Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

# Virtual Reality experience for Spatial Data Visualization and Annotation

*Author:*
Luca Simonato

*Supervisor:*
Dr. Thomas Heinis

Submitted in partial fulfillment of the requirements for the MSc degree in MSc in Computing (Software Engineering) of Imperial College London

September 2020

**Abstract**

As human beings, we are immersed everyday in data. Everything we see surrounding us could in fact be collected as information and classified as spatial data. Collected spatial data is analyzed everyday by companies and researchers that make decisions based upon what they see, this could be about data marine currents, archeological findings, animal migratory flows or climate changes throughout the years. Visualizing such data is not trivial, usually this is done with simulations or by building 3-dimensional models that can be interacted with using specific software.

Virtual Reality is a fast growing field that is rapidly becoming more and more accessible. Its experiences are usually related to digital entertainment but are not limited to that. Thanks to the immersion that can be provided with recent hardware, the applications could be virtually endless. Spatial Data Visualization is one of the experiences that gives the best results when applied to Virtual Reality. It projects the user into a new way of visualizing data that is very natural as it mimics the real world. Moreover the user could have access to additional functions to improve the functionalities of the experience.

In this project I propose a Virtual Reality based experience to allow users to Visualize and Annotate Spatial Data of the stratified Earth's ground of the Volve field, a decommissioned oil extraction field. The software uses this dataset just as a demonstration of the potential applications and the ease of use of Spatial Data Visualization in Virtual Reality. The same piece of software could easily be used with different datasets and extended to include more appropriate functionalities.

# Acknowledgments

I would like to thank my Supervisor, Dr. Thomas Heinis, for the opportunity to work on such a project.

I would also like to thank Riccardo Bovo and his availability for the support provided during the development of the project.

Finally and most importantly I want to personally thank all the people that were close to me during this period of time: my parents, my sister, my friends and Manuto, his silence during every debugging session was the key to moving forwards in the project.

Grazie Padre, Madre, Sorella, Zia, Deca, Lollo, Lucone, Federica, Guti, Cami, Dev, Gna, Victor, Mark, Mary, Giulio, Dani, Fede, Marco, Andrea, Flavio, Stefano, Fabrizio, Luca, Valerio, Manuto.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Spatial Data Visualization is widely used by researchers and companies in a variety of fields. Specific pieces of software are normally used to visualize such data, learning how to use these applications could be tedious or even hard for some users that do not have experience with navigating 3-dimensional models using a computer. Virtual Reality, as well as Augmented Reality, provides a possible platform to give users an easier and more intuitive way to navigate Spatial Data.

The main reason why Virtual Reality systems are not currenlty used for this applications is the fact that VR itself is a very recent field of development. Only recently big tech companies started investing, researching and pushing software development is this direction. This led to the production of more competitive headsets as well as more affordable ones, making such products easily available for the common public.

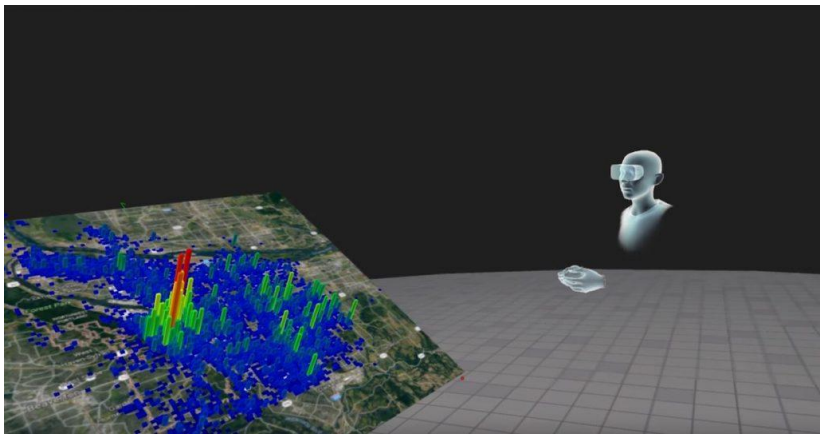

**Figure 1.1:** Source: https://www.zdnet.com/article/data-visualization-via-vr-and-ar-how-well-interact-with-tomorrows-data/

Ease of use is not the only factor that contributes to the choice of using VR of AR for visualizing data The amount of data that can be visualized at any given time is also increased as the users can have data distributed around them at 360 degrees as

well as above or below him. Data manipulation can also be improved by providing a wide range of new inputs thanks to gestures, giving the users the possibility to modify data by using intuitive and more natural input sources.

When developing a data visualization software, data annotation is arguably the most important feature that can be implemented if multiple users should access the same dataset. When the data is shared between a group of users, expecially if the objective of such group is to analyze the data to make decisions based upon it, it is vital to provide the possibility to comment the data in order to highlight something, give personal opinions, and discuss decisions.

There are multiple ways to annotate data, and a lot of research has been done in the past to understand what is the best way, but when is comes to Virtual Reality there are some new considerations to be done. This project gave me the opportunity to reason about this and to give an example of how spatial data could be annotated in Virtual Reality.

## 1.2 Aims

The aim of this project is to produce a piece of software to highlight the possibility and more importantly the benefits of Spatial Data visualization and annotation in a Virtual Reality environment.

In order to reach this objective I first of all had to research on the subject, data annotation is a well researched and documented field. Starting from existing pieces of research I had to adapt my finding to make them suitable for a VR environment. After that, I used some real world data to generate a 3-dimensional model that would act as the visualizable spatial data. Being able to interact with such data in order to improve the visualization was a key requirement of the development.

Finally, implementing the previously researched annotation system, and creating a stable VR experience would have marked the end of the project.

## 1.3 Outcomes

Using real world data provided by Equinor [1] I managed to produce a stable apk that can be installed on any Oculus Quest VR headset. The VR experience is developed thinking about researchers and their needs to explore stratified ground data in order to decide where to place oil wells. Even if the dataset used allows to explore Volve, a decomissioned oil field in the North Sea[2], the experience is designed to provide access to different datasets with a single installation.

Once the user is inside the experience he is able to interact with the desired data using intuitive gestures that recall real life ones. The user also has the ability to annotate such data by placing audioclips anywhere in the 3D dataset. These clips can be listed to and followed up by other researchers using the same headset during different sessions.

To provide evidence of the ease of use and to highlight the benefits of the immersion provided by using Virtual Reality, a similar experience with the same functionalities has been developed to be used in a non VR scenario. In this experience users can use any computer running a Windows Operating System to achieve the same objectives.

Finally, the VR experience was evaluated in order to ensure a smooth execution on the Oculus Quest device. The computational power provided by the hardware put some limits to the graphics of the system. As a consequence the application was adapted in order to ensure a stable framerate that would not impair the user experience.

# Chapter 2

# Background

When it comes to annotating data, the immediate thought goes to writing. Placing some text right next to the data we want to say something about, seems like the obvious action to take. But there are actually many different ways to annotate data, expecially when we think about specific environments.

Various researchers tried to identify the best way to annotate data, but it really depends form the context. Before deciding and commiting on an annotation system, we need to ask ourselves the following questions:

- Who and how many people are accessing and annotating the same data?
- What kind of information will be annotated on the data?
- How is the data being visualized from the users?

With these questions in mind I will run trough the various pieces of research I have read, and I will highlight some key contributions I absorbed from every academic text analyzed, considering only what is applicable to my specific domain of development. Finally I will explain the reason behind my choice.

## 2.1 Previous research

In 2013, J. Bamber described a possible way of annotating data accessible from different people[3]. Although this article reports information about spatial annotations used in order to have users from different backgrounds, and with different levels of expertise and knowledge, communicate and have a confrontation of points of view (not applicable in our intended use case, since the software should be used only by experts), it does provide some useful inputs on how such annotations are placed and handled:

- Annotations are spatially placed on a 2D map in this specific case, by using 3 different not editable icons;
- Different icons are used to categorize the issue that is being described, but they do not provide detailed information;

- Each icon is linked to a text annotation that describes the user's point of view in detail;
- Each annotation, with the first text comment, starts a sort of forum thread like conversation in which different users can answer, contradict or confirm other users' comments;

This article's annotation method is really straightforward and intuitive, but the research does not face some issues that consequentially arise after adopting such a method, for example: what if different users want to annotate some opinions that belong to different categories on the same point? should they create another icon on the map on the same point (but this could be visually unpleasant) or should they respond to the same thread even if the issue is of a different nature?

On the other hand, there are also some more complicated but also more flexible annotation methods, like 3D sketching. As described in a research from 2001[4], 3-dimensional sketching gives the user a lot more freedom, but it also makes the system possibly much more complicated to use. Writing a long and detailed description of some data for example, would result in a not so immediate act, so this is probably to be used only as a secondary way to annotate data.

There are also past researches on cooperative annotation methods, such as this article[5] that outlines an idea for freely annotating content on public screens from personal devices. It is important to note that this research is from 2004 and I personally think it would be very unfeasible to deploy this today. Moreover this scenario is not really relevant to the project's scope, but it can give interesting clues on cooperation techniques such as: handling more than one user in the same simulation; sharing annotated data back and forth; or similar modalities of data sharing.

Another research[6] that is very relevant to our intended final usage is not so recent, but in spite of being from 2002, it's still very relevant. It describes two dated pieces of software for 3D navigation and asynchronous annotation based in a web browser. Firstly developed thinking about architects, RedLiner and Space Pen both aim at the same thing but with different means. RedLiner only provides users with spatial comments that are described like post-its in the real world (indicated by a blue sphere inside the 3D model). Space Pen is, on the other hand, the evolution of RedLiner and it also allows 2D sketching in the model as well as other secondary features. This proposition of not only one way of annotation to the user, has apparently been received in a very positive manner by the testing subjects, and it confirms the hypotesis that a multimodal annotation system is generally prefered by the end user even if it may seem redundant or not necessary.

The next piece or research[7] focuses on annotating a scenario which is really close to this project's as it is set in an Augmented Reality experience. Despite of this, I personally don't think that the ideas introduced by this research could be useful in this specific project. The annotation method here described proposes the insertion of audio stickers in a 3D environment. Audio annotations are proven to be very useful as we will see in the findings of the following researches, however audio stickers behave in such a way that may distort the user experience in our software.

I would rather have audio annotations attached to specific points, just like audio stickers, but it's not necessary to have such annotations continuously play sound in a spatially aware manner. It would be easier and clearer to play those audio on user command.

Just as the previously discussed research, many other studies helped the development by giving an insight on some possibilities rather than a complete answer, just like this research about sketch-based annotation[8]. Again, this research does not propose a really applicable annotation method since it describes a way to generate sketches. This may be useful in other scenarios, but not in our case, since even if we decide to introduce sketching, free-hand drawn sketches are probably a more informative and intuitive approach. However the research also discusses a way of storing such annotations by using a second layer that stores the sketches (or annotations more generally). This is on the other hand very applicable to our project since we could actually think about storing annotations, whatever kind, on a separated layer instead of saving them directly on the model. The upside of using this technique is that if we have to implement annotation sharing between users that are working on the same model, we could share or merge annotations in a lighter fashion, without having to work on the model itself.

In another study from 2007[9], the usage of reinforced learning is proposed to automatically annotate 2D images by dividing the whole image into regions and annotating those regions singularly. This is clearly not applicable to this project but it glimpsed to the idea of annotating not a specific point, but rather a region, or an area. The stratification model that is used used in the proposed software shows a layered view of the Earth's ground, this technique could be implemented in order to indicate a portion of the surface of one of the layers.

Not only annotation methods are discussed in this research field. The following research[10] for example, does not propose an annotation method, but it gives clear evidence on which annotation method could provide more information when it comes to sharing annotations in a spatial system. Here we mainly look at the difference between spoken and written annotations: written annotations are generally prefered when users want to annotate or communicate something about a very specific thing that is non controversial (such as pointing out an error, or stating clear evidence); spoken annotations on the other hand are preferred by users when they want to argue something or propose an idea or opinion that could bloom into a discussion. Spoken annotations are also preferred when users want to annotate something very generally, that is not objectively clear.

On a similar note, this research[11] highlights the idea of allowing multiple annotation techniques for the users to use. Some of them were already seen (such as free-hand drawn sketches), others are innovative, yet not applicable to this specific project (such as 2D snapshots of 3D views to later annotate). Finally there is one idea that is not an asynchronous annotation method (all of the other methods are) but is instead a way of highlighting a certain portion of the model or scene with a tool that gives the effect of a flashlight. This of course is not a good way to annotate a model if we want to share such annotation, but since the software could

also be used for presentations, this could be useful to indicate particular areas of the model to people which are not inside of the VR environment but are just looking to a projection of what the user is seeing.

On the same topic of multimodal annotation, an article from 1999[12] raises many interesting points. However the most curious one is that building a multimodal system will not necessarily push users into multimodal interaction. Based on this research, when annotating something, apparently only 20% of inputs are expressed multimodally (using both speech and text) when this is an option. The remaining 80% of interactions remains unimodal, either spoken or written. However, it is also proven that most users prefer the option to interact multimodally when possible, especially in spatial domains, where 95 to 100% of users prefer to have this interaction option available to them.

The features proposed by this final text[13] could most likely be seen as optional yet interesting pieces of implementation to improve usability of our end software. The above research proposes the idea of simply exporting the annotations from inside the VR environment to outside of it and vice versa. This way a user could look back to the annotation made inside the experience without having to wear the headset and experiencing the environment again. This is useful as sometimes it is just unfeasible to do so, the headset may be momentarily unavailable for example. To import annotations on the other hand could be useful if users want to create a checklist of things to do inside of the VR environments. Being able to access such annotations during the experience itself, could save time and be useful overall.

## 2.2 Existing Software

Just like past research, there are some pieces of software that I belive could be useful to look into and analyze in order to obtain information about user controls to reproduce in this project. Unortunately, most of these softwares are not publicly available due to their nature (results of academic researches, unreleased prototypes, etc.), such as the end product of the studies by Sebastian Pick[14].

There is however one piece of software that is very well known and easily accessible, that is Google Tiltbrush[15]. Google Tiltbrush is a software that lets you paint in 3D space with virtual reality, transforming the Virtual room you are immersed in, in your canvas. Other that studying this application for its possibility to draw 3D sketches (wich is a good option for annotting data), we can also take inspiration from its intuitive user interface. Just as when painting, users only use their primary hand for drawing. Instead of leaving the off-hand unused and without a purpose, Google decided to include a sort of palette on the hand itself, so that the entirety of the user interface is included on it. Users can change mode, color, tools and even quit drawing just by pointing to their off-hand with their primary one. This is a solid design choice but it has a downside, it never leaves the user with both of his hands free to use.

A similar approach is used by IrisVR[16]. The annotation function of this software also allows the user to draw free-hand sketches but opposed to Tiltbrush, it is actually designed for annotating. Unfortunately it is not designed for specific data annotation. As a consequence, this application can only be used to load up a 3D model and draw free-hand sketches onto it, it doesn't allow for any other kind of annotation or manipulation. If we take a look to the UI, it looks just like the one from the Google application, this makes a lot of sense given the fact that once again, the user never needs to use both of his hands at the same time.

A different approach is the one taken by Oculus[17] itself (leader in the VR field) in their experience called First Steps[18], which is as the name suggests, an experience used to teach new users how to use their VR device. In this experience, users will be able to see a reproduction of their hands inside of the application even if the headset is not tracking the user's hands. This is done using the Oculus controllers and their set of buttons and triggers. Even if the reproduced hands can only mimic a small set of hand positions and gestures, this is more than enough to give the users a feel of reality and to immmerse them even more in the virtual space.


**Figure 2.1:** A screenshot from the Oculus' First Steps experience

A very interesting VR poject I discovered is called Glossy[19]. Glossy is only available for HTC Vive[20] but it's simple yet useful functionality allows users to place audio annotations anywhere in a loaded 3D model. This still has a lot of limitations: users cannot place more than one annotation in the same place, users cannot zoom or intetract with the model in any way, and finally users can only place annotations where their player is in the VR space. The strenght of Glossy is that it allows users to place or listen to annotations without interfeering with the exploration of the environment. Typing some text is unfortunately a very awkward action to do in VR, as it would require to invoke a virtual keyboard on wich to type, the longer the text is, the sooner the user will get tired of the system. On the other hand, speaking does not require the user to use his hands, which is the main way the user can give inputs

to the system. A similar concept can be used to argue what is the best way for users to retrieve annotated information, more about this will be explained in section 2.4.

## 2.3 Building blocks

It is a waste of time to reason about the best annotation method if it is not clear what data needs to be annotated, why and what tools can be used. I will quickly introduce the dataset provided for this project and give a backgroud idea about the work that it should be used for. I will also introduce the software and hardware used to bring this project to light.

### 2.3.1 The Volve Dataset

As much as we would like to deny it in a world that is realizing that ”Green” and ”Eco” are the keywords for a brighter future, oil is still our primary source of energy. Energy companies such as Equinor[1] still operate various platforms in order to extract oil from the ground using wells that are dug down into the E arth. According to the data provided by the U.S. Department of Energy[21], the average depth of an oil well is slightly less than 2 kilometers, this means that the drill has to dig through different ground layers, each one with different characteristics. Coring the ground allows researchers to have an idea of the shape and depth of these layers to then study them in order to decide where to place new wells to reach oil deposits.

In 2018 Equinor released their dataset about the Volve[2] field to the public. This dataset contained a wide range of information, but we will only focus on two things

- The ground Layers;
- The existing oil Wells.

Using this dataset we can build a 3-dimensional model of the decomissioned field to demonstrate how scientists could use Virtual Reality to explore and study such data, with the final intent of picking the best locations for new wells.

### 2.3.2 Used software and hardware

Virtual Reality had an enormous growth in the past 5 years, and it is expected to grow even more in the close future[22] as we can see in figure 2.2. In 2014[23] even the tech giant Facebook placed a 2 billion dollars bet in this field of the industry buying the company Oculus. As of today Oculus is producing and selling three different VR headsets which are hard to get your hands on given the number of requests. The Oculus Quest[24] is the midrange of the three. Given its standalone nature and good enough computing power, the Quest is the preferred device for many users. It is also the device that I was given for the completion of this project, so I had to take into consideration advantages and limitations of this device.

As for software used I mainly relied on three platforms:

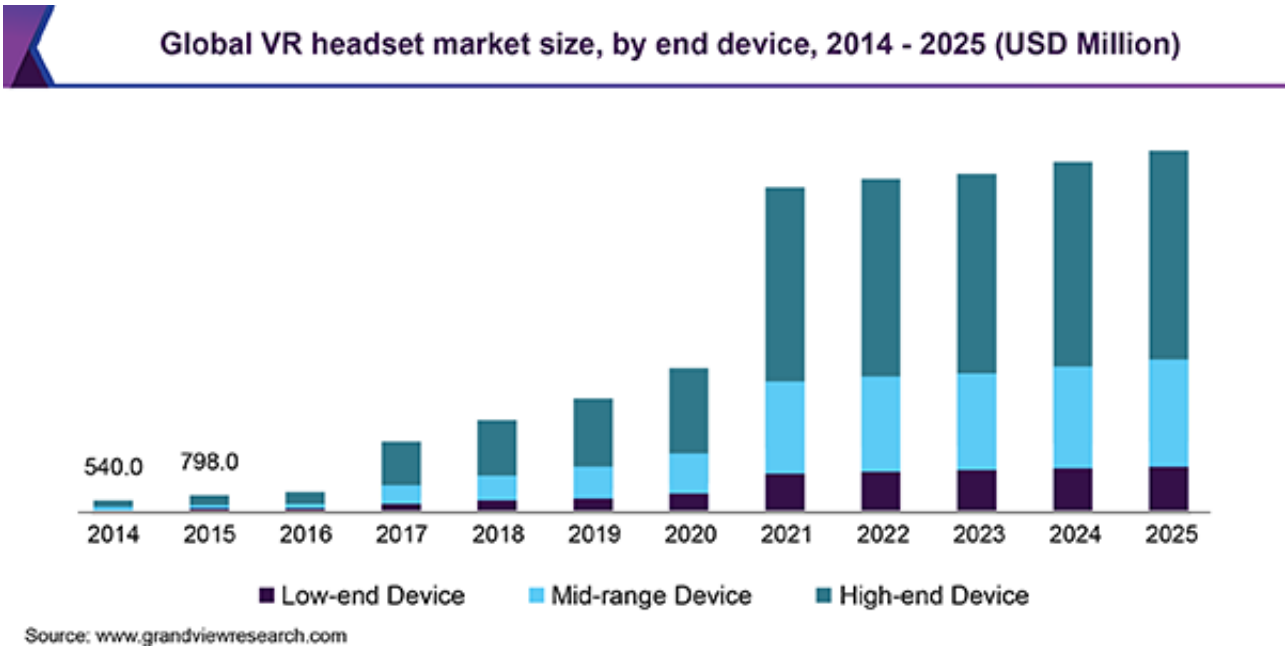


**Figure 2.2:** Expected growth of the VR market

- Paraview[25] is a data analysis and visualization application. I used ParaView to build a raw visual representation of the data contained in the dataset (more on this in section 3.1).
- Blender[26] is a well known 3D creation suite. I used blender to produce and combine higher quality 3D models used in the VR experience.
- Unity[27] is a real-time 3D development platform mainly used in the industry to develop videogames and other interactive digital entertainment applications.

  Unity uses C# as it's scripting language so to allow users to give the desired behaviour to every aspect of whatever they are trying to build. This was the software used the most during the development of the project, everything that can be seen inside the final product is part of the Unity project.

## 2.4 Design choices

### 2.4.1 Annotation method

Having both past researches and existing softwares in mind, I decided what the best annotation method for my specific application would be based on the 3 questions that I previously highlighted.

- **Who and how many people are accessing and annotating the same data?**

  The spatial data and annotations included in the experience will mostly be accessed by researchers belonging to the same company, most likely from an

office room. In some situations the software may also be used to show findings or decisions to people from different backgrounds or with different expertises. In these specific situations however, users should only navigate the data and do not need to annotate.

- **What kind of information will be annotated on the data?**

  Since the main users will be researchers from the same company, annotations will most likely contain non trivial information. This includes, but is not limited to opinions, doubts or uncertainties about some aspects of the data. It is then necessary to give users a way to express their thoughts without limiting them. Also, a way to answer previous annotations could be very useful to build a chain of opinions, just like in a forum thread.

- **How is the data being visualized from the users?**

  The data will be visualized in a VR experience. This means that the users will be potentially surrounded by the data, their vision shouldn't be impaired. This also means that users won't have access to any physical mean of input other than the headset and the controllers. The immersion part is never to be forgotten, the more the user feels the experience as natural or real, the more he will be immersed.

Now that we have a clear understanding of what we want to do, it is easier to pick a solution for our use case scenario. My final decision was to usa a single annotation method working with localized audio annotations, and here's why.

- **Single over multiple annotation methods**

  The most important idea, is that I wanted to ensure an experience that would be as easy as possible for users. VR is a new technology that many people have never encountered, if researchers were suddently be thrown in a VR scene, and find themselves surrounded by loads of options and icons while they don't even know how to move around, they would probably just want to go back to the software they were previously using. As a consequence I decided to give the whole application a minimalist look with a natural feel to it. I didn't want to provide users with secondary annotation systems that were not necessary. Especially knowing that users were probably going to get confidant with the primary system and exclusively use that one. That's why I think it is better just to concentrate on a single annotation method.

- **Localized annotations over Region annotations**

  Identifying a wide region and describing its characteristics may be a useful feature in some cases. Identifying a point and giving information about that specific point is, on the other hand, necessary. Going back to the previous point, I think that the process of identifying a region in VR is very complicated, and the advantages of placing a region annotation are not so relevant. Placing a localized annotation and discussing about the region surrounding that point

has the same result of a region annotation. Because of this, I decided only to include localized annotations.

- **Audio annotations over Text annotations**

  As noted above, users placing annotations will be experts of the field. Just like the experiment in one of the studies previously described[10], I came to the conclusion that because of the user base, most of the annotations will be part of discussions or non trivial facts. These genre of annotations are better to be annotated using voice rather than text. Also, placing a text annotation, even if short, would force the user to invoke a virtual keyboard and distract him from the environment. Placing an audio annotation on the contrary, does not require users to stop exploring the data since it allows full freedom for their hands. Similarly, reading or listening to an annotation are two very different processes. The first requires the user to use his eyes to extrapolate the annotated data, while his eyes should be focused on what the annotation is talking about instead. Listening only requires the user to use his earing to understand the annotation, so he can still use his eyes to focus on the data surrounding him while moving around. Moreover, the Oculus Quest is a device containing both a microphone and a speaker, so there is no need to use any external devices to record or listen to annotations.

- **Answering to an annotation**

  Users will most likely have discussions about ideas or opinions. Because of this I think it's very important to give users the possibility to answer to an annotation. I belive the best way of doing this, is to allow multiple annotations in the same location, these annotations will be displayed to the user in chronological order. If a user wants to access an annotation in a specific point, he may find himself accessing a thread of annotations instead, he can then listen to all of the annotations in that place, or only listen to the ones he's interested in. To achieve this, every new annotation placed in a never annotated point, will generate a collection of annotations that can be accessed to and visualized from other users.

- **Where to place annotations**

  Users should ideally have the possibility to place annotations wherever they want in the world space, but will most likely want to comment on some ground's layer characteristic, or some existing object (like an old well in this case). Since the experience is supposed to be used to allow the user to be surrounded by data, it is also important that users have the ability to not only annotate something on their current location, but also something further away from them. As a conclusion I decided to allow users to annotate any point on a layer or on an existing object that they have visual access to. If users want to annotate a point in space however, they have to be in that exact point.

### 2.4.2 Virtual Reality UI

User interface design is very important since a clear and easy interface is key for a smooth user experience. In VR, this concept gets exagerated since UI gets a whole new level of importance, the more minimal the interface is, the more immersion in the experience the user is going to feel. In order to keep the experience as natural as possible, I decided not to include any menus or user interface inside the application. A quick read of the user manual should be more than enough to understand how to get the best out of the scene. When inside of the Virtual world, users will be able to see a representation of their hands, and with those hands they should be able to access all the required functionalities. Users should have access to 3 abilities mainly: moving around in the space of the experience, manipulating the model for better visualization of the data, having access to annotations. Moving the user around the data and moving the camera in order to focus on different data is the most important feature. It is the one that allows the user to properly visualize the data. This function should be always accessible. I decided to distribute the remaining functionalities over 3 different modalities in order to reduce to zero the presence of a traditional user interface.

- **Layers mode**

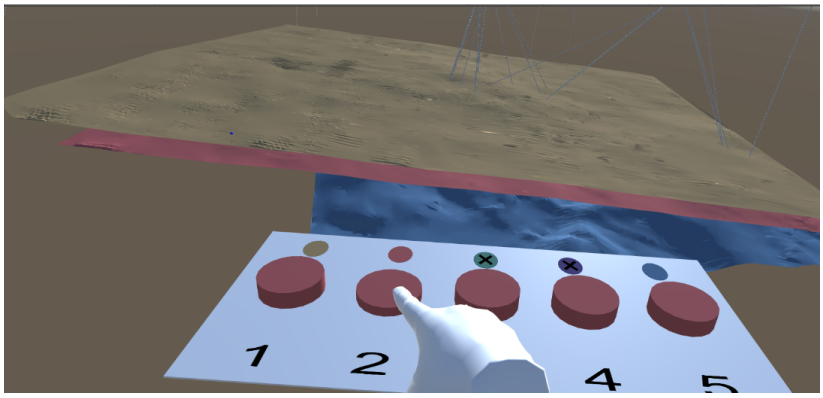


**Figure 2.3:** A Screenshot of the final software in Layers Mode

  As explained earlier, the model in question has a number of layers, users may want to focus on one layer or on multiple ones depending on the needs of the session. In this modality the user should have the possibility to select which layers to see and which not to. In order to exclude UI selections, I decided to place a Virtual desk inside the scene. Users can use this to select or deselect layers thanks to the use of virtual buttons (more on this in section 3.6).

- **Model mode**

  After the selection of the necessary layers, the user should switch to the following mode, where he is given the possibility to move and scale the 3D model. This way he can be surrounded by the data and reach the desired level of immersion. Depending on the purpose of the session, the user may want to view the model from far away, or possibly dive into the data so to be able to

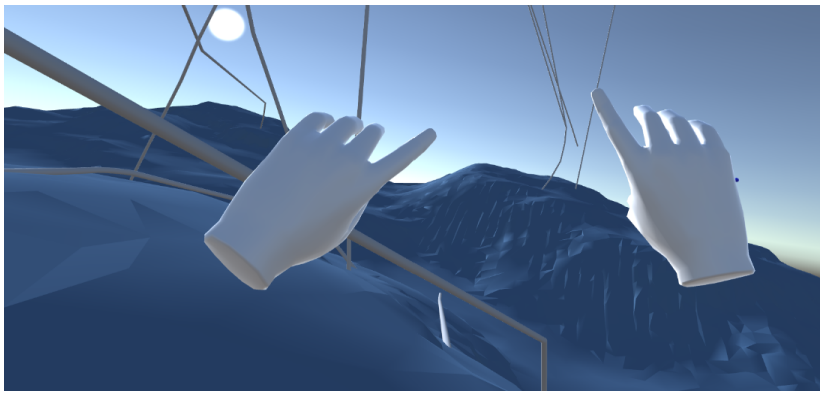

**Figure 2.4:** A Screenshot of the final software in Model Mode

see small details about the ground's shape. No matter the objective, in this mode the user will be able to use some simple gestures to move and stretch the model. Using gestures allows us once again the remove completely the traditional User Interface and give a natural feel to the experience (more on this in section 3.5).

- **Annotation mode**

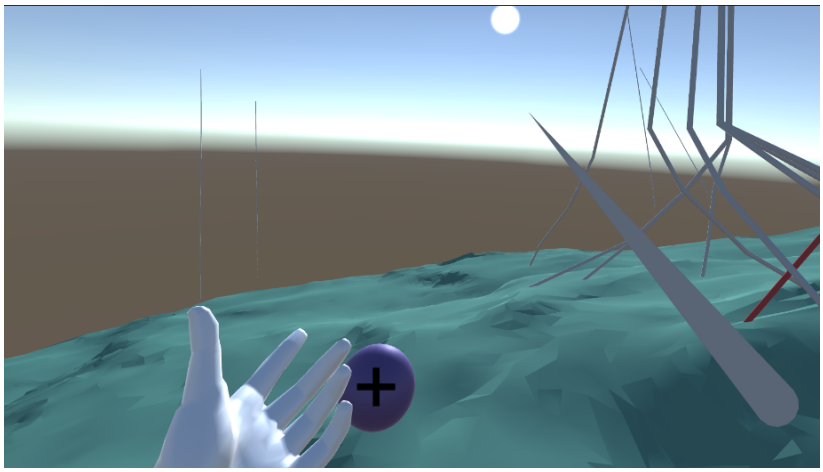

**Figure 2.5:** A Screenshot of the final software in Annotation Mode

Finally, the user likely wants to annotate something he noted about the data. He switches to the third modality to see one of his hands to be replaced with a virtual pointer, and a floating sphere in front of him. All of the annotations in the scene are represented by a sphere. Users can use the pointer to select a sphere and start listening to annotations. With the very same pointer they can also click on the floating sphere to start recording a new annotation. After the recording finishes, they can select where to place the annotation by using the pointer (more on this in section 3.2).

### 2.4.3 Other design choices

Since the software is probably going to be used inside of an office, the whole application is designed to be experienced from a sitted position. Users do not need to move around since office space is usually not very abundant, but they are required to be sitting on a swivel chair in order to get the best out of the experience.

To have a smoother transition to the virtual world, I decided to have users placed in an office like room when opening the application. Here they will find themselves sitting on a swivel chair, just as they are in the real world. This should give users a warm welcome in the experience and act as the main menu. I will refer to this as the Welcome Scene 2.6. In the same room they will see some tables. On top of every

**Figure 2.6:** Two Screenshots of the final software inside of the Welcome Scene

table is a model wich is a preview of the accessable data. This can be visualized by clicking a button in front of the table. This is also to show that the experience is not limited to a single dataset. Researchers in a company will most likely work on different projects, probably even at the same time. Because of this I decided to shape the experience in such a way that multiple datasets could be visualized and annotated, without having to access a completely different experience.

## 2.5 Ethical considerations

Despite all of the entries in the ethics checklist has been answerd as ”no”, I belive it is correct to address some points about the environment. Enviromental consciousness has been a recurring topic in the past years, a branch of this topic focuses on finding reliable sources of renewable energy in spite of more traditional ones, given their impact on the environment. This software can be used to study the field to dig new wells for oil extraction, but it doesn’t in any way promote the use of oil as a source

of energy. The dataset used in the project contains information of a decomissioned dataset and will not be used for installing new wells. The purpose of the project is not to promote the use of this software for this specific scenario, but rather to demonstrate its utility in all sorts of fields.

# Chapter 3

# Contribution

The contribution of this project consists of the following:

- the main contribuition is a complete VR experience that allows users to visualize and annotate spatial data of the Volve field;
- a secondary contribuition is a partial interactable experience that allows users to visualize and annotate spatial data of the Volve field from a Computer.

The secondary contibuition has similar functionalities when compared to the main one. It allows a demonstration of how an immersive experience in Virtual Reality can improve spatial data visualization. Many different parts of the project have been reused in the two pieces of contribuition. In this chapter I will firstly run through the development of components used in both of the experiences. Then I will conclude on the ones needed for the intended functionality of the VR version. Finally I will quickly run through some details about the Computer based version.

## 3.1 Rendering the dataset

After downloading the open source datset [28] of the Volve field, I localized the necessary files (ground layers and wells locations). These file had the .dat file extension. After converting them in .csv format and getting rid of all the unnecessary fields, the files looked like 3.1. As it's easy to guess from the figure, the three fields correspond to the coordinates of some points in space. Using Paraview I could convert these coordinates to a graphical representation. Then, using the Delunay[29] algorithm I generated a representation of the ground's layer. Unfortunately the generated 3D model was centered on the origin of the coordinates (the point 0,0,0). The coordinates system used in soil science uses as origin a point that is far away from the described layer. Because of this I decided to calculate the mean point for every axis, taking into consideration all of the layers, and then recalculate the coordinates using the mean point as the new origin. Once the data about the layers was normilized, I could then generate the centered 3D models for each of the 5 layers.

Importing this models to Unity it was easy to see that the dimentions were very big. I then opened the models with Blender to scale them down enough to make them

| | A | B | C |
|---|---|---|---|
| 1 | x | y | z |
| 2 | 431128.4 | 6477383 | 3252.197 |
| 3 | 431131.5 | 6477395 | 3251.346 |
| 4 | 431134.5 | 6477407 | 3252.683 |
| 5 | 431137.5 | 6477420 | 3253.441 |
| 6 | 431140.5 | 6477432 | 3253.322 |
| 7 | 431143.6 | 6477444 | 3251.137 |
| 8 | 431146.6 | 6477456 | 3248.951 |
| 9 | 431149.6 | 6477468 | 3247.163 |
| 10 | 431152.6 | 6477480 | 3245.323 |
| 11 | 431155.7 | 6477492 | 3242.853 |
| 12 | 431158.7 | 6477505 | 3237.087 |
| 13 | 431161.7 | 6477517 | 3234.281 |
| 14 | 431164.7 | 6477529 | 3231.94 |
| 15 | 431167.8 | 6477541 | 3230.942 |
| 16 | 431170.8 | 6477553 | 3230.433 |
| 17 | 431173.8 | 6477565 | 3228.846 |

**Figure 3.1:** Example of layer spatial data

entirely visible once inserted in a scene in Unity. Then I could see another problem. Running a scene containg only one layer in Unity would allow me to view said model from top to bottom, but it would show as transparent when looking at it from below. After researching about the problem, I found out that the reason for this was because of the auto generated normals[30] of the model. To fix this issue I used Blender to

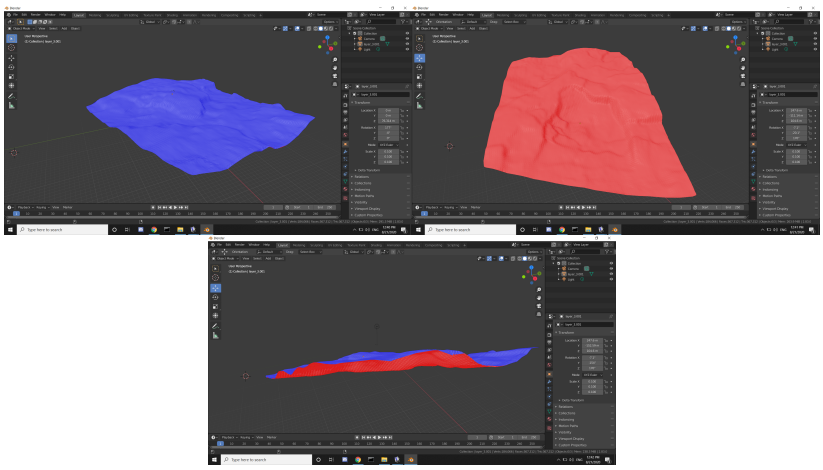

**Figure 3.2:** Examples of layer model's normals visualized in Blender.

duplicate each layer, flip the normals and merge the original layers with their flipped normals copies. Figure 3.2 shows how normals are visualized in Blender, polygons highligthed in blue will be visible once exported, red ones will not. Now that the layers were ready to use, a similar process had to be applied to the oil wells.

Once again I converted the data from .dat to .csv; normalized the coordinates, so that the models could be placed in the correct locations on the layers; and used Paraview to visualize the points included in the files. In order to generate a proper 3D model, I connected the points in space and made the edges look tubular. Once the 3D models were exported and placed in a Unity scene together with the layers, the model was ready to be used.

Individual layers, just as individual wells, were imported to Unity as single models. This way I could control the behaviour of every object individually rather than as a collection. Every layer and well has a mesh rendered attached to it inside of Unity. I used the mesh renderers to control the material and color of the objects as well as to decide whether to hide or display each layer.

## 3.2 Annotation system

### 3.2.1 The final result

In both the VR and Computer based experiences, what the user sees is controlled with a camera. While the camera is moved in two very different ways in the two experiences (moving the player or the headset in the first, and through the keyboard in the second), some objects are anchored to the camera in both of them. One of this objects is the audiobox. The audiobox is an invisible container used to display audio spheres, its content can be moved left or right so that the view of the user is never impaired, this is shown in Figure 3.3. The most important audio sphere that is represented in the audiobox is used to record new annotations. The user can click on the sphere to start or stop a recording. Once a new annotation has been recorded the user can click on a location to place the annotation. In VR, clicking is done through the use of a virtual pointer. If the location does not contain any other annotations, a

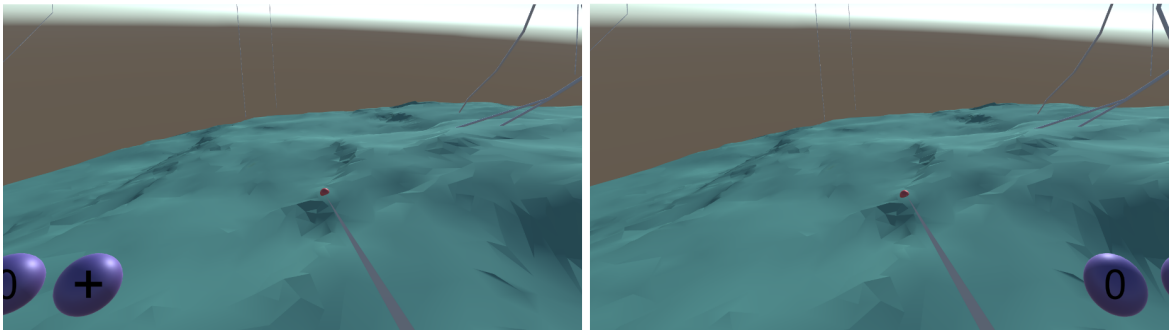


**Figure 3.3:** A screenshot from the final software showing how annotations are displayed to the user.

new collection will be created in that location, and the newly recorded audio will be added to that collection as the first audio. Alternatively, if when placing the audio the user selects an already existing collection, the new audio will simply be appended to the list of audios belonging to the collection. Collections are displayed on the model as spheres, if a user navigating the model selects a collection, it will change color

to display it is currently selected. Wells behave just like collections from this point of view, a user can select one and it will change color to display it is selected. Only one collection or well can be selected at any one time, selecting a different one will deselect the old one. When a collection or well is selected, the audiobox will display the contents of the list storing all of the annotations of that location. Next to the audios belonging to the selected object, an audio sphere that can be used to create new annotations will be displayed, if the user clicks on that sphere to record a new audio, the new annotation will be directly added to the currently selected location.

### 3.2.2 How recording works

When the user starts recording, the application will identify the primary microphone used by the system and start a Coroutine (Asyncronous function) that keeps the microphone open until the recording is stopped or a predefined time has elapsed. If when a recording is about to begin, the application does not have access to the microphone permissions, it will by default prompt the user a request to allow those permissions. To to avoid this and avoid disruptions, permissions will be asked as soon as the application is accessed the first time.

### 3.2.3 Saving data in between sessions

Without annotations being saved in between sessions, it would be useless to place annotations in the first place. To achieve this, the system needs to do two things:

- save the actual audio clip;
- remember which well or audio collection the annotation belongs to, and its index.

I decided to use two different kinds of storages to provide this feature: permanent storage and Player Preferences. Permanent storage is used to save resources or files and can be used to both write and read data from (the default resource location can be accessed as read only during runtime, so it is not suitable for our scenario). Player Preferences is a light and fast key value storage used to save informations in between sessions.

Permanent storage was used to save the audio files. In order to do this, I used an open source library named SavWav[31] that allows to save audio clips in .wav format. When audio annotations have to be retrieved, since their location is not in the resources folder, which would be easily accessible, I use asynchronous web requests specifiyng the local path of the clip as the target. This path contains the filename of the audio that is specified as it's being saved. Every annotation's filename is composed of three parts in order to form a unique name:

- an ID specifying to which model the audio belongs to;
- a type identifier (either **CollectionAudio** or **WellAudio**) to identify what kind of location the annotation belongs to;

- a numerical ID.

These three components together form a unique name accross different locations or even models.

We use the type identifier to know if an audio belongs to a Collection or to a Well, but we get no information from the audio itself about which specific Collection or Well it belongs to. In a similar way, we also don't know in which order audios belonging to the same location should be displayed. To solve this issue, I used Player Preferences. Every time a new audio is created, the Player Preferences file is updated to store the order and the IDs of audios belonging to every Collection or Well. Every Well or Collection has a unique ID, for every ID there is an entry in Player Preferences storing the list of audioclips IDs belonging to said Well or Collection. Eg. Well represented with ID = 8 will have a key value pair in player preferences with key = "Well_8" and Value "12/32/36/" so we know that the well with ID 8 will have 3 audios named "WellAudio_12", "WellAudio_32" and "WellAudio_36" in this order.

### 3.2.4 Loading annotations

At the beginning of the experience, all annotations are loaded at launch time with asynchronous requests. Firstly Player Preferences is queried and then web requests are generated to locate the audios and assigning them to the objects in the scene. While Wells are part of the model and are always present in the scene, even if no annotations are assigned to them, audio Collections are creates by the user and are not part of the model when the experience begins. This means that in order to recreate the proper scene when the experience is loaded, it is necessary to generate the audio Collections during runtime. For this we use a Prefab of a Collection, with the correct behavioural script attached to it, and we keep a reference of this in the same class that loads the audioclips. A field in Player Preference stores the number of audio Collections present in the scene, for every Collection, the AudioLoader generates a copy of the Prefab in the scene and assigns it the correct ID. Subsequently the Collections are populated with annotations together with the Wells. But where to place these Collections?

In section 3.2.1 we touched on how when a user places a new annotation in the scene, if the selected location in empty, a new Collection is created. It's in that moment that the Player Preferences are updated to increment the number of Collections in the scene. Also a new field is created, it has the ID of the new audio Collection as the key, and its position in the 3D space relative to the center of the model as the value. Since the user is able to move and stretch the model, we have to do some adjustments to the position before storing it. The solution I implemented constists of some simple steps:

- firstly, I istantly generate the Collection in the scene;
- then I register the Collection as a child of the model (in this way every alteration to the Transform component of the model (position, scale or rotation) will also be applied to the Collection);

- the model's Transform is then restored to its original values, that is the position, location and scale of the model at the beginning of the experience;
- only at this point the local Position of the Collection is read and saved in the Player Preferences;
- finally the model's Transform is changed back to the values it had at the beginning of this process.

Because this process happens in a single function call, the model is never rendered with a different Transform, and the user experience is unaffected.

### 3.2.5 Removing annotations

There are many reasons for which users should be able to remove annotatios, so it was obligatory to include this function. Once a Collection or Well is selected, a user can select one of the spheres in the audiobox to remove the desired annotation. This could create a situation in which a Collection has no annotations in it. In this case the Collection itself will be removed both from the Scene and from the Player Preferences, so that it won't spawn during future sessions. If this is not the case, and an annotation is removed from either a Well or a Collection, Player Preferences are still updated. This is to ensure that the location the audio was in, won't recognize it as an annotation belonging to that location anymore. Also, the audio itself gets removed from the Permanent Storage so that the memory of the system won't get clogged with useless files.

## 3.3 The Oculus Library

From this section until otherwise specified, only features belonging exclusively to the VR software will be discussed.
Before continuing, it is necessary to open a small pharentesis about the Oculus Library[32] for VR development in Unity. This Asset contains multiple classes, scripts, prefabs, 3D models and Scenes that are used to ease developers' way into VR development with Unity. A very important part of this library is the OVRInput class that allows to easily read inputs from the Oculus controllers' buttons. The OVRGrabber and OVRGrabbable scripts are used for allowing players to grab and move objects inside the scene, in the project I used modified versions of these classes so I could adapt them to my needs. Also the OVRPlayerController Prefab has been modified and used so that it would be perfect for achieving the desired behaviour.

## 3.4 The Welcome Scene

As user boot up the experience they will find themselves sitting on a chair in a small room. Here they can move around and decide which model's data to visualize. Every different dataset will have a dedicated table with a small reproduction of the model

floating on top of it, users can grab this model but it will always return back to it's table once released. Every table has a button and a label with information of the dataset, the button uses a system composed of a spring and an anchor, together with its behavioural script, to simulate a real world button. Users can use their index finger or their fist in the VR scene to push the button, this will return to the original position if it's not pressed all the way. An example of buttons in the Welcome Scene can be seen in figure 3.4. The behavioural script will stop it from moving


**Figure 3.4:** A screenshot from the final software showing buttons interaction in the Welcome Scene.

past the intended depth and from moving in the opposite direction. The spring component will ensure a smooth and natural movement to the original position. Some constraints fixed in the Rigidbody component of the object will stop the button from rotating or moving in other directions. Once a button is pressed all the way, the scene containing the dataset will be loaded and the user will leave the Welcome scene. Users can always return to the Welcome scene from any other scene.

## 3.5 Moving the Model

### 3.5.1 The final result

As the data visualization scene loads up, the camera should point right to the model, that will be displayed in the user's view in its entirety. From here, users will be able to move their hands towards the model, grab it and move it around. This scene will be entirely unaffected by gravity and users won't be able to throw objects, they will remain steady where they are released and won't float in the air. The model is the only grabbable object present in the scene, so I decided to make users able to grab the object from a distance as well. Users are able to grab the model with a single hand using a gesture that is really natural (just closing the hand grip). Finally users can scale the model up or down. Once again this can be achieved using a natural gesture that requires both hands (pinching thumb and fingers with both hands and moving hands apart to enlare, or closer to reduce the size). These gestures can be seen in figure 3.5. The two gestures can also be used at the same time. Users are free to move the model as they wish, but the experience was designed to have the

users zoom in the model so much that they should find themselves surrounded by the data, floating inside of it.

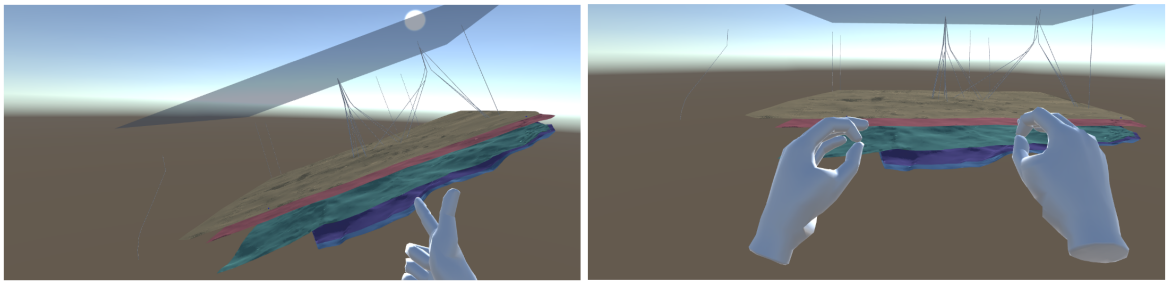

**Figure 3.5:** A Screenshot of the final product showing gestures for grabbing (left) and scaling (right).

### 3.5.2 How it works

The user in the Scene is represented by a Player Controller object. Just like a person in the real world uses his eyes to watch his surroundings, the camera is placed inside of the player, where the head should be. Rotating the head in the real world directly translates to rotating the camera just as in all of the VR experiences. This gives the most immersion in the scene. The other factor that allows users to dive deep into the Virtual world is the ability to see their hands represented in the scene. Thanks to the Touch controllers of the Oculus Quest, and the 3D hand models contained in the Oculus library, I managed to reproduce the behaviour used in most other VR experiences, where the 3D models in the Scene mimic as much as possible the hand movements of the user in the real world. Going back to the Player Controller object, also the hands are part of it. Each hand uses a different scripts for controlling each of the two different gestures used in the Scene: Grabbing and Scaling. Each of the two hands also has two Collider components, one for the fingers (that are subject to flexing) and one for the palm and wrist (which can only rotate). The two behavioural scripts are similar. In order to work they both require the user to have at least one of the hand's colliders overlapped with the collider of the object the user wants to interact with. Then, the correct input gesture will trigger the behaviour. Since the model is the only grabbable and scalable object in the scene, I decided to increase the size of the model's collider so that the gestures would work everywhere in the Scene and not only on the model.

### 3.5.3 Decimating the model

After testing the model's movement in VR, I noticed that the framerate of the scene was very unstable and extremely low. This problem would not present itself when using the computer's hardware instead of the Quest headset. The reason for this is in the different computing power. The cause of the problem was the size of the model and the precision used to generate the polygons that composed the layers. When the entire model was inside the camera view, too many triangles were being

rendered and the headset was stuggling. This problem would obiously not present itself when only a small part of the model was in the camera view. Maintaing the definition of the layers was of vital importance, analyzing the details of the model is the whole purpose of the experience. To solve the issue I decided to generate 3D models of the layers with less details and only use them when the scale of the model was below a certain threshold. This way the model only has the original definition when the user scales it enough to actually see the difference. In those situations, it is impossible to have the whole model included in the camera view, so the original models could be used and the framerate would be stable. To reduce the definition of the meshes without losing too many details, I used Blender once again and applied a Decimation filter to the originals. I also included a trigger in the behavioural script of the model, when its scale changes over or below the threshold, the meshes rendered by the layers would change accordingly. The difference can be seen in figure 3.6.

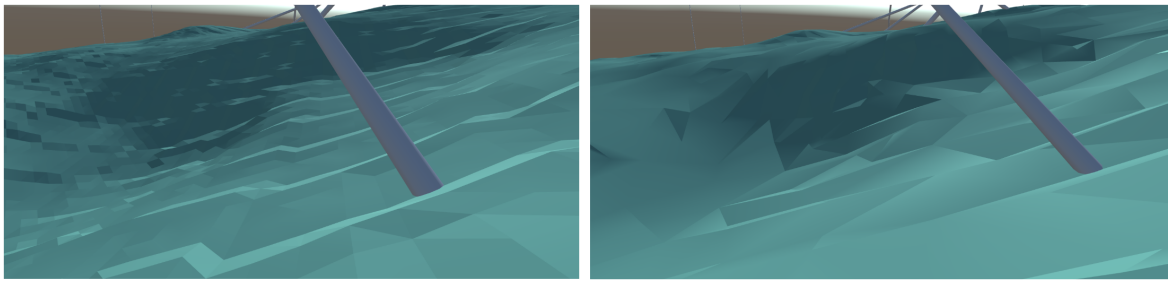

**Figure 3.6:** Comparison between original (left) and decimated (right) definition layers

## 3.6 A different User Interface

Recalling the focal intent to maintain the Virtual experience as immersive as possible, I decided not to include a traditional user interface. However, the amount of features to be included in the experience requires either a sort of Interface or a complex use of the controllers. To keep the application easy to interact with, I decided to provide users with some sort of interface that would not disrupt the immersion and yet, provide all of the features. An example of this is the audiobox and audiospheres described earlier. While the audiobox is only present in the Annotation Mode, this other piece of interactable interface is only present in Layers Mode. To select which layer to display or hide, the user will have a desk in front of him with one button per layer (as previously seen in figure 2.3), these buttons work in a similar way to the ones earlier described, but as they are pressed, they will enable or disable the Mesh Renderer of the layers.

Both the audiobox and the table are in a stable position in the view of the player, but their position moves in the world space. In the audiobox's case, its position is simply anchored to the camera, no matter how the player moves or rotates the camera, the audiobox will always appear in the same spot in the user's view (althought it can be moved if required). In the other case however, to maintain the illusion of the reality in the Virtual scene, the table should always be placed at the same height and with the same local rotation, but it should still follow the player's movement

and horizontal camera rotations. In this case I used a behavioural script to give the table's Transform component the desired behaviour.

## 3.7 Handling Colliders

Colliders are used throughout the project for a variety of reasons. The player's hands and body, the pointer, the individual layers, the wells, audio collections and audio spheres, buttons and tables all have collider components. One of the main difficulties in this project was finding a way to efficiently handle colliders. In some occasions we may want some colliders only to register collisions with other specific colliders. To give an example why this was necessary, I will give a simple scenario. In the Welcome scene the player should be able to grab the models, hence the model and the player's hands neet to have colliders. At the same time the player should be able to move, but not to move through the tables, hence both the player's body and tables need to have colliders. When grabbing the model and moving it towards the player, the collider of the player's body would be pushed away by the collider of the model that is moved towards it. This is definitely not the intended outcome. To solve the issues I relied on the combination of two solutions:

- The creation of a new layer to only interact with the Pointer (more on this in section 3.8).

  Many different colliders including Wells, audio Collections and audio spheres, only need to collide with the pointer in order to be selected while the user is in Annotation Mode. Unity gives the possibility to use a Layer system to handle objects interaction. Objects belonging to one Layer only interact with objects belonging to a specific set of Layers. I created a new Layer and assigned it all of the objects that were supposed to only collide with the Virtual Pointer, while leaving every other object in the Default Layer. I then set the new Layer to only collide with the Pointer. This way those objects would correctly collide with the Pointer, but would ignore all other collisions.

- Using Unuity's Physics system to ignore collisions

  In other cases, creating a Layer wasn't enough. Some collisions needed to be active at times and inactive at other times. Unity's Physics system provides the possibility to ignore collisions between two specific colliders. I used this function to ignore the remaining unnecessary collisions such as the ones between the buttons and the model or the player's body.

## 3.8 Virtual pointer

During Annotation Mode the user should be able to point to objects in the scene, as described earlier. The Pointer created for this reason replaces one of the user's hand in this mode. It can be naturally controlled by users just as a person would control a stick or a laser pointer in real life. To make the Pointer interact with the

scene's objects I first had to identify what object the Pointer was pointing to at any given time in the scene. This was done by using a Unity's Raycast spawning from the player's hand towards **Vector3.Forward**, and checking which collider was hit by it, if any. Then, to render the Pointer, I used a Line renderer that would create a tube from the hand to the point hit. If no point was it, the tube would just be rendered for a fixed lenght in the **Vector3.Forward** direction. Finally I had to override the Event System in the scene and give the pointer as a new input to easily handle collision events. This way I could assign to every desired object an event script that specifies its behaviour on collision with the Pointer.
To give a clear visual confirmation of what the user is pointing to, all of the selectable objects in the scene change color when the pointer is hovering on them. Most of the functionalities are triggered when the user pulls the controller's trigger while pointing to an object. When Wells or Collections are selected, they notify the corresponding parent object to update the system to show that a Location is currently selected. Audio spheres where instead designed to start or stop recording. Each object has its own different onClick function.

## 3.9 Mouse clicks as inputs

Now, I will discuss features belonging to the Computer based experience.
Features that are working thanks to the pointer in the VR experience, will work in a very similar way in the Computer based one, but instead of having a virtual pointer, users will just use the mouse one. This means that the Event System needs to listen to mousclicks as inputs. This is much more simple and it is already integrated in Unity's default Event System. To detect collisions with objects in the scene, I once again used Raycast, generating a Ray from the center of the user's view (Main Camera object), in the direction of the mouse pointer. The first Collider hit in by the Ray will be the point clicked by the user. At that point the mouse click would just trigger the object's behaviour, just as in the Pointer's case.

## 3.10 Camera and Model movement

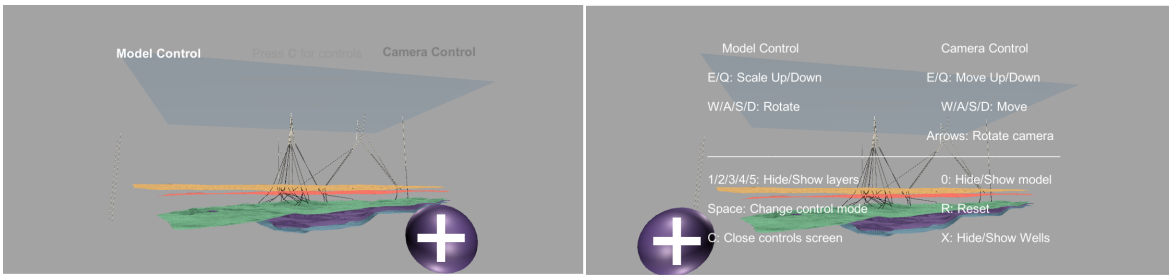


**Figure 3.7:** Two Screenshots of the Computer based version of the software

Given the amount of possible inputs in the Computer based version of the experience (thanks to the mouse and keyboard), I decided to remove the Modality controller

and merge all of the features in a single mode. For this software I had to change all of the commands for the player and model movement. First of all, there was no need to have a player and a camera object. Only using a camera would be more than fine to achieve an acceptable result. But I still had to handle movement and rotation on 3 different axis for the camera, and movement, rotation and scaling for the model. Since I wanted users to be always allowed to annotate (as the modes were removed), I reserved the mouse inputs for annotations. As a consequence, all the movements of Camera and Model had to be handled through keyboard inputs. Since too many commands had to be handled, I finally decided to give to the users the choice to either control the Camera or the Model. Users can switch in between the two with a simple key press.

# Chapter 4

# Project structure

A clean and simple project structure allows for a clearer readability, as well as easier reusability for future work. In this chapter I will give a quick overlook to how the Unity project was organized. I will also glimpse to how the scripts were written and the behaviours coded. Finally I will outline how the project's workload was distributed over the course of the past months, and what developing approaches were employed.

## 4.1 Scenes structure

The two experiences (Computer and VR), were developed as two different Unity projects. The Computer one had to be compiled as an executable file for Windows, the VR one had to be compiled to an .apk since the Operating System of the Oculus Quest in Android. In the following images I tried to extend the objects as much as possible. In some cases however, it would have either been redundant or too much space would have been occupied. In those cases I will explain the reason and briefly run through the hidden structure if necessary. This time I'll discuss the Computer version first.

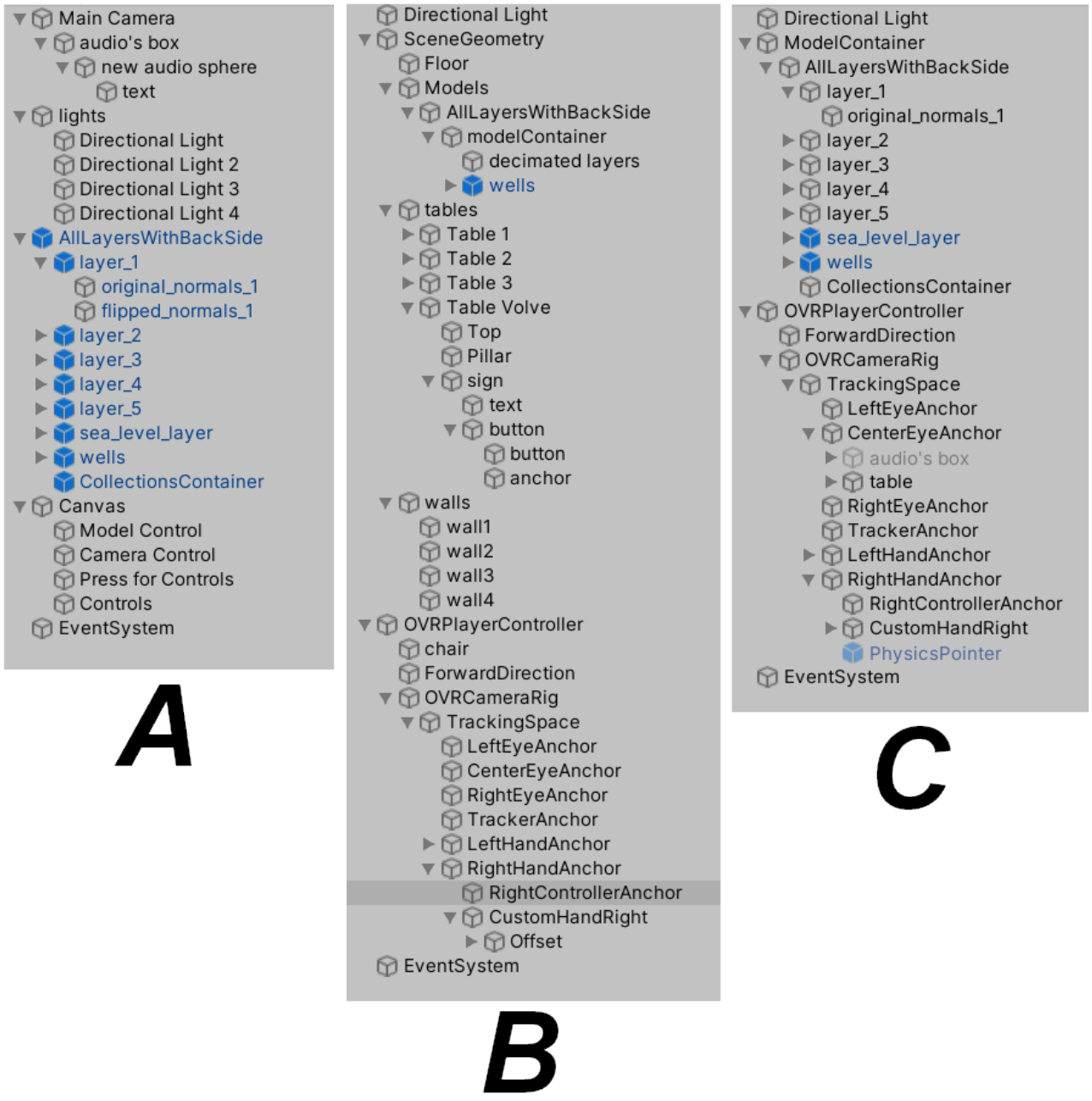


**Figure 4.1:** Scene's structure in Unity: A-MainScene; B-WelcomeScene; C-VolveScene

### 4.1.1 Main Scene

The Computer based experience is not as complete as the principal VR based one. There is no Welcome scene. This is because the aim of this application was mainly to show the difference in usability during the data visualization scene. That scene is the only one present in this Unity project. Figure 4.1-A shows how the scene hierarchy is structured in Unity's inspector. Reading from top to bottom we have the following objects:

- **Main Camera:** the Main Camera is mainly used to change the user's field

of view. The **audiobox** is directly attached to the camera so that it always remains in a fixed position in the user's view. Inside the audiobox there is the **new audio sphere**, this is always present inside of the audiobox and is used for creating new annotations. The child **text** is only used to display text on the sphere.

- **Lights:** the lighting system in this experience uses 4 different directional light to produce a homogenius lighting. This is not necessary in the VR experience since some situations are unlikely to happen (eg. the camera is moved to the top of the scene and is looking down towards the model, in VR the user will almost always look along the x axis since it is a more natural position).

- **AllLayersWithBackSide:** the blue color of this object in the hierarchy indicates that this is a Prefab. The reason behind the name is in fact to differenciate it from another Prefab present in the project, that one only contains layers meshes with mono directional normals (only observable from top to bottom). This object is used to move and scale the model. It has one children per layer, each one containing one mesh per each normals variation. Moreover, an additional layer, **sea_level_layer** is a mesh I created to include the sea level in the model. The **Wells** object has 33 childrens, one for each well. Every one of them is used to render a single well. The last child, the **Collection Container** is populated with audio Collections when the scene is loaded, and it's used to control the behaviour or Collections.

- **Canvas:** the Canvas contains all the components of the user interface. In the Computer version there is no need to maintain a sense of immersion. The software uses a simple User Interface to display basic information. The same interface is also used to show details about the controls of the experience. **Model** and **Camera Control** are the two text fields used to communicate to the user if he's currenlty controlling either the Camera or the Model. This is done by changing text color. **Press for Controls** and **Controls** are instead used to show how to control the model and the camera in the Scene.

- **EventSystem:** this is used to handle user inputs, it is contained in every Unity Scene by default.

### 4.1.2 Welcome Scene

The VR based experience provides the user with a much more complete experience. The Welcome Scene earlier described is the first scene that is loaded when the user launches the application. Figure 4.1-B shows how the scene hierarchy is structured in Unity's inspector. Reading from top to bottom we have the following objects:

- **DirectionalLight:** this scene only uses a single directional light to simulate a natural light source.

- **SceneGeometry:** inside this object are included all the models that compose the physical structure of the scene. The **Floor** uses a realistic material to

trasport the user into the experience. **Models** contains all of the small model reproductions floating on the tables. In the **wells** collection of the Volve model, some wells have been removed to have a more contained model.**Tables** contains all of the tables in the scene, each table has a single leg, the top part, a **sign** displaying information about the dataset and a **button**. The **button** is composed of the button itself, and the anchor of the spring component. Finally, inside of **walls** are the 4 walls of the scene.

- **OVRPlayerController:** probably the most interesting object in the scene. This is what holds the main camera (which is a component residing inside of the **CenterEyeAnchor**) and the objects used as the two hands: **CustomHandRight** and **CustomHandLeft**. Inside of the Custom hands' **Offset** are the components that construct the model of the hand, each finger, joint, or movable part is condidered as a single component so to achieve every possible hand position in the scene. Moreover the **OVRPlayerController** contains the **chair** object, which has no fuctionality except being visible to the player and enhancing immersion.

- **EventSystem:** is the default Event System of Unity.

### 4.1.3 Volve Scene

This scene gets loaded when the user clicks the button of the table with the Volve model on top of it. It is the scene where users can navigate the data and annotate it. Figure 4.1-C shows how the scene hierarchy is structured in Unity's inspector. Reading from top to bottom we have the following objects:

- **Directional Light:** just like in the Welcome Scene, a single directional light correctly emulates a real world light.

- **ModelContainer:** this object contains all the necessary objects to handle the model, similarly to the Main Scene. The only differences are: an additional container that is used for handling movement of the model in an easier way; and the fact that a single mesh is used for every layer instead of two. This is because in order to have the decimated (lower quality) meshes be replaced by the normal ones when the scale of the model changes, it is more efficient to just have a single one changed instead of two. I then improved the meshes by merging the two into a single one.

- **OVRPlayerController:** similarly to the Welcome Scene, main Camera and hands are included in this object. But in this scene, the **CenterEyeAnchor** also has the **audiobox** and the **table** inside of it, to follow the user's view. The **table** is used in Layer Mode to control the visibility of the individual layers. It is composed of a plane and five buttons. Each button also has an anchor for its spring component, as well as a coloured dot and a number to identify the linked layer. Finally the **cross** is a text field used to show the status of the layer (hidden or displayed). The **RightHandAnchor** also contains an inactive

**PhysicsPointer**, this is activated in Annotation Mode while the **CustomHandRight** gets deactivaded. This is the Pointer used for selecting elements in the scene.

- **EventSystem:** lastly the Event System is overriding the default one and allowing the use the **PhysicsPointer** as an input for triggering Events.

## 4.2 Code structure

All of the code used for developing the project is in form of C#scripts used to control the behaviour of objects, cameras or the Scene in general. Some objects in the projects have been assigned more than one behavioural script. This was done to properly apply **separation of concerns** so that in the future, it could be easier to modify or remove pieces of behaviour. In some occasions, behavioural scripts have to comunicate with each other. To follow **encapsulation** principles, proper getters and setters methods have been included. Unity allows to initialize fields declared in scripts with GameObjects or components present in the Scene. This can be done direclty in the graphic engine if the fields are declared as public. Since this feature eases many processes, but we ideally wouldn't want fields to be declared as public just for this reason, I used Unity's feature to **Serialize Fields** to reach the same objective with private or protected fields.

To promote **reusability** the code has been cleaned where it was possible to. Scripts were also properly commented to describe their function as well as the functionalties of every method. Some classes however include much more functionalities than needed in a project. This is because in some cases (such as the script that controls the model's movement) the same class has been reused for both the Computer and the VR based experience.

## 4.3 Development approach

### 4.3.1 Workload distribuition

Before starting off working on the project I planned out the development to always have an idea of the status of the progress.

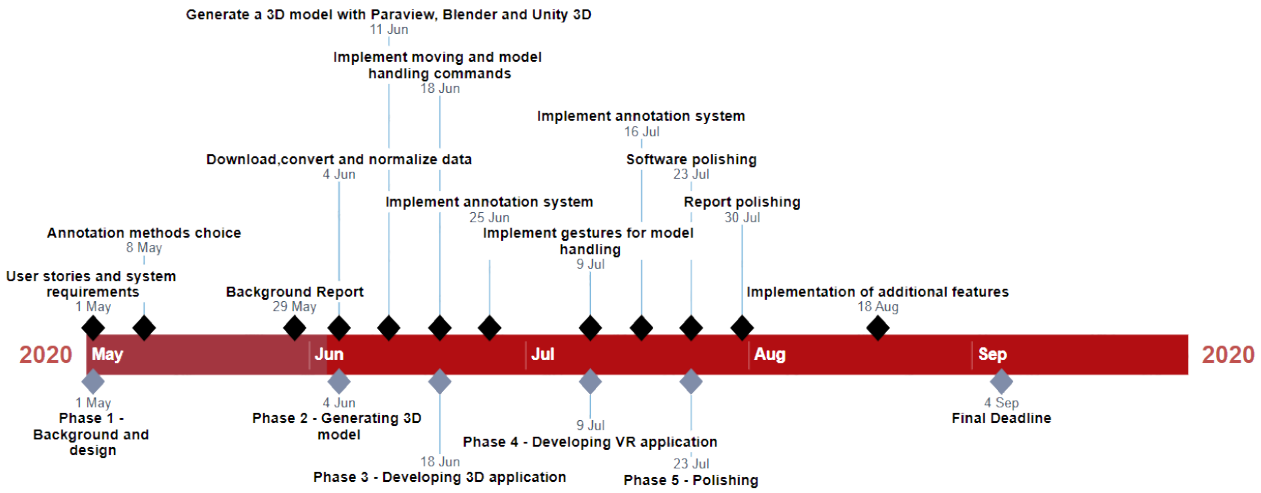


**Figure 4.2:** Gantt chart showing the plan of the development

Figure 4.2 shows the initial plan for the project. Unfortunately due to organization issues and general delays in production and shippings caused by the COVID-19 situation, the hardware used for development took longer than expected to arrive. The original plan has been adapted to cope with the disruptions.

### 4.3.2 Agile and CI

The whole development of the project has been done using an Agile approach, specifically the Kanban method. Visualizing each individual functionality one at a time and implement them in single sessions or sprints hepled a lot to keep focus on the progress. I have not used any automation of a Continuous Integration. However, after having set up the building process, following each functionality integration I would build and test the software.

### 4.3.3 Debugging and Testing

Unity has a feature in which users can test the current scene without building the whole application, this is called Game mode. While in Game mode users can also change details and values of Game Objects in the scene. Testing is made much simpler by this feature. Since this experience is a sort of interactable piece of digital entertainment, ideal testing is done by repeatedly running the software and trying to trigger or reproduce known unwanted behaviours. This is exactly what I did. Many different bugs have been found and removed during the development. Almost every bug has been discovered after the implementation of a new feature and has been removed before the implementation of the following one.

Debugging on Unity is usually done by logging errors and warnings. Unfortunately, when testing on the Oculus Quest, this process becomes harder. Given the standalone nature of the device, the apk should be built and loaded onto the device

using **adb**[33] after every new integration. This process takes a lot of building time. It also gets much harder to view error messages since it has to be done once again through **adb** in the command line. Fortunately a better solution exists, even if it's still in Beta and it's unstable. Oculus Link is a feature provided to the Oculus Quest that allows it, with the use of a cable, to use a computer's hardware to run applications that would otherwise not be supported. This also includes Unity's Game mode, ensuring both faster testing and the possibility to use Unity for analyzing logged messages.

### 4.3.4 Evaluation

Once again, a software of this kind is hard to evaluate without the help of some target users. I tried to have some people that were oblivious to scope of the project help me by trying the software. I got feedback about the user experience and how it could be improved by asking some questions. This has been of great help but none of this has been recorded. The only other way to evaluate the software was to test it's framerate during the execution of the experience.

To do so, I needed to disable Oculus Link, because otherwise I would be using the computer's hardware instead of the headset for rendering frames. Doing this meant that I had no acces to Unity's statistics in Game Mode. I had to find a different way to obtain information about time and frames elapsed. My solution was to take advantage of the Update method used in Unity's scripts. This method is called on every script in every game object in the current Scene before any frames get rendered. Only after all the Update methods have finished execution, the new frame is rendered. This means that I could keep a counter of frames generated and time elapsed between each frame generation. Using this data I could calculate an average framerate (frames per second) during execution and log the results. I had to use the standard Oculus Quest debugging system to view the logged debugging messages, using adb through the command line. I used this system to calculate framerate in different scenarios:

1. having one entire layer rendered in the view;
2. having one entire layer rendered in the view while moving the it;
3. having the whole model rendered in the view;
4. having the whole model rendered in the view while moving the it;
5. having the model scaled up so that it overflows the view;
6. having the model scaled up so that it overflows the view while the camera is moved around;
7. having the model scaled up so that the decimated meshes are replaced with the high resolution ones while the camera is moved around.

| User's view scenario | FPS |
|---|---|
| 1 | 72 |
| 2 | 72 |
| 3 | 72 |
| 4 | 72 |
| 5 | 72 |
| 6 | 72 |
| 7 | 61 |

**Table 4.1:** Average framerates in different user's view situations

The average framerates have been calculated using the final piece of software over a 5 seconds time frame. The results can be seen in table 4.1. I consider 60 frames per second providing an absolutely great experience. The results provided show that the rendering process is fast enough to consistenlty ensure a smooth experience for users. The model's scale value that acts as the treshold for the switch between decimated and original meshes was chosen examining results from this test. The value was adjusted so that it could be as small as possible while always ensuring 60 frames per second.

# Chapter 5

# Conclusion

This project succesfully obtained the development of a Virtual Reality experience for Oculus Quest that can be used to visualize and annotate spatial data from the Volve field dataset. However the software could easily be used to visualize and annotate any other dataset of spatial data. Moreover, to prove the upsides of using Virtual Reality for visualizing spatial data, a similar experience based in a non VR environment has beed produced.

This software could ideally be used by researchers to analyze data and discuss choices about future oil wells placement. But the greatest achievement of the project is demonstrating the value of using Virtual Reality for visualizing spatial data. The immersion provided by the Oculus Quest, combined with a properly designed experience, could potentially make data navigation much easier and more enjoyable. Moreover, using detailed models, spotting tiny details in meshes becomes an easier process as well.

By comparing navigation in VR with the navigation in the additional computer based experience, any user can easily tell the advantages provided by Virtual Reality. This is considering that users get used to the controls and feeling of the headset in the first place. To ease this process the VR experience has been designed to be as intuitive as possible, also for users that have never worked in VR before. This was done by complying to navigation standards where possible, or designing new gestures that recall real life movements. To enhance immersion, the User Interface has beed designed to reduce the disruption in the view to a minimum.

## 5.1 Future work

The amount of additional features that could be implemented in this project could range from simple navigation options to much more complex functionalities. Here I will quickly describe a few interesting ones.

### 5.1.1 Importing datasets

Since the application can be used to analyze multiple datasets, it would be interesting to implement a feature that allows users to import datasets in .csv format and automatically generate 3D models and a new scene. This would not be straightforward to implement. It would require to use the same Python scripts used by ParaView to generate the models and then saving the models on the device's storage. Then the scene has to be procedurally built every time the software is run. This is very complex but the upsides are remarkable.

### 5.1.2 Sharable annotations

This software is designed to be used in a single office on a sharable device. Users may want to use personal devices or simply multiple devices owned by the company. In order to do this, annotations should be shared either by being saved on a remote database accesible from everyone, or an annotations set could be exported and manually loaded into the system. Moreover, given the growth of remote jobs during this period of time, this may become even more useful.

### 5.1.3 Multiplayer experience

Another idea is to allow multiple user to access the same simulation at the same time. Users don't necessary have to be in the same physical place for this to happen, but they would need to have an headset each. Visualizing other poeple's avatar inside the experience could give an opportunity to discuss decisions in real time, making annotations of secondary importance. A real time voice chat could be implemented to enhance this feature. A relaxation of this could be used for demonstrations: a primary user has acces to all the functionalities of the software, while a number of secondary users are in the same simulation and in the same position but can only move the camera while immersed in the dataset. Data visualization is improved in such a way that also stakeholders or outsiders can visualize the data without the need of having experience with the application's controls.

# Bibliography


[1] URL `https://www.equinor.com/`. pages 2, 9

[2] URL `https://www.norskpetroleum.no/en/facts/field/volve/`. pages 2, 9

[3] J. Bamberg. Engaging the public with online discussion and spatial annotations: The generation and transformation of public knowledge. *Planning Theory & Practice.*, 14, 2013. pages 4

[4] D. Bourguignon, M-P. Cani, and G. Drettakis. Drawing for illustration and annotation in 3d. *Computer Graphics Forum.*, 20, 2001. pages 5

[5] S. Carter, E. Churchill, L. Denoue, J. Helfman, and L. Nelson. Digital graffiti: Public annotation of multimedia content. *Conference on Human Factors in Computing Systems - Proceedings*, 2004. pages 5

[6] T. Jung, M.D. Gross, and E.Y.-L. Do. Annotating and sketching on 3d web models. *Proceedings of the 7th international conference on Intelligent user interfaces - IUI '02*, 2002. pages 5

[7] T. Langlotz, H. Regenbrecht, S. Zollmann, and D. Schmalstieg. Audio stickies: Visually-guided spatial audio annotations on a mobile augmented reality platform. *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration, OzCHI 2013*, 2013. pages 5

[8] C.X. Ma, Y.J. Liu, H.A. Wang, D.X. Teng, and G.Z. Dai. Sketch-based annotation and visualization in video authoring. *IEEE Transactions on Multimedia.*, 14, 2012. pages 6

[9] J. Yuan, J. Li, and B. Zhang. Exploiting spatial context constraints for automatic image region annotation. *Proceedings of the ACM International Multimedia Conference and Exhibition.*, 2007. pages 6

[10] B.L Chalfonte, R.S. Fish, and R.E. Kraut. Expressive richness: A comparison of speech and text as media for revision. *In Proc. of CHI'91*, 1991. pages 6, 12

[11] M. Tsang, G.W. Fitzmaurice, G. Kurtenbach, A. Khan, and B. Buxton. Boom chameleon. *Proceedings of the 15th annual ACM symposium on User interface software an technology - UIST '02*, 2002. pages 6

[12] S. L. Oviatt. Ten myths of multimodal interaction. *Communications of the ACM.*, 42, 1999. pages 7

[13] D. Clergeaud and P. Guitton. Design of an annotation system for taking notes in virtual reality. *3DTV Conference 2017: Research and Applications in Future 3D Media.*, 2017. pages 7

[14] S. Pick. Interactive data annotation for virtual reality applications. 2019. pages 7

[15] URL https://www.tiltbrush.com/. pages 7

[16] URL https://help.irisvr.com/hc/en-us/articles/213570187-Annotations. pages 8

[17] URL https://www.oculus.com/. pages 8

[18] URL https://www.oculus.com/experiences/quest/1863547050392688/. pages 8

[19] URL https://github.com/buzzfeed-openlab/glossy. pages 8

[20] URL https://www.vive.com/. pages 8

[21] URL https://www.eia.gov/dnav/pet/pet_crd_welldep_s1_a.htm. pages 9

[22] URL https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-headset-market. pages 9

[23] URL https://www.forbes.com/sites/briansolomon/2014/03/25/facebook-buys-oculus-virtual-reality-gaming-startup-for-2-billion/. pages 9

[24] URL https://www.oculus.com/quest/. pages 9

[25] URL https://www.paraview.org/. pages 10

[26] URL https://www.blender.org/. pages 10

[27] URL https://unity.com/. pages 10

[28] URL https://data.equinor.com/?_ga=2.207664082.1955451383.1597512538-1925792752.1590613667. pages 17

[29] URL https://www.mathworks.com/help/matlab/math/delaunay-triangulation.html. pages 17

[30] URL https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Workbench/!FME_Geometry/Vertex_Normals.html. pages 18

[31] URL https://gist.github.com/darktable/2317063. pages 20

[32] URL https://assetstore.unity.com/packages/tools/integration/oculus-integration-82022. pages 22

[33] URL https://developer.android.com/studio/command-line/adb. pages 35, 42

[34] URL https://developer.oculus.com/documentation/native/android/mobile-device-setup/. pages 43

# Appendix A

# Installation Guide

The proposed software is distributed as an .apk and it has to be installed on an Oculus Quest device. To do this users can use the command line, navigate to the folder containing the apk (finalproject.apk) and run the command **adb install finalproject.apk** . This requires adb[33] to be installed.

# Appendix B

# VR Users Guide

On the Oculus Quest device, navigate to the library and then to **unknown sources**. This could require the device to be set to developer mode[34]. Players should be seated on a swivel chair while using the software and they are required to be able to spin, however, moving is not necessary. Use both Touch controllers to run the application and allow permissions to access the microphone if annotating is necessary. The Controls are as follows:

- Left controller's thumbstick is used to move forward/backwards/left/right;
- Right thumbstick is used to move up/down or to move the audiobox left/right;
- Side button is used to close the hand's grip, this allows to grab items;
- Index trigger is used to grip with the index finger;
- X and A buttons are used to close the thumb finger.

Gestures:

- **Pointing:** to point a finger, use X/A buttons and side buttons to simulate a pointing position.
- **Grabbing objects:** to grab an object move one hand on the object (or on the object's collider) and grab it using the side button.
- **Pinching:** to pinch, close index and thumb finger at the same time.

Once out of the Welcome Scene, users can swap between the three modes in this order: Layers, Model and Annotation, by pressing the Y Button.
Functionalities available in every Mode:

- **Moving:** use thumbsticks as described before to move around the scene, users are not affected by gravity.
- **Speed Control:** press the right thumbstick as a button to increase the movement speed modifier, press the left one to reduce it, speed ranges from 1 to 10.

Functionalities in Layers Mode:

- **Buttons pressing:** To press a button, point the index finger and press the button using that finger. Pressing a button will hide/show one of the layers.

Functionalities in Model Mode:

- **Moving the model:** to move a model, grab and move the hand used to grab it, the object will follow the hand's position until released.
- **Scaling the model:** while in a room with a scalable model, pinch with both hands and move hands apart or closer before releasing the pinch to enlarge or reduce the size of the model.

Functionalities in Annotation Mode:

- **Recording a new annotation:** use the right hand index trigger while pointing to the "+" audio sphere to start recording. Stop recording by doing the same.
- **Placing annotation:** point to the desired Well, Collection or any point on a layer and use the right hand index trigger to place the new annotation. To place an annotation in space, point to your left hand and pull the index trigger.
- **Selecting/Deselecting a Location:** to select a Well or a Collection, point to it with the pointer and pull the right hand index trigger. Selected objects are indicated by a different colour. To deselect the current selection, pull the left hand index trigger.
- **Playing/Stopping audio annotation:** to play or stop an audio annotation, first select the Collection or Well the annotation belongs to. Then pull the right hand index trigger while pointing to the desired annotation.
- **Deleting an annotation:** to delete an annotation, first select the Collection or Well the annotation belongs to. Then point to the desired annotation and keep the right hand index trigger pulled for 3 seconds, the annotation will just disappear.
- **Going back to the Welcome Scene:** to return to the Welcome Scene simply keep the left hand index trigger pulled for 5 seconds.