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 visualization is an important step in the data analysis process. Whether used on it's own as a tool for analysis, or as a final stop to clearly articulate and display results. It is in spirit a tool for communication, allowing abstract and complex ideas to be easily communicated to a wide audience, usually requiring minimal expertise to understand (Based on the visualization design done). It's ubiquity across all industries is then no surprise [28]. Yet limitations and shortcomings exist, the accessibility of tools to create visualizations are often complex and at odds with the accessibility of the created artifacts. This limits creators by requiring them to have considerable knowledge of programming fundamentals, or be limited by what they are able to create. This is further compounded with 3D+ Multidimensional data visualization, Which has an even smaller selection of accessible tools.

Another challenge, when considering all tools regardless of user accessibility, has been the effect of Big Data on the user requirements of these applications. The ever increasing size and complexity of datasets doesn't invalidate the value provided by visualization to smaller datasets. In fact, It's importance as a communication aid is greater than ever before. Yet, current visualization tools and techniques cannot keep up with this requirement [5]. In a way, this is just another accessibility problem- an inaccessibility of certain data types.

The aim of this project then was try to address the roots of these inaccessibility problems by creating a new market contender application focused with this problem in mind. This consisted of an academic long software development project undertaken by the student- with a resultant Multidimensional Scatterplot application created. This paper aims to highlight the decisions made in design and development while also recording the entire process followed by a look back on if the project was successful.

2 BACKGROUND

The result of background research done for this project can be subdivided into two distinct sections being 'Related Work' and 'Tools and Technologies'. This was one of the earliest phases of the project whose results motivated the specific goals of this project-beyond just creating a Multidimensional plotting application as is required by the honours assessment that this project falls under, and explore what opportunities (in terms of tooling) exist to develop them.

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 2.1.3)- 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 A1, 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 [30] shows that a large majority of the most used libraries and frameworks were for data analysis or data-based projects, specifically Python. Which is also the third most used language identified by the Survey [30]. 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 [13].

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 visualization aspect, but either ensure that adequate analysis tools are available or importing or exporting data is easy and straight forward. Visualization 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 A1 for more information. It is important instead to create an alternative that is powerful enough but makes the process of data visualization 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 2.1.2. 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 study was done by analyzing a small subset of important, state of the art 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 2.1.2 points above):

- Current visualization techniques are not capable of handling big data [5]. New solutions are being created and investigated by both industry and academia. Strategies for creating new visualization systems that handle each of the troubling aspects of big data have been put forward in [21]. 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 by taking into consideration human cognitive psychology. [28]
- Solving the problems to visualization poised by big data would require the combination of current visualization solutions merged with new technologies. [28]

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 [18] (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 [4], 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 2.1.2 Market Research phase a lot of different solutions with a large variety of graphing types were seen. An analysis of the most common ones seen, which also had some overlap with The seven basic tools of quality (A collection of seven charts that need minimal training to use [19] for analysis) 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 in Appendix A2 but in summary- It was found that Scatterplot's fit these needs the most while also being part of the Seven tools meaning increased accessibility in who could use the application. The other main options included the Radar chart which was suitable but not part of the Seven tools.

2.2 Tools and Technologies

Research was also done to highlight what technologies and design paradigms would be available for the development of this project. This also included a comparison between them. Though it should be noted that no decisions were made during this research stage of the project- The research and analysis recorded here act to justify and shortlist the final design in section 4.1

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 selected 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, can distribute software on USB, Disc etc..) thus can be ran offline. But computing resources are limited to only what the user has and no inherit way to synchronize data among multiple users and devices- 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 supported by Microsoft [23]. It is cross platform among Windows, MacOS and Linux (With support for iOS and Android using frameworks such .NET Multi-platform App UI / MAUI [11]). 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 is installed on that device). .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." [27] It is the most popular web technology among the Stack Overflow 2022 Survey respondents [30]. 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 with node.js acting as a server side environment). 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. [3]

Browser Engine. Browser Engine's- Modern browsers all have some JavaScript runtime engine, with V8 being the most widely used engine [1] [2] 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". [3]

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. [6] [29]

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 [40]. 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. [31]

Vulkan. Vulkan is a cross-platform low level open standard rendering API that has superseded OpenGL in active development by the Khronos Group, but does not replace. 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 [39]. Being much newer, with the initial release date being 2016 [16], it can

be assumed to be much less supported among older devices than OpenGL.

Direct3D. Direct3D is a subset API of the proprietary DirectX family of multimedia APIs developed by Microsoft [14]. 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 [35].

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, WebGL 1.0 is based on the OpenGL ES 2.0 standard (OpenGL ES is an embedded subset version of OpenGL) while WebGL 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. [24] [15]

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. In terms of language bindings; Objective-C, Swift and C++ are the only supported options. [8] It was created by apple to replace OpenGL on their hardware which was depreciated in 2018. [7]

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 [25]. It is highly extensive and offers a wide array of useful functions and constructs to that end. It is available as a npm package [38] 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. [26] [37]

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 critical applications. It is Object-Oriented but can be used to also write functional code. It is arguably the de-facto standard for systems programming. [36]

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). [20]

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). [6] [29]

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. [9]

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. [10]

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 in the same way. [22]

2.2.6 Automated Testing. Automated testing is used in software development to run a set of pre-configured tests repeatedly, usually every time before code is committed to a repository or as needed. Once tests are setup and written, it is an easy way to quickly test all cases including edge cases that would otherwise be tedious or take too long to do manually. Automated testing is not always possible for all aspects of an application but nevertheless a number of options were looked at in case an opportunity to integrate automated testing arises.

Test Complete. Test complete is a GUI automation tool for testing a wide range of platforms and application types across Desktop, Web and Mobile platforms. [33] But it's in-accessible price and no free tier limits it's adoptability for this project. [34]

Robot Framework. Described on it's website as a "Robot Framework is a generic open source automation framework. It can be used for test automation and robotic process automation (RPA)." [12] Being open-source it is free to use and has a wide range of libraries made for testing different device types and environments. [12]

Selenium. Selenium is a set of browser automation tools and libraries that can be applied to automate almost any part of browser

interaction. It is open source and free to use- It functions by providing a "WebDriver" interface for writing testing instructions that can be run on different browsers. Tests can be written in a variety of languages but can only automate applications running in a browser. [32]

2.3 Summary

The background research mentioned in this section acted to ground the other-wise open-ended project to a set of goals, both mandatory and optional, to create an impactful and original project. The following paragraph then summarizes the goals previously brought up into a short, guiding statement.

> The purpose of this project is to create a new market entry application for the creation of visualizations. This application would focus on user accessibility first and foremost, to be an application that is usable without programming knowledge. It will allow Multidimensional data to be plotted through a Scatterplot plot and possibly more chart options. In it's creation it will not ignore the current problems poised by Big Data and will attempt to support it thorough the application of state of the art techniques to support it's visualisations or be designed in such a way as to support their future integration into the application.

Otherwise the research mentioned thus far also sets an origin from which all further decisions are justified by. For the final selection and analysis of the tech stack chosen by the student to develop the application, see section 4.1. To see how market research affected the application design, see section 4.5. These are just two of the bigger sections building upon the research set here- thus, other references linking here are to be expected as different parts of the project are focused on.

3 SPECIFICATION

This section considers the formal project plan that was created for the development of this projects, and the decisions on how it was structured. This phase of the project came after the specific project goals were identified during research (See 2.3) and act's to create a development plan based on achieving these goals.

Each of the subsections then focus on one particular aspect of that planning.

3.1 Development Methodology

PXP as introduced in the research phase 2.1.4 was selected for its well defined (extensively documented) 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 planned to be often collected. As is a fundamental principle of agile this methodology was adapted to the project at hand (And the students work style) with some main changes and additions being:

- 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 applied to 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.
- No automated testing. This was a decision that was retroactively added here after prototyping during the design phase (See 4.4). The justification for this controversial change is covered there but in summary, It was found that Graphics development is fairly tricky to automate the testing of as the large majority of testing is completely visual and subject to changes.
- Reworked development process in general to better fit the students workflow. See 3.4 and 3.5 for specific details

To see the resultant development flow with these changes in action see Section 5.

3.2 Requirements Gathering

With the goals that the developed application needs to achieve set as per the research phase (See 2.3)- 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 one which was inspired by the development methodology chosen to be followed by the student in section 3.1, PXP, the methodology in question, describes the process for a developer to stand in for the client if the client is unavailable for planning. So, The student took on the role of the client to create a client brief using the insight gained during the research phase. This had a number of key advantages listed as follows:

- Minimal Preparation and much faster compared to planning and hosting an interview.
- Can take advantage of previous research into market competitors to inspire requirements.

- With the "Client" also being the developer it is possible to think more deeply about how possible the stories are and how well they fit together.

With this technique, a planning stage was ran and a requirements brief was created. 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. It was considered that only a minimal version of the application should be planned at the beginning with further improvements then coming in a more agile way through user testing and analysis by the student based on how the project is progressing.

But the main reason to have created this brief 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 meeting the goals set during the research phase (See 2.3).

With a brief set, the next step was to extract user stories (formal requirements) from the brief. Those initial user stories all formed the MVP (minimum viable product) feature set which were scheduled as further covered in section 3.4. The specific extracted user stories and their analysis follow as so (See Table. 1)

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 for it's supporting tools (See 3.3.2 and 3.3.3).

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 but did not offer any further advantages that the student considered worth the learning downtime.

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.

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.

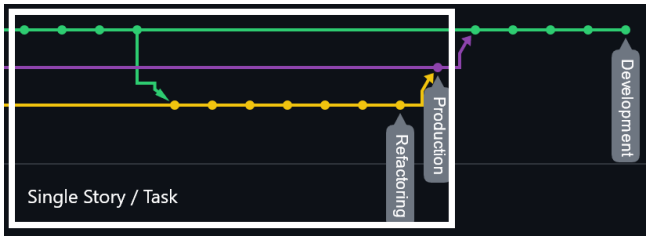

Figure 1: Development Timeline of a Single Story

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.

The specifics of when each feature sets deadline was set and how it changed can be seen on a sprint by sprint basis in the implementation section (See 5).

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, 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 would then be fully tested and only then would be pushed to the final Production branch, which was build and 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 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 (See 5) This phase itself though consisted of the following key sections.

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 if any problems arose during prototyping (See 4.3), nevertheless set what the stack should be otherwise. This decision was done by comparing valid options built up from those identified

ID	Title	Estimate (Relative)
1	As a User, I want to be able to import a dataset into the application to graph it in a 3 dimension scatterplot	3
2	As a User, I want to be able to navigate around the generated graph in 3D space while having the axis stay accurate	6
3	As a User, I want to be able to view a scatterplot of data set against an axis with accurate scales	5
5	As a User, I want to move the 3D view of the scatterplot in 3D using on screen controls	3
6	As a User, I want to be able to zoom in and out of the 3D Scatterplot	10
7	As a User, I want to be able to rotate the 3D Scatterplot around all 3D axis individually	5
8	As a User, I want the application to have easy to use on screen controls for interacting with the application	3
9	As a User, I want to be able to access the application on landscape screens of different sizes	2
10	As a User, I want to be able to view instructions within the application on how to use the application	2
11	As a User, I want all interactable parts of the Application to be visible at all times	1

Table 1: Requirements Gathering #1 User Stories, MVP Feature Set

Figure 2: TypeScript [41] and WebGL logos [17]

during the research phase (See 2.2) The selection and justification of every element of the stack follow below:

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. This also would simplify testing as there would be less infrastructure moving parts that could break.

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 writing latex). 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 providing developer tools such as hot reloading on changes and helpful error screens.

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 Model loading in 4.3.2).

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, integration with Github, support for building on a node.js environment (The same as the development environment, allowing for continuous integration from the production branch) 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.

Testing Framework. With this project set to run in a browser as per the rest of the technology stack (See 4.1). It was decided that

Robot Framework would be the ideal choice for handling testing- This was expected to be done using WatchUI, a library for Robot Testing that runs selenium under the hood. This was done to make testing more flexible and less time consuming thanks to the wide selection of other helping libraries available with Robot Framework. Unfortunately, during prototyping (See 4.3.2) a number of issues cropped up that ultimately left user testing to be set aside. For further details see 4.4.

4.2.1 Additional Libraries. The other 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.

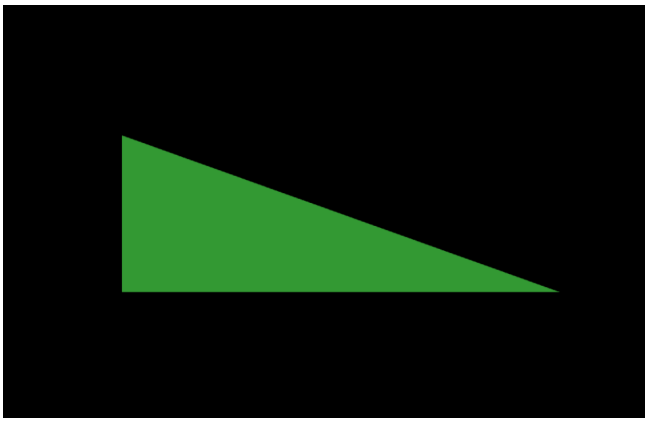

Figure 3: Prototype 1 - A simple 2D triangle drawn using the defined Technology Stack

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.

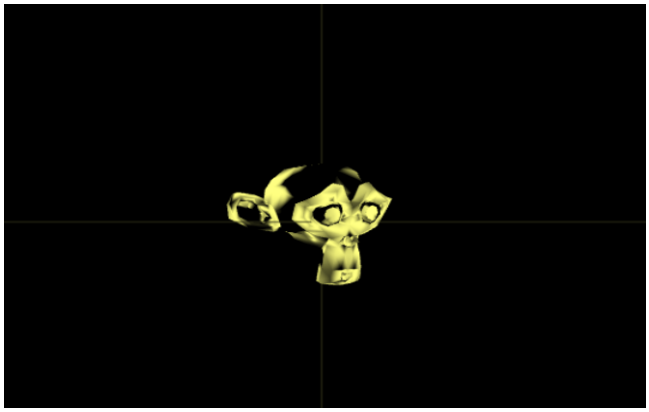

Figure 4: Prototype 2 - A 3D rendering engine

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

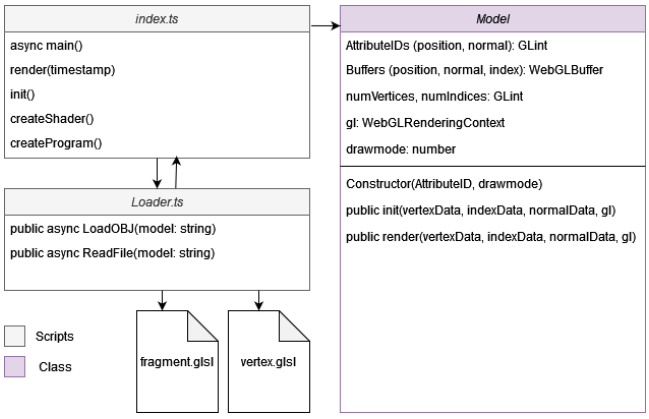

Figure 5: Prototype 2, Application Class Diagram

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.

One bottleneck though was identified during this development, which was to do with Automated testing. See 4.4 for the details. Like previously mentioned, This prototype then set the basis from which development was started.

4.4 Testing Methodology

Test Driven development is an important aspect of the PXP process when developing software. This is most often done with automated testing for the best time efficiency and accuracy. But, during the creation of prototype 2 (See 4.3.2) the student encountered a number of critical issues that contributed to the decision to not implement automated testing, subject to possible further consideration later on. Those issues were:

- Graphics Development was found to be difficult to write tests for due to it's visual nature. This on it's own was not a problem as a solution using WatchUI was identified (See 4.2). What ended up being complex though was how rapidly the visuals changed during development, this made making tests useless as they could not be reused since they depended on comparing what the application rendered to reference images.
- Minimal UI to test, Automated testing on the web is particularly well suited to testing UI. Unfortunately, there wasn't any UI to test.

This points were expected to become less true as the application developed. Nevertheless, the student considered it unwise to setup tests for the start of development. Another process had to be identified in the meantime to allow the student to focus on bringing value to the project (completing user stories). This was the application of manual testing during the refactoring branch as per the specified development flow (See 3.5).

4.5 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.

4.5.1 Visualisation Design. One of the project's goals (See 2.3) was to use a Scatter plot. To achieve that, two suitable rendering techniques were identified by the Student. Those are the following:

- 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 origin. This is a very easy to implement design but might be difficult to navigate for an inexperienced user and to appropriately combine together with UI.
- Classic cut-off figure. This is directly inspired by the way the majority of applications seen implement scatterplots. 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 7. 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.

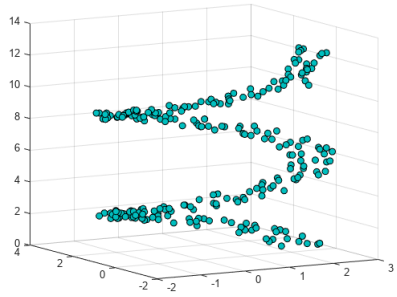

Figure 6: A 3D scatterplot generated through MATLAB, example of a classic cut-off figure

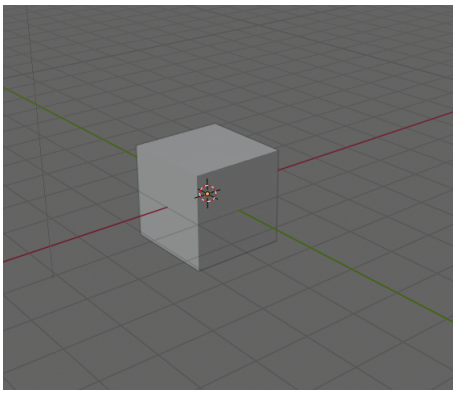

Figure 7: Screenshot from Blender, A 3D modelling application, similar to the Unlimited Axis World view presented

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

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
Difficult to explore	Exploration possible

Table 2: Comparison of design changes to how figures are created in the application developed compared to programmatic market alternatives

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. On input, the chart can be regenerated allowing exploration to be done in a similar sense, instead of moving the camera the chart is modified to show subsections of the data (See Table 2)

4.5.2 User Controls. The design created by the student in the last section (4.5.1) requires a robust design of user controls. The design and implementation for UI then followed an agile approach, where UI was designed and implemented as the need arose and the rendering counterpart could support it.

4.6 Summary

This design stage was critical in finalizing the last big questions that needed to be answered before development. With the application of the prototype phase (See 4.3), the student gained experience and confidence in working with the tools selected, meaning development could proceed much more smoothly.

5 IMPLEMENTATION

This project was developed over 8 sprints between 7-10 days each. 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 per sprint:

- 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 (See stories on Table 3). 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 (See 4.3.2) was chosen to be extended into the application being developed. It was copied into the development branch and work was 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 tend to favor 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 8: Arial font bitmap glyph image

Figure 9: Glyph Structure

This is where a texture atlas with pre-rendered glyphs is embedded into the application and used to apply individual glyphs to flat surfaces (usually two triangles of geometry, See 9). 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 many 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 support large character sets as each character set would need to be saved into an image and embedded into the application meaning the whole application takes longer to load- and if changing font sizes is required, then that means different sized copies of the character sets need to be further 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 take-in, store and apply texture data on render
- 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. See Figure 11.
- An Glyph set image and accompanying .JSON file for glyph data was embedded into the application. It's this data that allows the Font Class to function. See Figure 11 for more details.
- A new set of vertex and fragment shaders were created for just text. This was to allow texture data to be applied in the fragment shader.

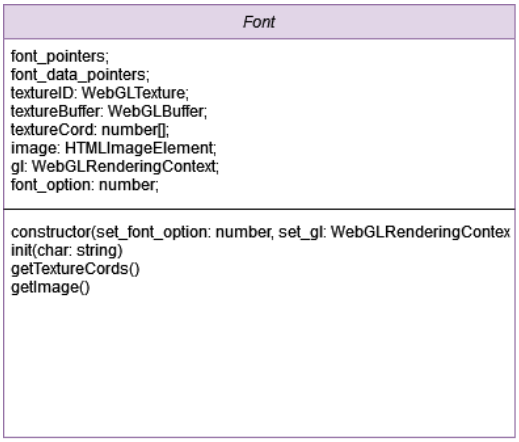

Figure 10: Font class created

The implementation of the Font Abstraction makes it very easy to have multiple different fonts (As in style) in the application. As long as font data is embedded in the application- It is trivial to change the font by simply changing a variable. With this, two fonts options were added- Arial and a bold variation of Arial.

5.1.3 Summary. The new technique for rendering text ended up fixing the accuracy issues caused by the previous label system, 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 the project progressed.

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

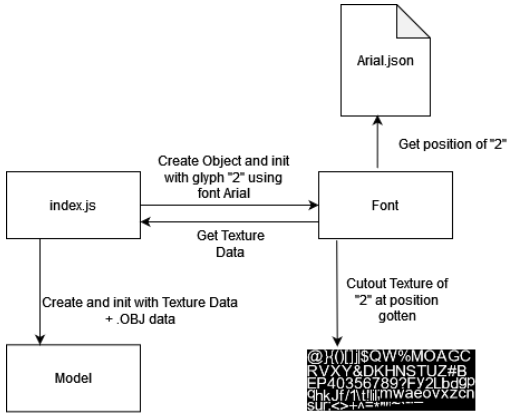

Figure 11: Process to prepare a single glyph model before rendering, Glyph "2" is taken as an example

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 (See table 4), 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 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 label rendering in 4.3.2) 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. See below for an example Event Listener:

```
// Event Listener for a rotate x axis button
(<HTMLElement>document.getElementById("right"))
.addEventListener("click", function () {
       x_rotation += 0.1;
   });
```

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 was solved by writing a custom shader to be used when rendering text glyphs, implementing billboarding.

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

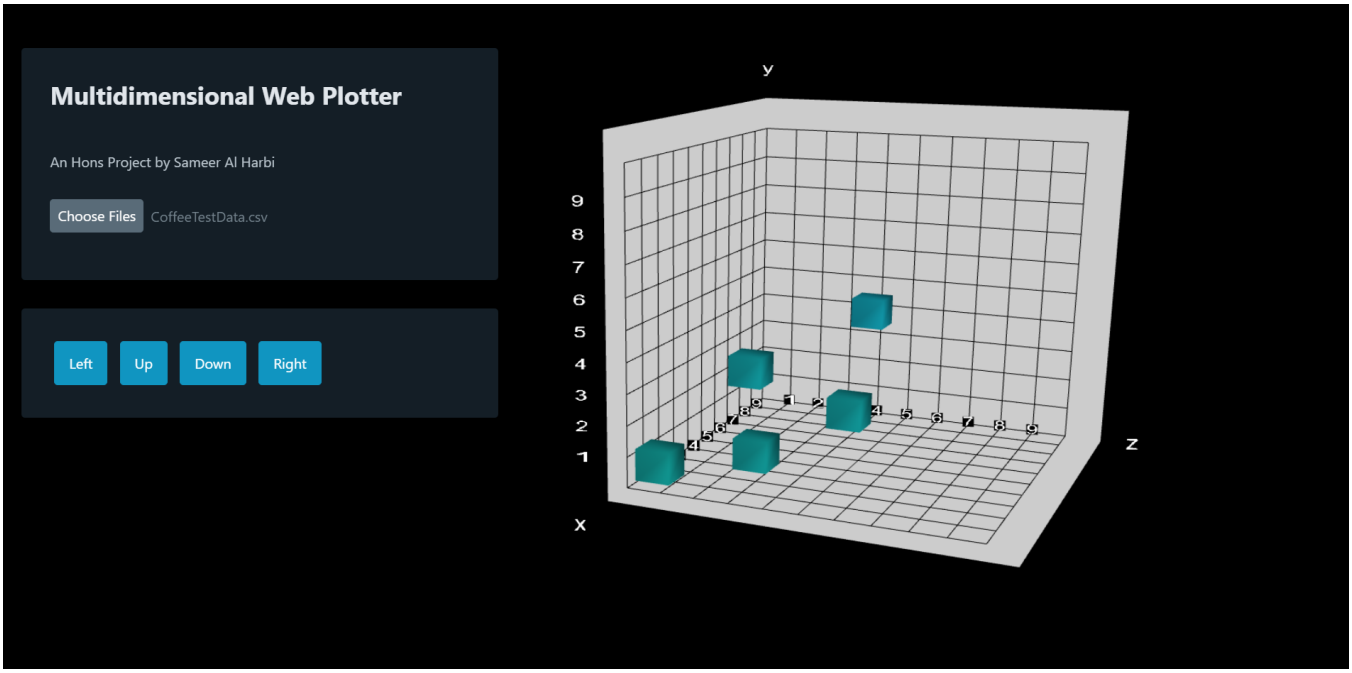

Figure 12: The application at the end of Sprint 2

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. See Figure 12

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 5 User stories were selected (See table refsprint3) with one of them, story 6 alone being ranked at a workload of 10. In total, the sprint consisted of 24 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.

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 (See 4.5). This was 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.

This did not take full advantage of the Model class prototype abstraction pattern, Technically it would have been possible to just store the unique texture data of each glyph in a number and repeatedly apply it to a single model then rendering each glyph in turn, which would have meant less repeating of data through multiple Model objects- but this would have a greater performance impact as the one model would need to be re-initialized for every glyph every frame before rendering.

Moving the chart was next to be implemented. This was done by adding new movement buttons (2 for each axis that either moved the axis ahead by 1 or 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 (See Figure 13) where axis lines would move out to give the illusion that the camera was moving closer without actually changing the 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(
```

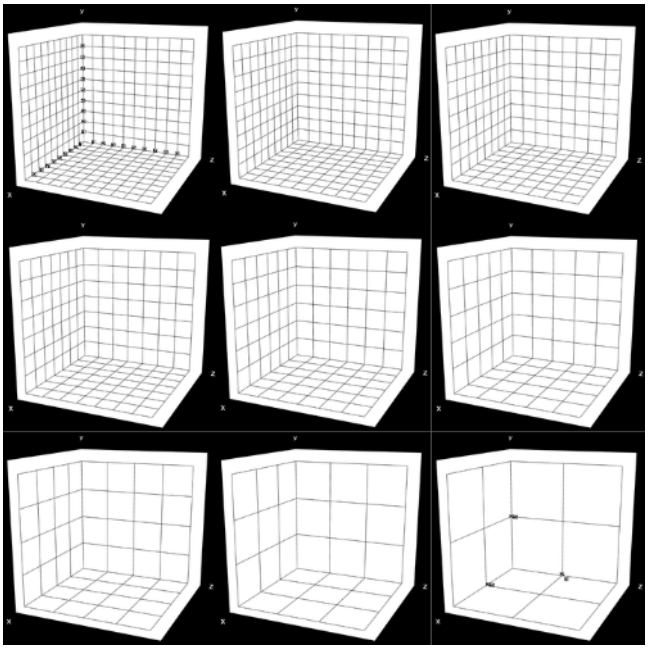

Figure 13: Zooming effect formed by Axis Lines (Zooming Out left to right, top to bottom)

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

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 6.2

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.

5.7 Sprint 6 - Start March 05

5.7.1 Plan. This was the 6th Sprint, and the first sprint starting work on the second version of the project. Two new FeatureSets were now available for user stories to be selected from. One composed from user testing analysis as further described in section X and A further improvements feature set created in a similar fashion as the first MVP feature set (By the student through the research phase). For this Sprint, 7 Stories were selected from both feature sets with a priority for Must. In terms of workload, a wide range were selected for a total workload of 27 Points over a 5 day sprint period.

5.7.2 Implementation. The first important story done was reworking how zooming worked. User testers asked for a zoom that showed labels for each level which was found to be difficult to calibrate with the zooming visual effect as mentioned in Sprint 3. So instead, the zoom was reworked to always show 10 axis lines and only change position of data points and labels on zoom- This was generally straight forward and created a robust zoom that hit the criteria, even if it didn't look as visually impressive.

The next addition was to modify the renderer to allow picking / highlighting data points. To achieve this, two main techniques were identified by the student.

- Raycasting - This works by taking the screen coordinates clicked on by the user and projecting them into the 3D scene. From this scene position a ray is then shot and a check done to get any intersections with objects.
- Colour Picking - This is a technique where Object ID is encoded into the colour of an object. When a user then clicks on an object, it is possible to get what object was clicked on by picking the colour at the click position (pixel clicked on) and matching it to an object in the scene.

After analysis of the two, Colour picking was chosen over ray tracing for the following key points:

- More Complex implementation with raycasting.
- Raycasting is highly CPU dependant, Lots of Math- which isn't good with JavaScript.
- Something else

To implement colour picking then, the student extended the renderer to render data points twice- The first render would render to a texture in a different buffer, this first render would also encode an ID for every Model rendered into 3 rgb Channels giving a unique colour for every model drawn (up to around 16 million). After that, render is called again but to the screen buffer instead. This means

that the user only see's the the second render (Which can freely use colour) instead of the first one. *Have image comparison here*

glGetPixel would then poll what colour pixel the user's mouse is hovering over. During the second render phase as each model is rendered, it's id is checked to see if it matches the selected id extracted from the colour of the hovered over pixel. If it does, the model is highlighted and it's data saved to be displayed.

5.7.3 Summary. All User stories were completed as expected and the application pushed to production on March 11. With this sprint almost all Must stories from both Feature sets making this a very productive sprint and one that set the project on good track to be completed by the end of the month.

5.8 Sprint 7 - Start March 12

5.8.1 Plan. This was the 7th Sprint. The highest priority user stories were once again selected from both feature sets. This included the last 2 Must stories and 2 Should stories. In total, the workload was 21 over a 5 day period. The main focus of this sprint was to expand the application to support one more dimension and create a new suite of user controls to view and assign data to chart axies.

5.8.2 Implementation. To add a new dimension, the student had to start looking at different ways to represent dimensions. The main options identified were Alpha/Transparency, Single Channel / Saturation component, Full Colour component and Shape component. Of those, Shape was dropped due to limited range (Have to have a different model to represent each possible values). Alpha was also dropped due to difficulties in how the browser handles Alpha. Finally, Saturation was chosen over colour as having all 3 channels might be confusing for users to extract values.

During the following implementation- the first problem encountered was how to plot an infinite range of values between two Saturation values on a single channel. For this problem the student identified the use of a squashing function that formed an asymptote at the limits. By then also allowing the user to change those limits, any data can be effectively plotted with saturation.

$$\frac{1}{1 + e^{-mod * value}}$$

Figure 14: Squashing Function, where mod is based changed by the user while value is the value of the data point plotted

Once the mathematical basis was solved, the implementation was relatively straight forward. The next main addition worked on was to create a new section of user controls. To that end, a tabular design was adopted for aspects of the application that demanded large screen real estate and focus from the user. This consisted of three total tab sections as follows:

- Graph View - This is what has been shown so far, the rendered graph.
- Data Management - This will be a new tab. Here, the data uploaded is displayed in a table fashion with data point IDs for cross reference with the graph. Additionally, a set of selectors were created on the page that allowed the user to select which column from the table would apply to which axis to be graphed (or to saturation).
- Help tab - This is a FAQ based help section, Answers to common (or expected to be) questions.

This was also relatively straight forward to implement, although with the data management tab- the UI system was pushed to it's limit in terms of state management. The student had to design the UI in such a way as to minimize UI elements depending on each other for state. Although there still are nevertheless some less than ideal control implementations. *Have example here*. Any considerable future UI work will preferably need to bring on some helping tool to minimize development time and complexity.

5.8.3 Summary. All user stories were completed as expected and the application was pushed to production on March 18th. The usability of the application took a considerable upgrade this sprint- with the application itself being almost fully completing both feature sets.

5.9 Sprint 8 - Start March 19

5.9.1 Plan. This was the 8th sprint and expected to be the last sprint before user testing 2 could start. Only 2 user stories were selected for development with a total workload of 3 points over a 5 days span.Most of the time this sprint was dedicated to preparing for user testing over development. Nevertheless this was still an important sprint that covered some of the last few to be implemented Should stories.

5.9.2 Implementation. Implementation went as expected. The stories mainly needed some minor UI additions.

5.9.3 Summary. All stories were completed and the application was pushed to production on March 29th. With this sprint, the application was ready to be user tested again.

6 EVALUATION

An important driving factor for this project and the application developed was a system of Evaluation at all levels of development and planning. To that end, the student employed a number of techniques and phases to convert feedback into actionable input that would then influence how the project progressed. Each of those techniques is covered in detail in the following subsections.

6.1 Development Testing Process

Under PXP, the refactor branch was adapted to also serve as a final testing stop before the student pushed code to production (See specification for more detail on PXP). Automated testing was considered as mentioned in the Design section but was found to not be suitable for this project due to the heavy focus on rendering development. A technology review of some options was considered in the Design section but a suitable option was not found.

Instead, refactoring was done manually on the refactoring branch based on visual inspection for rendering artifacts. To help with this process, a python command line tool was written to generate a wide range of datasets for edge case testing. The datasets are all attached in appendix X but some notable ones are as follows:

ID	Title	Estimate (Relative)
3	As a User, I want to be able to view a scatterplot of data set against an axis with accurate scales	5
11	As a User, I want all interactive parts of the Application to be visible at all times	1

Table 3: MVP Feature Set, Sprint 1

ID	Title	Estimate (Relative)
8	As a User, I want the application to have easy to use on screen controls for interacting with the application	3
1	As a User, I want to be able to import a dataset into the application to graph it in a 3 dimension scatterplot	3
18	The Axis should scale with the values shown	5
7	As a User, I want to be able to rotate the 3D Scatterplot around all 3D axis individually	5

Table 4: MVP Feature Set, Sprint 2

ID	Title	Estimate (Relative)
2	As a User, I want to be able to navigate around the generated graph in 3D space while having the axis stay accurate	6
5	As a User, I want to move the 3D view of the scatterplot in 3D using on screen controls	3
6	As a User, I want to be able to zoom in and out of the 3D Scatterplot	10
9	As a User, I want to be able to access the application on landscape screens of different sizes	2
24	UI doesn't scale very well and not all controls visible	3

Table 5: MVP Feature Set, Sprint 3

- DataCube - This is a data set that creates a 10x10x10 Cube of data points. If the application renders correctly, this dataset would fill every visible space at the initial zoom and slice position.
- Terrain Data - This is a Stress test dataset consisting of perlin noise generated terrain coordinates in a 25x25x12 sized chunk. It is a test for not only performance but also ability to handle data points with decimal values.

For UI testing, It was considered by the student that there was minimal value based on how little UI the project had for the majority of development, In only the last 2 sprints did the UI develop to a point where automated testing could be applied beyond it's simplest state. See the design section for further analysis on this during the start of the project.

These datasets also served to provide a wide range of example data for user testers to use.

6.2 User Testing

User testing was an important step in gathering requirements and paving the way for further development beyond the most basic, fundamental requirements. In total, 2 User testing phases were ran. The first one after the completion of the MVP feature set post Sprint 5 and the second one after the completion of the further additions feature set and user testing 1 feature set post sprint 8.

The process for these two phases revolved around an anonymous questionnaire with data collected as per the prepared ethics documentation for honours students. To support the testing process, the student created a simple static webpage to act as a central source for all links to the application, the questionnaire, testing data and ethics documentation that guided a tester through the entire process. Having this page really simplified testing as the student would only need to share a single link which self-contained everything needed. The page itself and HTML source is available to view in the appendix X.

The questionnaire itself (See appendix X) asked a tester to download a specific dataset and use the application to identify some factor or knowledge placed in the data by the student. The datasets were also designed in such a way that the questions could only be solved if the tester used some specific controls, or understood some aspect of the graph. This was also followed by the testers personal thoughts on a variety of questions concerning usability and how they went about solving the problems set out.

This allowed the student to construct a critical understanding of how real users used the application to solve typical problems the application was designed in mind with. Were users able to get the correct answer? If not, did they think they got a correct answer? What controls did they use? Could those Controls have misinformed them or gave them a false understanding of the data?

Though it should be mentioned that this analysis likely is not fully representative of the demographic tested due to the limited number of testers. Of which, a majority were computing students. This likely shifted results for usability specifically.

The specifics of responses and analysis carried out leading to the creation of Feature sets are covered in more detail below for each of the testing phases undertaken.

6.2.1 User Testing 1 - Starting February 17th. This was setup and ran right after sprint 5 was complete. The focus for this testing phase was to identify what non-obvious controls are still missing to help users do real work using the application. A questionnaire was created to that end and user testing began. The user testing was then concluded a week later with 5 total responses.

6.2.2 User Testing 2 - Starting March 30th. TODO WIP

7 FINAL PRODUCT

The Application developed is a client-side, web-based plotting application rendering a 3D cube graph showing 10 values on each axis-.CSV data can be uploaded and the user can select which columns to render on which axis's, 3 of which are positional and one modifying the saturation of the plotted point. Provided user controls include buttons to allow the user to move each axis separately to show a different slice of values, zooming controls to increase or decrease the difference between each line on the whole graph together, rotation through mouse dragging or button clicking, changing data point sizes, modifying how saturation is applied to labels, selecting points on hover to view the values they represent. The application also allows the user to view the data uploaded in a table. The application can be accessed on any device with different screen sizes including mobile devices as long as OpenGL and a reasonably modern browser is available on that device. Although highly dependant on device specifications, the application can be reasonably expected to render data with 1,000+ Data points. With Modern laptops found to be capable of rendering at least 40,000 non-decimal Points with acceptable performance.

8 CONCLUSION

8.1 Appraisal

8.2 Future Work

The application as is has a great potential for future development. At the very minimum the following would be the most important modification to make:

- Optimize rendering

REFERENCES

[1] StatCounter Global Stats [n.d.]. *Desktop Browser Market Share Worldwide.* StatCounter Global Stats. https://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-202110-202110-bar (Accessed 4/12/2023).

[2] v8.dev [n.d.]. *V8 JavaScript engine.* v8.dev. https://v8.dev/ (Accessed 4/12/2023).

[3] webassembly.org [n.d.]. *WebAssembly.* webassembly.org. https://webassembly.org/ (Accessed 4/12/2023).

[4] Ravikant Agarwal and David Umphress. 2008. Extreme Programming for a Single Person Team. In *Proceedings of the 46th Annual Southeast Regional Conference on XX* (Auburn, Alabama) *(ACM-SE 46)*. Association for Computing Machinery, New York, NY, USA, 82–87. https://doi.org/10.1145/1593105.1593127

[5] Syed Mohd Ali, Noopur Gupta, Gopal Krishna Nayak, and Rakesh Kumar Lenka. 2016. Big data visualization: Tools and challenges. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. 656–660. https://doi.org/10.1109/IC3I.2016.7918044

[6] Amazon. [n.d.]. *What is Java? - Enterprise Java Beginner's Guide - AWS.* Amazon Web Services, Inc. https://aws.amazon.com/what-is/java/ (Accessed 4/12/2023).

[7] Apple. [n.d.]. Apple Developer Documentation. https://developer.apple.com/documentation/apple-silicon/porting-your-macos-apps-to-apple-silicon (Accessed 4/13/2023).

[8] Apple. [n.d.]. Metal. https://developer.apple.com/documentation/metal/ (Accessed 4/13/2023).

[9] BillWagner. 2022. A tour of C# - Overview. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (Accessed 4/13/2023).

[10] MDN Contributors. [n.d.]. JavaScript. https://developer.mozilla.org/en-US/docs/Web/javascript (Accessed 4/13/2023).

[11] davidbritch. [n.d.]. *What is .NET MAUI? - .NET MAUI.* learn.microsoft.com. https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0 (Accessed 4/12/2023).

[12] Robot Framework. [n.d.]. Robot Framework. https://robotframework.org/ (Accessed 4/14/2023).

[13] Morteza Ghobakhloo. 2020. Industry 4.0, digitization, and opportunities for sustainability. *Journal of Cleaner Production* 252 (2020), 119869. https://doi.org/10.1016/j.jclepro.2019.119869

[14] GrantMeStrength. 2021. Direct3D - Win32 apps. https://learn.microsoft.com/en-us/windows/win32/direct3d (Accessed 4/12/2023).

[15] Khronos Group. 2011. WebGL. https://www.khronos.org/api/webgl (Accessed 4/12/2023).

[16] Khronos Group. 2016. Khronos Releases Vulkan 1.0 Specification. https://www.khronos.org/news/press/khronos-releases-vulkan-1-0-specification (Accessed 4/12/2023).

[17] ™/®Khronos Group. 2017. The Official WebGL Logo. https://commons.wikimedia.org/wiki/File:WebGL_Logo.svg (Accessed 4/13/2023).

[18] W.S. Humphrey. 1996. Using a defined and measured Personal Software Process. *IEEE Software* 13, 3 (1996), 77–88. https://doi.org/10.1109/52.493023

[19] Kaoru Ishikawa. 1985. *What is total quality control? The Japanese way.* Prentice-Hall.

[20] Steve Klabnik and Carol Nichols. 2023. *The Rust Programming Language, 2nd Edition.* No Starch Press.

[21] Tingting Liang, Shan Lu, and Quansheng Liu. 2020. Data Visualization System Based on Big Data Analysis. , 76-79 pages. https://doi.org/10.1109/ICRIS52159.2020.00027

[22] Microsoft. [n.d.]. TypeScript - JavaScript that scales. https://www.typescriptlang.org/ (Accessed 4/13/2023).

[23] Microsoft. [n.d.]. *What is .NET? An open-source developer platform.* Microsoft. https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (Accessed 4/12/2023).

[24] Mozilla. 2019. WebGL: 2D and 3D graphics for the web. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (Accessed 4/12/2023).

[25] mrdoob. [n.d.]. three.js. https://github.com/mrdoob/three.js/blob/dev/README.md (Accessed 4/13/2023).

[26] Benjamin Nicoll and Brendan Keogh. 2019. *The Unity Game Engine and the Circuits of Cultural Software.* Springer International Publishing, Cham, 1–21. https://doi.org/10.1007/978-3-030-25012-6_1

[27] Node.js. [n.d.]. *Node.js.* Node.js. https://nodejs.org/en (Accessed 4/12/2023).

[28] Ekaterina Olshannikova, Aleksandr Ometov, Yevgeni Koucheryavy, and Thomas Olsson. 2015. Visualizing Big Data with augmented and virtual reality: challenges and research agenda. *Journal of Big Data* 2 (2015), 22. https://doi.org/10.1186/s4053701500312

[29] Oracle. [n.d.]. *What is Java technology and why do I need it?* Java.com. https://www.java.com/en/download/help/whatis_java.html (Accessed 4/12/2023).

[30] Stack Overflow. 2021. Stack Overflow Developer Survey 2021. https://insights.stackoverflow.com/survey/2021 (Accessed 4/12/2023).

[31] Mark Segal and Kurt Akeley. 2022. The OpenGL® Graphics System: A Specification (Version 4.6 (Core Profile) - May 5, 2022). https://registry.khronos.org/OpenGL/specs/gl/glspec46.core.pdf (Accessed 4/12/2023).

[32] Selenium. [n.d.]. The Selenium Browser Automation Project. https://www.selenium.dev/documentation/ (Accessed 4/14/2023).

[33] SmartBear. [n.d.]. TestComplete | SmartBear Software. https://smartbear.com/product/testcomplete/ (Accessed 4/13/2023).

[34] Smartbear. [n.d.]. TestComplete Pricing | Automated Software Testing Tool. https://smartbear.com/product/testcomplete/pricing/ (Accessed 4/14/2023).

[35] stevewhims. 2021. Direct3D 12 programming environment setup - Win32 apps. https://learn.microsoft.com/en-us/windows/win32/direct3d12/directx-12-programming-environment-set-up (Accessed 4/12/2023).

[36] Bjarne Stroustrup. 1986. An Overview of C++. *SIGPLAN Not.* 21, 10 (jun 1986), 7–18. https://doi.org/10.1145/323648.323736

[37] Unity Technologies. [n.d.]. Unity - Manual: Graphics API support. https://docs.unity.cn/Manual/GraphicsAPIs.html (Accessed 4/13/2023).

[38] threejs.org. [n.d.]. three.js docs. https://threejs.org/docs/index.html#manual/en/introduction/Installation (Accessed 4/13/2023).

[39] Johannes Unterguggenberger, Bernhard Kerbl, and Michael Wimmer. 2023. Vulkan all the way: Transitioning to a modern low-level graphics API in academia. , 155-165 pages. https://doi.org/10.1016/j.cag.2023.02.001

[40] OpenGL Wiki. 2022. History of OpenGL — OpenGL Wiki,. http://www.khronos.org/opengl/wiki_opengl/index.php?title=History_of_OpenGL&oldid=14895 (Accessed 4/12/2023).

[41] ™/®Microsoft. 2020. Typescript logo 2020. https://commons.wikimedia.org/wiki/File:WebGL_Logo.svg (Accessed 4/13/2023).