start of this project.
- Minimal Value added. As previously mentioned the current system worked as needed and did not need any changes to provide further functionality to the user. As such, it would have been hard to justify work done of porting the application to react over only future concerns. PXP prioritizes doing the simplest design and not worrying about future requirements as those tend to change constantly.

The final point, to find an overarching architecture, the student found that the application fit very well into the MVC pattern. As such, the student starting working on implementing the pattern-which went smoothly as the application had already been unknowingly made to partially fit the pattern by the student. The application was structured as follows:

- View - Static Written HTML index.html
- Model - The application render loop rendering chart. The chart rendered is a result of multiple control variables which make up the model.
- Controller - This was the new class created. All event listeners and control functions were moved to be managed by this class.

Have chart showing all this here TODO

5.6.3 Summary. This was an important sprint that worked to refactor the application to be easier to work with, especially when adding new UI elements. The 2 user stories were thus completed and the application pushed to production as Version 2 on March 4th. It was completed just in time as user testing has come to a close and analysis begun.

REFERENCES

A APPENDIX 1

Include any relevant appendices here (or delete if you want...)