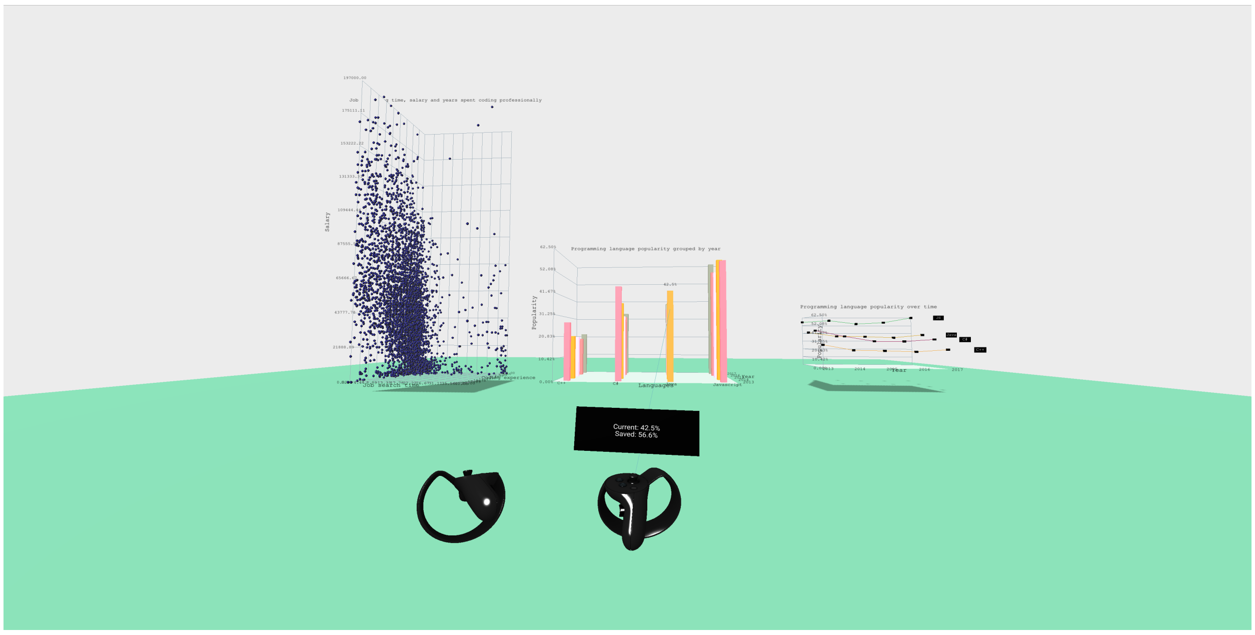
CHALMERS
UNIVERSITY OF TECHNOLOGY



graphVR

A VR web application for interactive 3D-graphs

Bachelor's thesis in Computer Science and Engineering

MIRANDA ALDRIN HENRIK BOSTRÖM ROBIN EDQUIST
OSCAR JOHANSSON JACOB NILSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Bachelor's thesis DATX02-18-16

graphVR

A VR web application for interactive 3D-graphs

MIRANDA ALDRIN, HENRIK BOSTRÖM, ROBIN EDQUIST,
OSCAR JOHANSSON, JACOB NILSSON

Department of Computer Science and Engineering
Division of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

graphVR
A VR web application for interactive 3D-graphs
MIRANDA ALDRIN, HENRIK BOSTRÖM, ROBIN EDQUIST, OSCAR JOHANSSON, JACOB NILSSON

Supervisor: Thommy Eriksson, Department of Computer Science and Engineering
Examiner: Morten Fjeld, Department of Computer Science and Engineering

Bachelor's Thesis DATX02-18-16
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The open world scene in the application graphVR.

Typeset in LaTeX
LaTeX-template created by DAVID FRISK, 2016
Gothenburg, Sweden 2018

graphVR
A VR web application for interactive 3D-graphs
MIRANDA ALDRIN, HENRIK BOSTRÖM, ROBIN EDQUIST, OSCAR JOHANSSON, JACOB NILSSON
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

The study presented in this paper is a Bachelor's thesis at Chalmers University of Technology. The purpose of the study was to find out how to transform statistical tabular data into interactive VR graphs. It was achieved by creating the application *graphVR* using the framework *A-Frame*. The project was carried out as a technical development project, which used user tests and continuous evaluation to guide the design of the application.

The project was based on an iterative development process, and produced two prototypes, used during the user tests, and one final product. The thesis discusses the complete method of developing *graphVR*, including the system architecture, the environmental models used, and the graph components which were created specifically for the application.

Keywords: virtual reality, information visualization, JavaScript, A-Frame, VR, Three.js.

Sammandrag

Studien som presenteras i den här rapporten är ett arbete gjort på kandatnivå på Chalmers. Studiens mål var att undersöka hur man kan presentera statistisk data i en interaktiv VR miljö. För att uppnå målet skapades applikationen *graphVR* som använde sig av ramverket *A-Frame*. Projektet var en teknisk utvecklingsstudie som använde sig av användartester och kontinuerlig utvärdering som en guide för designen av applikationen.

Projektet använde en iterativ utvecklingsprocess och producerade två prototyper, som användes för användartesterna, och en slutprodukt. Rapporten diskuterar hela metoden som använts för att producera *graphVR*, inklusive systemarkitekturen, 3D modellerna som användes och grafkomponenterna som var specifikt skapade för applikationen.

Acknowledgements

We want to thank our supervisor Thommy Eriksson for his extensive help throughout the project. We also want to thank the people who helped us in the user tests and the IT faculty for giving us access to their media lab.

Miranda Aldrin, Gothenburg, May 2018
Henrik Boström, Gothenburg, May 2018
Robin Edquist, Gothenburg, May 2018
Oscar Johansson, Gothenburg, May 2018
Jacob Nilsson, Gothenburg, May 2018

Contents

List of Figures

List of Tables

1
Introduction

Data has been collected by humans for thousands of years [1, p. 1-2]. To make this information easier to interpret, data visualization has been used to form a tangible representation of the information [2]. It helps to manage the collected data as well as predict what could happen if a set of conditions are met [3, p. 1]. The point of data visualization is to visually show the relationships between variables, and the more variables analyzed, the more discoveries could be made.

One way to analyze the collected data is to use information visualization, a type of data visualization. **Information visualization** is a computer-supported interactive visual representation of abstract data to help amplify cognition [3, p. 13]. The fact that information visualization is interactive aids the understanding of the data. However, currently most diagrams are viewed on 2D platforms, such as computer screens or tablets and because of the nature of the 2D platform, there are some limitations with the interactions. This is where we believe a virtual reality information visualization application could shine.

Virtual reality (VR) has developed significantly since the 1990's [4, p. 158]. Early VR headsets were clunky, caused motion-sickness and disorientation. However, with new headsets such as *Oculus Rift* [5] and improved motion tracking, VR has become a viable option for game development and virtual experiences. Because of this, VR is making its way to the commercial markets [6] and might even become mainstream in the future. It lends itself well to situation awareness and interactivity [7, p. 610]. Thus, the nature of VR could provide a way for us to display multi-dimensional graphs and see trends in the data not seen on a 2D platform. When viewed as a virtual experience it would be possible to use this technology to continue the exploration of multi-dimensional graphs and bring out new patterns.

It is with this premise that we decided to develop graphVR, a VR web application for interactive 3D-graphs. The idea was to see if *A-Frame* [8], a web framework used to build virtual experiences, is suitable for developing a VR application for information visualization.

1.1 Purpose

The purpose of this project was to develop a web application to explore the grounds of information visualization in VR. The application should allow the user to analyze and interact with statistics in a virtual environment by using VR controllers. One of the difficulties to overcome was to transform statistical data sets to 3D graphs. To ensure that the best possible design choices were made during the design process, user tests were performed to evaluate the application further with external test subjects.

1.2 Problem Statements

Questions connected to the design of the application:

- How to design useful interactions for information visualization software
- How to create a VR application that is easy to use for the user
- How to design an environment suitable for a virtual reality application

Questions connected to the development of the application:

- How to transform tabular data into graphs
- How to model graphs as *A-Frame* components
- How to create environments with lights and shadows in *A-Frame*

1.3 Scope of the Project

The statistical viewing tool, *graphVR*, is a Bachelor's thesis project of five students from Chalmers University of Technology. All members of the group have a background in computer science and have spent 400 hours per person on the project. The 400 hours consisted of developing and documenting the application *graphVR*, writing this report, giving two oral presentations and attending lectures held by the university. No group member had prior experience in developing nether web applications nor VR applications.

In *graphVR*, the user is able to interact with data presented in the form of graphs to facilitate the understanding of the graphs. Since *graphVR* is a web application it is not accessible through *Oculus store* [9], *Steam* [10] or any other distribution platform. The headset we developed for was primarily Oculus Rift since this is what was accessible to us during the project. *graphVR* handles three types of diagrams: bar graph, line graph, and scatter graph. It allows the application to have some variation and still be developed within the short time frame.

Our product was developed in English and our target audience is English speaking users. English was chosen to be able to reach a broader audience. Physical and mental abilities or disabilities were also not taken into account. Making the product accessible to all was not a priority.

Statistics may not directly harm people, but manipulated statistics could be indirectly harmful. It could result in decision makers coming to the wrong conclusion. In this project our aim is not to handle these ethical questions but instead to make the graphs as correct and easy to read as possible. This works indirectly towards handling these ethical dilemmas but is again not our goal.

1.4 Tools

To create *graphVR*, specific tools were needed. The application as developed using the framework *A-Frame* [8] together with the programming language JavaScript. However, not all parts of the application was possible to produce using JavaScript. 3D models were used throughout the application and the models were made using the 3D modelling software *Blender* [11]. The application was also mainly developed for the VR head mounted display (HMD) Oculus Rift.

1.4.1 Frameworks and Programming Languages

Choosing what framework and the programming language to use when developing an application is a crucial part of the development process. The framework *A-Frame* was used because of its simplicity and cross-platform capability. Other frameworks like Unity is also a good choice for developing VR applications, but the advantage with *A-Frame* is that even though there are homemade solutions for porting VR apps created in *Unity* [12] to run on the web with *webVR*, *A-Frame* does this native. *A-Frame* is built on top of *Three.js* [13], which is a graphics library written in the JavaScript programming language.

There are many frameworks made for unit testing in *JavaScript*, but we settled on using *Mocha* [14] to run the tests and *Chai* [15] for assertion. *A-Frame* uses both *Mocha* and *Chai* for testing of the framework, therefore it was easy to find information on how to adapt the tests to *graphVR*.

1.4.1.1 A-Frame

Three.js is a lightweight graphics library that uses WebGL to create GPU-accelerated 3D animations using the JavaScript language. The library provides *<canvas>*, *<svg>*, *CSS3D* and *WebGL* renderers, which commonly used for rendering on the web [13].

A-Frame is a web-framework that enables developers to make WebVR with HTML and entity components. WebVR is an open specification that is used to bring VR to the web environment. Just because *A-Frame* is based on HTML, it is easy to understand for a developer as well as for a designer. Since it is built on top of HTML, it also allows the use of other frameworks and libraries as *Node.js* [16] and *React* [17]. *A-Frame* also supports most VR headsets such as *Vive*, *Rift*, *Windows Mixed Reality*, *Daydream*, *GearVR*, *Cardboard* and could also be used for augmented reality. The main core of *A-Frame* is its component entity system and is built on top of *Three.js* [18].

1.4.1.2 JSON

Javascript is a convenient language when it comes to data collection with *JSON*. *JSON* stands for JavaScript Object Notation which is a format that is commonly used when transmitting data in web applications[19]. In this project *JSON* is used as a way to collect data from statistical databases and transfer it to our application. The way graphVR does this is to translate *JSON* objects to *Javascript* objects and then use the object for visualizing the data [19].

1.4.1.3 ECMAscript

ECMAscript is a standardization that was mainly created to standardize *JavaScript*. This scripting standard is object-based, which means that an *ECMAscript* program uses clusters of objects that has the ability to communicate with each other. Large *ECMAscript* programs are supported by modules, where each module explicitly identifies declarations it uses that need to be provided by other modules and which of its declarations are available for use by other modules [20].

The modules are managed by *Webpack*, which is a static module bundler for modern *JavaScript* application and the server that we used was managed by *Webpack Dev Server*. When *Webpack* processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles. In this project *Webpack* was used to easily add new *Javascript* modules and keep track of which ones are used [21].

1.4.2 VR Hardware

The headset and controllers we used was Oculus Rift, henceforth referred to as the Rift. The Rift is a VR platform that allow for six degrees of freedom (DOF) [22] giving it full mobility and makes it work in a room-scale VR experience. **Degrees of freedom** refers to the different movements tracked in the VR environment. They are categorized into two groups: pitch, yaw, roll and sway, surge, heave. It is called 3-DOF (three degrees of freedom) when pitch, yaw and roll allows are present, which allows you to rotate in all three space axis. 6-DOF (six degrees of freedom) is the presence of the first three DOF combined with sway, surge, and heave which not only let you rotate in all three space axis, but also move in all three spatial directions [23, p. 68]. A visual explanation can be seen in Figure 1.1.

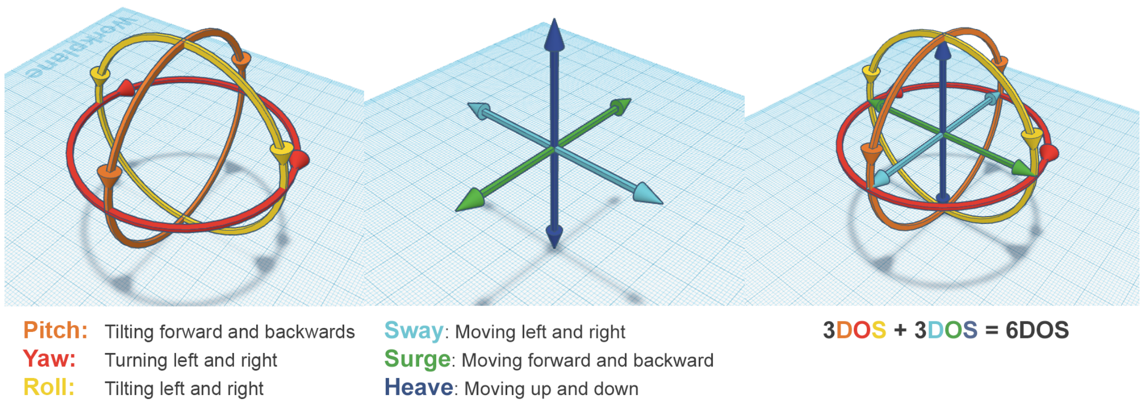

Pitch: Tilting forward and backwards
Yaw: Turning left and right
Roll: Tilting left and right
Sway: Moving left and right
Surge: Moving forward and backward
Heave: Moving up and down
3DOS + 3DOS = 6DOS

Figure 1.1: Figure describing the different Degrees Of Freedom.

The Rift controllers enables six degrees of freedom and while the application would be manageable with the first three, six degrees makes for a much better experience. The headset has a resolution of 1080x1200 per eye, 110 degrees field of view and a refresh rate of 90 Hz [24]. The resolution affects how well you can see details and in our case we had to consider how well text would be readable. The high refresh rate is good since it reduces nausea from motion sickness. The motion sickness occurs mainly when the camera is moving while the user's head is still, making what the users see not align with what the user feel.

1.4.3 Modelling Software

Some parts of *graphVR* needed to be made with a 3D modelling software. We used two types of methods to produce models. Most models were made by our own designers, specifically for *graphVR*. The second method we used was to modify models taken from online 3D modelling libraries.

A-Frame [8] recommends using one of four types of programs to make 3D models: *Blender* [11], *MagicaVoxel* [25], *Autodesk Maya* [26], or *MaxonCinema4D* [27]. Both *Autodesk Maya* and *MaxonCinema4D* are expensive software with an extensive library of features. Most of these features were unnecessary for the purpose of creating models for *graphVR*. That left us with *Blender* and *MagicaVoxel*. They are both open-source software with a supportive community and multiple tutorials. Ultimately, *Blender* was chosen because of its intuitive modeling workflow [28]. There were also a lot of learning resources for *Blender* which contributed to the decision.

Most models were made from scratch in *Blender*. However, there was one model which was quite complicated and thus part of the model was taken off of *Thingiverse* [29], a 3D sharing platform, and was modified to fit our needs.

When exporting a model, there are multiple file formats that can be used. *A-Frame* supports *glTF*, *OBJ*, and *COLLADA* when using 3D models in an application

[30]. The *glTF* format is used as it provides an advantage over the other two formats. It is an open-source “specification for the efficient transmission and loading of 3D scenes and models by applications” [31]. In *A-Frame*, *glTF* offers “scene information”, such as information about lights and cameras, and provides “more robust materials and shaders” [32], which is why it is the recommended format. The Khronos *glTF* exporter extension [33] was used to be able to export the model to *glTF* format from *Blender*.

1.4.4 Version Control

Since a big part of this project was the development of the web application, a lot of work was done through coding and multiple people was working on the application at once, it is important to have a clear and structured workflow when working with the code. There are many tools available to help with version control, like *Github*, *Bitbucket*, and *Cloud Source by Google*. All of these would have provided sufficient support, as the differences between them are not too big but as management of *Github* is taught at Chalmers all members have good knowledge of how to work with that system. Therefore to accomplish this the we used Github[34] for version control as it saved time that would be needed to learn a new version control system. For code management within the repository the following workflow was used (see figure 1.2).

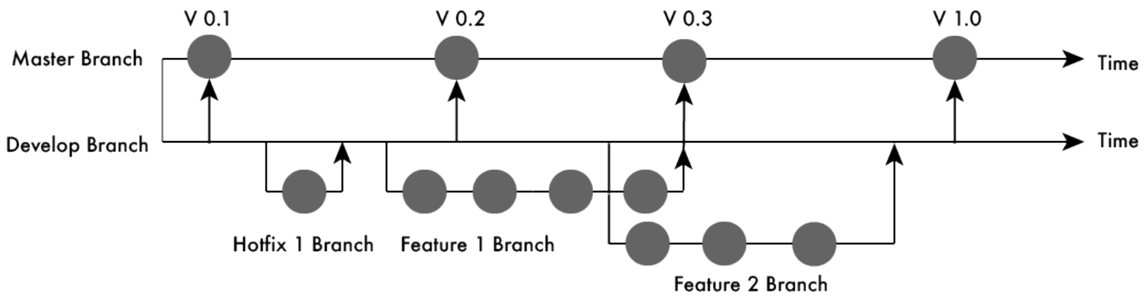


Figure 1.2: Illustration of the *Github* workflow used during the project.

2
The Design Process and Evaluation of the Application

According to Jerald in his book *The VR Book: Human-Centered Design for Virtual Reality*, the recommendation for designing a VR application is to use an iterative design process [35, p. 369]. Designing for virtual reality is quite different from designing for a 2D platform as there are few standardized design rules and generally there is a lack of knowledge when it comes to designing for VR. Because of the lack of knowledge, both for us as designers, and in the business as a whole, it was important to achieve continuous discovery. **Continuous discovery** is the ongoing process of engaging users during the design process [35, p. 374]. We always strove towards discovering new insights. The goal was to continuously understand what users wanted to do in the the application, why they wanted to do it, and how they could do it in the most efficient way possible. Using continuous discovery helped us as designers to understand the users, and it was then possible to apply that knowledge to *graphVR*.

The iterative approach was also chosen because it worked well with our agile development process. More about the agile development process can be found in Section 2.2. It was important to rapidly prototype our designs, and due to time constraints, the development team always had to have new designs to work on. Therefore, the iterative design process was best suited for designing *graphVR*.

We followed the high-level iterative process Jerald recommends, as seen in Figure 2.1. It consists of:

- The **define phase**, which answers the question “What do we make?” and it includes creating requirements, storyboards, and user stories.
- The **make phase**, which answers the question “How do we make it?” and includes the actual process of designing prototypes and making the final product.
- The **learn stage**, which answers the question “What works and what does not work?” and the answers are then used in the define stage to help define the next iteration.

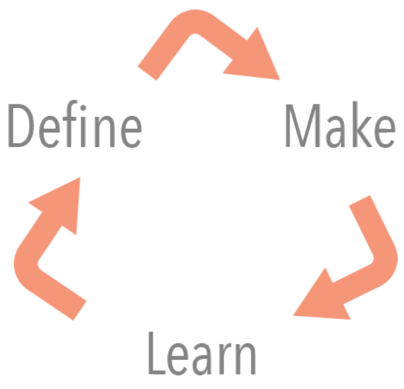


Figure 2.1: The high-level iterative process as seen in Jason Jerald's book *The VR Book: Human-Centered Design for Virtual Reality* [35, p. 370]

In theory, each phase is a separate phase and performed in sequence, however during our project some stages were interwoven and performed in parallel to each other. This is a common occurrence according to Jerald [35, p. 371]. For example, while designing a new element or fixing an old one during the make phase, it was not uncommon to learn from the existing designs while we were doing it, because we continuously kept testing the product during the development. This made it possible for us to find more insights of what worked and what did not.

Because of a limited time frame, we chose not to perform all the design methods suggested for each phase in *The VR Book: Human-Centered Design for Virtual Reality*, but hand picked the methods that were the most relevant for our project.

2.1 Define Phase

In the beginning of a project, the define phase is when the overall problem and the general design idea are defined [35, p. 379]. However, these definitions do not need to be concrete from the beginning since it is possible to narrow down the definitions in the next iteration. This means that throughout the project, the definition of the problem and the design will evolve as the project and application evolves.

This was how *graphVR* developed. When the project began, the scope of the project was very wide. Because of this, the first iteration was based on a general idea of the application and we kept defining the project throughout the design process. It took many iterations before it was possible to form a concrete idea of the final design.

During each iteration, many changes to the design can occur, so throughout this process, it is important to continuously justify these changes [35, p. 379]. We tried to thoroughly justify each change to the design so that it was known why the change was made. When justifying changes, we also tried to define what it was we were trying to achieve and why it was being done.

To aid the define phase, we used multiple design methods: user stories, storyboards, and design requirements. All methods were chosen to aid in the communication between designers and developers

In agile development, **user stories** are often used to define short concepts or functions a user might want to see in the application [35, p. 392]. However, this method can also be used for the design elements of an application. We used user stories for both the development and design of *graphVR*. All stories were written from the point of view of the person who would benefit from the element (developer, user, designer). The story also stated a goal (what the person wants) and a reason (why the person wants it). A collection of some of our user stories can be seen in Appendix A.

The storyboard method is a method where interactions are sketched out in a similar manner as a comic strip, where each interaction is one frame in the strip. It is used to easily convey how an interaction will lead to an event, which can be quite an abstract idea to try to pitch to the rest of the development team. The storyboard used can be found in Appendix H.

There are both strengths and weaknesses to this method when it comes to storyboards and VR development. On one hand, they can be a good supplement to clearly convey interactions with objects. But if those actions are non-linear, a linear method, such as a storyboard, is not the optimal method. Because of this, storyboards were mainly used in the design of the more linear scenes in the application, as seen in Appendix H.

Based on our user stories and the storyboards we developed, design requirements were created for the application. A list of **requirements** is a list of expressions "such as descriptions of features, capabilities, and quality" [35, p. 395]. They should be concise and definitive statements to be used to determine if the system is working as expected.

During each new iteration, the design requirements were updated for the project to reflect what we had learned from the previous iteration. The design requirements can be seen in Chapter 3.

2.2 Make Phase

The make phase is where all the implementation of the application or prototype occurs [35, p. 401]. For *graphVR*, two prototypes were produced for the user tests discussed in Section 2.3 as well as a final product, using the design requirements decided upon in the previous phase. It was during this phase that all the tools presented in Section 1.4 were used. For example, the graphs used in *graphVR* were developed and the environments surrounding those graphs were modelled.

The traditional way of working in projects are called waterfall. This is when the development is split into 4 phases: analysis, design, code and test. Then each phase is performed consecutively, one phase at a time [36]. This in contrast to agile development where all phases are done continuously throughout the project from start to end [36]. Studies have shown that an agile development style improves both the quality and development time of the product compared to the traditional waterfall model [37]. Improved quality and speed is desirable and therefore it was decided that an agile development process should be applied. The agile process used during this project is not a complete, already defined process such as *Scrum*, instead it was based on previous competences in the project group of agile development processes. We decided to do it this way since learning the complete process of a new method would be time consuming and not worth prioritizing in this type of project. A visualization of this work methodology is presented in Figure 2.2.

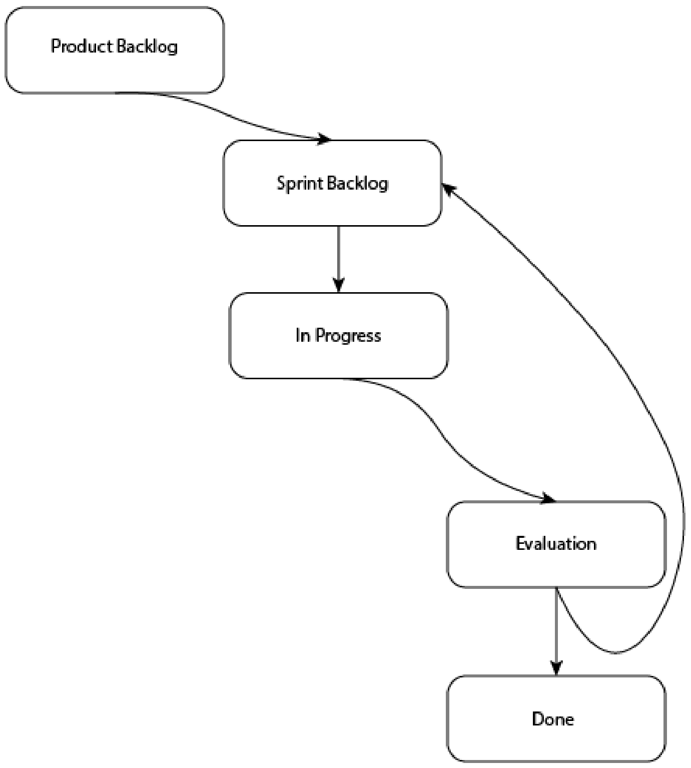


Figure 2.2: An illustration of the workflow used during the project.

A **sprint** is a set period of time for each iteration of the process. In our case, this was one week. A **backlog** is a list of items that is a suggestion or a requirement to finish. There is a general **product backlog** where any possible additions to the application can be added and **sprint backlog**, which is the list of items that should be completed during the sprint. In the start of each sprint, a meeting was held and the **product backlog** was discussed to decide what should be moved to the **sprint backlog**, the previous sprint was also evaluated during this meeting. When the backlog item is implemented, it needs to be evaluated and if any changes needs to be made, the item is placed back in the sprint backlog to be adjusted in the next sprint, otherwise it’s marked as done and no changes are made to this part of the application if not necessary.

According to the agile process, we continuously evaluated VR applications and statistics programs that already exist on the market, see Appendix B. Doing this gave us an understanding of what features would work well in *graphVR* as it allowed for testing and evaluation of features without having to implement the features in our application. This helped us to not only make better design choices, but to also implement them quickly. Design choices that worked well in other applications did not always work as well in *graphVR*. That is when the evaluation of *graphVR* came into play and we had to figure out a way to adjust the implementations to work well for our scenario, or simply find a solution that would work better in our case.

To test the quality of the code, specifically the vital data processing parts of the code that transform the tabular data to graphs, traditional unit testing was used. **Traditional unit testing** is the practice of writing unit tests after writing the application. A unit is the smallest working part of the application like a method or class [38, p.10], and the idea is to divide the application into units and write tests for all units.

Another method used was sketches. They were used to formalize design ideas and communicate them to the development team. Sketches are rough drawings, not intended to be finalized work, but only used as a rough guide for the actual development of the product.

Because of the short time span of the project, most of our implementations in the make phase aimed towards the final product. Not all implementations were as perfected as a final product, but we continued building on the existing design and implementation during each iteration.

2.3 Learn Phase

During the learning phase, the goal is continuous discovery [35, p. 427]. Because of the fact that there are fewer standards for designing in VR, it is important to constantly try to learn from all parts of the design process, even during phases outside of the learning phase. While it is a good idea to be focused on learning during all phases of the project, the learning phase is dedicated to it.

Throughout all phases, but especially in the learning phase, we actively sought out any difficulties for the user and anything that could be considered a failure in the application. Finding failures in an application can help with the design of a better solution [35, p. 428]. We did this by continuous evaluation during the development of *graphVR* as well as performing two user tests.

In order to gather qualitative information about our design choices, two user tests were performed. Before the tests were performed, test documents were written as per Hoa Loranger's recommendations in the article *Checklist for Usability Studies* [39]. In the test documents, general goals were decided upon as well as the tasks the users were to complete. The tests were qualitative studies and because of the limited time-span, the participants were students from the university at both Bachelor and Master levels. The test documents for user test 1 and user test 2 can be found in Appendices C and D.

According to Nielsen Norman Group, the optimal sample size for a qualitative user test is five participants [40]. The return of investment, the cost versus usability findings, is the greatest around five participants because the test will not produce enough new usability findings for the cost of increasing the sample size. In our case, time will be regarded as the cost, and since we do not have a lot of it, the sample size chosen was five participants. However, during the first test the sample size ended up being four people, as one user dropped out at the last minute.

The **thinking aloud** method was used for the user testing [41]. The method is frequently used when performing user testing on regular platforms such as computers and smartphones, and was also useful with a VR environment. The method urges the user to think out loud as they are performing the tasks of the test. The method gives the facilitators an idea of why the user performed the task as he did. While the observations of the test provide better understanding of what is wrong with the product, the thinking aloud method allows the facilitators to also understand why the users did as they did. Especially in a VR environment, where the users are disconnected with the facilitators during testing. They are disconnected because they can hear the surroundings, but not see it.

Our target audience for the application was people with a wide range of VR experience. We wanted the application to be accessible by both new users as well as experienced users. This was reflected in the user tests, as there were both novices as well as advanced users. In the first test, three people had only tried a VR headset one to two times, for roughly 15 minutes per time. The last user in the test owned an HMD. For the second test, all users had used a headset before. Two people had developed applications for VR and tried it thoroughly throughout the development process. Two people had limited experience, and had only demoed a VR headset 2-3 times before. The last person owned his own headset and has advanced experience with VR.

2.3.1 Ethical Testing Procedure

Testing in VR comes with some privacy issues, depending on how the data from a user is handled. In both tests, the user's physical and virtual presence was recorded. Because of this we formed a contract, seen in Appendix E, with each test subject giving us permission to use the data in our research. The contract included a clause, which allows the user to end all involvement in the test, including allowing the subject to request a deletion of the data collected.

We only recorded the users during the user testing. We never record anything from the user after this.

Before each user test, the user was informed about the possible nauseating effects the head mounted display can cause. The user was always able to abort the test temporarily or permanently at any time.

2.3.2 Camera Set Up

When a user uses an HMD, it is harder to read the person's face when they are performing a task. To be able to analyze the user's actions in accordance to what is happening on the screen, the test was recorded using both a video camera as well as a screen recording software. Both tests recorded audio as well. The camera used in the first test was a Canon 5D Mark IV and the second test was recorded using a Fujifilm X-T20. The screen recording software used was *OBS Studio* [42].

The floor plan of the set up of the video camera can be seen in Figure 2.3. As seen in the figure, the camera was set up behind the monitor, filming the user from the front and capturing the full body movements. The facilitator of the test was positioned to the left of the user and the scriber to the right. A **facilitator** is the person who interacts with the user during the test and tells the user what tasks to complete [43]. The **scriber** records observations and takes notes on the answers to the interview questions [43].

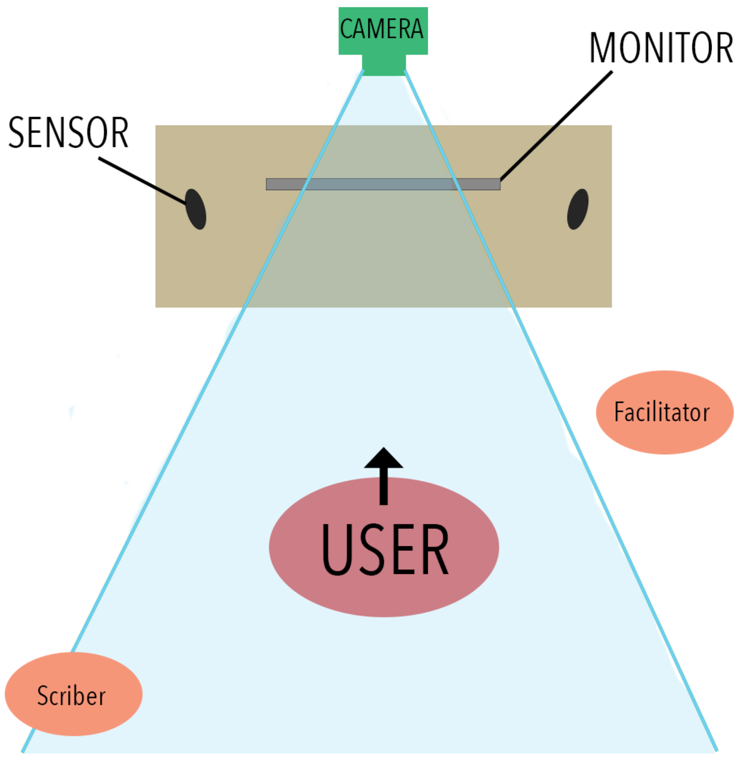


Figure 2.3: The camera set up for the user tests. The blue area indicates what the camera included in the recording.

The set up allowed us to use the recordings when evaluating the results of the test. VR is also much more physical than regular devices, and it is therefore important to capture the physical observations of the user to, for example, see if they tried to move the controller in a way that we had not thought of before.

2.3.3 Test structure

The two user tests were performed to ensure that the development of the application was going in the right direction. The general test structure can be seen in the flowchart featured in Figure 2.4.

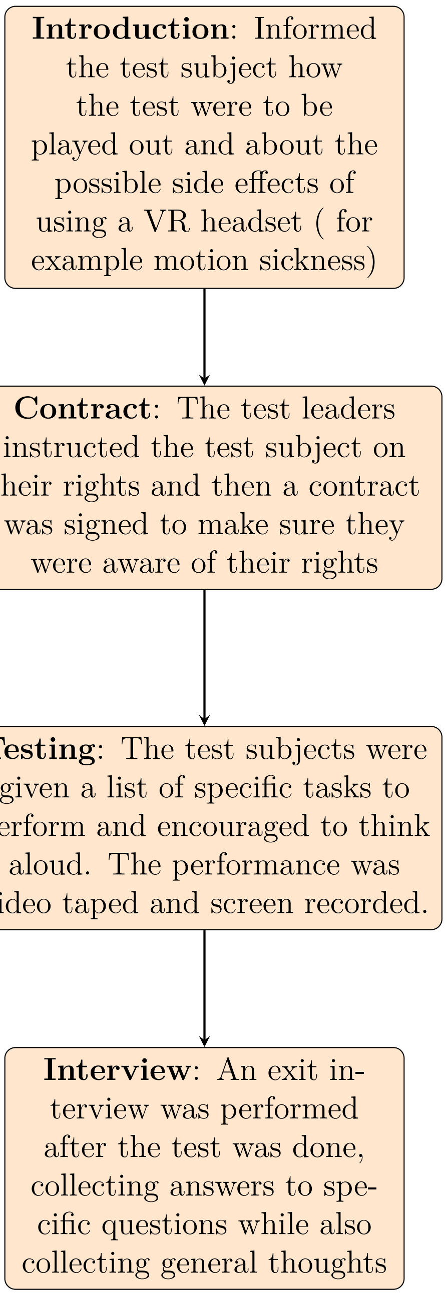

Introduction: Informed the test subject how the test were to be played out and about the possible side effects of using a VR headset (for example motion sickness)
Contract: The test leaders instructed the test subject on their rights and then a contract was signed to make sure they were aware of their rights
Testing: The test subjects were given a list of specific tasks to perform and encouraged to think aloud. The performance was video taped and screen recorded.
Interview: An exit interview was performed after the test was done, collecting answers to specific questions while also collecting general thoughts

Figure 2.4: Flowchart of participants' progress through the phases of the trial

The users were introduced to the test and were told about the side-effects of wearing HMD's as well as how the test was going to be performed. Then, a consent form was presented to the participants, as seen in Appendix E. They were also informed that they could stop the test at any time and to contact us if they want their recordings and data deleted.

The first test was set up in a way to help us understand how the users would interact with the VR system as a whole and had more **specific tasks**, tasks that have a narrower focus and usually results in a right or wrong answer [39]. The four tasks performed during the test, as well as the purpose of the task can be seen in Table 2.1.

Table 2.1: A table of the specific tasks used in User Test 1, and why they were performed.

Task no.	Task	Purpose
1	How many summer homes were sold in Stockholm during Q2 in 2016?	To check how the user interacts with the bar graph, if they use the interactions as they were intended to be used
2	What is the trend for summer home prices in Stockholm over the past decade?	To see if the graphs are able to be analyzed, even in a 3D environment
3	Give an example of a county with a large population and a high median wage.	To see how the user interacts with the scatter graph, as well as if they use the interactions present in the application
4	Give an example of a county with a high median rent.	To see how the user interacts with the scatter graph, as well as if they use the interactions present in the application

The main goal of the test was to see how users would use and interact with the graphs in the application. We wanted answers to general questions, such as would the user understand how to read the graphs, was the controls hard to understand, and did the room feel claustrophobic. We wanted to know if we should keep working on the existing design, or if we needed to redesign the application and move in a different direction. Since no one in the group had worked with VR before, we needed to test out fundamental things, such as if the navigation control we had chosen was a good fit for *graphVR*.

In the second test, the product was more developed and it was possible to do a more in depth test of the system. Therefore, there were both exploratory and specific tasks, testing interactions as well as the user's personal experience of the whole system. **Exploratory tasks** are open-ended tasks created to get the user to explore the system and find out how the user interacts with it [39]. The five tasks performed during the test and their purpose, can be seen in Table 2.2.

Table 2.2: A table of the specific tasks used in User Test 1, and why they were performed.

Task no.	Task	Purpose
1	Do the tutorial and then continue to the museum.	To check if the user is able to complete the tutorial without help and learn how to perform the interactions
2	Explore the museum and tell us your initial reactions	To get insights into what the user thinks about the museum as a whole
3	What was the three most common programming languages in 2017?	To see how the user interacts with the bar graph, as well as if they use the interactions mentioned in the tutorial
4	Can you see any general trend in the relation between job searching time, salary, and years spent coding professionally?	To see if the user still can analyze the 3D graphs, even after the changes made in the second prototype
5	Choose any data point and tell us how much time the person spends on looking for a new job.	To see how the user interacts with the scatter graph, as well as if they use the interactions taught in the tutorial

The test document for the second test can be found in Appendix D. Since the application was further along in the design process, it helped with confirmation of the design choices.

A structured interview was conducted at the end of each session. The questions and observations from the first and the second test can be found in Appendix F and G respectively. All questions collected subjective measurements, measurements about the ease of use or satisfaction of the product [39], and the questions were mostly questions about how the system was to use, if the controls were easy to learn and use, and if the application was frustrating to use.

Once the tests had been completed, a list of problems with the application was written and used in the design phase in the next iteration to design and develop fixes for those problems.

3
Design Requirements

The final version of *graphVR* is a tool for viewing graphs in a virtual environment on the web. The application consists of four different scenes: a tutorial, a museum, a world where you are placed within a graph and an open world. In these scenes three different types of graphs are presented. The graphs can be interacted with in various ways that allow for easier readout.

Design requirements are a set of statements that defines the features, capability, and quality of the product which is developed [35, p. 395]. The design requirements helped the development team to prioritize what to work on as the highest priority was to ensure the entire list was satisfied by the final product.

The requirements were established in multiple ways. Some were already specified by the project specification whereas others had to be specified during the project through research, testing, and evaluation. Since many of the requirements have been specified throughout the project through evaluation of earlier versions of the application, the list of requirements are fulfilled by the final product. The final list of requirements can be found in the list below.

- The application should have VR-Support
- The application should run on the web
- The application should support Oculus Rift
- The application should be developed using *A-Frame* combined with *Three.js*
- The application should clearly present data in a way where no data is hidden
- The application should be able fetch data and and transform the data to make it suitable for visualization
- The application should be able to visualize data
- The application should teach new users how to interact with the application
- The user should feel that the application is fun to use
- The user should feel that the application is easy to use
- The user should be able to draw some conclusion from analyzing the data
- The user should not feel claustrophobic when using the application
- The user should not have any issues understanding the controls
- The user should not become sick by using the application
- The user should be in a closed environment
- The user should be able to use use the application without help
- The user should be able to move around in the environment
- The user should be able to move a graph
- The user should be able to see the graph from every angle
- The user should be able to rotate a graph

- The user should be able to see the value of a data point
- The user should be able to select a data point and compare it to other data points

4 Using the A-Frame Framework and System Architecture

A-Frame [8] is a framework that is built on the *Three.js* [13] application programming interface (API) for *Javascript* [44] that uses the entity-component-system (ECS) architecture [18]. This architectural type is a common choice within 3D application and game development [18]. ECS follows the composition over inheritance and hierarchy principle [18], which can be read about more in detail here in Java; desing: objects, UML, and process by Knoernschild [45].

As stated in the name, this architectural pattern is built up by three parts, *entities*, *components*, and *systems* [18]:

- An **entity** is a container where components can be attached to construct something new.
- **Components** are reusable modules which can be attached to entities, to give that entity some sort of attribute, like appearance, behaviour or functionality.
- **Systems** are global services that can be used to handle groups of components, in our case this would be the scene in which all the entities are placed.

A-Frame includes many pre-built components, where you can modify the components of an entity when placing it in the virtual world. However, the pre-built components are not enough for every project, like this one. Therefore some of the entities used within *graphVR* have been built from scratch using *Three.js* but they can be used the same way as the pre-built components. An example of components which are developed specifically for this project are the graphs. Using only pre-built components could also introduce some problems in the form of performance issues as these are built for as general of purpose as possible, therefore rewriting simple components and using those instead improved the performance of the application notably.

To create custom components in *A-Frame*, *Three.js* is used. To create an object in *Three.js* **materials** which describe the appearance of an object, is combined with **geometries** which describe the shape of the object, to create a **mesh**. This mesh combined with behaviour and logic can then be used as a component in *A-Frame* [46].

As stated above, the components are created using *Three.js* [47]. When placing the components in the virtual world, it is done through a *HTML* [48] document, an example of which can be found in Listing 4.1 [18].

,

```html
    <a-entity
        grid bars="color:blue" position="1 2 0";
    ><a-entity>
```

Listing 4.1: Code example of how objects are placed in the virtual environment using *HTML*

<a-entity> is the element name of the tag, explaining to *A-Frame* that a generic entity should be created. The **grid**, **bars**, and **position** then tells the tag what attributes that be included. In this case, the entity should be a grid entity, consisting of bars placed at a specific position. In some cases an attribute value is needed, like in the case of position. This value tells the graph at what position the graph should be placed relative to the scene. The way this is used in *graphVR* is that a graph is defined as a type of grid depending on the data in the graph is numerical or categorical and also what content the graph should contain and present the data as, namely bars, point-clouds or lines. The way this is built makes future expansion of *graphVR*. All that is needed is to build the new contents to present the data in a new way. Also, thanks to this way of building the application, it makes it really simple to present the same data in many different ways, as not all data is suitable for all types of graphs. Therefore numerical and categorical data is separated in our implementation of *graphVR*, as some operations used to generate the graphs only work with one type of data.

5
3D Graphs

Graphs are visualizations that present the relationship between groups of numerical data [49, p.164]. The term 3D graph, can according to Harris [49, p.401], be interpreted either as a graph with three axes, or as a graph that has two axes and an added cosmetic depth, to look 3D [49, p.401]. From this point on, we will refer to 3D graphs as graphs that have three axes. 3D graphs can cause problems if they are not able to be seen from different angles, since it could cause data points to be hidden from the reader. Hidden data points are a problem if the graph is displayed in a static context, for example, on a paper or in a computer application where the camera or the 3D graph can not be moved or rotated.

We chose to implement 3D graphs in *graphVR* because hidden data points can be seen by moving around a 3D graph in a VR environment or by rotating a 3D graph using a VR controller. The types of 3D graphs that have been implemented in *graphVR* are the following: a 3D bar graph, a 3D line graph and a 3D scatter graph. Every 3D graph has some common attributes, such as a 3D grid, a title, axis tick labels and axis legends. Each 3D graph also has some unique attributes such as different data point visualizations (bars, lines, points, etc.). In this project we have explored how these 3D graphs can be implemented as combinations of different *A-Frame* [8] custom components.

5.1 Text

A text component is needed for each 3D graph to have a title, axis legends, tick labels for each axis and a value indicator above all data points. *a-text* [50] is a native *A-Frame* [8] component that can be used for all 3D grid text purposes. The problem with *a-text* is its limited amount of stock font families and that the text can not be outlined. Another approach to create text is to make a custom text component using the *HTML5 Canvas* [48], which can render text decorated by *CSS fonts* [51], making it possible to outline text and to use all *CSS* generic fonts [51]. The *HTML5 Canvas* can be rendered onto a *Three.js* texture[52] and then be attached to an *A-frame* component.

Billboarding is a technique that can be used on 2D objects to make them always face the viewer [53, p.257]. This technique is useful to use on text in VR since stationary text can only be read from certain angles. A downside to billboarding is that it makes a scene more unrealistic since text in the real world does not always face the viewer.

In our implementation of a custom text component, we chose to use the

HTML5 Canvas to render text because of the flexibility we got by using *CSS fonts*. We chose to billboard our text component since it made it easier to read all types of text on the 3D graphs.

5.2 3D Grid

3D grids can be used as helping lines for users to read and compare data points in a 3D graph. A problem with 3D grids is that data points which are placed in the middle graph can be difficult to compare to the grid (to approximate a value) since they are far away from each other. This can lead to data points values being misread.

A 3D grid component in *A-Frame* can be built by combining three 2D grid components. A 2D grid component can be created with the help of the *Three.js* [13] grid helper object [54], which sole purpose it to create grids. The problem with using the grid helper object is that grids can only be made quadratic [54]. Another approach to create grids is to make a 2D grid component out of *Three.js* line meshes and cross them to form grids with any dimensions, and with any amount of lines in any direction.

We chose to implement a 3D grid component with line meshes. This made it possible to create 3D grids that only had two vertical lines (see Figure 5.2), which was useful when 3D graphs did not need 3D grids with vertical lines. It also made it possible to create 3D grids with more lines on one axis (see Figure 5.1), which was useful for graphs with high precision data.

A problem that occurred when we used line meshes was that intersections with ray casters only occurred directly on the lines. We wanted intersections to happen as if the 2D grids were planes. We solved this by adding three transparent planes as a hemicube on top of the 3D grid, to fix the intersection problem.

This was also useful since the planes, when made half transparent, could make it easier for the user to distinguish data points from the environments if they had similar colors.

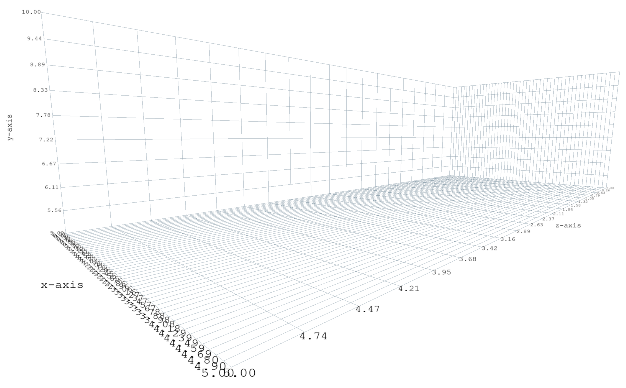


Figure 5.1: A 3D-grid with more lines on the x-axis compared to the other axes.

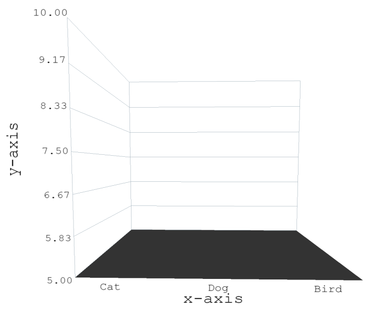


Figure 5.2: A 3D grid with two vertical lines for each 2D grid.

We added line labels to graph axes by using two different methods. The first method was to create an axis of labels for an interval of values, which was useful for numerical values. The second method used was to add a fixed array of values to an axis, which was useful for categorical data. Both methods could be used on any axes, making it possible to create 3D graphs that had differently placed label axes.

5.3 3D Bar Graph

A 3D bar graph displays rows of bars that represents grouped data series [49, p.80-81]. All bar rows can be separated by color to make it easier for the user to identify bars that belong to the same data series [49, p.403]. Transparent bars can be used to give the user a more complete overview of all bars in a 3D bar graph [49, p.403]. The downside to transparent bars is that it can be difficult to distinguish individual bars since the bars' colors and lines are blended together [49, p.403].

We decided to create a 3D bar graph with bar rows with unique colors and the option for all bars to be semi-transparent, for the reasons mentioned above. This was implemented by combining a 3D grid component with a matrix of *Three.js* box meshes. We input the statistical data into the 3D bar graph as a matrix, where each row in the matrix represented one data series. Each row in the matrix could then be mapped to the heights of the bars in each bar row. Each bar was also made hoverable meaning when intersected with a raycaster it would change color and show the hovered bar's value. The result of the 3D bar graph can be seen in Figure 5.3 and Figure 5.4.

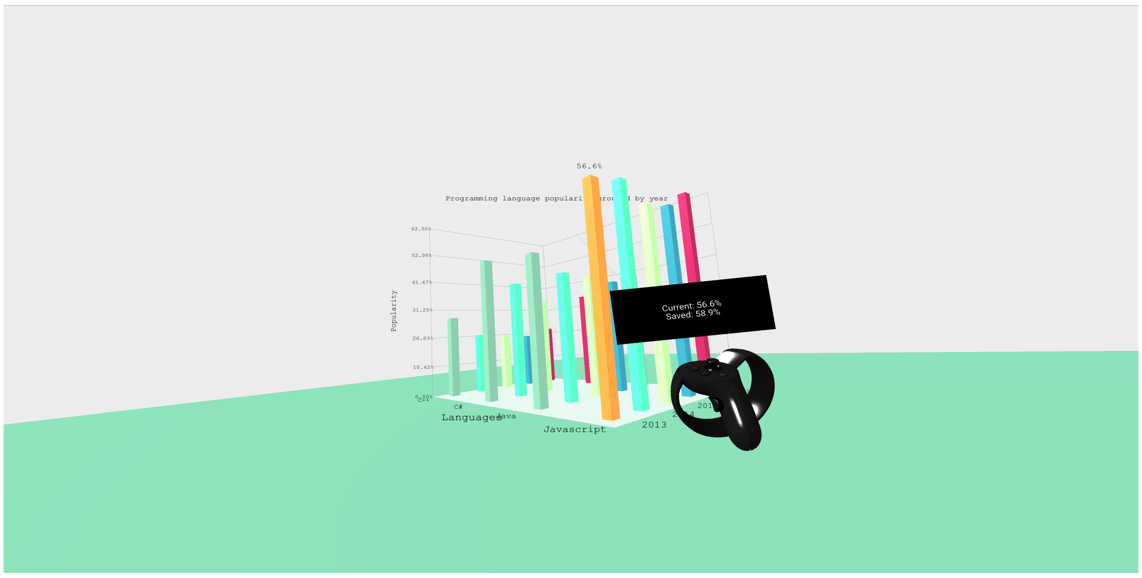


Figure 5.3: A bar graph.

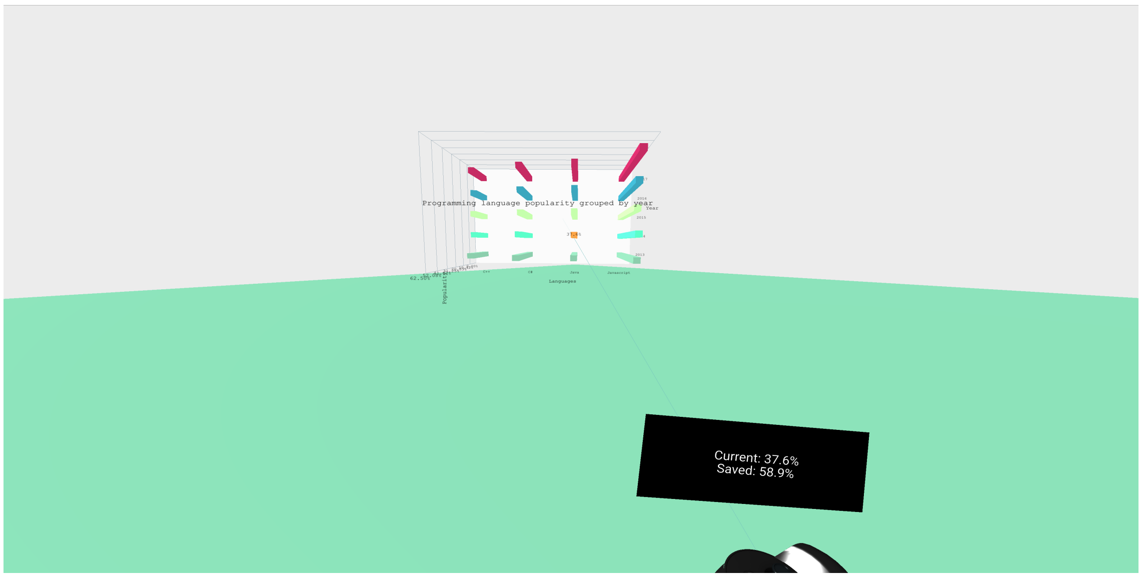


Figure 5.4: A rotated bar graph.

5.4 3D Line Graph

A 3D line graph displays lines that represents multiple data series [49, p.207-208]. Lines can have different colors to help the user differentiate between different data series [49, p.403]. A problem with the 3D line graph is that the same data can be displayed on a 2D graph by simply removing the z-axis and placing all lines on a 2D plane. In comparison a 3D bar graph can not directly be made as a 2D graph since the bars would overlap and some bars would not be visible.

We implemented the line graph by combining a 3D grid component and rows of continuous lines with unique colors using the *Three.js*[13] line meshes. We decided to not have an 3D grid z-axis with line names, instead we attached text labels to each line at their last heights (see Figure 5.5). Having the line name close to the line made it more clear which line belonged to which line label. To make it possible to hover each data point on the graph *Three.js* box meshes were added on each data point. When the boxes were hovered with a raycaster, the value of the point was shown as a label on the box.

We input the statistical data into the 3D line graph as a matrix, where each row in the matrix represented one data series. The heights of each point on each line segment was then mapped to their respective data series in the matrix and placed out on the z-axis of the 3D line graph.

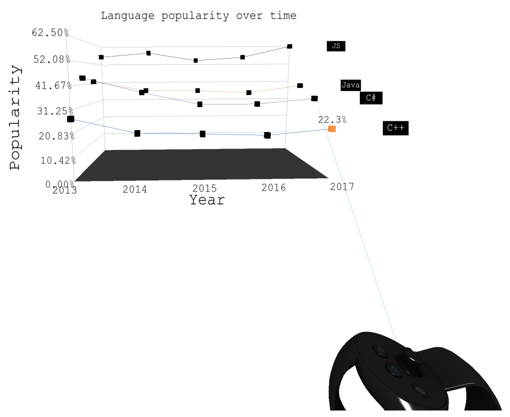


Figure 5.5: Line graph

5.5 3D Scatter Graph

A 3D scatter graph is a variation of a point graph with only quantitative scales or a combination of quantitative and sequence scales [49, p.291]. A 3D scatter graph is used to observe patterns of distributions among a large quantity of data points[49, p.291]. This makes it difficult to read individual data points [49, p.291], if that is something that the user wants to do.

One method of implementing a 3D scatter graph, which Canter [55] used in his 3D scatter graph project found here [55], is to attach *Three.js*'s *Points* [56] as a particle system to an *A-frame* component. Sprites which in *Three.js* always faces the viewer [57], can be attached to each point in the particle system. Another method to create a 3D scatter graph is to use sphere or box meshes from *Three.js* to create all points as individual *A-frame* components, compared to the *Points* system which would be one single *A-frame* component.

We implemented a 3D scatter graph with the possibility to be added to a scene as either a particle system or as a group of meshes. The reason behind the decision was that a particle system performed better with large quantities of data compared to meshes, and meshes worked better with *A-frame*'s raycaster component to handle interactions with the 3D graphs compared to the particle system.

We tried two textures used as sprites in the particle system, one that looked like spheres (see Figure 5.6) and one with a simple color (see Figure 5.7). A scatter graph implemented with sphere meshes, can be seen in Figure 5.8.

The statistical data input our 3D scatter graph component was an array of 3D points. These points were placed out directly either as *Points* particles or as *Three.js* meshes, into the 3D scatter graph.

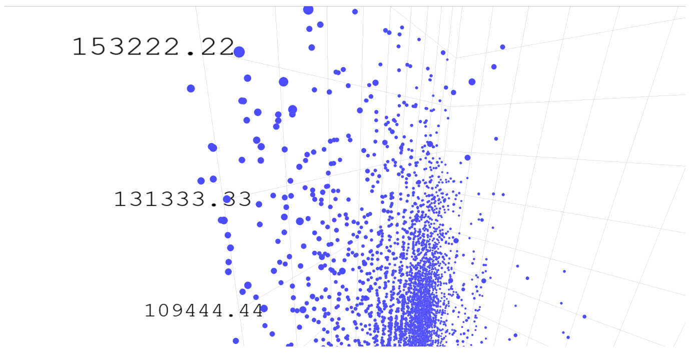


Figure 5.6: 3D scatter graph with flat colored sprites.

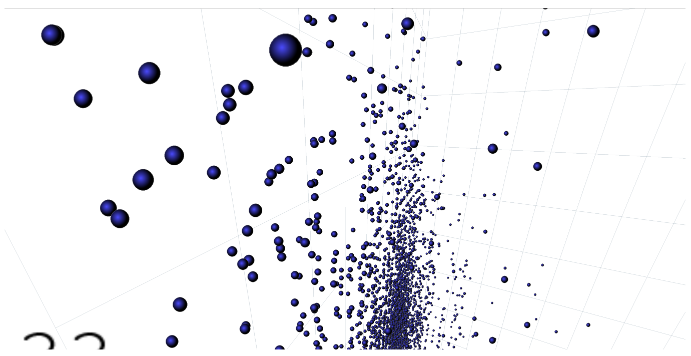


Figure 5.7: Scatter graph with sprites that look like spheres.

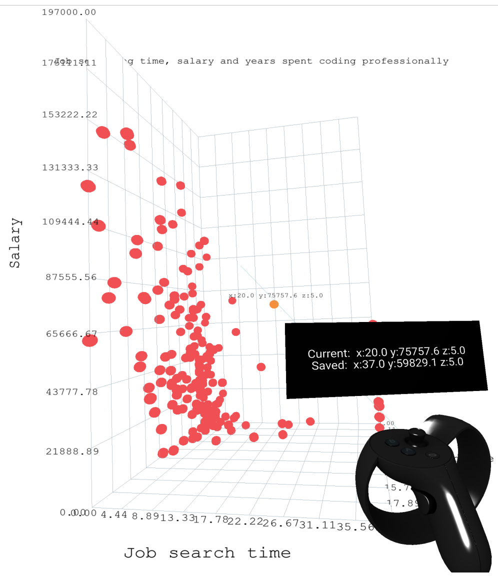


Figure 5.8: Scatter graph implemented with *Three.js* sphere meshes.

5.6 Data table

A table of statistical data can be used to create 3D graphs. This works by using a number of columns in a table to represent different data series, which can be placed and used in 3D graphs as spheres, points or any other data point visualization. Statistical data can be transferred to a data table with text files. Two examples of text file types used to transfer data are *JSON* [19] and *CSV* [58]. *JSON* can store tabular data as *JSON* objects where the keys in each object represent table columns and each object in the *JSON* array represent a table row[55], as seen in Listing 5.1. *CSV* can store tabular data as rows of comma-separated values where the table columns are the first row and all other rows are table rows. A problem with parsing *CSV* in *JavaScript* is that it requires a third party library, compared to JSON files that can be parsed by using a native *JavaScript* method.

```
[
  {
    "city": "Stockholm",
    "date": "2017-01-01",
    "temperature": 10.8
  },
  {
    "city": "Gothenburg",
    "date": "2017-01-02",
    "temperature": 12.0
  },
  {...}
]
```

Listing 5.1: *JSON* scheme with columns: city, date, temperature. Based on a JSON schema used for a 3D scatter graph by Canter [55].

In our implementation of a table, we used a *JavaScript* object as the table, with keys as columns and the values as arrays of statistical data. We used *JSON* to transfer statistical data into the table object since a third party *CSV* parser would have added an unnecessary dependency to the project.

We used the same method to choose which three columns in the table that would be used for the creation of a 3D graph component as Canter [55] did in his 3D scatter graph component [55]. This meant that the same *JSON* file of statistical data could be used for numerous 3D graphs.

Statistics from a table can not be used directly when creating a 3D graph. First, the tabular data has to be transformed into arrays of data point positions and labels. We solved this by having methods that could be used for both categorical and numerical data to get point positions that would fit each 3D graph with a fixed size. Another feature added was the possibility to offset data points, which was useful when the minimum value of a numerical value was greater than zero. Doing an offset on all data points reduced the amount of empty space in each 3D graph.

6 Environment

The environment plays a big part in a VR application, since the user is encapsulated by the virtual world. The environment consist of both virtual rooms or backgrounds, as well as attributes which contribute to the environment, such as lights and shadows.

Before designing the environment, we had to decide if we wanted a stationary environment or an interactive environment. One example of a stationary environment is *CalcFlow* [59], which is an open source software used to look at vectors and other mathematical graphs in 3D. The user is stationary and the only environment is a graph in front of the camera. The graph can be manipulated, but there is no navigation controls. This seems to work well for that application, but since the idea for *graphVR* was to be able to see more than one diagram in a space, a stationary environment did not work. So we narrowed it down to an interactive environment quite quickly.

There were two main types of interactive environments we had to choose between: an open world or an enclosed space. Before starting the development of *graphVR*, the application *Tilt Brush* [60] was studied. *Tilt Brush* is an open world painting tool designed by Google for painting in VR. The application is still designed for only one main object. When we tried the navigation tool, we easily got lost in the open world. This was a problem we did not want to have in our application.

That left us with an enclosed environment, where all the graphs were on display. We got the idea talked about in Chapter 3 from Circopia Solution's *the Hall* [61]. The initial idea was to make an enclosed museum with both the instructions and the graphs present.

For a large part of the project, the main goal for the environment was a room with three different graphs because of the reasons stated above. However, the end result was actually a combination of an enclosed space and an open world. We decided that, even though the enclosed space was the most user friendly environment, it should be up to the user if they want to have a distraction free environment or not. We also decided to have a scene where the user is in the scatter plot in an open environment, as there was a market for it.

Lights and shadows also play a big part in bringing depth to the scenes. They were both used in all scenes to create a more realistic environment.

6.1 How to Work in *Blender*

In *Blender* [11], a common type of object in a 3D scene is a mesh. Meshes are made up of vertices, edges, and faces and create a polygonal figure [62]. In Figure 6.1 it is possible to see that a vertex is created by two edges meeting, an edge is the connection between two vertices, and the face is the area created when closing three or more edges.

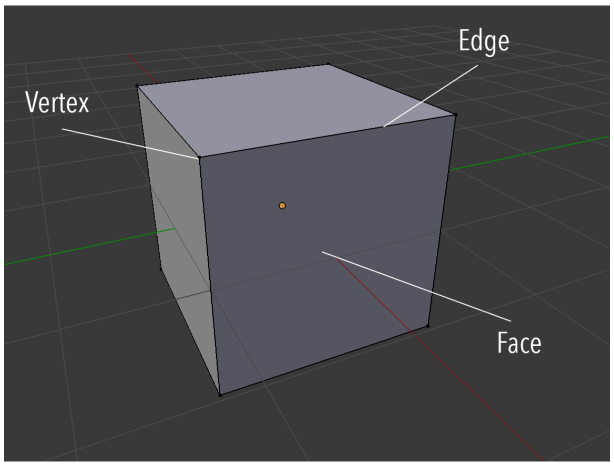


Figure 6.1: A mesh of a cube in *Blender*. It is possible to see the vertices, edges, and faces of the model.

The mesh used in Figure 6.1 is a **primitive**, a built in mesh of a commonly used shape, and it is common to use them to make more complicated models [63]. Because of the many ways it is possible to combine primitives, there are different ways to build a 3D model. In the case for the museum, mentioned in Section 6.3, the room was built out of a plane and then the walls were extruded from that plane. The plane, which at first consists of only one face, was subdivided, creating multiple faces on the plane. The squares closest to the edge was then extruded, making the plane thicker in those particular areas. The final mesh can be seen in Figure 6.2.

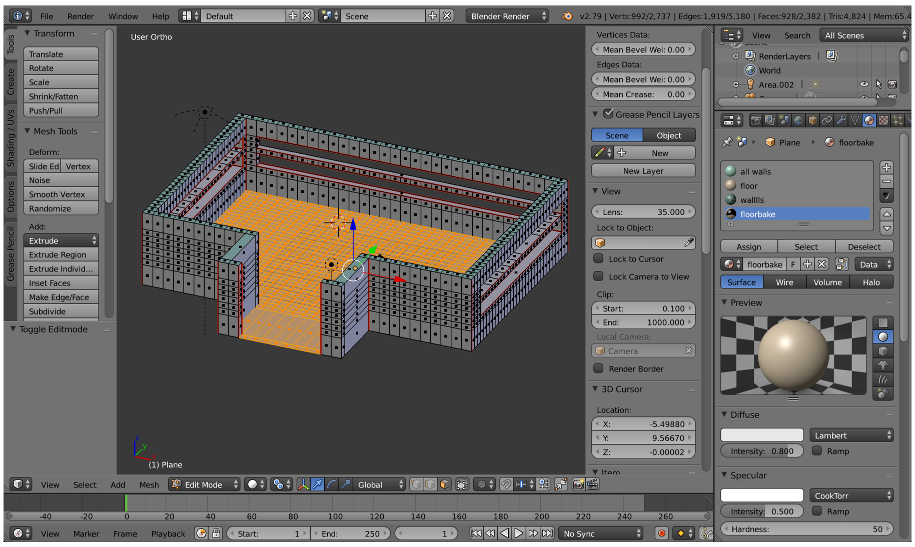


Figure 6.2: The room mesh in *Blender*. It is possible to see all the subdivided squares.

To add a texture to the model, a material color was assigned to the object and then baked into the texture of that object. **Baking** is used as a tool to speed up the rendering process by baking in the lights and shadows into the textures itself instead of having to render them every time [64]. It also helps to reduce the polygon count of an object, thus making it better for performance as A-Frame does not have to render as many polygons on the model. However, we mostly did it because it gave the room a depth that A-Frame's lights and shadow components could not provide. The difference between the room with baked textures and without can be seen in Figures 6.3 and 6.4. It was also used for the other 3D models present in the scene.

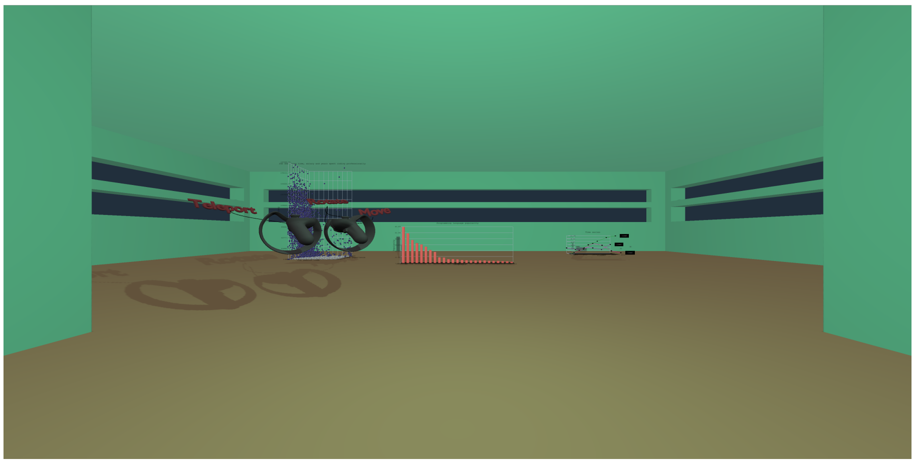


Figure 6.3: Without baked textures, the room does not have the same depth as the room with baked textures.

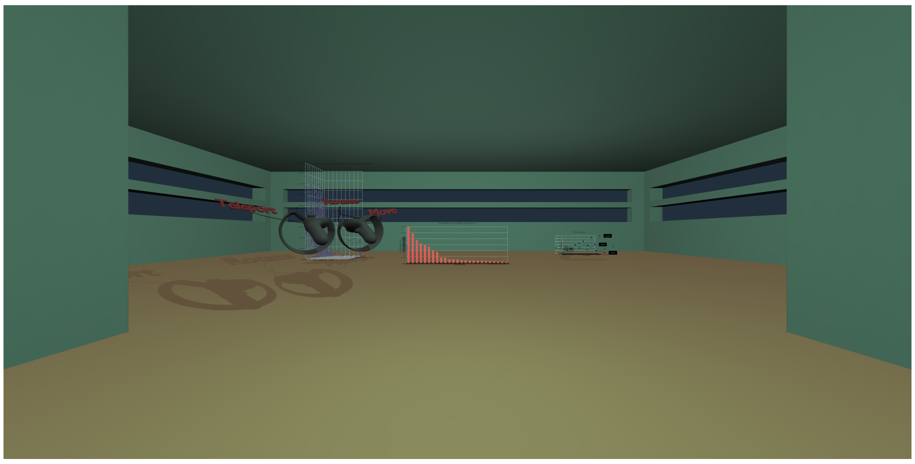


Figure 6.4: The room as seen in our application, with baked textures.

When a texture is baked, it creates an image, which is then applied to the model. However, since the model is in 3D and an image is flat, UV-mapping is used. **UV mapping** is a process to flatten out a 3D object, to then be able to apply a flat image to the flat model [65]. This creates a UV map, a 2D mesh mapping out all the vertices, edges, and faces of the model. An example of a UV map can be seen in Figure 6.5, which is the UV map for the walls of the room in the museum. This allows the image to put a texture on both the front and the back of the model all in one image. The image is then wrapped around the object according to the map.

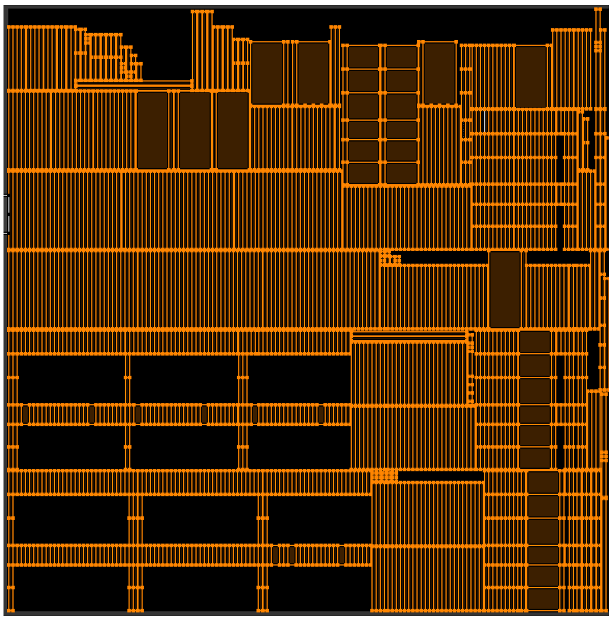


Figure 6.5: The UV map for the room in the museum.

To be able to bake a texture, the model first has to be UV-unwrapped. It is a method to unwrap the UV map from the model itself [66]. This is done by flattening the model at all the seams of the model. Two methods can be used to do this: it can be done automatically by *Blender*, which means that *Blender* assumes where the seams are, or by placing out seams by hand. Both methods were used as *Blender*'s automatic UV unwrapping worked well in some cases, but for the museum's room the UV-map was not correct when unwrapped automatically. Seams were placed on all corner edges as seen in Figure 6.6.

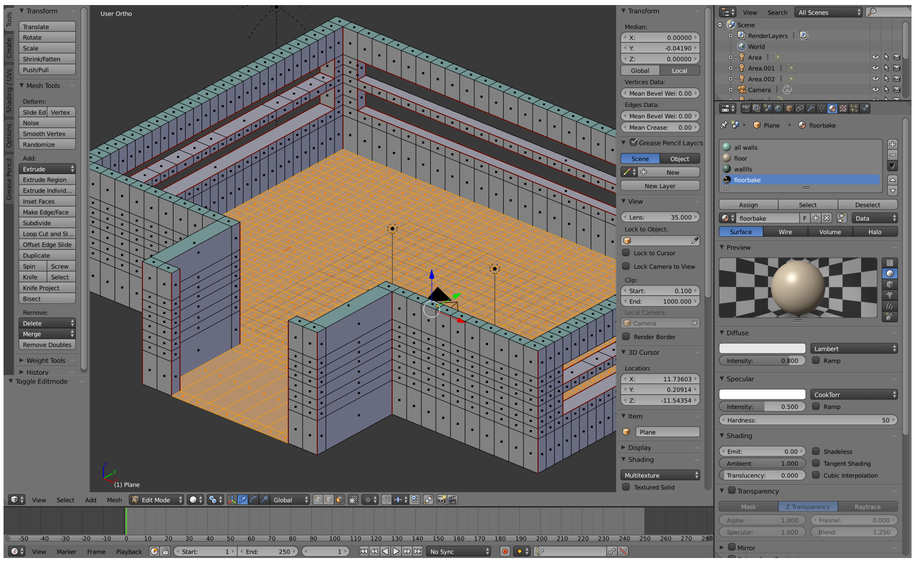


Figure 6.6: The red edges of the model are the seams that were put on the model by hand.

Once the model is unwrapped, then it is possible to bake the texture onto a canvas, creating an image to wrap around the model.

6.2 Tutorial

The tutorial evolved from a simple static board to a fully interactive separate scene. In the first iteration of our application, the tutorial was just a simple help board showing what each button did, as see in Figure 6.7. The idea was that the interactions would be intuitive and easy enough to only need a simple picture for the user to understand the controllers.

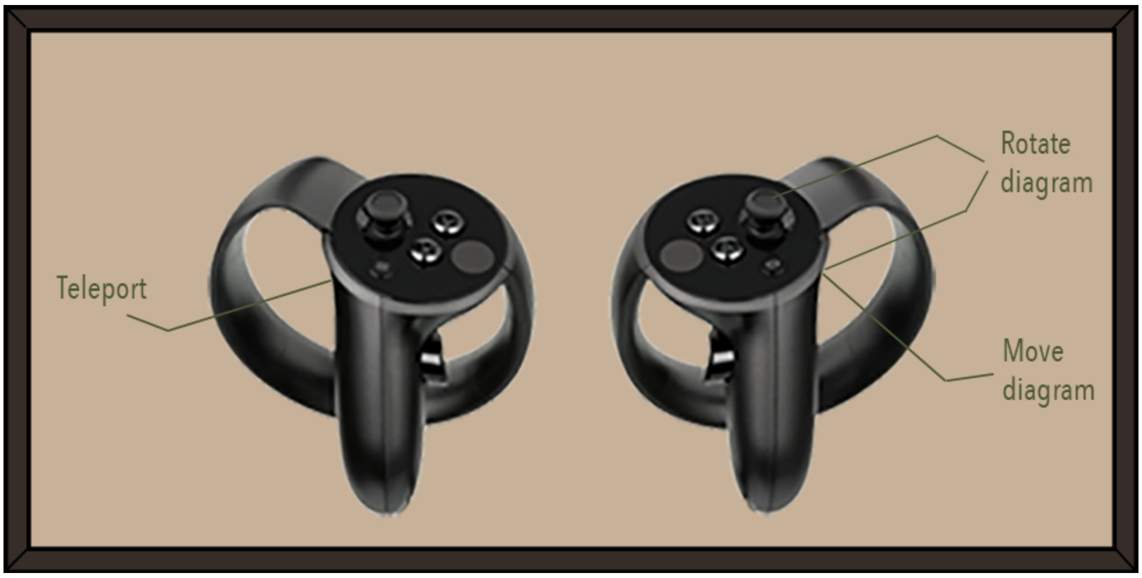


Figure 6.7: The help board designed for the first user test. The figures of the two Rift controllers are taken from a picture from Wikimedia [67] under public domain. Then the picture was modified in *Photoshop* [68], creating the border and the labels. The final image was then imported into *A-Frame* [8] with the image tag.

Unfortunately, it was apparent after evaluating the aplication, that the users did not read the whole board and quickly went on with the tasks they were given. More information about the insights from user test 1 can be found in Section 8.1. In the end, the result of having a simple help board was that the users only found the navigation command and did not use any of the other interactions.

The result of the first user test was an evolved tutorial with a separate scene before the main museum scene. Two early sketches of the tutorial can be seen in Figures 6.8 and 6.9. As can be seen in the figures, we considered two types of layouts. The layout seen in Figure 6.8 was an idea to have multiple rooms leading up to the main museum, where each room consisted of teaching the user one interaction. The second sketch, seen in Figure 6.9, featured a hallway leading the user from one diagram to another.

Both ideas were built around the fact that we wanted the tutorial to be interactive for the user, as well as specifically pointing out all of the different functions available in *graphVR*. First the user was forced to teleport, then learn how to move a diagram, and after that how to rotate a diagram. In both sketches, a game state system was to be used, where each introduction of an interaction was a state. Once the action had been performed, the current state would become invisible, and the next state would become visible. For example, in the first stage, once the user teleported to a specific location, the next state, the move state, would become visible. In the end, the user would had learned all of the necessary skills to use *graphVR*.

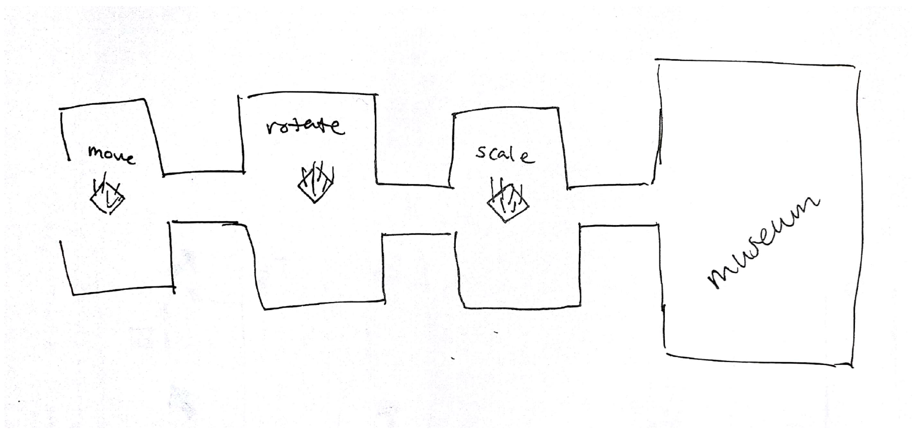


Figure 6.8: An early sketch of the tutorial

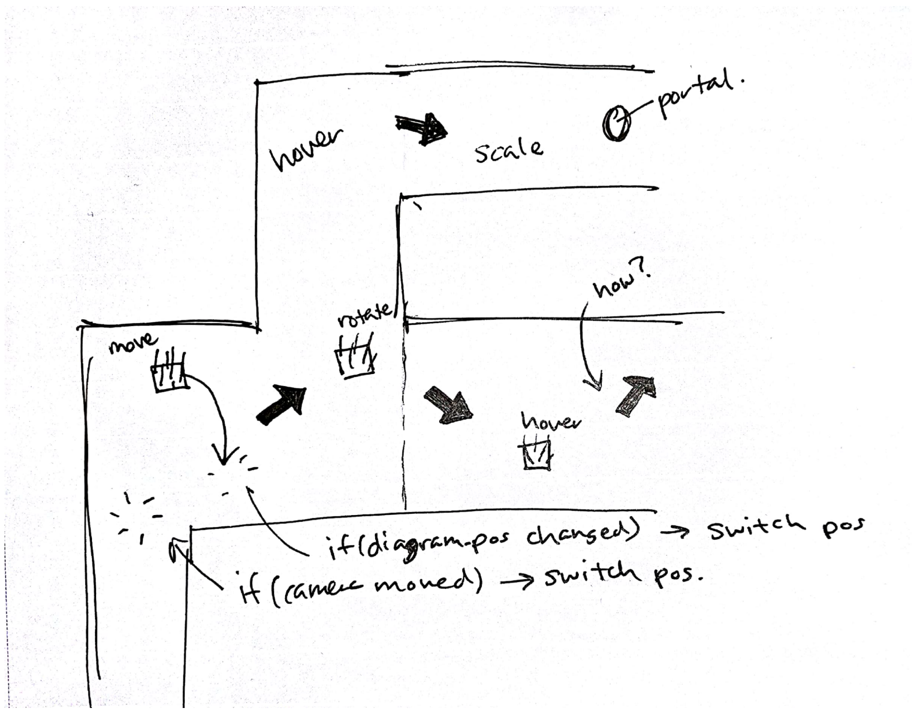
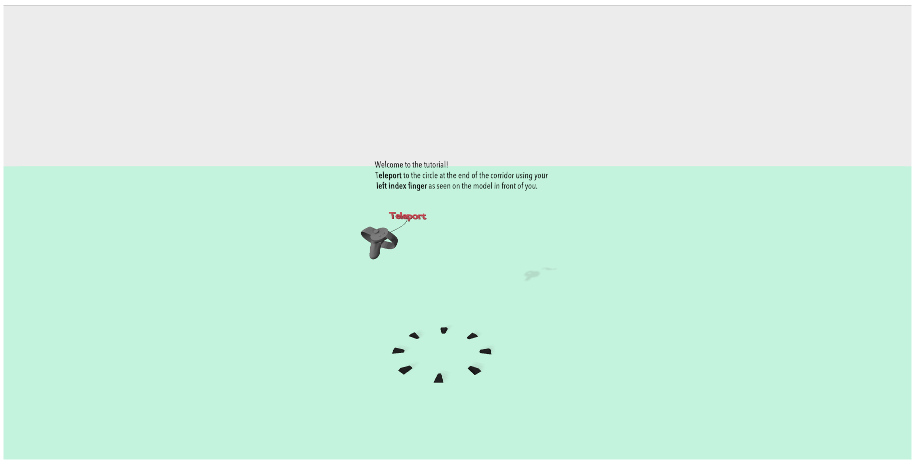

hover
scale
portal.
how?
rotate
move
hover
if(diagram-pos changed) → switch pos
if (camera moved) → switch pos.

Figure 6.9: An early sketch of the tutorial

There were some important aspects to consider when designing the tutorial. It is suggested to have as little text as possible so avoid overwhelming the user and make sure they remember what the game controls are after they have completed the tutorial [69]. We also had to come up with a way to force the user from one event to another. Since we wanted it to be interactive and similar to a game, it was not an option to just have the user press the **A** button and read the instructions. In the end, we decided that the second sketch would allow us to guide the user more than the first, and thus seemed like the better choice.

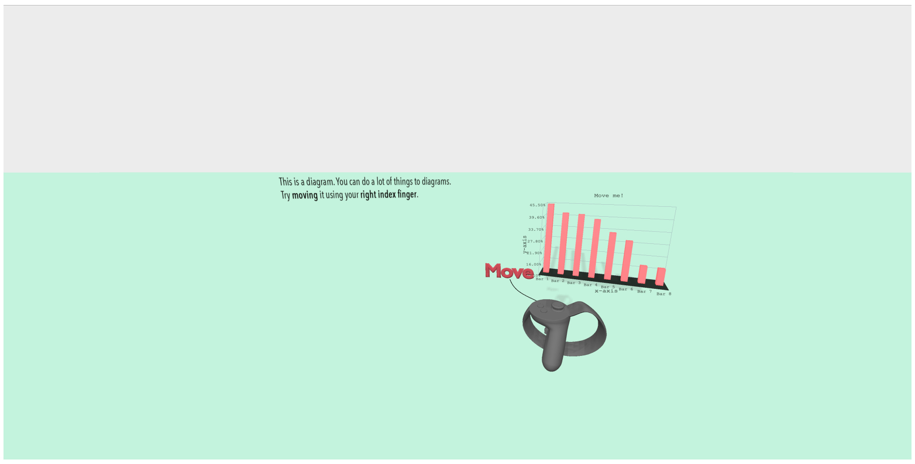
Because the idea for the tutorial was a linear series of event, a storyboard was drawn up to show the consecutive events and make the idea of the tutorial more clear to the developers. The storyboard can be seen in Appendix H.

The game states of the tutorial was implemented through an A-Frame system and each step in the tutorial was a state, which took the user from one step of the tutorial to another. Once an event was completed, another event started. The first scene can be seen in Figure 6.10. Once the player teleported to the circle, a new state began and the changed into Figure 6.11. All states in the tutorial can be seen in Appendix I.



Welcome to the tutorial!
Teleport to the circle at the end of the corridor using your **left index finger** as seen on the model in front of you.

Figure 6.10: The first state in the tutorial, encouraging the user to teleport towards the circle. In it, no other elements of the scene is visible.



This is a diagram. You can do a lot of things to diagrams.
Try **moving** it using your **right index finger**.

Figure 6.11: Once the user has teleported to the circle, the tutorial will enter the second state, the move state. The move diagram, move text, and move controller become visible and the user is encouraged to move the graph.

The final design of the tutorial is a simple environment consisting of a light plane and sky. The floor is at 50% opacity to still be able to see a graph if it has been pulled through the floor. Jason Jerald's book *The VR Book: Human-Centered Design for Virtual Reality*, suggests to have subdued colors as the surroundings and highlight the important objects in the environment with lighter colors [35, p. 238]. A subdued, darker color scheme was tried, but we felt that the lighter scheme suited our application more.

Because the tutorial was not connected directly to the museum or any of the other scenes, there needed to be a way to go from the tutorial to those other scenes. Fortunately, A-Frame has created an entity similar to VR version of the hyperlink, the **link traversal** [70]. It acts as a portal from one scene to the next, and worked well for connecting our different scenes together.

A controller model was made for the tutorial to ensure inexperienced VR users could visually see where the buttons for the action was. An example of this can be seen in Figure 6.12. Because the controllers were 3D models, the user could physically look around it by moving their head in real life. The Rift controller models are built by KingRahl [71] and downloaded from the digital design sharing website *Thingiverse*, under a creative commons licence. They were then modified with labels for the different interactions.

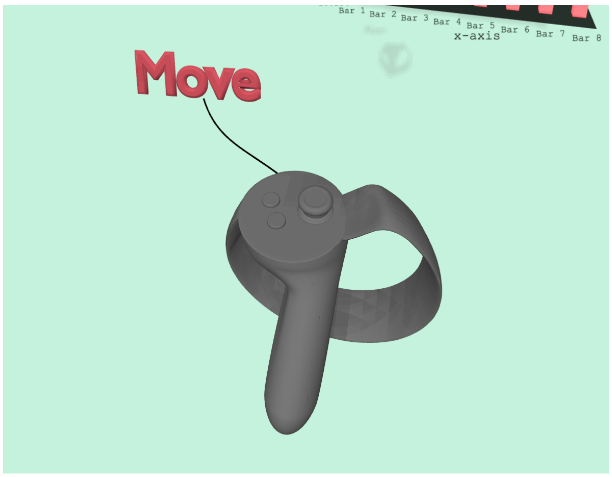


Figure 6.12: An example of the controller models used in *graphVR*. This is a depiction of the controller which tells the user where the interaction move is positioned.

6.3 Museum

As previously mentioned, the idea from the start was to create a room which displayed diagrams, much like statues or artworks in a regular museum. There have been multiple iterations of the actual museum model, going from a simple plane, to a square room, and in the end a large room with windows. An early sketch of the museum layout can be seen in Figure 6.13. The layout is actually quite similar to the final layout.

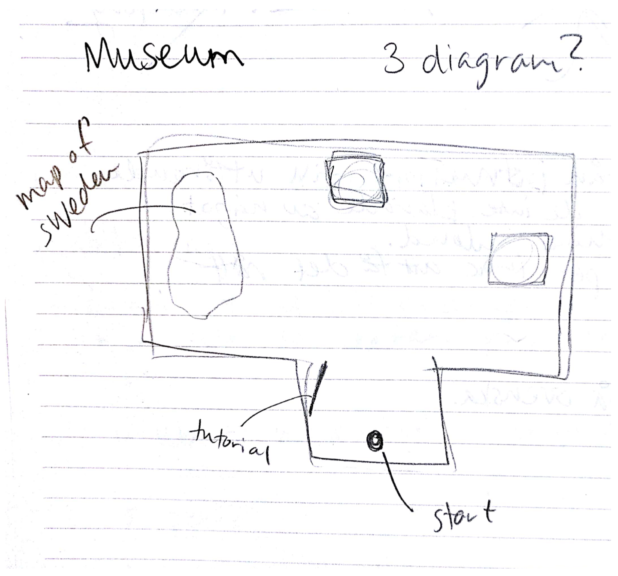


Figure 6.13: An early sketch of the layout of the museum environment in *graphVR*

At first, the goal was to have a photo realistic surrounding, especially when it came to the textures. However, we noticed that this interfered with the reading the graphs. To fix this a more minimalistic theme was adopted. The walls and floors were monochrome so there was not as much noise when looking at the graphs. A picture of the final museum can be seen in Figure 6.14.

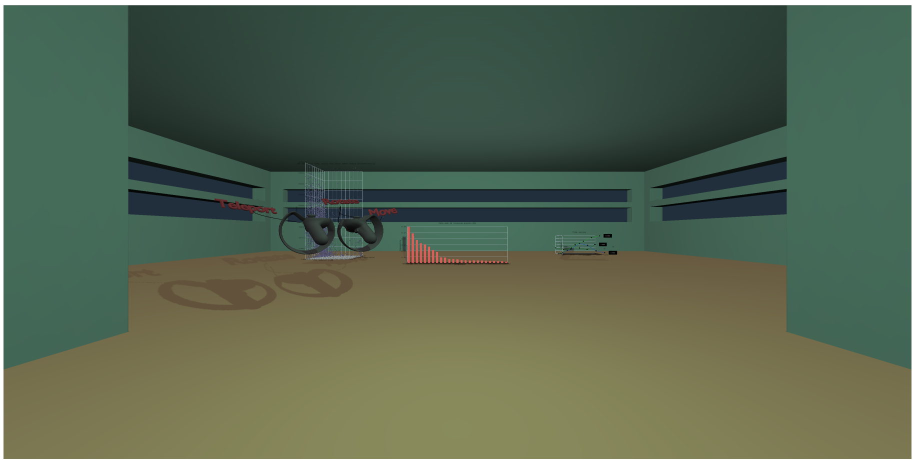


Figure 6.14: The final museum scene in *graphVR*

The current and final version of the museum consists of one room. In the room, there is a scatter graph showing the relationship between developers' salaries, how long they have worked, and how much time they spend searching for a new job each month. There is also a bar graph of the most popular programming languages in 2017. Finally, there is a line graph to the right of the bar chart. To the left of the starting position, a controller model shows the user all of the possible controller interactions.

During the design process, the idea to have windows in the museum was a conscious choice to make the room feel less claustrophobic. We tried making skylights, small rectangular windows and smaller windows, but in the end we chose large windows on the three largest walls. This also contributed to the lighting because we can have a light source, coming from the outside of the room.

6.4 Open World

Towards the end of the project, we decided that the users should have a choice whether they wanted to be in an enclosed space or in an open world. Therefore, an open world was added as an option and the user could choose between the museum and an open world after completing the tutorial.

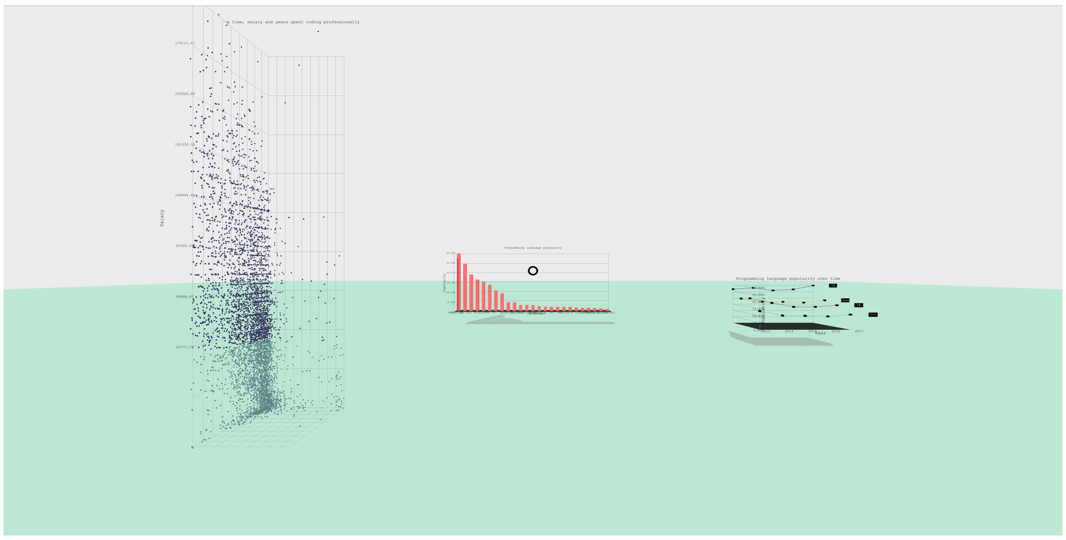


Figure 6.15: The open-world scene in *graphVR*

The open world shown in Figure 6.15 has a similar layout to the museum, with all the diagrams and the helper controller, but the environment is much more similar to the tutorial. The floor and background are the same colors, and the floor is the same plane entity as in the tutorial.

The choice to include an open world environment came from the idea to have a distraction free space to analyze the graphs in.

6.5 Immersive Scatter Graph

The immersive scatter graph environment was a product of the last user test. Many users mentioned that what they liked the most about the program was to be able to stand inside of a graph.

The environment is just a large scale scatter graph and all interactions work the same way as in the other scenes. A picture of the environment can be seen in Figure 6.16

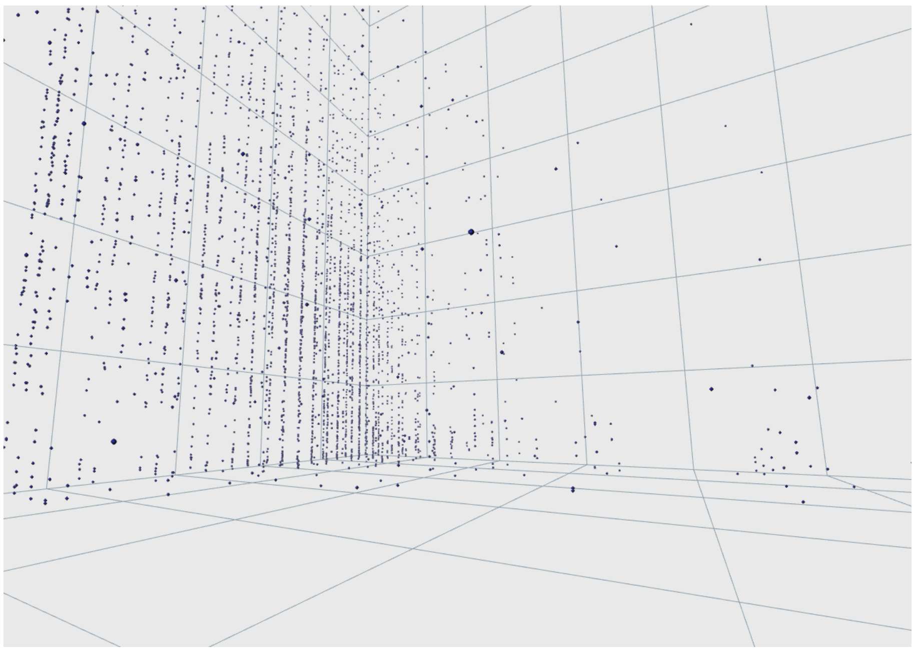


Figure 6.16: The immersive scatter graph environment. The user is surrounded by all the data points.

6.6 Lights and Shadows

In *A-Frame*, there already exists components which tell an entity to act as a light source [72]. Lights create shadows and helps enlighten the environment around the user. There are five different types of lights in *A-Frame*: directional, ambient, spot, point, and hemisphere.

The directional light component is light coming from one direction far away, similar to the sun or the moon. The position of the light cannot be changed, but the light can be pointed towards a specific direction by using the position component. They are the most efficient lights to use if the scene has real time shadows.

Two other directional light sources are the spot component and the point component. However, compared to the directional light component, they both act similarly to actual light bulbs, where the point component can be compared to a single light bulb and the spot component is similar to actual spotlights. The spot component casts light in a single direction and forms a cone shape as seen in Figure 6.17. The angle of the cone can be adjusted, which means that it is possible to decide how wide the light should shine. The point light seen in Figure 6.23, is an omni-directional light source, which means that it sheds light in all directions. Both of these are good to use in places where actual light bulbs would be present, but they can be used in other instances as well.

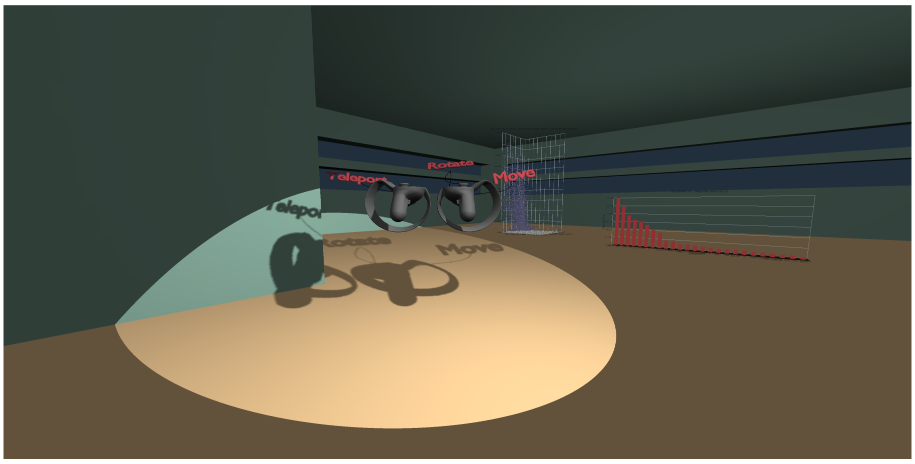


Figure 6.17: A spot light component directed towards the controllers in the museum. The component creates a cone of light and casts a sharp shadow.

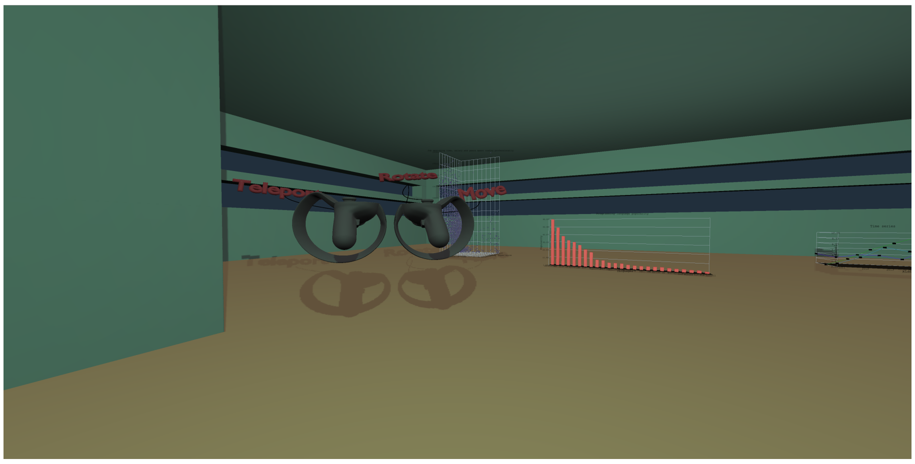


Figure 6.18: A point light component shining in all directions and creating a softer light.

The ambient and hemisphere components are similar to each other and cast light globally in the scene. These lights do not contribute to casting shadows, but contribute to a more natural feel in the environment.

Ambient light interacts globally with the components in the scene and is a process where a light tries to simulate indirect lighting. Ambient light brings about realism in the scene and makes sure the shadows do not appear black, but act as naturally as shadows do in the real world. It is recommended by *A-Frame* to have an ambient light present in every scene [72].

As previously stated, the hemisphere light is similar to an ambient light, but can be used for scenes where the sky and the ground are distinctly different colors. For example, if the ground is green grass but the sky is blue. It is possible to decide both the ground color and the sky color, which is why it is good for a scene where the ground and the sky needs two different types of light.

Because the three scenes in *graphVR* are built around different environments, we needed to use different types of lights in each scene. All scenes needed a directional light and an ambient light to create a natural look in the environment. The directional light component was used to simulate the light coming from far away and the ambient light was used to create indirect lighting in the scenes. However, in the museum another light source was chosen: the point light component. To create shadows, a light needs to have a direction. Because the scene is inside a room, the light outside was not enough to light up the inside of the museum. Either the point light or the spot light had to be used to create a brighter scene as well as create shadows in the museum. Because the different elements in the museum was to be moved by the user, the spot light seemed too focused on one point to work for the museum. Therefore a point light was chosen to light up the inside of the museum.

Having a directional light in the scene, either a directional light component or a point light component, was important to create depth in the bar graph. In Figures 6.19 and 6.20, it is possible to see the difference shadows make. To be able to separate the different bars in the bar graph, shadows were needed.

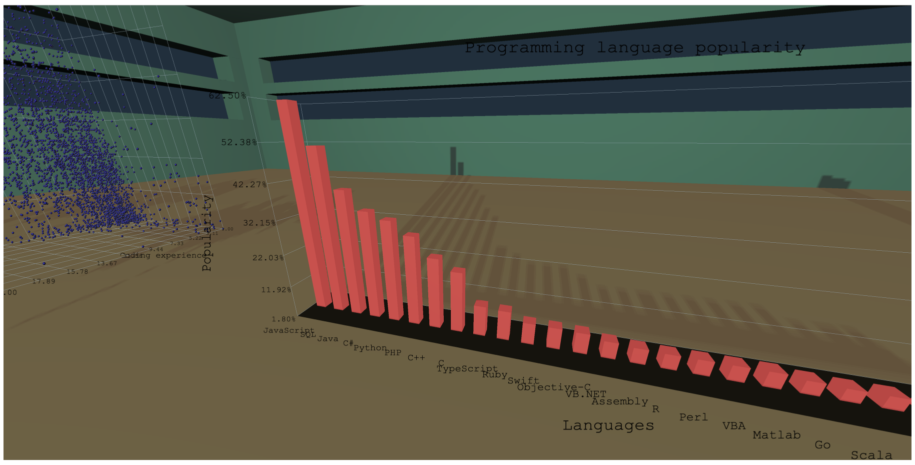


Figure 6.19: A scene with a point light and an ambient light. Each side of the bars in the bar graph can be distinguished.

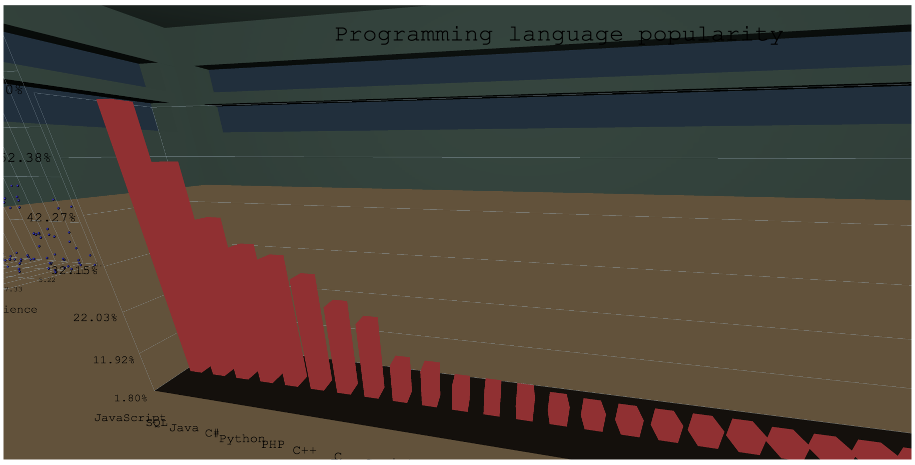


Figure 6.20: A scene with only ambient light. All sides of the bars on the bar graph are the same colors. To the left of the graph it is hard to distinguish which bar is which.

A shadow component is available in *A-Frame* and is simple to use. The component allows entities to cast shadows onto other objects and also receive shadows that are cast onto them.

In A-Frame, shadow maps can be set to only be calculated on the first frame or to update every frame. Updating in every frame costs more from a performance perspective, but is needed if the shadows are supposed to behave as expected when moving objects. However, in *Blender* it is possible to bake a texture and add the shadows onto the model itself, see Section 6.1. We used a combination of baked shadows and real time shadows. We wanted to have real time shadows because the diagrams would be moved by the user and it would look strange if its shadow stayed in place.

7 VR Controller Interactions

VR controller refers to the system implemented in the application, which adds the ability to interact with the environment by using the physical controllers, that the user hold in the hand while using the application. The keystone of these controllers are a method called raycasting. Raycasting is done by sending out an invisible or visible ray from a point and detect if any objects that can be collided with are intersected by the ray [73]. This is done by taking the origin of the ray and setting a direction for it. The program will then check for all objects in the ray's path and add them to an array [74]. The first item that is intersected is put in the beginning of the array, which also probably is the item that the user want to analyze further. Therefore all other data points are being ignored.

In the case of *graphVR* the raycaster is a laser pointer on the right hand, which helps the user know what he or she is pointing at, as there is a line to trace until it collides with an object. The values of this object can be used by *graphVR* and thereby be interacted with by the user. As it is most likely the first object that is being intersected that the user want to interact with, all other objects are being ignored. Without this technique, it would be difficult for the program to know what object the user wants to interact with.

In the beginning of the project *super-hands* by Will Murphy was used as it was a multi-purpose tool that implemented many additional features to *A-Frame* specifically for interactions, like allowing the user to grab an object or notifying when an object is being hovered [75]. Unfortunately, using *super-hands* introduced compatibility issues with other components used. Therefore we had to build a similar tool to *super-hands*.

The tool that we built to replace *super-hands* uses the laser-controls component [76] from *A-Frame* to determine what objects the user is pointing at and also showing the correct controllers matching the ones that the user is holding. The laser-controller component created is called *graph-controller*. ***Graph-Controller*** also, in addition to what the laser-controller implements, also implements interactions. The interactions **scale** and **rotate** are done from the *three.js* object when called. To **move** an object the right trigger button is pressed, and the entity becomes a child of the *graph-controllers* and follows the movement while the trigger button is pressed. When the trigger button is released the entity goes back to being a child of the scene and stops following the controllers. Due to the data being presented in a 3D world, it is possible that the user can not see data points that are hidden behind other points, therefore we find that the interactions are necessary for the user to get a fair view of the graphs. By scaling the graph the perspective isn't really changed but it makes it possible to focus on one portion of graph if needed.

As for analysis of the data, it is important that the user is able to get exact and correct data from each data point that is being presented in the graph. Therefore as a help tool for the user, a small display is mounted on one of the controls where the user can see all the values of the data point of which he or she has selected and then compare it to other values. This is also a kind of highlighting that is good for selecting points from large clusters of data and for hiding unnecessary details that is not in the scope of interest for the user [3].

Already in the pre-studies we realized that there are many ways to handle navigation within VR. Therefore we had to do some evaluation of each of the possible methods for VR-stat. We found that the possible options would be:

- Teleport, point where to go and press the trigger button to get there
- Move with the joystick, like in common first person shooter games
- One stationary spot, no navigation for moving further distances than the motions you do physically

As we tested other VR applications, the move with joystick option was quickly excluded. This made the users who tested this method motion sick very quickly. This was probably, as Wikström states in his study about walking- and rotation speed for effective navigation in VR, due to the sensory conflict theory [77]. Basically this theory tells us that when the sense of balance, sight and other nervous systems which determine the movement and direction of the body are stimulated by conflicting information. This would happen all the time if you were to navigate with the joystick since every time the user would want to move he or she would see that she is moving but the sense of balance and feel would conflict. Therefore this method was scrapped for this project.

The choice between the other two options was not as easy. We found that both worked quite well and did not cause any motion sickness. We think that both of these could have been used for this project successfully hat not been for the fact that we ended but using teleport as we wanted to showcase multiple graphs at once, as that would allow the user to compare the different graphs. To do this we found that if we placed all the graphs in the room and allowed the user to move around as s/he wished and also moved the graphs as they pleased, no functionality would be harmed by the navigation choice. Unfortunately we found some cases of users struggled with using the teleport navigation accurately.

To implement the teleport functionality, the teleport controller was used [78]. This allowed the user to use the left controller to navigate around the environment by pointing where to go and pressing the left trigger button. The development team did testing and evaluation of different ways to navigate around the environment and settled on teleport being the most convenient. Teleport was also appreciated by the test subjects as a tool for navigating around the environment.

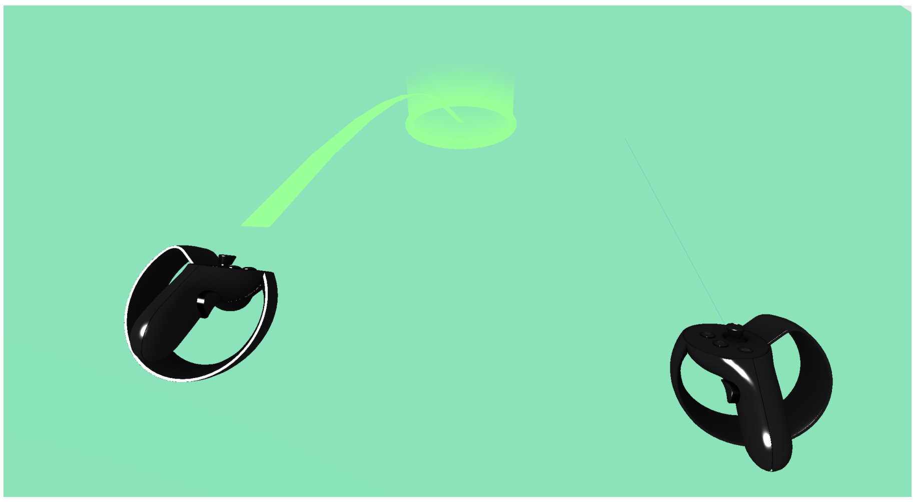


Figure 7.1: Screen capture of user navigating using teleport.

8
Evaluation of the application

Two user tests were performed to aid in the development of *graphVR*. They provided valuable insights into the current design state of the application as well as providied information about possible future design changes.

The results from the user tests were qualitative, with both observations from the facilitators as well as answers to the interview questions. The recorded footage of the tests was also evaluated after the tests were completed. All observations, interview questions, and answers can be found in appendices F and G for user test 1 and 2 respectively.

The findings helped form the final product of *graphVR* and resulted in a further understanding of the users.

8.1 Test 1

This section is a summary of the data that was gathered in the first user test. We found that the choice of using teleport as a navigation tool was appreciated by the test subjects and easy for the users to understand as noted in interview question 1 found in Appendix F. By inspecting the test subjects, it was found that even though there was a help board displaying all the controls, the test subjects did not read this and therefore struggled getting started. A picture of this board can be found in Figure 6.7. This was a problem we had not foreseen and unfortunately only one user tested the other features, apart from teleport and to read data from a hovered data point. This user was very familiar with VR, since he owned a headset of his own, which probably contributed to why he figure the controls out faster than the other users. As a result of this we decided to construct a tutorial to teach the user about all the features he or she could use.

One of the test subjects did not even read the data by hovering the data point, instead he looked at the axis marks which gave him a close estimation of the correct value. Apart from this, all test subjects managed to complete the tasks we assigned them. When one of the test subjects questioned our data in the graph, we realized that the graph was reading the data the wrong way, this had to be solved.

A positive remark was that the test subjects liked that they were bound by a room so that they knew that they did not miss anything. The room also was not too small so that the test subjects would feel claustrophobic. The room felt somewhat too bright and therefore made text difficult to read, to reduce this problem we decided to mute the colors of the walls and the ceiling.

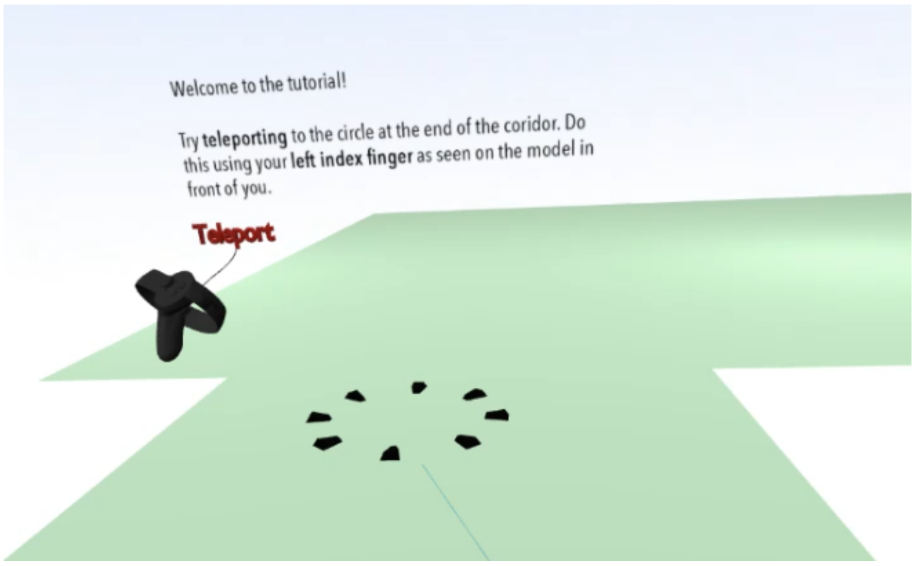
All the test subjects expressed that they were very impressed by the application and were entertained while using *graphVR*, this is a good sign that the application did not feel difficult to use and had a good flow for the user. It is also one of the design requirements found in Chapter 3 that should be met, this was primarily expressed in question 11 “Was the program easy to use?” in Appendix F . During the evaluation of this test to improve for the second user test, we found that introduction questions for the test subjects would have been useful to make the test subject comfortable with the test and also to get some information that could be useful to us before we began the test, such as if the test subject felt confident in using a computer. Having these types of questions are also suggested by Doody in her presentation at Harvard [79] and something we learned when designing the next test.

8.2 Test 2

The second user test generated many useful insights into what changes needed to be made to *graphVR*. Based on the post-test interview, the application was generally easy to use and it was, for the most part, not frustrating to use. The controls were easy to use and to understand. To see all questions and answers, see Appendix G. However, there were some major design flaws that surfaced during the test.

One of the main goals for the user test was to see if the newly implemented tutorial would help the user learn the controls in a better way. However, only 3 out of 5 users manged to complete the first task of going through the tutorial and continue to the museum without any help from the facilitator.

The biggest problem with the tutorial was that the circle used to indicate where to teleport, as seen in Figure 8.1, was also used in the next event to indicate where to move the diagram, as seen in Figure 8.2. This caused a lot of confusion, and user 1, 2, and 4 even teleported to the circle in the second figure. User 4 was even so confused he had to restart the test twice before understanding that the diagram was supposed to be moved into the circle. In the final product, the circle was only used for indicating where to teleport. This is described in Section 6.2.


Welcome to the tutorial!

Try **teleporting** to the circle at the end of the coridor. Do this using your **left index finger** as seen on the model in front of you.

Figure 8.1: Figure describing the different Degrees Of Freedom.

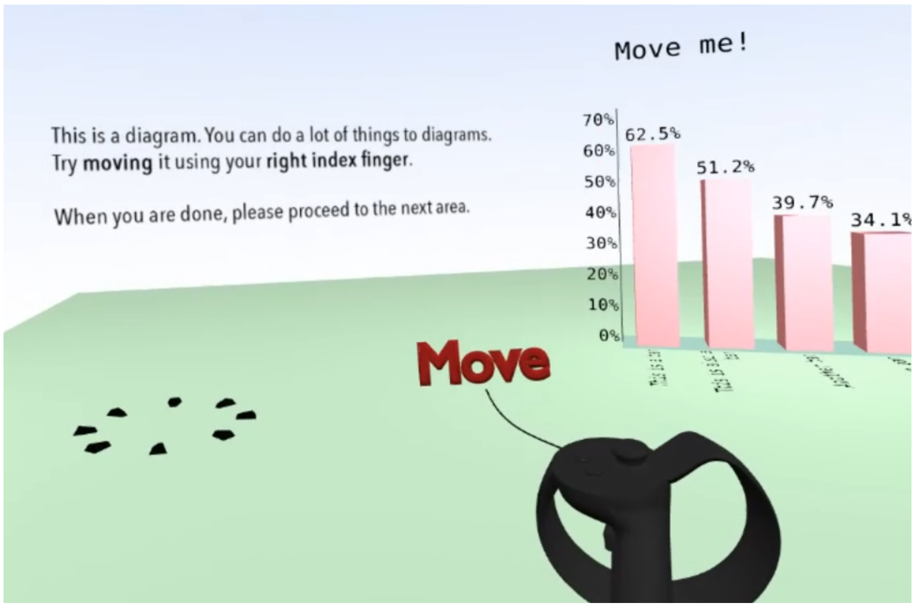

Move me!
This is a diagram. You can do a lot of things to diagrams.
Try **moving** it using your **right index finger**.
When you are done, please proceed to the next area.
70%
60%
50%
40%
30%
20%
10%
0%
62.5%
51.2%
39.7%
34.1%
Move

Figure 8.2: Figure describing the different Degrees Of Freedom.

Another problem people had was that the text disappeared as soon as the diagram was moved, however the next scene did not appear until the diagram was moved into the circle pictured in Figure 8.2. This also caused a lot of confusion and two users asked out loud if they had missed something in the text because they could not advance in the tutorial. In the final product, the text did not disappear and the next event was made visible as soon as the diagram is moved.

To see how previous VR experience and video game experience affected the user's performance, the completion time of the tutorial compared to experience was noted and can be seen in Table 8.1. As can be seen in the table, there does not seem to be any correlation between VR experience, video game experience, and the time it took to complete the tutorial. User 3, the user with the least amount of experience using VR, and users 1 and 2, the users with the most experience, have similar completion times. The same thing can be said about video game experience. Unfortunately, we did not have any user without VR or video game experience. Because of the lack of controlled variables, as well as the low amount of people used in the test, no real conclusion can be drawn. However, it does seem as if there is no correlation between the completion time of the tutorial and previous experience.

Table 8.1: The relationship between completion-time of the tutorial and previous experience with both VR and video games.

User	VR experience	Video game experience	Completion-time
1	Has demoed VR 3-4 times, developed for VR	No	2 min 20 sec
2	Has demoed VR 3-4 times, developed for VR	No	1 min 24 sec
3	Has demoed VR for 15 minutes once	Yes	1 min 57 sec
4	Has demoed VR 2-3 times	Yes	3 min 30 sec
5	has demoed VR 3-4 times	Yes	55 sec

Despite the problems the users faced in the tutorial, we saw a great improvement of the users learning the controls. Once they had made it to the museum, the users did not seem to have any problems with the controls, and the other four tasks were completed by all users without any problems.

During the second task, when the users were asked to explore the museum, two users actually stood in the scatter graph and explored it. Both user 1 and 4 said that what they liked most about the application was to stand inside the diagram. We took this information and created an environment consisting of only a scatter graph, as talked about in Section 6.5.

As seen in user test 1, the environment of the museum was still appreciated. It was not claustrophobic and user 1 even said "The last room [the museum] was better than the first room [the tutorial]. It was good because you could see the walls." The enclosed space seems to be the best option for our application.

A few other features that were brought up during the user test were the ability to scale, highlight data points that you are pointing to, and being able to pull the diagrams closer to you. Three users wanted to use a scaling function during the test. Two users also mentioned being able to pull a diagram closer because "what was tricky was to get the right distance to the thing you are looking at. Because you cannot do that with the selection, you have to do that with the teleport. For example, reading the text was too small, you had to move closer because you cannot pull the diagram closer." Both scale and highlight have been implemented in the final product, and the other interaction would have been implemented had there been enough time.

9
Result

graphVR is a VR web application that displays statistical data in three different graphs: scatter graph, bar graph, and line graph. The graphs can be viewed in three different environments: a museum, an open world, or a world where you are standing in a scatter graph. A tutorial is also available for users not familiar with the controls of the application.

There are several interactions for the user to utilize. The users can interact with the graphs through scale, move, rotate and read specific data points. It is also possible to move around in the VR environment using a teleport function.

To read more about the functionalities of *graphVR* see Chapters 5, 6 and 7.

graphVR satisfies all our design requirements, but there are several improvements that could be made to make the product look better visually, discussed in Chapter 10.

10 Discussion

As we can see in the result, we have achieved what we set out to do: to create a VR program for the web, which displays data in graphs. We found that our product proves that it is possible to visualize statistics and interact with the visualized data, which is presented in three different 3D-graphs. To be able to do this we needed the right tools and frameworks to build our application and A-Frame played a big role in enabling that.

10.1 Discussion of Results

A-Frame [8] was convenient when developing the application. It took very little effort to get a first demo up and running since the framework makes it quick and easy to create environments and entities, while being compatible with all VR hardware.

When the application got more complicated however, we noticed that when using *Three.js* [13], it was much easier to build the individual custom components instead of using *A-Frame*'s primitive components.

We found the three graphs available in *graphVR* to have different strengths and weaknesses. Both the bar graph and the scatter graph had added value in the VR environment, however the line graph was for the most part less clear to analyze in the VR environment.

The bar graph made it easier to see trends in categorical data when different categories was shown behind each other. In other words, when three dimensions of data was compiled into the bar graph, it was easier to see a general trend. However, this came with a drawback. Since some data could be hidden behind other data, it was sometimes difficult to see all bars at the same time. To counteract this, a semi-transparent component was created. The graph is most suitable for sets of two categorical and one numerical data.

The scatter graph also added some additional value when displayed in 3D, possibly more so than the bar graph. It did not have as big of a problem with hidden data points, since the point of the graph was to see patterns in the tabular data. It added a dimension of data and enabled the user to see the depth of the data points and clearly see clusters of data. In future projects, it could be possible to add another dimension by changing the size of the spheres or adding another data set and comparing the two sets. It is this graph that we believe works best in VR out of the graphs we have developed. This graph is most suitable for data sets of three numerical data.

However, we found that the 3D line graph added no extra value in the VR environment compared to a 2D platform. It contains no extra dimensions compared to a 2D line graph, where the lines are different colors. The 3D graph also has a disadvantage since it is easier to compare the lines if they are flat on a 2D surface than if they are distorted by perspective in 3D space. In its current state, the line graph is not suitable for a 3D environment. This could be researched further, and perhaps there is a way to display another dimension of the graph or possibly a different way of displaying two data dimensions, but currently the 2D version is better.

Although we find our product satisfies our goal, there are several things that we think could be improved with this product. Because of time limitations, there were multiple functions and ideas that were not implemented, but could be interesting to explore in the future.

During the last user test a few functions were suggested by the users. One function we did not have time to implement was a function to move the diagrams closer or farther away, similar to a lasso. This is something we as designers have discussed as well, even before the users mentioned it. This function would help make *graphVR* easier to use. The teleport function works well for large jumps and general navigation, however, it is quite difficult to fine tune your position and the position of the graph. The lasso would enable you to stand anywhere and move a graph anywhere with greater ease.

Currently, the museum scene is minimalistic in its appearance, the graphs as the only objects in the room. Since the original idea was to create a museum of diagrams, it would be a nice feature to have more than just three diagrams as well as possibly some lamps and other objects that could be found in a real museum. Another idea would be to put the diagrams on podiums, like statues in a museum. The user could grab the diagram they want and bring it into the center of the room for further inspection.

All graphs are currently implemented using only standard *Three.js* geometries and materials. No custom shaders are used, which could have improved the performance and the visual results of the graphs. A future work of improvement *graphVR* could be to implemented custom shaders to improve the overall quality of all graphs.

The way we have built the application, we are mostly focusing on how people can view the diagrams. It is possible to move two diagrams next to one another and compare them and it is also possible to pinpoint specific data points in the diagrams with the read data point or hover function. However, it is not possible to do extensive statistical analysis in *graphVR*. There would need to be more tools available, like a pen tool which can be seen in *CalcFlow* [59], a filter to filter the data shown, a way to zoom to a detailed level and crop the data shown.

Something we noticed when developing the controls for the application, was that there is a limit of functionality because of the lack of buttons. If there is to be more functionality added to the app, like a lasso tool, scaling, and a pen function, there would need to be a menu for all the functionality. For example, it would be possible to implement a menu on the wrists much like the the menu in Tilt Brush [60]. It is very intuitive and would probably work well in a more complex application.

Because the tutorial was a late addition to the application, we did not have as much time as we wanted to create the environment for it. Therefore the environment is a simple plane entity for the user to walk on. However, if time would have allowed it, the environment of the tutorial would have been further developed. The initial idea was to have a corridor which would take the user from one state to another. The corridor would provide visual queues to guide the user forward.

Another queue that would be a great addition to the tutorial would be the use of audio. Because of the virtual environment, it is possible to position audio where event happen. For example, once the user moves the diagram in the second part of the tutorial, a sound would come from the rotation diagram as it becomes visible. This would draw the attention of the user and act as another queue to guide the user forward, since they can hear that the sound came from the left.

If this product is viable as an option for viewing data at this time is hard to determine. On one hand you get some nice functionality, it is more fun and there are no major drawbacks. On the other you lack tools to properly analyze the data and you need the VR hardware to use the application. In future studies it would be interesting to compare these graphs to existing 2D graphs and 3D graphs on an 2D media. At this time we do not see *graphVR* as a product to be used commercially but perhaps as a demo for how data visualization is possible in webVR with *A-frame*.

In general the *graphVR* was well received by our testers, if it was the application or simply the experience of VR that they found enjoyable is hard to tell, but we are ultimately happy with the result of this project.

10.2 Validity of User Tests

It is important to understand what the result from the user tests could be originating from. Therefore analyzing the validity of the user tests is important.

For a start, the user test group of 9 people were quite homogeneous as all test subjects were younger than 30 years old and studied at Chalmers University of Technology. It could then be argued that all of these young people used to being in a technical environment and therefore might have had an easier time coping with new systems than the average person.

We set a goal to try the application on five test subjects in each of the two user tests. Unfortunately due to illness, only four out of the five test subjects showed up on the first test. This could result in less usability problems being found. The second test was done on a full set of five people.

The tests were performed on two separate groups of people to make sure that the second test didn't give false indications that the application was more intuitive and easy to understand due to experience from the previous test. The prior experience of VR applications were varied between all the test subjects.

Unfortunately the users did not test stable versions of the application in the tests. This resulted in that some users experienced performance issues and hard to know how much this affected the users' view of the application and if it made them more hesitant to explore further on their own as these could be very disorientating.

It was sometimes hard to interpret the results that were found in the user tests, especially in some special cases where we as test conductors clearly saw that the user struggled with the task but when asked about it in the post test interview the users did not acknowledge the struggle. There were also results from different test subjects that resulted in contradicting results. We mostly decided on follow our observations rather than what the the test subjects claimed they experienced if our observations and the interview answers contradicted each other. One possible reason for this is that some users might have been overwhelmed by using VR for the first time and enjoyed it even though they were clearly struggling and therefore did not experience that they were struggling. As Christopher Murphy writes, "listening and observing are both important and will provide you with different insights". He tells us that listening gives us as facilitators insight of the subjective preferences of the test subjects, like what features they like and so on, while observing tells us how the user will use the application [80].

11 Conclusion

The purpose of this project was to develop a web application for information visualization in VR. The application, *graphVR*, was developed using *A-Frame*. To help the design process both internal and external evaluation in the form of user tests of the application were done. The resulting application is an application with four different scenes: a tutorial, a museum, an open world, and a scene with an enlarged scatter plot. All scenes used *A-Frame*'s light and shadow components to create a more realistic experience, as well as baked textures on the models.

We learned throughout the process to continuously evaluate the product and try to learn from our mistakes. Performing user tests is a good way to understand the user and see how they respond to the design.

Multiple interactions were implemented in the final application. Move, scale, rotate, and read data point was the functionalities we deemed the most vital. However, it could be useful to implement more analytical interactions in the future.

Tabular statistical data was transferred to a data table component, where data series were gathered, transformed and lastly send to each 3D graph component to be visualized as different data points (bars, points, lines, etc.).

Our conclusion is that an iterative design process works well for developing a VR application for the web. *A-Frame* [8] is a good framework for creating information visualization in VR. The built in components are useful and it is easy to implement your own components as it is built on *JavaScript* and *HTML*. However, it is easier to create custom components in *Three.js* and that is what was done in this project.

Through completing the project, we have come up with a few recommendations for anyone trying to produce graph visualization software using *A-Frame*. We recommend:

- Whether a static or interactive tutorial is used, the controls need to be very obvious to the users. We tried both types of tutorials, and they both had problems. But we would recommend having an interactive tutorial as it is easier to encourage the users to try out all functions.
- To keep in mind that users do not read voluntarily. Keep all texts to a minimum and make sure the main point is easily understood.
- That that the environment consists of muted colors, with dark backgrounds and light objects, as they seemed to work the best in a virtual environment.
- To use *A-Frame*'s lights and shadows. They work well for VR environments.
- To keep the primary object in the view of the user. The user is able to look around the environment, but during our user tests, the users mainly noticed the objects right in front of them.

- Using teleport for navigation instead of using the joystick. Using the joystick disorients the user's mind and can cause motion sickness quicker than the teleport navigation can.
- Using a representation of the controllers in the application as the user's hands. The user is then able to reference the virtual controllers to understand where the buttons are on the real controllers. It is then possible to use the name of the fingers to indicate the trigger (index finger) and grab (middle finger) buttons when teaching the controls.
- To have more than one focus group during the user tests. It is then possible to try out all the different functionality you might think the application could need. This would be especially good if the developers and designers are not used to developing for a VR environment.
- To perform evaluations of the application continuously throughout the development process. This will help both designers and developers to know they are on the right track.
- To not create one custom *A-Frame* component per graph, instead separate graph parts and functionalities into different custom components and reuse them to avoid code duplication.
- To process statistical data on the client side of the application and make sure that all data processing methods are unit tested to give the application credibility.

Bibliography

[1] E. Olshannikova, A. Ometov, Y. Koucheryavy, and T. Olsson, “Visualizing big data with augmented and virtual reality: Challenges and research agenda”, *Journal of Big Data*, no. 2.22, 2015.

[2] A. Unwin, C.-h. Chen, and W. K. Härdle, *Handbook of Data Visualization.* Berlin, Germany: Springer-Verlag, 2008, ISBN: 9783540330363.

[3] S. Few, *Now You See It: Simple Visualization Techniques for Quantitative Analysis.* Oakland, CA: Analytics Press, 2009, ISBN: 9780970601988.

[4] Y. Rogers, H. Sharp, and J. Preece, *Interaction Design - Beyond human-computer interaction.* The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom: John Wiley & Sons Ltd., 2015, ISBN: 9780470665763.

[5] Oculus VR, *Oculus rift*, 2012. [Online]. Available: `https://www.oculus.com/rift/`.

[6] Statista. (2017). Worldwide virtual reality (vr) headset unit sales by brand in 2016 and 2017 (in millions), [Online]. Available: `https://www.statista.com/statistics/752110/global-vr-headset-sales-by-brand/` (visited on 02/02/2018).

[7] C. Donalek, G. Djorgovski, S. Davidoff, A. Cioc, A. Wang, G. Longo, J. S. Norris, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, and A. Drake, “Immersive and collaborative data visualization using virtual reality platforms”, p. 610, Oct. 2014.

[8] A-Frame, *What is a-frame?* [Online]. Available: `https://aframe.io/docs/0.7.0/introduction/#what-is-a-frame` (visited on 02/01/2018).

[9] Oculus. (2018). Rift experiences|oculus, [Online]. Available: `oculus%20store` (visited on 05/13/2018).

[10] Valve corp. (2018). Welcome to steam, [Online]. Available: `https://store.steampowered.com` (visited on 05/13/2018).

[11] Blender Foundation. (1998). Blender, [Online]. Available: `https://www.blender.org/` (visited on 05/11/2018).

[12] Unity, *Unity Technologies.* [Online]. Available: `https://unity3d.com` (visited on 05/14/2018).

[13] *Three.js.* [Online]. Available: `https://threejs.org/`.

[14] MochaJS, *Mocha - the fun, simple, flexible javascript test framework.* [Online]. Available: `https://mochajs.org` (visited on 05/14/2018).

[15] ChaiJS, *Chai*. [Online]. Available: `http://www.chaijs.com` (visited on 05/14/2018).

[16] NodeJS, *Node.js*. [Online]. Available: `https://nodejs.org/en/` (visited on 05/14/2018).

[17] Facebook Inc., *React - a javascript library for building user interfaces*. [Online]. Available: `https://reactjs.org` (visited on 05/14/2018).

[18] A-Frame. (2018). Entity-component-system - a-frame, [Online]. Available: `https://aframe.io/docs/0.8.0/introduction/entity-component-system.html` (visited on 04/18/2018).

[19] (2014). The javascript object notation (json) data interchange format, [Online]. Available: `https://tools.ietf.org/pdf/rfc7159.pdf` (visited on 04/15/2018).

[20] Ecma International, *Standard ecma-262*, 2017. [Online]. Available: `https://www.ecma-international.org/publications/standards` (visited on 05/13/2018).

[21] Open Source, *Concepts*. [Online]. Available: `https://webpack.js.org/concepts` (visited on 05/14/2018).

[22] B. Lang. (Feb. 2013). An introduction to positional tracking and degrees of freedom (dof), [Online]. Available: `https://www.roadtovr.com/introduction-positional-tracking-degrees-freedom-dof/` (visited on 04/10/2018).

[23] L. C. Hale, "Principles and techniques for designing precision machines", PhD thesis, MIT, Massachusetts, Feb. 1999.

[24] J. Martindale. (2018). Oculus rift vs. htc vive, [Online]. Available: `https://www.digitaltrends.com/virtual-reality/oculus-rift-vs-htc-vive/` (visited on 04/04/2018).

[25] ephtracy. (2015). Magicavoxel, [Online]. Available: `https://ephtracy.github.io/` (visited on 05/11/2018).

[26] Autodesk. (1998). Autodesk maya, [Online]. Available: `https://www.autodesk.eu/products/maya/overview` (visited on 05/11/2018).

[27] Maxon. (1990). Maxoncinema4d, [Online]. Available: `https://www.maxon.net/en/products/cinema-4d/overview/` (visited on 05/11/2018).

[28] Pluralsight. (2015). 3ds max, maya lt or blender - which 3d software should i choose for asset creation?, [Online]. Available: `https://www.pluralsight.com/blog/film-games/3ds-max-maya-lt-blender-3d-software-choose-asset-creation` (visited on 04/26/2018).

[29] Thingiverse, *Thingiverse*. [Online]. Available: `https://www.thingiverse.com/` (visited on 05/09/2018).

[30] A-Frame, *3d models*. [Online]. Available: `https://aframe.io/docs/0.8.0/introduction/models.html` (visited on 04/26/2018).

[31] KhronosGroup, *Gltf*. [Online]. Available: `https://www.khronos.org/gltf/` (visited on 04/26/2018).

[32] A-Frame, *Why use gltf?* [Online]. Available: https://aframe.io/docs/0.8.0/components/gltf-model.html#why-use-gltf (visited on 04/26/2018).

[33] KhronosGroup, *Gltf-blender-exporter.* [Online]. Available: https://github.com/KhronosGroup/glTF-Blender-Exporter (visited on 04/26/2018).

[34] GitHub. (2018). Github, [Online]. Available: https://github.com (visited on 02/01/2018).

[35] J. Jerald. (2016). The vr book: Human-centered design for virtual reality, [Online]. Available: https://dl-acm-org.proxy.lib.chalmers.se/citation.cfm?id=2792790 (visited on 05/04/2018).

[36] J. Rasmusson, *Agile vs waterfall.* [Online]. Available: http://www.agilenutshell.com/agile_vs_waterfall (visited on 05/07/2018).

[37] VersionOne. (2018). Version one 12th annual state of agile report, [Online]. Available: https://explore.versionone.com/state-of-agile/versionone-12th-annual-state-of-agile-report (visited on 05/03/2018).

[38] H. Ahmed, H. Saleh, E. C. (.-b. collection), and K. (.-b. collection), *JavaScript Unit Testing*, English, New. Birmingham: Packt Publishing, Limited, 2013, ISBN: 9781782160625;

[39] H. Loranger. (Apr. 2016). Checklist for planning usability studies, [Online]. Available: https://www.nngroup.com/articles/usability-test-checklist/ (visited on 05/10/2018).

[40] J. Nielsen. (2012). How many test users in a usability study?, [Online]. Available: https://www.nngroup.com/articles/how-many-test-users/ (visited on 02/01/2018).

[41] ——, (2012). Thinking aloud: The #1 usability tool, [Online]. Available: https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/ (visited on 02/04/2018).

[42] Open Broadcaster Software. (2012). Obs studio, [Online]. Available: https://obsproject.com/ (visited on 05/10/2018).

[43] D. Nessler. (Mar. 2016). A guide to paper prototyping & testing for web interfaces, [Online]. Available: https://medium.com/digital-experience-design/a-guide-to-paper-prototyping-testing-for-web-interfaces-49e542ba765f (visited on 05/11/2018).

[44] Mozilla, *Javascript | mdn.* [Online]. Available: https://developer.mozilla.org/bm/docs/Web/JavaScript (visited on 05/14/2018).

[45] K. Knoernschild, *Java; desing: objects, UML, and process.* Addison-Wesley, 2002.

[46] A-Frame, *Writing a component-a-frame.* [Online]. Available: hhttps://aframe.io/docs/0.8.0/introduction/writing-a-component.html#example-follow-component (visited on 05/14/2018).

[47] ——, *Javascript, events, dom apis - a-frame.* [Online]. Available: https://aframe.io/docs/0.8.0/introduction/javascript-events-dom-apis.html (visited on 05/11/2018).

[48] W3C, *Html 5.2*. [Online]. Available: `https://www.w3.org/TR/html5/index.html#contents` (visited on 05/14/2018).

[49] R. L. J. Harris, *Information Graphics: A Comprehensive Illustrated Reference*, English. New York: Oxford University Press, Incorporated, 2000, ISBN: 9780195135329.

[50] A-Frame, *<a-text>*. [Online]. Available: `https://aframe.io/docs/0.7.0/primitives/a-text.html` (visited on 05/10/2018).

[51] (Mar. 15, 2018). Css fonts module level 3, [Online]. Available: `https://www.w3.org/TR/css-fonts-3/` (visited on 05/13/2018).

[52] *Creating text*. [Online]. Available: `https://threejs.org/docs/#manual/introduction/Creating-text` (visited on 05/31/2018).

[53] T. McReynolds, D. Blythe, S. (.-b. collection), E. C. (.-b. collection), and I. ebrary, *Advanced graphics programming using openGL*, English, 1st ed. San Francisco, CA: Elsevier Morgan Kaufmann Publishers, 2005, ISBN: 1558606599; 9781558606593;9780123814999;0123814995;0080475728;9780080475721;

[54] *Gridhelper*. [Online]. Available: `https://threejs.org/docs/#api/helpers/GridHelper` (visited on 05/14/2018).

[55] Z. Canter. (2018). Aframe - scatterplot, [Online]. Available: `https://github.com/zcanter/aframe-scatterplot` (visited on 05/31/2018).

[56] *Line*. [Online]. Available: `https://threejs.org/docs/#api/objects/Points` (visited on 05/08/2018).

[57] *Sprite*. [Online]. Available: `https://threejs.org/docs/#api/objects/Sprite` (visited on 05/13/2018).

[58] (Oct. 2005). Common format and mime type for comma-separated values (csv) files, [Online]. Available: `https://tools.ietf.org/html/rfc4180` (visited on 05/13/2018).

[59] Nanome. (2016). Calcflow, [Online]. Available: `http://store.steampowered.com/app/547280/Calcflow/` (visited on 04/18/2018).

[60] Google, *Tilt brush*. [Online]. Available: `https://www.tiltbrush.com/`.

[61] C. Solutions, *The hall*. [Online]. Available: `https://aframe.io/examples/showcase/museum/`.

[62] Blender Foundation, *Structure*. [Online]. Available: `https://docs.blender.org/manual/en/dev/modeling/meshes/structure.html` (visited on 05/13/2018).

[63] ——, *Primitives*. [Online]. Available: `https://docs.blender.org/manual/en/dev/modeling/meshes/primitives.html` (visited on 05/13/2018).

[64] ——, *Render baking*. [Online]. Available: `https://docs.blender.org/manual/en/dev/render/blender_render/bake.html` (visited on 05/14/2018).

[65] Blender, *Overview*. [Online]. Available: `https://docs.blender.org/manual/en/dev/editors/uv_image/uv/overview.html#uvs-explained` (visited on 05/13/2018).

[66] ——, *Mapping types.* [Online]. Available: https://docs.blender.org/manual/en/dev/editors/uv_image/uv/editing/unwrapping/mapping_types.html#unwrap (visited on 05/13/2018).

[67] Evan-Amos. (Sep. 2017). The vr book: Human-centered design for virtual reality, [Online]. Available: https://commons.wikimedia.org/wiki/File:Oculus-Rift-Touch-Controllers-Pair.jpg (visited on 05/10/2018).

[68] A. Systems. (Feb. 1990). Photoshop, [Online]. Available: https://www.adobe.com/Photoshop (visited on 05/09/2018).

[69] D. Jamieson. (Aug. 2015). The designer's notebook: Eight ways to make a bad tutorial, [Online]. Available: https://gamedevelopment.tutsplus.com/tutorials/4-ways-to-teach-your-players-how-to-play-your-game--cms-22719 (visited on 04/27/2018).

[70] A-Frame. (2017). Link traversal, [Online]. Available: https://aframe.io/blog/aframe-v0.6.0/ (visited on 05/12/2018).

[71] KingRahl. (Jun. 2015). Oculus touch controller mockups, [Online]. Available: https://www.thingiverse.com/thing:878939 (visited on 05/09/2018).

[72] A-Frame, *Lights.* [Online]. Available: https://aframe.io/docs/0.8.0/components/light.html (visited on 04/27/2018).

[73] D. N. Schuurman. (2013). Unity - raycasting, [Online]. Available: https://unity3d.com/learn/tutorials/topics/physics/raycasting (visited on 04/10/2018).

[74] A-Frame. (2018). Aframe/raycaster.js at master aframevr/aframe, [Online]. Available: https://github.com/aframevr/aframe/blob/master/src/components/raycaster.js (visited on 04/17/2018).

[75] W. Murphy. (2017). All-in-one natural hand controller, pointer, and gaze interaction library for a-frame, [Online]. Available: https://github.com/wmurphyrd/aframe-super-hands-component (visited on 02/15/2018).

[76] a-frame. (2018). Laser-controls, [Online]. Available: https://github.com/aframevr/aframe/blob/master/docs/components/laser-controls.md (visited on 04/02/2018).

[77] S. Wikström. (2017). Gång- och rotationshastigheter för effektiv navigering i vr, [Online]. Available: https://liu.diva-portal.org/smash/get/diva2:1083448/FULLTEXT01.pdf (visited on 04/20/2018).

[78] Fernandojsg, *Aframe-teleport-controls.* [Online]. Available: https://github.com/fernandojsg/aframe-teleport-controls (visited on 03/15/2018).

[79] S. Doody, *Starter questions for user research.* [Online]. Available: http://projects.iq.harvard.edu/files/harvarduxgroup/files/ux-research-guide-sample-questions-for-user-interviews.pdf (visited on 05/13/2018).

[80] C. Murphy. (2018). A comprehensive guide to user testing, [Online]. Available: https://www.smashingmagazine.com/2018/03/guide-user-testing/ (visited on 04/26/2018).

A
User Stories

- As a user I want to see a scatter graph in the museum.
- As a user I want to scale a bar graph with the help of a measurement that already exist (i.e a constant).
- As a user I want to rotate a bar chart to be able to see all bars in the graph.
- As a user I want see Stack Overflow Developer survey statistics in a bar graph to be able to analyze it.
- As a user I want to see multiple words in correct order along the x-, y- and z-axis, so I can know what each row in the bar graph is.
- As a user I want a skybox to be able to separate the background with the ground.
- As a user I want to see two bar charts in the same world to be able to compare them.
- As a user I want to press a bar chart with a "laser" and get some type of response to be able to get more information about a data point.
- As a user I want to use my controller to teleport around in the world to be able to move around in the environment.
- As a developer I want clean code and good code structure for better understanding.
- As a developer I want to set the bar size (width/depth) of the bar chart, to be able to decide how big a graph will be in the the environment.
- As a user I want to see two graphs in a "museum" to be able to interact with them.
- As a user I want to see one word in VR to be able to use it in the graphs.
- As a user I want to scale a graph with my controllers to make it intuitive to scale it.
- As a user I want to see a sign with some basic controls in the beginning of the museum to be able to learn how to interact with graphs.
- As a user I want a nice museum with textures to be able to walk around in an enclosed environment.
- As a developer I want a 3D model to work with for the "museum" to be able to view it in VR.
- As a user I want to see a scatter graph with data from the Stack Overflow survey to be able to analyze it.
- As a user I want to rotate a graph with my controllers to be able to see hidden data in the graphs.
- As a user I want to rotate an object without gravity and around the y-axis to be able to rotate the object from far away.

- As a user I want to see the data(height), x-label, z-label on my controller to be able to see the data more clearly.
- As a user I want to press a bar and see the data(height), x-label and z-label of the bar to be able to see the data more clearly.
- As a user I want to be able to read the text of the graphs to understand what they graphs represent.
- As a user I want windows in the museum to get a look outside so the room does not feel claustrophobic.
- As a user I want to read messages of all tutorials that explains how the program works so that I can learn how the interactions work.
- As a user I want simple colors in the museum to be able to see the text of the graphs more clearly.
- As a developer I want states to move between the different steps of the tutorial to be able to clearly move from one event to another.
- As a user I want to see controller models that show how I interact with all graphs to be able to understand where the controls are on the controller.
- As a user I want one color of all bars in the graph so as to not create any confusion as to what the colors mean.
- As a user I want the models and text in the tutorial to be positioned in a good way to make it easy to see and read the text and the models.
- As a developer I want organized event handling to make the code more structured.
- As a user I want to portal between the tutorial and the museum to act as a hyperlink between two scenes.
- As a developer I want all graphs to be implemented with THREE.js or shaders to improve performance.

B
Applications Evaluated

App Evaluated	Conclusion of Evaluation
Guns'n'Stories	It felt somewhat disturbing being stationary in one place. This made it quite boring since you were stuck with the same view the entire time It was difficult to aim accurately.
Space Station Explore	It was difficult to judge distance to things in this app which made it feel like everything was out of reach that you should grab. Also since a lot of movement was done without our physical body moving made it very disturbing and the motion sickness came quickly.
Tilt Brush	In this app being stationary didn't feel disturbing, you were able to spin the object that you were handling and the painting felt accurate. The environment that you were placed in felt spacious and gave a feeling of freedom which was nice, but because it felt so spacious and you were trapped in a stationary position it felt difficult to fill the room with paint.
Space Pirate Trainer	In this game it felt very appropriate to be stationary in one place. The room felt open but you were stuck on a spaceship and I didn't feel any need that it would be better if I moved overthere. The spaceship made it feel like there was boundraries.
The Hall	Teleporty controls allowed the user to move around quickly and at the same time not feel motion sickness. The museum environment that was being presented felt spacious and no feeling of being trapped occured. Also since it was so easy to move around they could have multiple areas showcasing different items and no difficulty to move between these.
Dirt Rally	Having the head move (car) without the user's actual head move in real life became very disturbing quickly. The experience was very immersing though.
D3js	This site allowed for many inspirations of how data could be visualized in different ways. Although being in 2d many of them, it gave some clue to how it would look in a 3d environment.
A-Blast	See Space Pirate Trainer.
Lights on aframe.io	This app gave great insight in how lights could be used in A-Frame. For navigation, in this app you were stationary but in a completely open environement. This was very disturbing because there was no floor to stand on, this made it feel very disorientating when looking down.

Freedom Locomotion VR	This game allowed for motion through moving your hands back and forth jogging in place as well as through joystick. The first option was just very clumsy and didn't feel natural at all. Moving through using the joystick caused motion sickness in just a couple of minutes and also didn't feel natural since you were moving like on rails.
Aframe - Scatterplot	Example of how particles can be used in a scatter graph.
Aframedc	Showed that graphs can be split into multiple components that should be coded individually to avoid code duplication.

C
Test Document 1

Test plan for user test #1

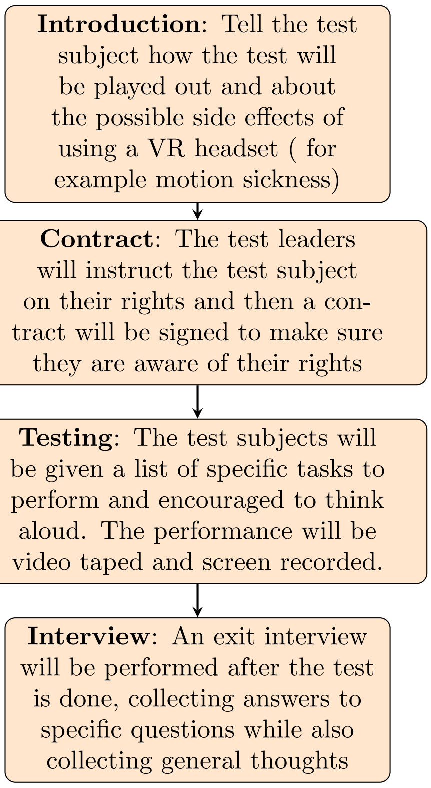
1 The Goal of the test

The goal of the test is to see if a user can use prototype #1 to read both of the diagrams (bar graph and scatter plot). We want to see if the user can figure out the current controls and be able to teleport, move a diagram, and rotate it. Specifically, we want to see if a user can intuitively use the prototype to find the correct data in the diagrams. The results from this test will be used to enhance the usability of the next prototype.

2 Logistics

There will be two test times: **Thursday 22/2** between 12:45 and 17:00 and **Friday 23/2** between 13:00 and 14:00. The test will take place in Medialabbet at Kuggen at Chalmers Lindholmen.

We will be performing a moderated, in-person test. The test will be performed in Swedish because that is the language the participants are most comfortable with, but all the test subjects have been asked if they speak English. The workflow is seen in the following flow chart:


Introduction: Tell the test subject how the test will be played out and about the possible side effects of using a VR headset (for example motion sickness)
Contract: The test leaders will instruct the test subject on their rights and then a contract will be signed to make sure they are aware of their rights
Testing: The test subjects will be given a list of specific tasks to perform and encouraged to think aloud. The performance will be video taped and screen recorded.
Interview: An exit interview will be performed after the test is done, collecting answers to specific questions while also collecting general thoughts

3 Participant profiles

All of the participants are Bachelor and Master students at Chalmers University of Technology. They have limited to no previous experience with virtual reality applications before the test. All of them speak both English and Swedish.

4 Tasks

- How many summer homes were sold in Stockholm during Q2 in 2016?
- What is the trend for summer home prices in Stockholm over the past decade?
- Give an example of a county with a large population and a high median wage.
- Give an example of a county with a high median rent.

5 Metrics and questionnaires

We have chosen to do an interview after the test is completed. This choice is based on the fact that we want to know about the user's general experience when using the application. However, we have also come up with a few questions to bring up during the interview which will allow us to get some quantitative data from the study as well. During the interview we will ask the following questions:

- How easy were the controllers to use? (scale 1-5) Did you understand how the controllers worked straight away? Did you experience any specific difficulty when using them? Did you use the help sign before starting your tasks?
- Was the program easy to use? (scale 1-5) Did you feel you could follow our instructions without any problems?
- How frustrated with the application did you feel during the test? (scale 1-5) Is there any specific action or event you can point out as frustrating?
- Which function did you think was the most useful?
- What did you most like about the application?
- What did you think about the environment? How trapped did you feel by it? (scale 1-5)
- How easy was it to navigate? (scale 1-5)
- Do you have any other insights you want to share?

6 Description of the system

The application is a web application that can be used with an Oculus Rift.

D
Test Document 2

Test plan for user test #2

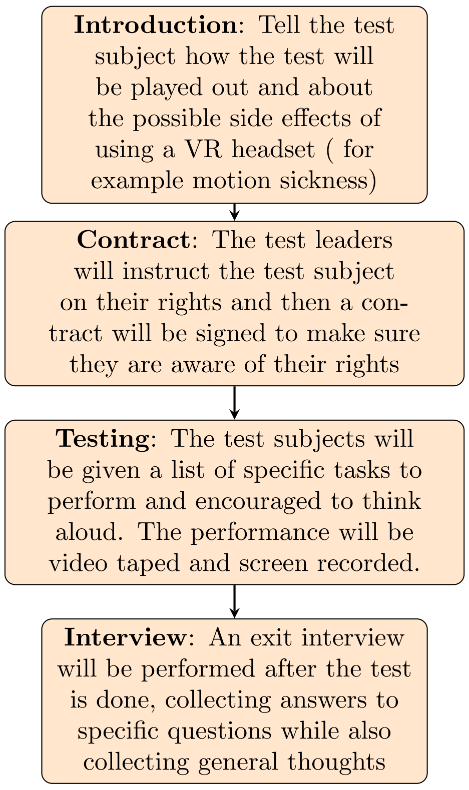
1 The Goal of the test

The goal of our second test is to see if the tutorial we have developed will help the users to learn the controls better than the last prototype we tested. Because our users didn't manage to use all controls last time, we also want to check the functionality of the controls and see if the users think they are useful.

2 Logistics

The test will take place on: **Tuesday 27/3** between 11:00 and 14:00. The test will be performed in Medialabbet at Kuggen at Chalmers Lindholmen.

We will be performing a moderated, in-person test. The test will be performed in Swedish because that is the language the participants are most comfortable with, but all the test subjects have been asked if they speak English. The workflow is seen in the following flow chart:



Introduction: Tell the test subject how the test will be played out and about the possible side effects of using a VR headset (for example motion sickness)

Contract: The test leaders will instruct the test subject on their rights and then a contract will be signed to make sure they are aware of their rights

Testing: The test subjects will be given a list of specific tasks to perform and encouraged to think aloud. The performance will be video taped and screen recorded.

Interview: An exit interview will be performed after the test is done, collecting answers to specific questions while also collecting general thoughts

3 Participant profiles

All of the participants are Bachelor and Master students at Chalmers University of Technology. They had a various amount of experience with virtual reality applications before the test. All of them speak both English and Swedish. They all have a good understanding of how to use a computer.

4 Tasks

- Do the tutorial and then continue to the museum.
- Explore the museum and tell us your initial reactions
- What was the three most common programming languages in 2017?
- Can you see any general trend in the relation between job searching time, salary, and years spent coding professionally?
- Choose any data point and tell us how much time the person spends on looking for a new job.

5 Metrics and questionnaires

Before the test starts, we have composed a few introductory questions for the users:

- Have you ever used a VR headset before?
- Do you have a good understanding of computers?
- Do you play video games?

Our tutorial is based on tutorials for video games, and it would be interesting to see if there is any difference between people who usually play video games and people who don't. Hopefully, the tutorial will be helpful to everybody. We also assume good knowledge with computers when developing the application.

We have chosen to do an interview after the test is completed. This choice is based on the fact that we want to know about the user's general experience when using the application. However, we have also come up with a few questions to bring up during the interview which will allow us to get some quantitative data from the study as well. During the interview we will ask the following questions:

- Was it easy to learn how the controls worked? How easy were the controllers to use? (scale 1-5) Did you think the tutorial gave you enough information to use the controllers correctly? Did you experience any specific difficulty when using them? How easy was it to navigate using the teleport function?
- Was there any part of the controls that you did not use? If so, why and what controls?
- Was the program easy to use? (scale 1-5)
- Was there anything that went wrong during your test?
- How frustrated with the application did you feel during the test? (scale 1-5) Is there any specific action or event you can point out as frustrating?
- Which function did you think was the most useful?
- What did you most like about the application?
- What did you think about the environment?
- Do you have any other insights you want to share? Was there anything missing from the application?

6 Description of the system

The application is a web application that can be used with an Oculus Rift.

E
Consent Form for User Testing

User test consent form

During the user test:

- You will be asked to perform tasks on a web application with virtual reality goggles.
- Your performance will be both physically and virtually recorded using a camera and a screen capturing program.
- We will conduct an interview after you have the tasks have been performed.

Participation in this user test is completely voluntary. You may at any point stop the test.

The information gathered in this test will be used in a bachelor's thesis report at Chalmers University of Technology. The results will in no way be linked to you personally and no names or other confidential information will be mentioned in the thesis. You also have the right to request the deletion of any record of your involvement.

For any further questions or information, please contact Miranda Aldrin (email omitted).

I have understood the above information and agree with it.

______________________________________	_____________
Subject's signature	Date

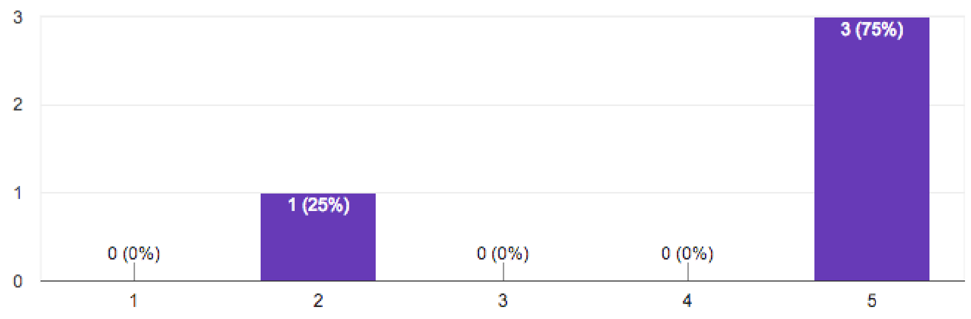
F
Results of User Test 1

The questions with scales from 1-5 goes from negative to positive. For example, in question 1, the scale goes from not easy to very easy.

1.

Hur enkla var kontrollerna att använda?

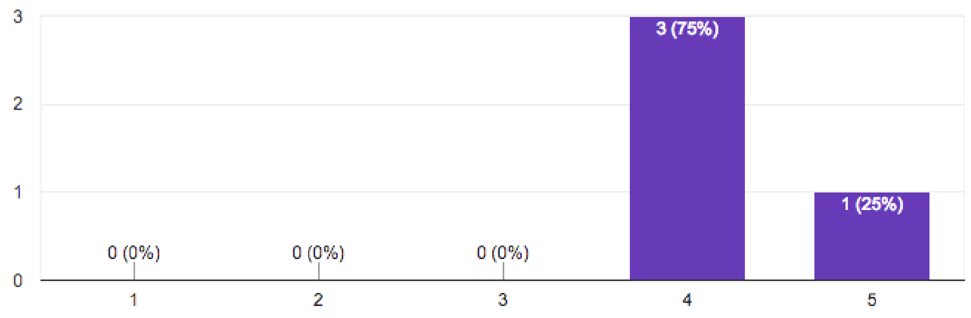
4 responses



2.

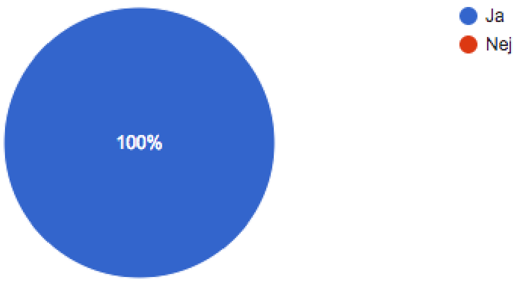
Hur lätt var det att navigera?

4 responses



3. Använde du hjälp-skylten i början av museumet?

4 responses



Var det något specifikt som var extra svårt att göra eller att använda?

4 responses

Väldigt svårt att avläsa i scatter plot

Nej

Prickade ibland och ville inte ändra ibland

Missade att det vara 3 axlar i scatterploten och färgerna i scatterplotens färger smälte in

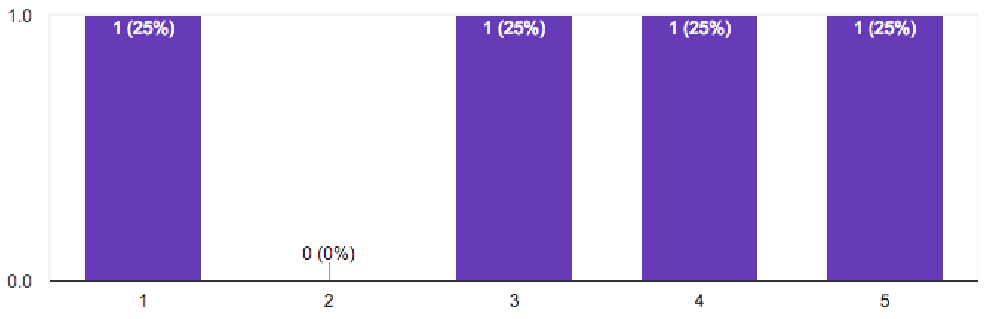
4. Var det något specifikt som var extra svårt att göra eller att använda?

4 responses

Väldigt svårt att avläsa i scatter plot
Nej
Prickade ibland och ville inte ändra ibland
Missade att det vara 3 axlar i scatterploten och färgerna i scatterplotens färger smälte in

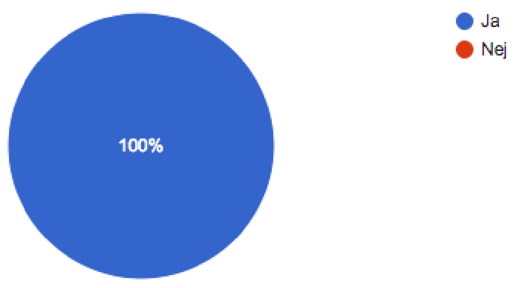
Hur lätt var det att använda programmet?

4 responses



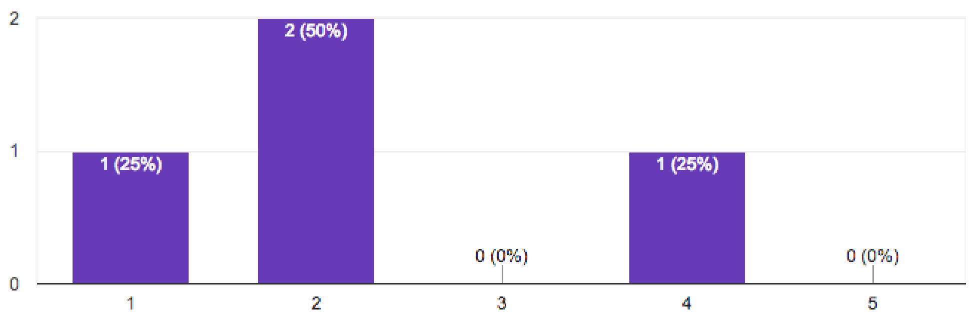
5. Kände du att du kunde följa våra instruktioner väl?

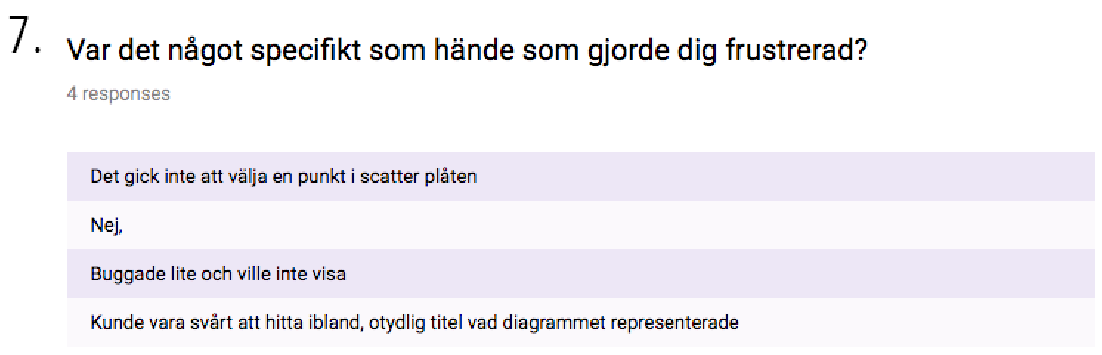
4 responses



6. Hur frustrerad blev du under testets gång?

4 responses

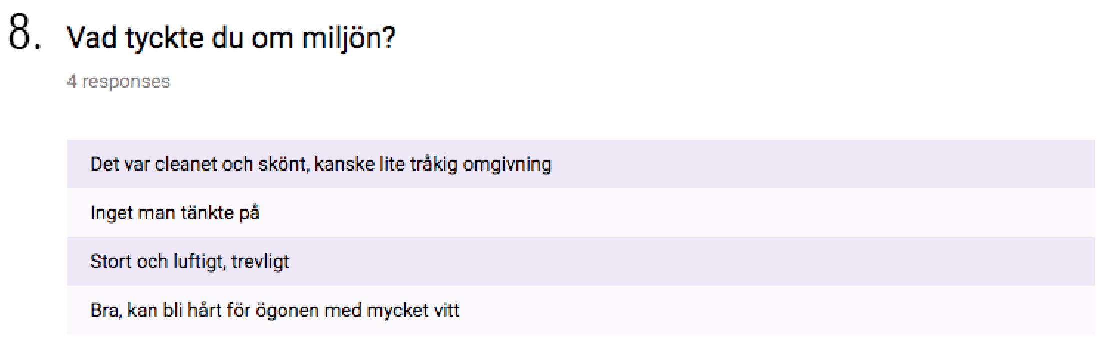




7. Var det något specifikt som hände som gjorde dig frustrerad?

4 responses

Det gick inte att välja en punkt i scatter plåten
Nej,
Buggade lite och ville inte visa
Kunde vara svårt att hitta ibland, otydlig titel vad diagrammet representerade



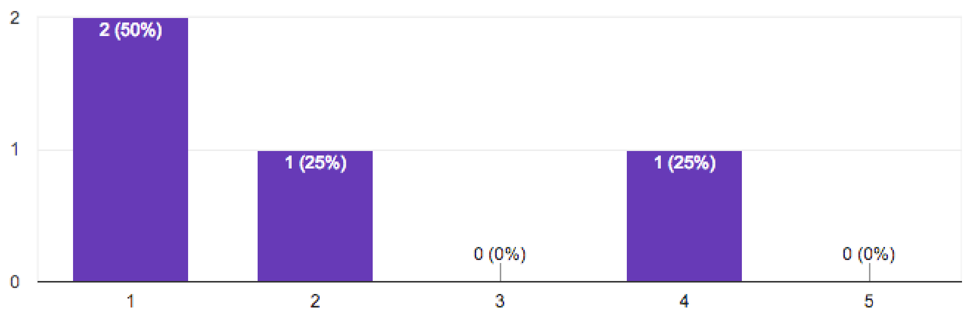
8. Vad tyckte du om miljön?

4 responses

Det var cleanet och skönt, kanske lite tråkig omgivning
Inget man tänkte på
Stort och luftigt, trevligt
Bra, kan bli hårt för ögonen med mycket vitt

9. Hur instängt tyckte du det kändes att vara i ett rum och kolla på diagrammen?

4 responses



10. Vilken funktion tyckte du var mest användbar?

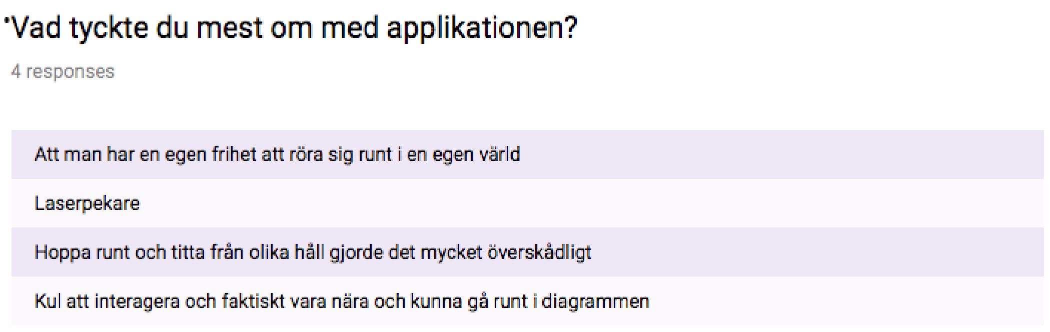
4 responses

Nice att man kunde teleportera sig runt

bra med laspekare, exakt och lätt att användare

Peka på grejer

Teleportering,

11.

Vad tyckte du mest om med applikationen?

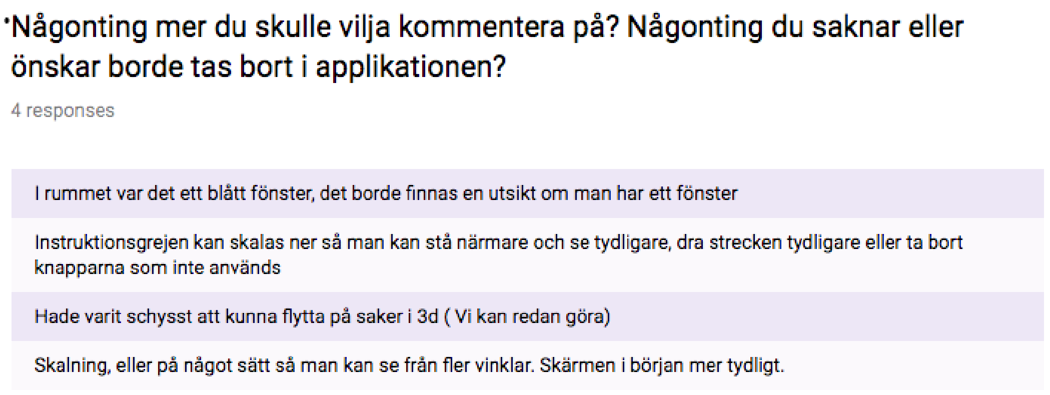
4 responses

Att man har en egen frihet att röra sig runt i en egen värld

Laserpekare

Hoppa runt och titta från olika håll gjorde det mycket överskådligt

Kul att interagera och faktiskt vara nära och kunna gå runt i diagrammen

12.

Någonting mer du skulle vilja kommentera på? Någonting du saknar eller önskar borde tas bort i applikationen?

4 responses

I rummet var det ett blått fönster, det borde finnas en utsikt om man har ett fönster

Instruktionsgrejen kan skalas ner så man kan stå närmare och se tydligare, dra strecken tydligare eller ta bort knapparna som inte används

Hade varit schysst att kunna flytta på saker i 3d (Vi kan redan göra)

Skalning, eller på något sätt så man kan se från fler vinklar. Skärmen i början mer tydligt.

Subject #1

Answers to the tasks:

1. 543
2. No
3. Nybro
4. Järfälla

- Had used VR before, but limited
- Did not read the tutorial
- Only used teleport (did not notice there were other controls)
- “Why are they different colors?” referring to the bar graph
- Had a hard time navigating

During interview:

- “Learned while time went on” (lärde sig under tidens gång)
- Had a hard time with the scatter plot. Suggested a change like a grid
- Frustrated over not being able to read the scatter plot
- The environment is a little bit boring
    - Maybe have a view from the windows?
- Teleport was useful
- Felt free to walk around the diagram, which was nice

Subject #2

Answers to the tasks:

1. 540-ish
2. Q2-Q3 seems to have more buyers
3. Nybro
4. Järfälla

- Has used VR before, but limited
- Didn’t find the teleport button (or any other button), and needed help
    - Was told that there might be a hint

During interview:

- Learning-curve
    - The controls got easier after a while, but in the beginning it was not something you are used to
- Wants a more streamlined controller
- Make the tutorial board smaller, so you can go up to it and
    - The lines on the tutorial board were difficult to see

Subject #3

Answers to the tasks:

1. 543
2. More sold in later quarters
3. Soruman
4. Sorsele

Interview:

- Difficult to know which button to use
    - Wanted to use thumb-button to teleport
    - Used the tutorial sign, but not to its full extent. Did not see if
- Seems like a better way to look at statistics
- The laser pointer was good
- It was nice to be able to jump around and look at things from a different angle
- Could use an interactive tutorial

Subject #4

Answers to the tasks:

1. 543
2. Ökar under sommarmånaderna
3. Nybro
4. Järfälla

Interview:

- Controllers were intuitive
- The tutorial was not intuitive however
    - The lines were too close to the color of the controllers on the tutorial board
- In the scatter plot: one of the axis (z axis) was not eye catching enough and was missed when trying to perform the task
- It would have been hard to use if you wanted to use make your own statistics and visualize it
- Sometimes you can get lost
- The data points in the scatter plot that are outliers are hard to see because the data point is yellow and the background is white
    - They are also high up (because the scatter plot is large)
        - Maybe be able to scale it?
- En tydligare turorial behövs

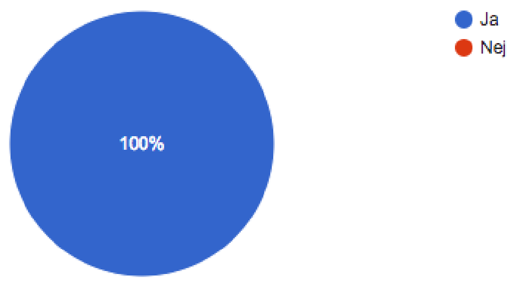
G

Results of User Test 2

The questions with scales from 1-5 goes from negative to positive. For example, in question 1, the scale goes from not easy to very easy.

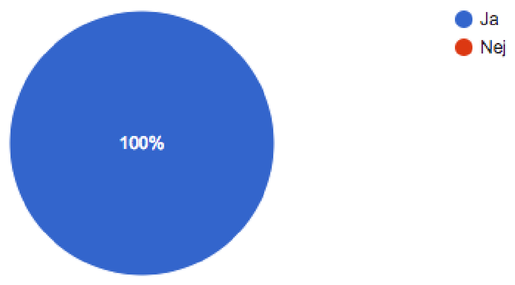
1. Har du använt något VR headset tidigare?

5 responses



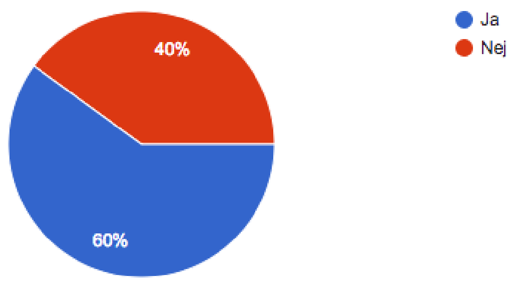
2. Känner du att du kan använda en dator på rätt sätt?

5 responses



3. Spelar du tv/datorspel?

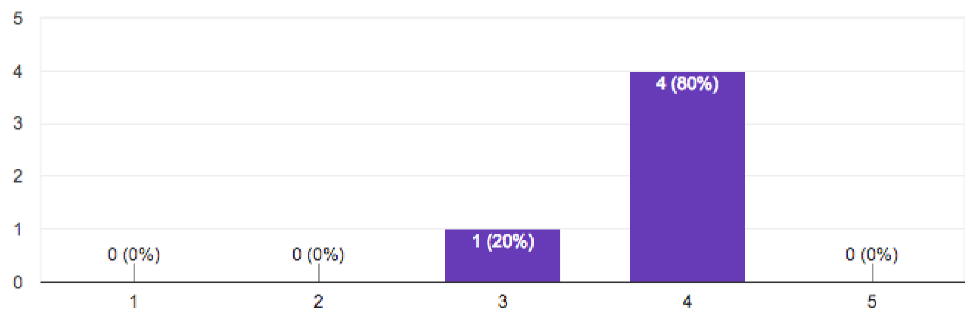
5 responses



Frågor efter testet

4. Hur lätt var det att använda programmet?

5 responses



5. Var det något som gick fel medan du använde testet?

5 responses

Nej, inget större fel

Moved around too quickly in the tutorial

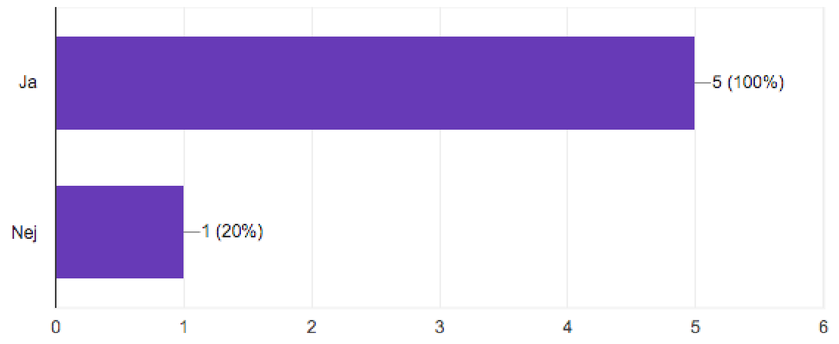
När man teleporterar till ett dumt ställe så är man fast där

jag råkade flytta en grej under marken, och den försvann. Råkade rotera väg saker lite. Svårt att tyda tutorial:en

Texten åkte igenom golvet

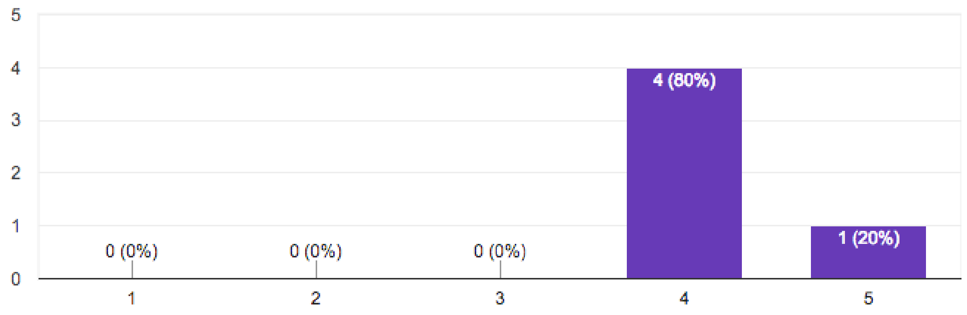
6. Var det enkelt att lära sig kontrollerna?

5 responses



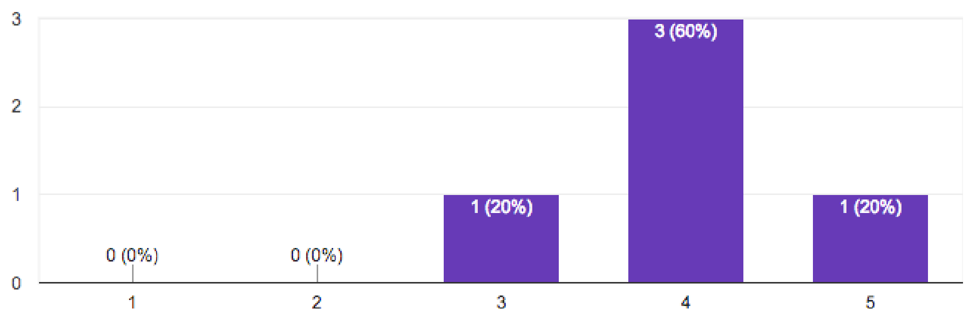
7. Hur enkla var kontrollerna att använda?

5 responses



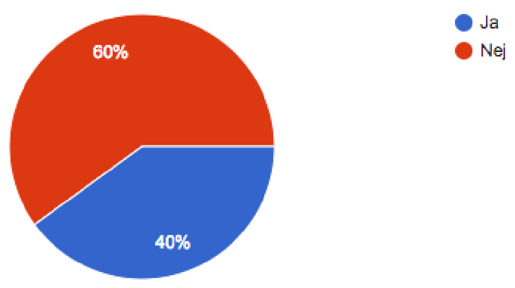
8. Hur lätt var det att navigera med hjälp av teleport?

5 responses



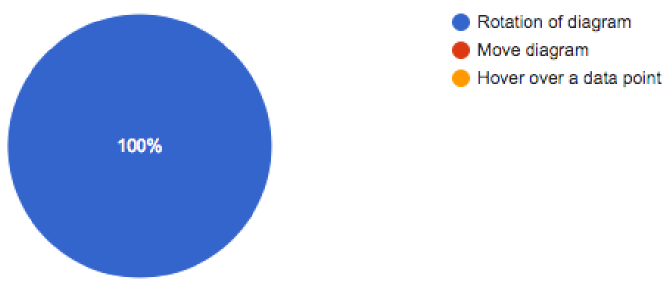
9. Var det någon funktion du lärde dig av tutorial:en som du sedan inte använde?

5 responses



10. Om så var fallet, vilken?

2 responses



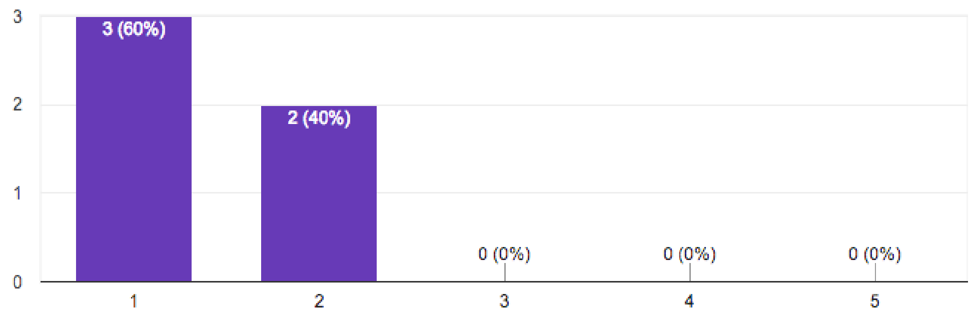
11. Var det något specifikt som var extra svårt att göra eller att använda?

5 responses

Nej, det dröjde lite innan jag förstod att jag skulle aväanda joystcik. Försökte använda handen först.
Getting the ight distance to the thing that you are looking at. Can't pull the diagram closer
Teleporten, det var ite intuitivt att an skulle lägga sakerna i cirklen
Rotera var extra svårt. Hög känslighet. Sen var det svårt att flytta saker närmare
Det var mest bara läsa

13. Hur frustrerad blev du under testets gång?

5 responses

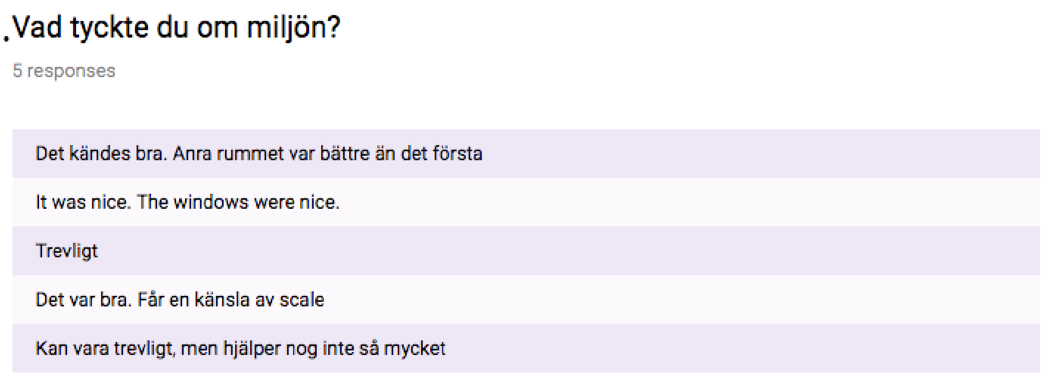


14. Var det något specifikt som hände som gjorde dig frustrerad?

2 responses

När tutorial texten försvann och jag hade hoppat iväg

Person var svårt att identifiera

15.

Vad tyckte du om miljön?

5 responses

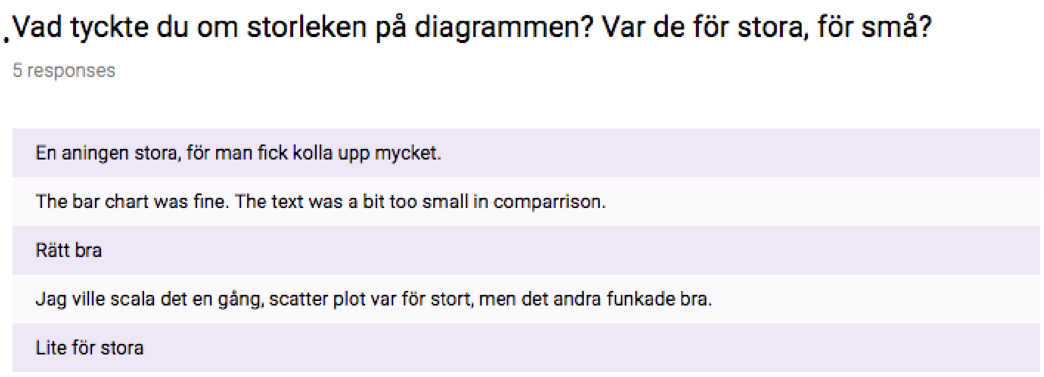
Det kändes bra. Anra rummet var bättre än det första

It was nice. The windows were nice.

Trevligt

Det var bra. Får en känsla av scale

Kan vara trevligt, men hjälper nog inte så mycket

16.

Vad tyckte du om storleken på diagrammen? Var de för stora, för små?

5 responses

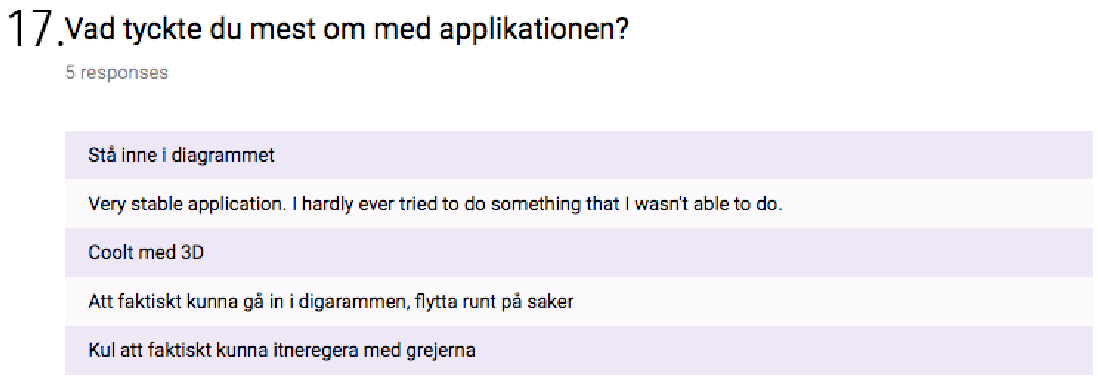
En aningen stora, för man fick kolla upp mycket.

The bar chart was fine. The text was a bit too small in comparrison.

Rätt bra

Jag ville scala det en gång, scatter plot var för stort, men det andra funkade bra.

Lite för stora



17. Vad tyckte du mest om med applikationen?

5 responses

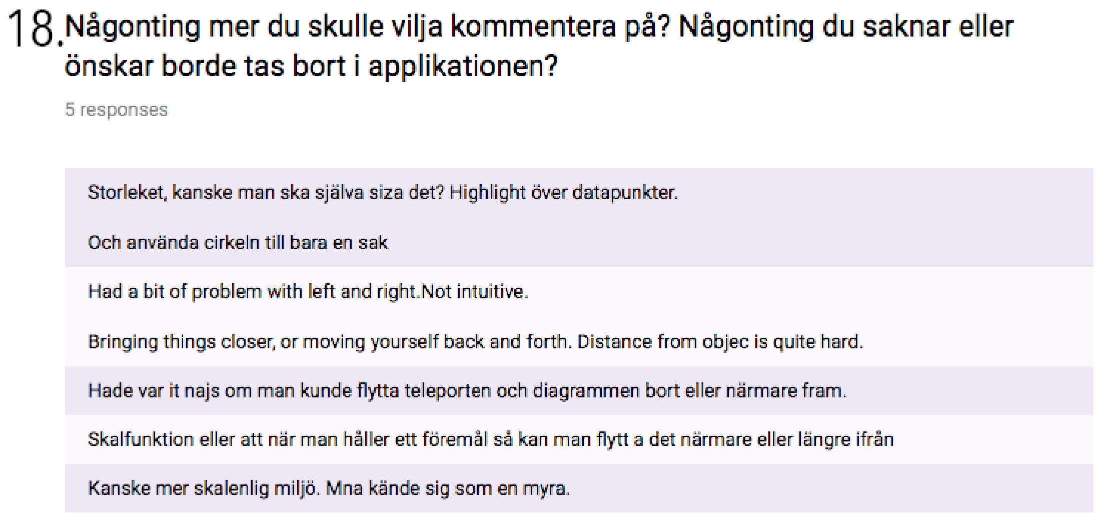
Stå inne i diagrammet

Very stable application. I hardly ever tried to do something that I wasn't able to do.

Coolt med 3D

Att faktiskt kunna gå in i digarammen, flytta runt på saker

Kul att faktiskt kunna itneregera med grejerna



18. Någonting mer du skulle vilja kommentera på? Någonting du saknar eller önskar borde tas bort i applikationen?

5 responses

Storleket, kanske man ska själva siza det? Highlight över datapunkter.

Och använda cirkeln till bara en sak

Had a bit of problem with left and right.Not intuitive.

Bringing things closer, or moving yourself back and forth. Distance from objec is quite hard.

Hade var it najs om man kunde flytta teleporten och diagrammen bort eller närmare fram.

Skalfunktion eller att när man håller ett föremål så kan man flytt a det närmare eller längre ifrån

Kanske mer skalenlig miljö. Mna kände sig som en myra.

Person 1

— TUTORIAL—

- [ ] Texten försvann så hon inte kunde rätta till sitt fel i att hon inte förstod var hon skulle ställa diagrammet i tutorialen
- [ ] Hon förstod inte var hon skulle ställa diagrammet i tutorialen - hon hoppade istället dit
- [ ] Tryckte på fel knapp när hon skulle portalera,ville teleportera in i den

— MUSEUM —

- Hade varit nice om punkten man pekar på highlightas på något sätt
- Klarade alla frågor utan svårighet
- Ville kunna skala diagrammet

— POST INTERVIEW —

Föredrog instängt rum över öppenheten i tutorialen
Diagrammen var lite för stora
Feedback på vilken datapunkt som är selectad

Person 2

— TUTORIAL—

- Förstod inte var han skulle flytta diagrammet först

— MUSEUM —

- Ville flytta diagrammet närmare sig själv
- Klarade alla tasks på egen hand utan att vi ens frågade
- Använde alla funktioner

— POST INTERVIEW —

- Dåligt att texten försvann då han ville bara börja göra saker och när han sen gick tillbaka så jhade texten försvunnit
- Inte intuitiva då kontrollerna var olika mellan vänster och höger hand och hade ingen aning om vad man hade för toolset
- Vill kunna dra saker närmare
- Nice med fönster
- Bar graphen var bra storlek
- Gillade hur bra att flöt på, allt han ville göra kunde han och allt bara flöt på
- Vill kunna flytta sig närmare

Person 3

— TUTORIAL—
Förstod inte var man skulle ställa diagrammet

— MUSEUM —
Kunde inte se någon trend

— POST INTERVIEW —
- Jobbigt att teleportera tillbaka
- Tyckte inte teleportens kurva var intuitiv
- Alla behövdes men rotation minst
- Vill ha en knapp för att flytta saker längre bort eller närmare

Person 4

— TUTORIAL—
- Läste inte instruktionerna, förstod inte att man skulle flytta diagrammet till punkten
- Texten försvann
- Ville ställa rotationsdiagrammet på markeringen
- Använde sig av alla funktioner

— MUSEUM —
- Ville skala ner scatter - diagrammet
- Läste inte av texten för datapunkten utan jämförde mot skalorna på sidan

— POST INTERVIEW —

Person 5
— TUTORIAL—

— MUSEUM —
- Tyckte texten var svår att läsa
- Tyckte färgen på texten smälte in i taket
- Grafen åkte igenom golvet
- Använde sig av alla funktioner

— POST INTERVIEW —

H
Storyboard of the initial stages of the tutorial

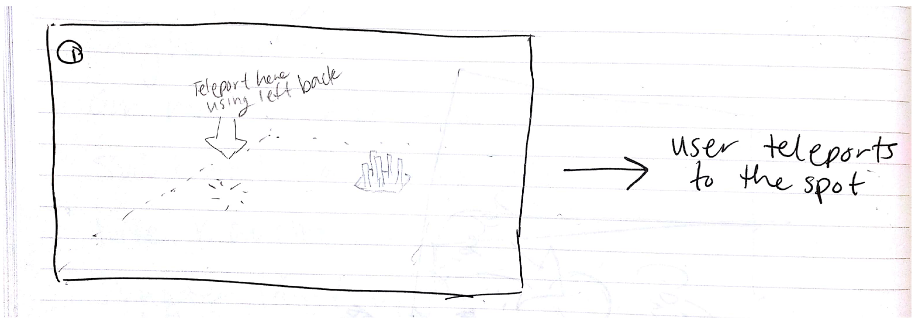


Figure H.1: The user enters the tutorial and is asked to teleport to a designated place.

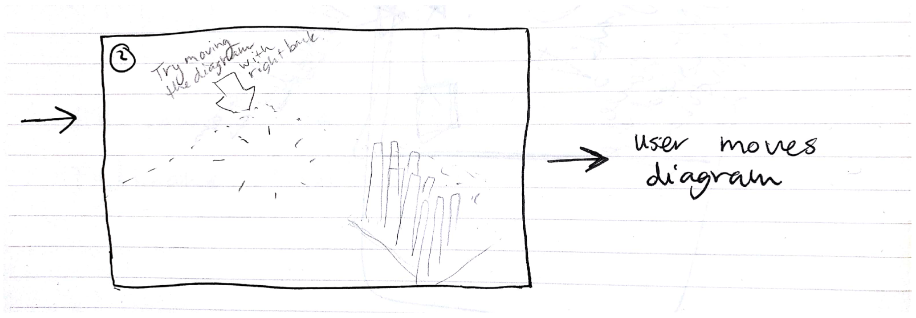


Figure H.2: The user is asked to move a bar graph to a designated place.

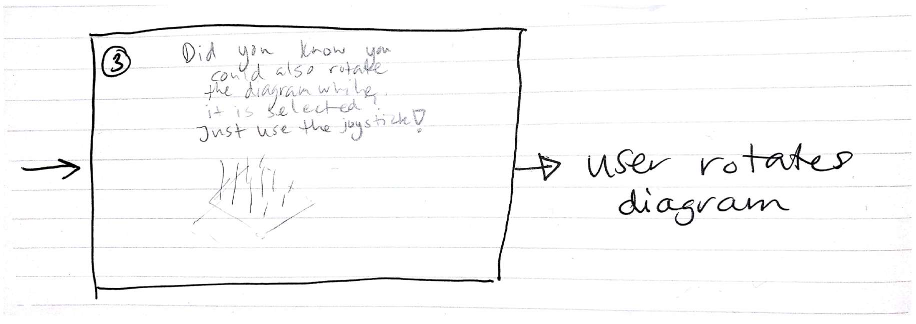

3
Did you know you could also rotate the diagram while it is selected? Just use the joystick!
user rotates diagram

Figure H.3: The user is asked to rotate a bar graph.

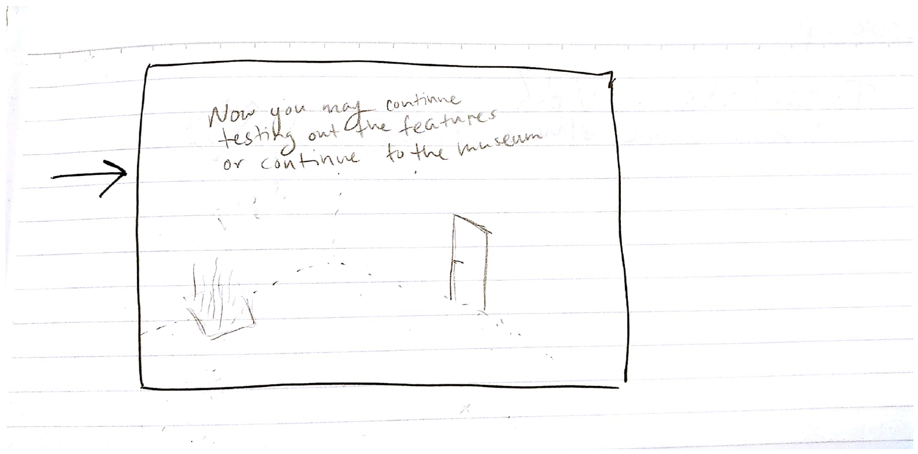
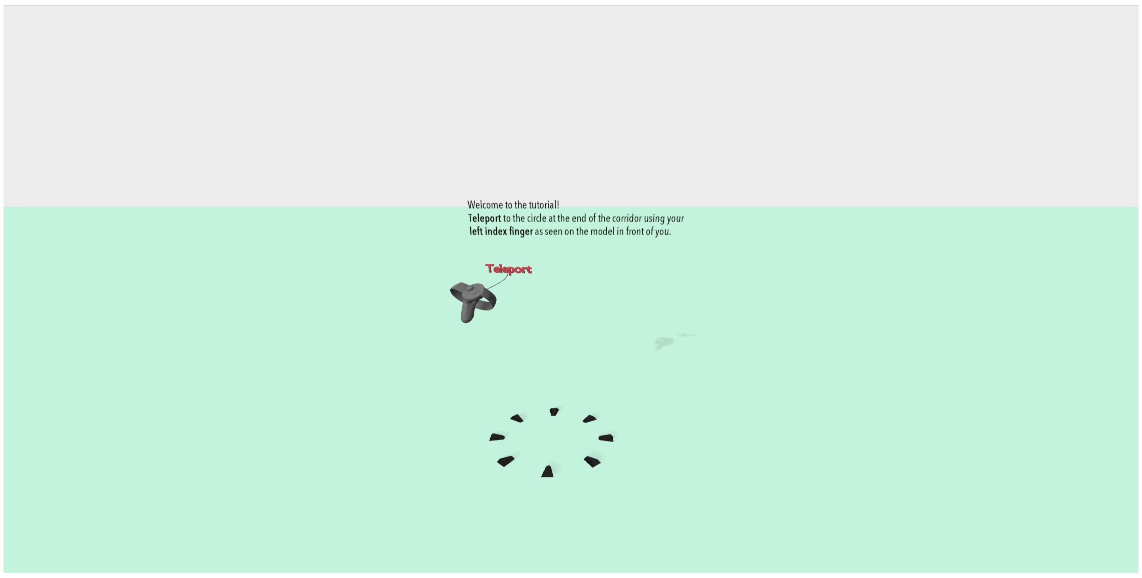


Figure H.4: Once the user has completed the tutorial, they will go to the museum.

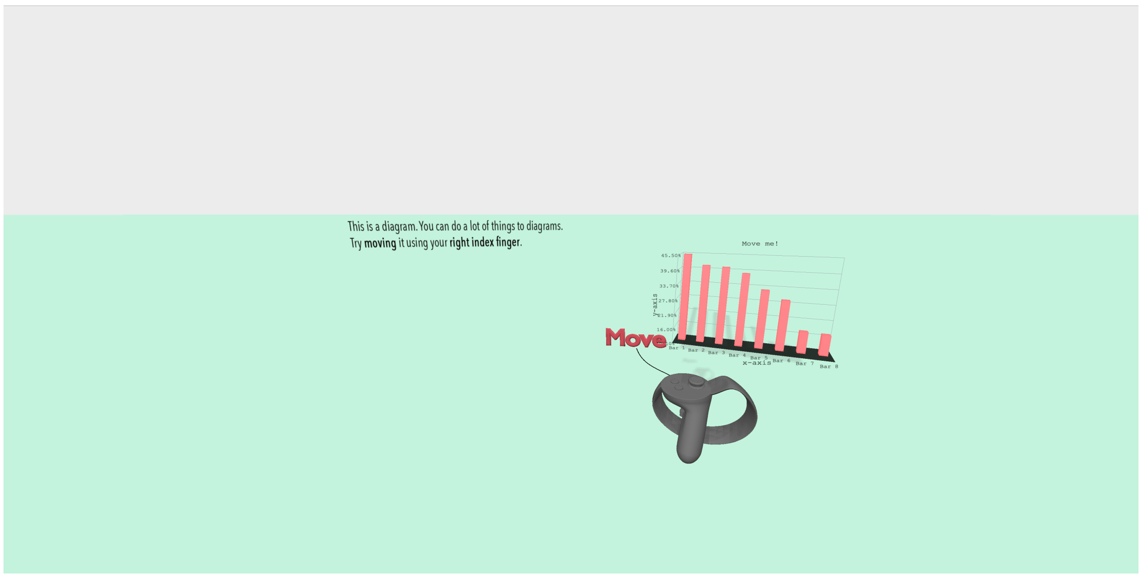
I

Steps of the tutorial



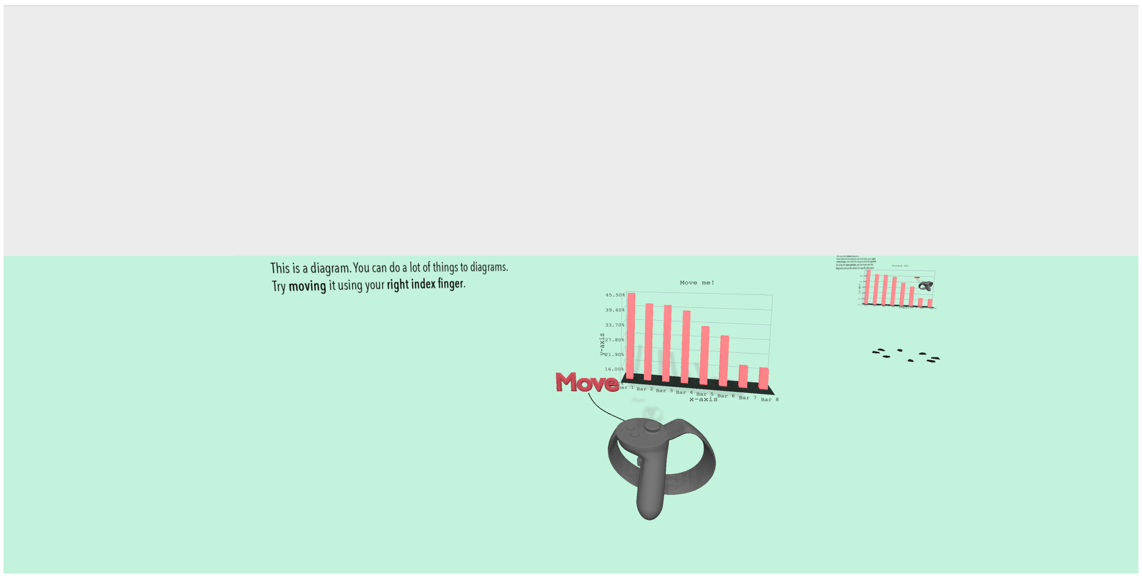
Welcome to the tutorial!
Teleport to the circle at the end of the corridor using your
left index finger as seen on the model in front of you.

Figure I.1: The first state in the tutorial, encouraging the user to teleport towards the circle.



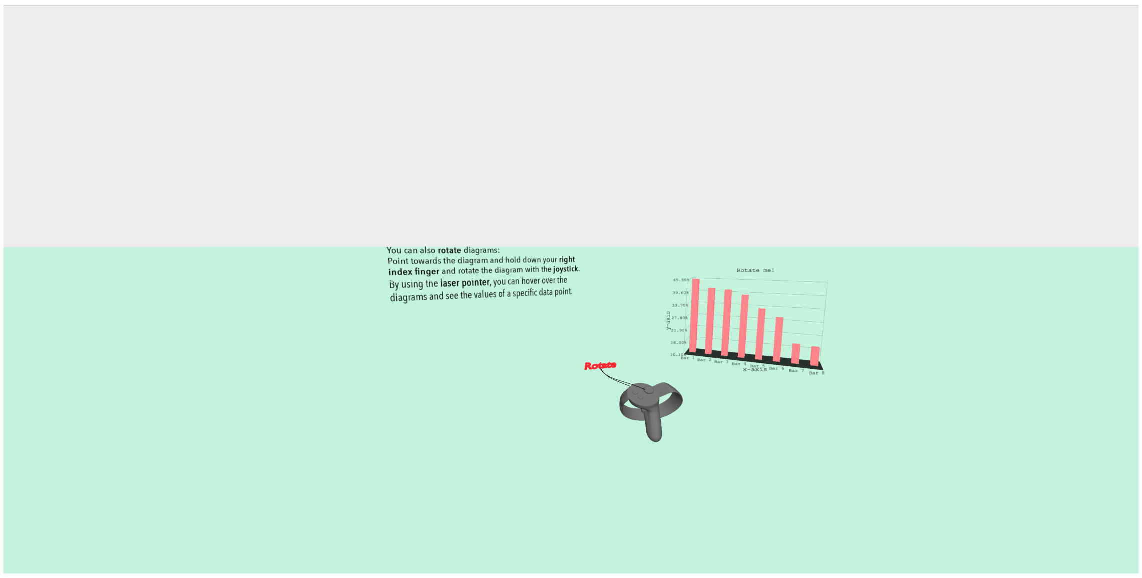
This is a diagram. You can do a lot of things to diagrams.
Try **moving** it using your **right index finger**.

Figure I.2: The second state in the tutorial, encouraging the user to move the graph.



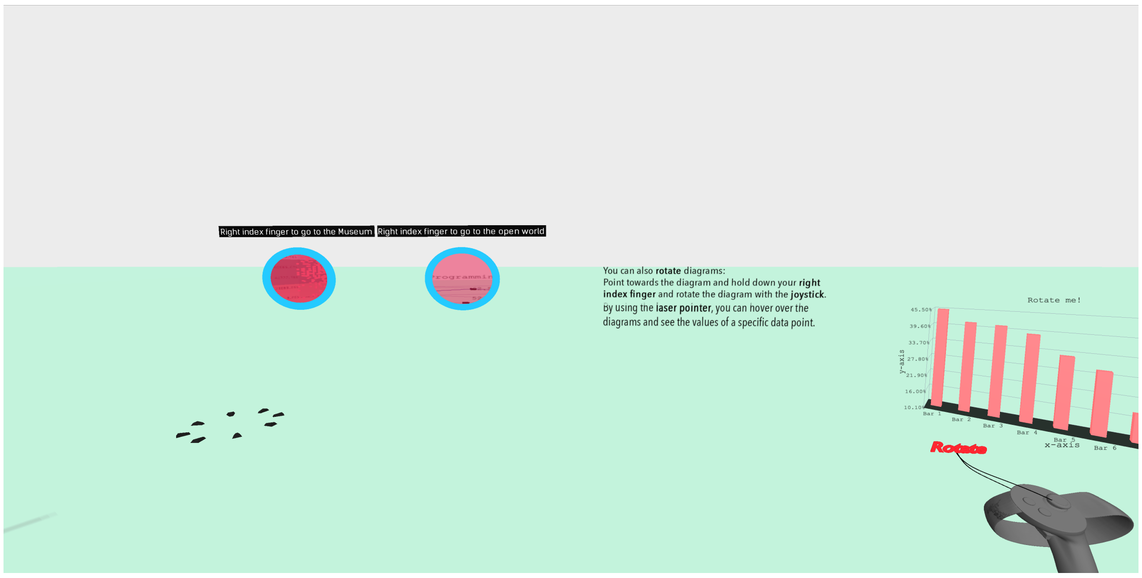
This is a diagram. You can do a lot of things to diagrams.
Try **moving** it using your **right index finger**.

Figure I.3: The third state in the tutorial, showing the view after the user has moved the graph in state 2.



You can also **rotate** diagrams:
Point towards the diagram and hold down your **right index finger** and rotate the diagram with the **joystick**.
By using the **laser pointer**, you can hover over the diagrams and see the values of a specific data point.

Figure I.4: Still the third state,encouraging the user to rotate the graph.



You can also **rotate** diagrams:
Point towards the diagram and hold down your **right index finger** and rotate the diagram with the **joystick**.
By using the **laser pointer**, you can hover over the diagrams and see the values of a specific data point.

Figure I.5: The fourth state in the tutorial, showing the view after the user has rotated the graph.

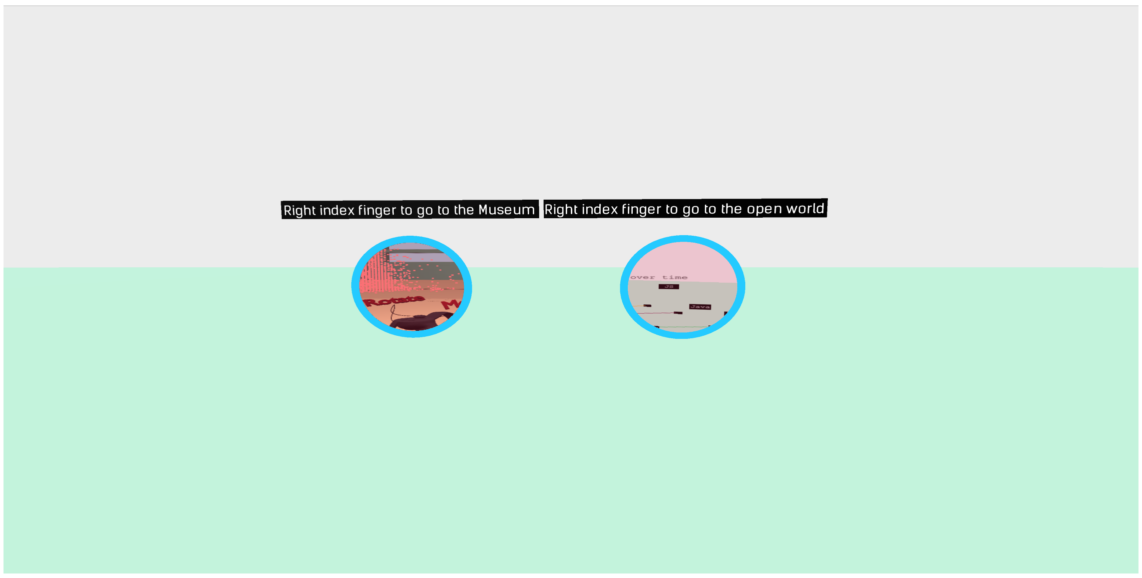

Right index finger to go to the Museum
Right index finger to go to the open world

Figure I.6: The fourth state in the tutorial, encouraging the user to go to either the museum, the open world, or the immersive scatter graph.