jamk.fi

Data visualisation in virtual reality

Antti Mäkelä

Bachelor’s thesis
December 2018
Technology, Communication and Transport
Degree Programme in Software Engineering

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Mäkelä, Antti	Bachelor's thesis	December 2018
		Language of publication: English
	Number of pages 60	Permission for web publication: x

Title of publication

Data visualisation in virtual reality

Degree programme

Software Engineering

Supervisor(s)

Lappalainen-Kajan Tarja, Rantala Ari

Assigned by

Huotari Jouni, Rantonen Mika

Abstract

Data visualisations are needed more than ever to understand the vast abundances of data that have surfaced. Visualisations can be used to convey the meaning behind the data. They can also be used as a method of data analysis. For long, data visualisations have been presented with two-dimensional media such as paper, chalkboards and computer monitors. As virtual reality technologies become more advanced and commercially available for the average consumers, visualisation can be brought into a whole new dimension of representation and interaction.

The project was started based on the idea of building a data visualisation application that can be viewed through head-mounted displays and interacted with motion controllers. The application was constructed with Unreal Engine 4 game engine and the tools associated with it. During the development of the application, different features and data visualisation methods were evaluated and included in the final application when appropriate.

The resulting application was a three-dimensional scatter plot visualiser that could be operated through a head-mounted display and motion controllers. With head-mounted displays, the users of the application could easily view the visualization from different perspectives. Motion controllers could be used to move, rotate and stretch the visualisation. The application supported visualising variables on four axes: three spatial dimensions and one colour axis. An implementation of t-SNE dimension reduction algorithm was also included in the application among other data analysis tools.

Although the project finished successfully and the final result was functional, many ideas and feature requests from the assignors were left for future development.

Keywords/tags (subjects)

computer graphics, Unreal Engine 4, virtual reality, visualisation

Miscellaneous (Confidential information)

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Mäkelä, Antti	Opinnäytetyö, AMK	Joulukuu 2018
	Sivumäärä 60	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Data visualisation in virtual reality

Tutkinto-ohjelma

Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)

Tarja Lappalainen-Kajan, Ari Rantala

Toimeksiantaja(t)

Jouni Huotari, Mika Rantonen

Tiivistelmä

Datavisualisointia tarvitaan enemmän kuin koskaan ennen massiivisten tietomäärien ymmärtämisessä. Visualisoinneilla voidaan viestiä tiedon merkitys graafisessa muodossa. Niitä voidaan käyttää myös data-analyysin aputyökaluina. Datavisualisointi on perinteisessä merkityksessään ollut aina datan projisoimista kaksiulotteisille pinnoille, kuten paperille tai tietokonenäytöille. Virtuaalitodellisuusteknologioiden ja kaupallisten toteutusten yleistyessä datan visualisointi voidaan viedä lähemmäksi käyttäjää kuin koskaan ennen.

Projektia lähdettiin rakentamaan datavisualisointiohjelmana, jota voitaisiin käyttää virtuaalilasien ja liikeohjaimien avulla. Ohjelma rakennettiin Unreal Engine 4 pelimoottorin ja sen kanssa käytettävien työkalujen avulla. Ohjelman kehityksen aikana testattiin useita eri kokeellisia ominaisuuksia ja visualisointimetodeja. Näistä lisättiin lopulliseen ohjelmaan vain ne, joista todettiin olevan hyötyä.

Lopullinen datavisualisointiohjelma oli kolmiulotteisten pistekaavioiden piirto-ohjelma, jota pystyi käyttämään virtuaalilasien ja liikeohjaimien kautta. Virtuaalilasien avulla ohjelman käyttäjät pystyivät katsomaan visualisaatiota valitsemistaan perspektiiveistä. Liikeohjaimia voitiin käyttää kaavion liikuttamiseen, kääntämiseen ja venyttämiseen. Ohjelma tuki muuttujien piirtämistä neljälle eri akselille: kolmeen tilaulottuvuuteen ja yhdelle väriakselille. Ohjelma sisälsi myös muutamia data-analyysityökaluja, kuten t-SNE-ulottuvuuksien vähennysalgoritmin.

Vaikka projekti päättyi onnistuneesti ja lopputulos oli täysin toimiva, runsaasti ominaisuuksia ja toimeksiantajien toivomuksia jätettiin jatkokehitykselle.

Avainsanat (asiasanat)

tietokonegrafiikka, Unreal Engine 4, virtuaalitodellisuus, visualisointi

Muut tiedot (salassa pidettävät liitteet)

Contents

Figures

Terminology

CPU	Central Processing Unit. A computer component that executes the logic of the computer program.
CSV	Comma-Separated Values. A textual data storage format where fields of data are delimited by commas.
GPU	Graphics Processing Unit. A device used to accelerate the computer generation of images and video.
HMD	Head-Mounted Display. A VR display device that is mounted on the user's head, providing separate viewports for both of user's eyes. (Head-Mounted Displays.)
RAII	Resource Acquisition Is Initialization. A programming language concept where lifetime of a resource is bound to the lifetime of a variable used to access it. (RAII 2017.)
UI	User Interface.
VR	Virtual Reality. The concept of creating simulated experiences that feel immersive and even realistic to the user, by the means of head-mounted displays, motion controllers and other accessories. (What is Virtual Reality?.)

1 Introduction

In the age of computers and big data, visualising the data has become an invaluable tool to understand the massive volumes of information available. Machine learning and applied statistics are used to solve the mysteries within these large data sets algorithmically; however, sometimes a pair of human eyes can do just as well or even better. Visualising data is a method of transmitting information to others; when the facts are laid out before the eyes, they are infinitely more believable. (Data vizualisation beginner's guide.)

With emerging technologies such as virtual reality, the visualisation can be taken even closer to the viewer. In addition to countless entertainment uses virtual reality has, its immersiveness has many practical applications as well, such as training medical personnel and aircraft simulation. Analysing masses of data inside the VR environment certainly is possible, however, the question remains how to leverage the full potential of the virtual reality headset without it becoming an unnecessary burden to the user. (Cashorali 2018.)

This thesis attempts to explore the potential use cases of VR inside the data visualisation world, and to provide an understanding of how virtual reality and data visualisation are combined. These thoughts are then condensed into a functional VR data visualisation application. The assignment came from Institute of Information Technology at JAMK University of Applied Sciences. Jouni Huotari, principal lecturer, and Mika Rantonen, senior lecturer, assigned the research project.

2 Direction of research

2.1 Goals and design

The primary requirement for this thesis was to investigate data visualisation in virtual reality and to build a VR data visualisation application. This requirement leaves much freedom for the developer to discover what VR data visualisation is about and how to utilize VR efficiently. A virtual reality scatter plot visualiser was chosen as a goal for the project.

The visualisation program would have to display a data set in a three-dimensional scatter plot, and the user would have the ability to control how the data set is displayed. The user would also be able to move around the VR world, looking at the data set from different angles. The motion controllers would act as manipulators for the visualised data: adjusting the position and rotation of the visualised scatter plot would be simply a matter of moving the controllers.

A user interface of some kind would also have to be provided inside the virtual reality world. With the interface, the user could inspect and manipulate the data further. Adjusting what variables are displayed, how the data is projected and viewing auxiliary information such as correlation coefficients would be the duties of this user interface.

The general research process used in this thesis can be divided into three steps. First step is to qualitatively gather information from various sources. Solutions to problems and other improvements would be synthesised based on the gathered information. Advantages and disadvantages of the new improvements would be evaluated. Based on the results of the evaluation, further research and adjustments would be made.

Documentation of used software was adopted as the primary source of information; however, internet search engines were consulted as well when needed.

2.2 Existing VR data visualisation applications

As virtual reality technologies fully emerge into the world of information and big data, an abundance of VR data visualisation programs have appeared as well. Below are listed a select few applications with similar goals with this thesis project.

One VR data visualisation program is the DatavizVR. This application is capable of plotting three-dimensional scatter plots inside the virtual environment. The user can control the appearance of these plots: the variables to be plotted and colours used can be adjusted with a user interface. (DatavizVR.)

Bertrand (2017) demonstrates a virtual reality data visualisation software titled Project NEO. The demonstration shows an application in which a data set is displayed simultaneously in multiple scatter diagrams and the user can explore these in VR.

There have also been pushes for VR applications that can be used straight from a web browser (WebVR; A-Frame). VR-Viz is a framework for web developers that attempts to bring VR data visualisations straight into the users' browsers (VR-Viz).

3 Background

3.1 Virtual reality

Virtual reality is the simulation of reality or creating an illusion of truth and presence, into which the users may immerse themselves. The idea of virtual reality has existed for centuries; however, the idea has been brought into a more concrete form in the past few decades. In these days, virtual reality is achieved with the aid of computers and a multitude of accessories: head-mounted displays that project visions into eyes, motion controllers that convey the ability to touch and interact, and treadmills that allow us to walk inside the virtual environment. These accessories must be finely tuned to provide a consistent experience in the virtual world, as human brains are very particular about the surrounding reality. Minute mistakes and desynchronizations can easily break the virtual reality experience and it can be jarring, unpleasant or even nauseating to the user. (What is Virtual Reality?.)

Augmented reality is a close relative of virtual reality. As the term “augmented reality” indicates, AR focuses on augmenting the real world with supplemental content and information, instead of wholly replacing it with a virtual one.

Head-mounted displays (HMD) are a form of virtual reality, as they try to simulate the experience of seeing and looking around oneself in a computer-generated environment. This is typically achieved by mounting two displays, one for each eye, in a chassis worn on the head. In addition, sensors such as gyroscopes and accelerometers may be used in the HMD to accurately measure the movement of the user. With the aid of the sensor feedback, the virtual reality simulation is updated to faithfully

replicate the visual experience associated with moving and looking around. (Head-Mounted Displays.)

HMDs have existed in some form since the 1960s; however, they have been unavailable to the general consumer. Late 20th century showed the rise of consumer-grade head-mounted displays, such as Nintendo Virtual Boy. These works of craftsmanship did not quite cut it in the market, as the ergonomics to use them were far too cruel for the average human, and the display fidelity left a great deal to be desired. With the rapid progress of display technology in the early 21st century, the HMD technology became more viable than before. (History of Virtual Reality.)

Using virtual reality and HMD devices in a software project brings many design challenges. One common problem that arises from HMD devices is *virtual reality sickness* or *cybersickness*, which has many common symptoms with motion sickness. These symptoms include, but are not limited to, headache, vertigo, nausea and even vomiting. (LaViola 2000.)

There are methods a developer can follow to reduce the symptoms of virtual reality sickness. Maintaining good performance and a high frame is important as low frame rates can be a cause for virtual reality sickness. Moving the user's viewport without consent can be another offending factor: cinematic camera angles and shaking the viewport for dramatic effect should be avoided. Intense lighting and post-processing effects may also induce virtual reality sickness. (Virtual Reality Best Practices.)

3.2 Motion controllers and positional tracking

Motion controllers are input devices that function by tracking the position of the controller. This is typically achieved with internal or external motion tracking sensors. When the user swings the controller, the movement of the controller is measured. The controller then forwards the movement information to the computer, which then acts upon this information.

Positional tracking in motion controllers can be described by the count of degrees of freedom in them. Degrees of freedom measure the number of tracked movement axes. For example, a controller that only tracks its position has three degrees of freedom: up and down axis, forward and backward axis and left and right axis. Tracking

the rotation of a controller has three degrees of freedom: pitch, yaw and roll. Therefore, to accurately track the exact movement of a motion controller six degrees of freedom are required: three for positioning and three for rotation. (Batallé 2013.)

The sensors involved with motion controllers and positional tracking can vary greatly. Magnetic sensors that rely on the orientation of magnetic fields are used in Razer Hydra motion controllers. Inertial trackers include devices such as accelerometers and gyroscopes. Accelerometers measure acceleration, which is useful when attempting to accurately measure position, as acceleration is the second derivative of position. Gyroscopes in turn measure angular velocity, which will find its use when estimating angle. Optical tracking is a group of methods utilising cameras that track visible markers on the tracked object. (Overview of Positional Tracking Technologies for Virtual Reality 2014.)

Motion controllers are typically used in tandem with virtual reality, as they can provide a more natural method of interaction unlike conventional input methods such as keyboards or gamepads. Virtual reality software utilising motion controllers has access to the three-dimensional position and rotational information of the controller in addition to other buttons and input axes. This information may be used to allow the user to interact with the environment more naturally. For example, the user might move the controller close to an object in VR and depress the trigger button to grab the object. The user may also position objects naturally by moving and rotating the controller.

3.3 Data visualisation

Data visualisation is the art of transforming information into a visual representation that may be viewed and understood easily. With visualisations, a form is given to the data: points, bars, lines, colours, angles, and many other visual features may be utilised when building a graphical representation of raw data that has no corporeal shape. Visualisations can be crafted to convey a message and show the truth behind the data. Raw numerical values can be more accessible after visualising them, and in a good visualisation, the nature of the data can be discerned immediately with a glance. (Few.)

Visualisations can also be used as tools for data analysis. Sometimes the mystery behind the data is revealed by just looking at the data through different visualisations. Human subconsciousness is very good at autonomously seeing patterns and detecting if something deviates from those patterns. Location of objects, similarities in appearance, connections and other traits are detected and used to build a mental grouping of the objects that are perceived. Much information is gathered from the visualisation pre-attentively, thus reducing the workload of the conscious mind when interpreting the visualisation. (ibid.)

3.3.1 Scatter plots

Scatter plots (Figure 1) visualise data by positioning points in the graphic. Typically, scatter plots exist as two-dimensional plots where the positioning of the data points along horizontal and vertical axes are dependent on two variables. These plots give immediate feedback on how the variables are distributed and how they interact with each other.

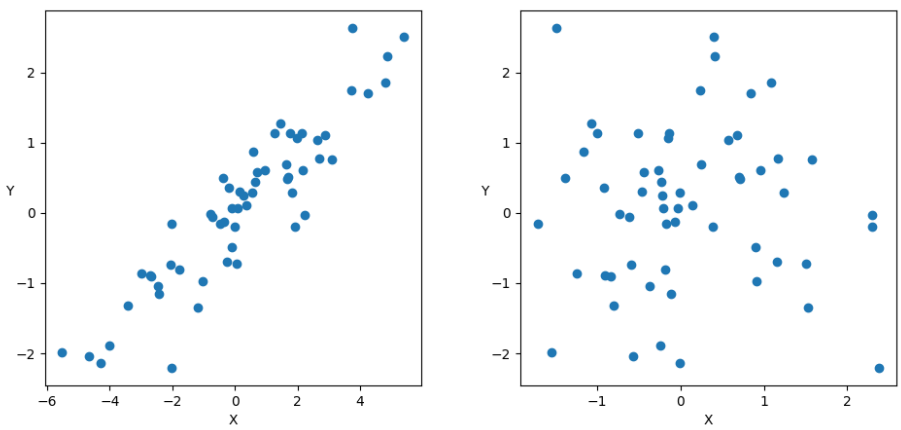


Figure 1. Examples of scatter plots

The figure above (Figure 1) displays two examples of scatter plots. The example on the left shows a strong correlation between two plotted variables which shows that the values interact with each other somehow. The example plot on the right displays no visible patterns or correlations.

Scatter plots can exist in three dimensions as well. A third dimension is extended perpendicularly to the other two dimensions. The typical problem in these higher dimensional scatter plots is their representability: a two-dimensional display method,

e.g., a computer monitor cannot accurately portray the minutiae of complex three-dimensional shapes. The higher dimensional shape must be first projected onto a two-dimensional plane before it can be presented to the user. In this process, some of the data may be lost or be harder to read: data points may occlude each other, and the relative depth of two data points may be lost in the representation. In many cases, it is more useful to use a two-dimensional plot with additional dimensions represented as some other feature, such as colour, shape or size. (Wilke.)

3.3.2 Data visualisation and VR

When visualising data in virtual reality, one obvious advantage surfaces; namely, the capability for the user to move, look and adjust his or her perspective freely. Interacting with the data can be much more natural in a virtual environment, as the user can grab and pull the visualisation closer for a better look. As VR data visualisation is a relatively new concept, the true effectiveness of virtual reality visualisations is yet to be seen.

Warfel (2016) argues that VR allows intuitive perception of different quantities. He says that VR representations are interactive by nature and dense data may be viewed more easily as the user can adjust the perspective of the visualisation to appear more convenient. He additionally states that VR gives users the possibility to comprehend scales that are otherwise unimaginable to the human mind.

Hackathorn (2015) discusses the negative side effects of VR technologies, such as possible nausea and physical fatigue that may distract the user from the visualisation. Hackathorn also states that VR visualisations should not be an extension of two-dimensional visualisations but rather leverage the unique advantages of virtual environments.

4 Tools and technologies

4.1 Virtual reality systems

Many virtual reality head-mounted displays exist on the market, such as Oculus Rift or HTC Vive. This thesis focuses on the HTC Vive virtual reality headset.

HTC Vive, or simply Vive, is a VR system co-developed by HTC and Valve corporation. The headset (Figure 2) comes with a combined display resolution of 2160x1200, 1080x1200 for each eye, front camera, gyroscope and sensors for acceleration and proximity (Vive VR System). Two motion controllers (Figure 3) with a trackpad and multiple buttons are included in the Vive system. The location and movement of the headset and controllers are tracked with two base stations. The headset requires wired connection to the computer to work while the controllers work wirelessly.

Figure 2. Front view of the HTC Vive headset

Figure 3. HTC Vive controllers

OpenVR is a virtual reality hardware API that provides a layer of abstraction between the programmer and various kinds of VR hardware. The users of this API do not need to know the exact specifications of the targeted hardware and may easily program VR applications that work on every hardware device supported by the OpenVR implementation. SteamVR is the runtime implementation of the OpenVR API. It is distributed via the Steam digital distribution service. (OpenVR SDK; OpenVR - API Documentation)

4.2 Unreal Engine

Unreal Engine is a game engine family from Epic Games. Its first iteration was developed in late 1990s, and in 1998 Unreal was released, the first game using the Unreal Engine. The original iteration of the Unreal Engine is currently known as Unreal Engine 1. The newest Unreal Engine version is Unreal Engine 4, released in 2014 along its complete source code. Unreal Engine 4 is a very widely used game engine supported on many different platforms, such as desktop computers, a wide variety of game consoles and mobile devices. The volume of features and the licensing policy of Unreal Engine makes it a compelling choice for both smaller, independent developers and enterprise scale producers. (This is Unreal 2016.)

The engine comes with a plethora of tools to aid in development of games and software. Many of these are available for use within the Unreal Editor application (Figure 4) which is the primary development environment for Unreal Engine developers.

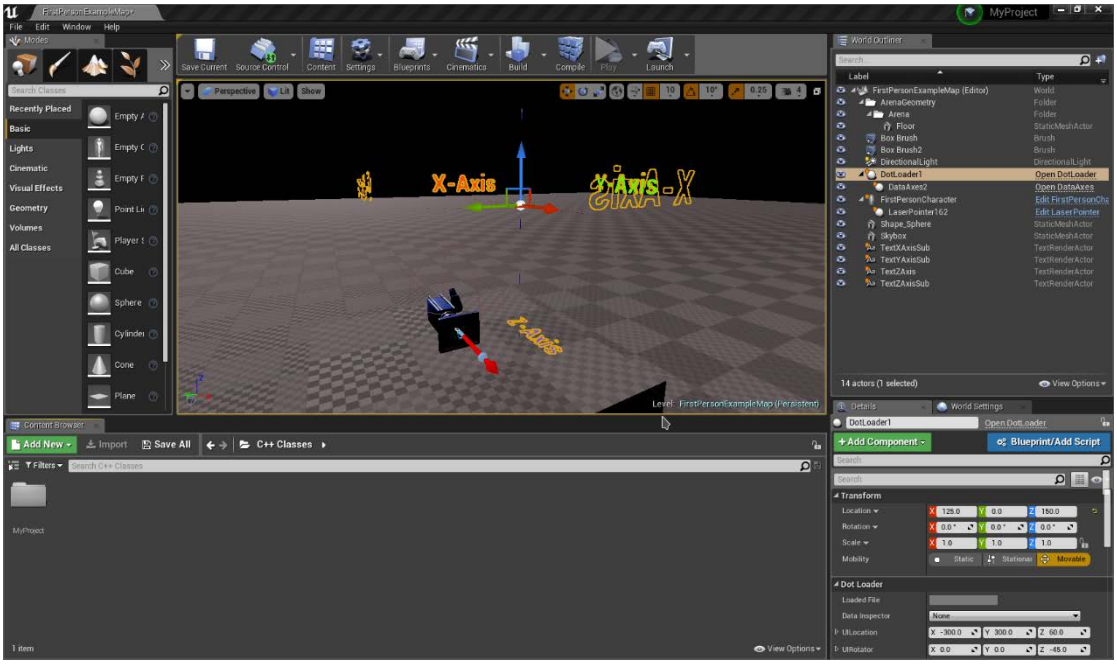


Figure 4. Unreal Engine 4 editor user interface

Unreal Engine 4 was chosen for this data visualisation project for many reasons. The primary reasons were its ability to interface directly with C++ code, compatibility with Linux systems and support for VR development for HTC Vive.

4.3 Performance profiling tools

When developing an interactive real time application, maintaining good performance is essential for a good user experience. When the performance is non-optimal, the developers may use *profilers* in order to discover the underlying reasons for the poor performance.

Unreal Engine 4 provides performance profiling tools (Figure 5) that may be used for investigating what causes slowdowns and hitches in the application. The profiler included shows a hierarchical view of the application with function level timing information. This view can be used to identify the exact cause of CPU related performance bottlenecks. Unreal Engine 4 also includes internal frame rate displays that can be used quickly measure the general performance of the application.

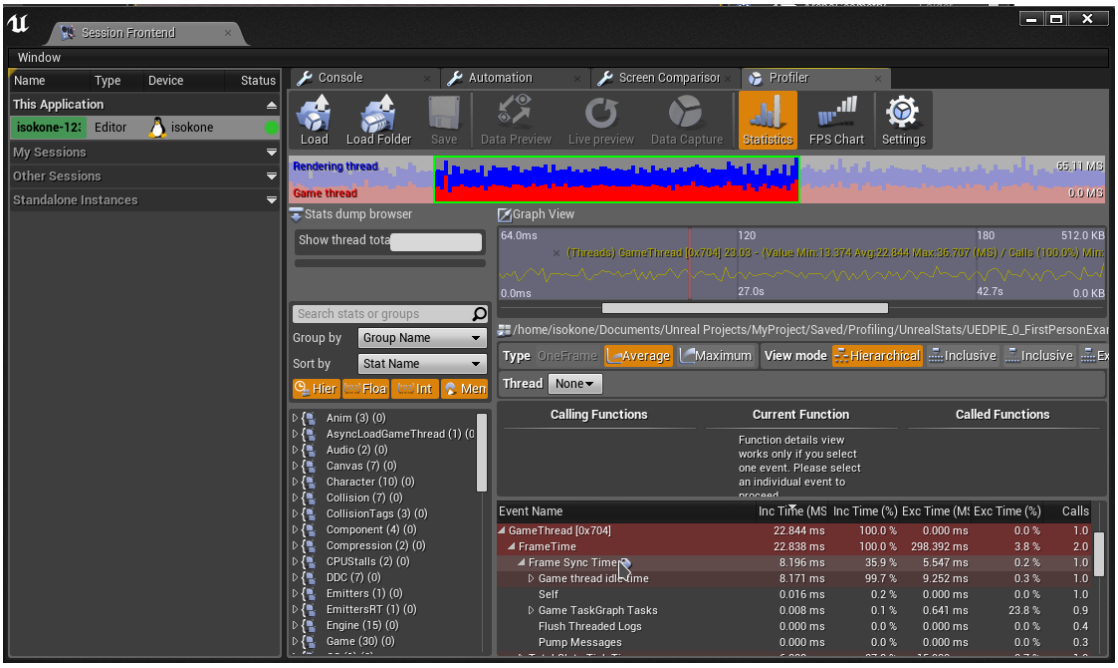


Figure 5. Unreal Engine profiling tools

SteamVR includes a simple VR frame rate display that can display whether the current frame rate is adequate. When more detailed statistics are desired, SteamVR also provides a user interface for viewing more comprehensive frame timing information (Figure 6). This tool gives insight where the possible bottlenecks are, whether they are CPU or GPU bound problems and what could be done to alleviate them.

Figure 6. SteamVR frame timing display

4.4 Multithreading

Multithreading is the technique of launching separate threads of execution in one program, as opposed to a conventional single threaded linear model execution. It can be a dangerous trade when not following proper safety precautions. New types of

bugs surface from a threaded environment that can be challenging to trace; however, when done correctly it is one of the most useful utilities for heavy processing tasks. Running computationally expensive tasks in separate threads from the user interface thread can help a program to function much more smoothly. These tasks may include some long running statistical calculations or input/output operations that may stall due to external circumstances.

A race condition is a usually unwanted phenomenon that occurs when multiple threads use the same unprotected piece of data simultaneously. This may manifest as some indeterministic quirk in the program, complete program malfunction or anything in between. The effects may be very hard to reproduce as they may depend on some very specific condition on multiple threads at the same time. (Description of race conditions and deadlocks 2012.)

To counteract race conditions, many threading libraries include implementations for *locks*. Locks can be *acquired* by only one thread at the same time and other threads must wait until the lock is *released*. Locks are used in multithreading contexts to ensure that a certain piece of code is never executed simultaneously in multiple threads. Race conditions can be prevented by guarding accesses to data that is visible to multiple threads with locks. However, with locks a new type of bug may surface: *deadlocks*. A deadlock occurs when threads are holding locks and trying to acquire new locks in a fashion that all participating threads start waiting for each other to release the lock. The most noticeable symptom of a deadlock is the program freezing. (ibid.)

Completely eliminating deadlocks is not as easy when compared to race conditions; however, they can be avoided with good programming practice. One of the more common deadlock situations is when one thread tries to acquire the same lock multiple times. This is fixed by following the trail of program and ensuring that the lock will not be recursively acquired twice. When needing multiple locks simultaneously, the order of acquiring locks should be established and that order should be used across the code base. This is to prevent a deadlock when multiple threads acquire the same locks in different order simultaneously. (Avoiding Deadlock.)

4.5 Statistical methods

Various statistical and data-analysis methods find their use when handling and analysing masses of data. Finding mean, variance, minimum and maximum can provide a quick piece of information what sort of data is in question. Like many compact representations of data, these should not be trusted blindly as is but they should be a tool used to form a greater picture. (Ali & Bhaskar 2016.)

One more sophisticated statistical quantity is correlation. Pearson's correlation coefficient, or simply correlation, is a measure of how two variables of a dataset relate to each other. This coefficient quantifies the amount of linear relationship between the two variables, and if no such relationship exists, the coefficient is zero. Values deviating further away from zero indicate stronger linear relationship, and when the coefficient is exactly one or negative one, the relationship is fully linear. In more complex situations with non-linear relations, some other measurement method must be used to fully capture the essence of the relationship. (Pearson Correlations – Quick Introduction.)

4.6 t-SNE

t-SNE or t-distributed stochastic neighbour embedding is a data visualisation and dimension reduction algorithm. t-SNE may be used to find lower dimensional representations embedded in the higher dimensional space. This may be very useful when exploring data sets with many variables. (Van der Maaten 2018.)

t-SNE has one parameter that may be tuned by the user, perplexity. In general terms, perplexity is used to describe how close the points should be to each other. In many cases, the user should try running t-SNE multiple times with different perplexity values to understand the nature of the data fully. t-SNE may give different results for the same input data set even if the same perplexity is used on both runs. This is due to the algorithm's random initialization. (ibid.)

Barnes-Hut-SNE is an implementation of t-SNE using a variation of the Barnes-Hut approximation algorithm for n-body problems. In terms of big O notation, this improves

the computational time complexity of t-SNE from $O(n^2)$ to $O(n \log n)$. (Van der Maaten 2013.)

5 Assembling a VR data visualisation application

This chapter gives an overview of how the VR data visualisation application was built and where different technologies were utilized. How the application is used, and the full extent of its features are described in the user guide (See Appendix 1).

5.1 Design process

Much of the thesis work was developed in a rather impromptu manner: many features were included and subsequently removed only for the sake of experimentation. Dimension reduction algorithm t-SNE, spherical projection, and other tools were introduced as quick prototypes that formed into fully functional features in the final version.

The early parts of the development process focused primarily on the data visualisation aspect of the program. Virtual reality development was pushed aside, and the plan was to attach VR to the finalized data visualisation engine. This was rationalized with the fact that a modern game engine like Unreal Engine 4 has features to transition a project from a 3D only project to a VR project seamlessly.

The project was developed on a desktop Linux machine running Ubuntu 18.04.1. Source control software Git was used to maintain version history of the project in cases of discovered regressions or data loss. The primary tools used during the development were Qt Creator integrated development environment and Unreal Engine 4 editor. GIMP image manipulation program and Blender 3D modeler was occasionally used when creating or adjusting graphical assets. Visual Studio 2017 was used when building the Windows builds of the application.

5.2 Using Unreal Engine 4

This chapter describes the general process when using Unreal Engine and Unreal Editor in a Linux environment. The Unreal Engine version used in this thesis is version 4.20.2 built from source.

5.2.1 Acquiring Unreal Engine

Unreal Engine 4 can be installed via Epic Games Launcher on supported platforms. Using the launcher requires the user to create an Epic Games account. Alternatively, the engine can be built from source code. The source code is available in GitHub source hosting service. To gain access to the engine source code the user must link a GitHub account to the Epic Games account and agree to the Unreal Engine End User License agreement. Afterwards the user will gain access to the Unreal Engine GitHub repository and subsequently access to the entire Unreal Engine source tree. (Installing Unreal Engine; Downloading Unreal Engine Source Code)

As the thesis project was developed in Linux environment where no Unreal Engine binary distribution was available, the engine had to be built from source. The build process is explained in detail on the Unreal Engine GitHub page. When building the Windows distributions of the application, Unreal Engine binary downloaded via the Epic Games Launcher was used.

5.2.2 Operating the editor

Launching Unreal Editor is a matter of executing the binary. On the first start-up of the editor, the materials and shaders for the engine are compiled, and the process can take many minutes. Unreal Editor provides a guided project creation wizard that builds a new project from a template.

Opening a project can be done from the Unreal Editor user interface or by supplying the project name as a parameter to the editor executable:

```
./UE4Editor ~/MyProject/MyProject.uproject -nosound
```

The *-nosound* flag is used to disable audio in the engine to avoid audio related bugs and crashes.

Sometimes the engine will refuse to load the project and inform the user that building the project was unsuccessful and the user should manually build the project. Occasionally, this problem simply disappears after starting the engine again; however, if this does not work, a rebuild is necessary.

Unreal Engine 4 editor provides an integration with source control software. SVN and Perforce repositories are supported, and Git support can be installed with a separate plugin. Assets, Blueprint scripts and other files included in the project are automatically tracked in the version control. When merging version control branches, many of the Unreal Engine project files, namely assets and levels, are stored in a binary format and therefore cannot be merged with automatic methods. Collaborating with others require manual inspection of conflicting files, and Unreal Editor has tools for comparing differences between assets. (Unreal Engine - Source Control.)

The project for the data visualisation application was created from one of the Unreal Engine template projects. This gave a foundation for the application as many features such as the movement of the camera and VR integration were already functional. Excessive and unnecessary features were removed, and the project configuration (Figure 7) was adjusted to work better for data visualisation purposes. Unreal Editor includes a sizeable quantity of adjustable rendering options. One important adjustable option is the presence of forward shading. Forward shading adjusts the way Unreal Engine internally renders the scene and Unreal Engine documentation states that this method of rendering gives typically faster results (Unreal Engine - Forward Shading). This feature is experimental in Unreal Engine version 4.20.

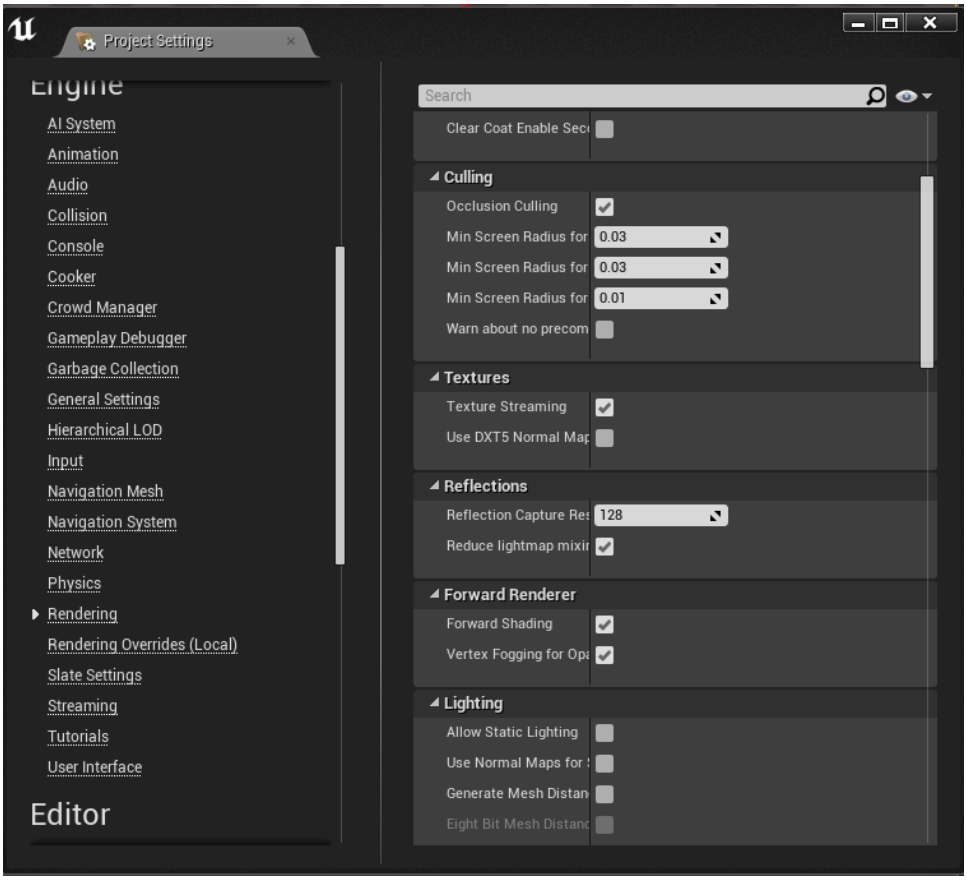


Figure 7. Unreal Editor project settings view

5.2.3 Programming

Unreal Engine 4 uses an actor-component based design. Actors are tangible objects that can exist in the world, while the shape and much of the behaviour of these actors is provided by including one or more components in the actor. The actor-component-system is hierarchical in the sense that components can contain other components. (Unreal Engine – Actors and Geometry; Unreal Engine – Components.)

Both actors and components can be programmed to include behaviour and event handlers. Events that can be handled include collision events, input events and many others. Actors and components that require periodic updates can use the engine feature known as *ticking*: actors and components are ticked at regular intervals, which in practice means executing a piece of code inside the actor or component in question. (Unreal Engine – Events.)

5.2.4 Blueprint scripts

Unreal Engine 4 Blueprints Visual Scripting, or simply *Blueprints*, is a visual programming method that can be used to build game logic without writing a line of code (Figure 8). This can be a useful tool for artists and developers who are not familiar with C++ programming. Additionally, the blueprints system is integrated directly into the Unreal Editor. They can be used, edited, tested and debugged without any external tools, which can be a very streamlined and user-friendly way to program additional features. As the features grow more complex, the Blueprint scripts might become visually cluttered and unmaintainable. When a more complex feature is required, it should be programmed in C++ and its interface exposed to the Blueprint scripts.

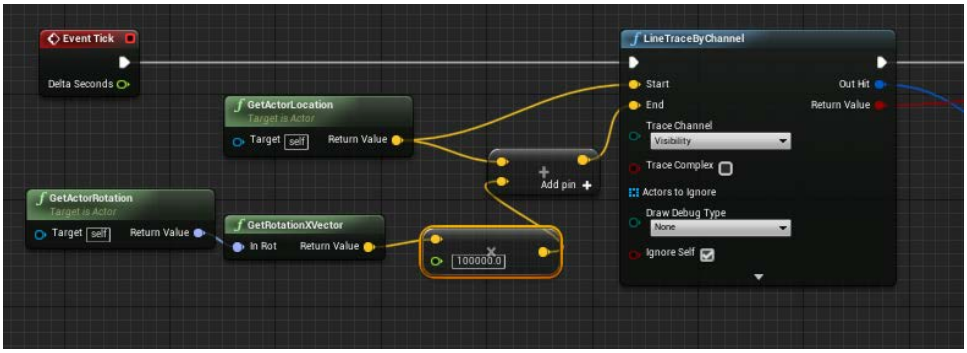


Figure 8. Blueprints Visual Scripting example script

Blueprint scripts are created by connecting nodes with *wires*. Wires indicate the flow of data inside the script. The program execution flow is also visualised with wires. In the figure (Figure 8) the node *Event Tick* at the top left corner is connected to node *LineTraceByChannel* at right with a white *execution wire*: whenever Event Tick is triggered, function LineTraceByChannel is called. The parameters to the function call are provided by other nodes connected below the white execution wire. Function calls can also be sequenced by connecting functions to each other with execution wires.

5.2.5 Using C++

Unreal Engine 4 has its own flavour of C++, giving the user many utilities to use across a project. While the use of the C++ standard library is possible, Unreal Engine provides alternative versions of many features to accommodate additional engine features. Unreal Engine 4 also extends C++ with a feature known as reflection: the

ability to inspect the compiled program during runtime. Unreal Engine utilizes reflection in memory management, the internal object model used by Unreal Engine and many other features (Noland 2014). Many of the new features are provided by Unreal Engine's own build system, the UnrealBuildTool and UnrealHeaderTool (Unreal Engine – Build Tools).

Manually building Unreal Engine 4 C++ code via the shell requires knowledge how to use and utilize the UnrealBuildTool. This can be avoided by using the variety of project files Unreal Engine automatically generates. If the project files were not generated or if they are out of date, they can be regenerated with the following piece of shell script:

```
# Replace file names and paths as necessary. Note that Un-
realBuildTool
# prefers absolute path names instead of relative path names.

mono "/home/user/UnrealEngine/Engine/Binaries/DotNET/Un-
realBuildTool.exe" -projectfiles -game -engine "/home/user/My-
Project/MyProject.uproject"
```

The generated project files will appear in the project root directory. These project files can be used to build the project source code outside of Unreal Engine editor, which allows more flexible development. For example, to use the generated Makefile project can invoke the make command in the project root directory:

```
make MyProject-Linux-Debug
```

This command builds a Linux debug build of *MyProject* internals. Note that when building the project source code outside of Unreal Engine editor in a Linux environment, the changes are not visible immediately to the editor. The editor uses a technique known as *hot-reloading* to load newly compiled code to replace the original. When the code is built outside of the editor, there is no trigger to start hot-reloading the freshly built code, which can lead to some peculiarities when the hot-reload is finally activated.

Unreal Engine user can use an integrated development environment to aid in development. Project files for Qt Creator, KDevelop and other development environments are generated alongside the Makefile. (Unreal Engine - How to Set up Qt Creator for UE4.)

5.2.6 Debugging Unreal Engine

As with all software development, debugging the Unreal Engine project should be possible in some fashion. For Blueprints, the engine provides easy and very visual debugging tools. For C++ code, the developer must rely on third party debuggers or more Spartan methods, such as logging.

The nature of Unreal Engine editor can cause some frustration when bugs in C++ code appear, as the editor itself will be compromised when flaws exist in project code. This is explained by the fact that the editor runs the project code directly instead of spawning a new process. If a segmentation fault or a memory corruption occurs in the project code, it usually manifests as a crash of the whole editor. If the faulty code is in the constructor of an actor or a component, it can cause the editor to crash immediately after loading the project. This happens because the editor must instantiate the object in order to display them in various editor viewports. There exists a possibility to launch the project in a separate process to avoid some of these problems; however, that includes some start-up time and is not optimal for fast testing cycles.

5.2.7 Packaging the project

The project can be built as a standalone executable via the Package command. This can be found in the File menu of the editor. This build process may take a very long time if the engine must rebuild all assets and project code. Compiling materials for different rendering interfaces can be the main cause for long-lasting build processes. The developer can expedite this process by removing unused materials and disabling unneeded graphical options and rendering interfaces. After packaging, the project will be available as a standalone program in the selected directory.

5.2.8 Quirks in the Linux environment

The application was developed with somewhat experimental Linux version of Unreal Engine. This subjected the development process to some additional challenges such as compiling the whole engine from source or unexpected behaviour

There are some other challenges that surface on Linux environments, such as Unreal Engine exceeding the Linux per-user limits for open file handles, resulting in a sudden engine crash. On KDE desktop environments, Unreal Editor disables desktop compositor resulting in broken desktop toolbars and frozen system clock.

5.3 Virtual reality setup

While this section describes methods used specifically with HTC Vive headset and SteamVR, the development environment is not dependent on this exact setup. Unreal Engine provides automatic integration with other virtual reality providers, such as Oculus VR (Unreal Engine – Oculus Rift Quick Start).

The Vive environment must be prepared before VR development can be started. After setting up the base stations and connecting the headset to the computer as described in the setup guide, the only requirement on the system is a functional SteamVR installation. This can be done by installing it with the Steam client, which requires a Steam user account.

After installing SteamVR, the user should perform some basic configuration of the head set. This includes running the *room setup*, a process which measures the dimensions of the room. This phase is important to perform correctly, otherwise the users may find themselves standing too tall, too short, or dislocated in the VR world. SteamVR also has the *Room Overview* utility, which displays a map of the room with VR devices marked in (Figure 9). This tool has some use when adjusting the positioning of the base stations to be optimal.

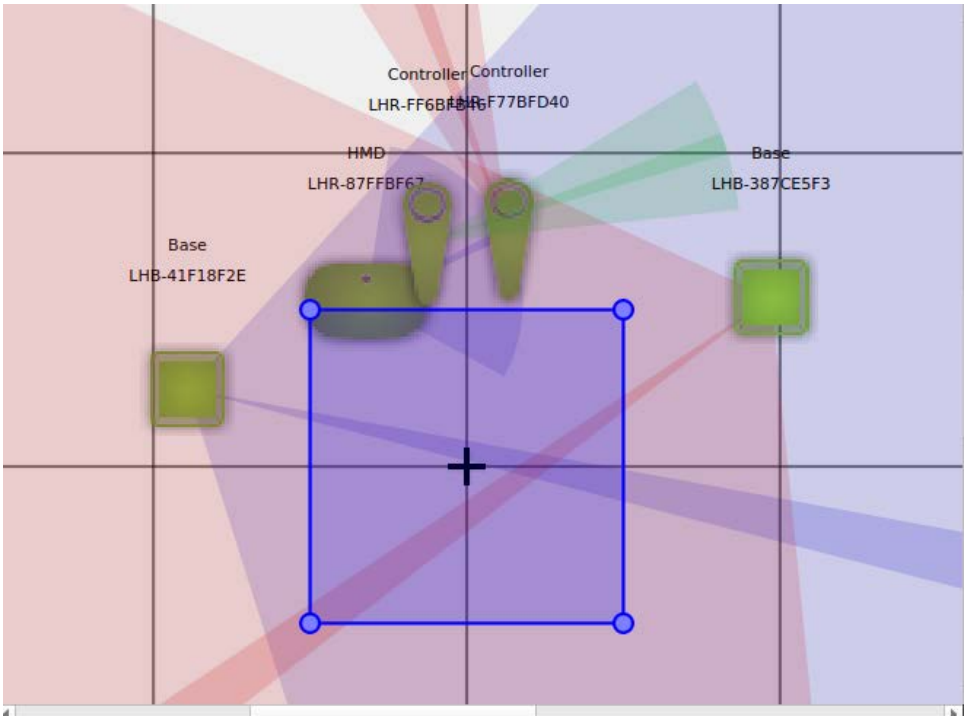


Figure 9. Steam VR Room Overview

In Linux systems, some special care must be taken when running SteamVR. SteamVR requires special permissions for accessing the headset's USB devices and the graphics card drivers should be of specific version range for SteamVR to function. (SteamVR Release Notes and Known Issues 2018.)

When using different VR headsets with different kinds of controllers, problems inevitably arise from controller layout incompatibilities. For example, Oculus Rift controllers do not have a button that acts like the menu button on the Vive controllers. In many cases, this is solved by taking multiple controller types into consideration when developing the application. Alternatively, SteamVR provides a tool for controller rebinding that the end user may utilise (Ludwig 2018).

5.4 Architecture

This chapter explains the overall program architecture of the data visualisation program (See Appendix 2).

5.4.1 Components

The project consists of four primary areas of interest: *the player character*, *central logic*, *controlling data* and *displaying data*.

The virtual reality camera and controls are implemented in the blueprint class FirstPersonCharacter. This class controls the primary ways of user interaction. Much of the simple logic pertaining to direct user interaction is provided here. This is also one of the few Blueprint classes in the project: features that are too complex to be implemented in Blueprints are consolidated into C++ classes.

The central logic is implemented in the C++ class *DotLoader*. This class is the central monolithic amalgamation of features, many of which could be split into separate classes. Many features that were too complex to include in blueprints were included here, such as VR interaction with multiple controllers. This class also works as an interface between the data and various user interfaces.

Displaying the data set is performed in the C++ class *DotRunner*. Available interfaces mainly control the displayed data set columns and other rendering related properties. This class also contains a single instance of the C++ class *DotMeshComponent* that is responsible for generating the mesh for display from the data set.

Controlling data refers to multiple individual systems that contain and move data set related information. There are three C++ classes for this: *ScalarDataTable*, *ColumnCollection* and *ColumnRedirector.* These three classes represent the way the data is stored in the program. ScalarDataTable is the raw base type for a data table while ColumnCollection is an abstraction over ScalarDataTable. ColumnRedirector is a system for joining multiple ScalarDataTable instances together. This design allows a programmer to easily add new data columns. Some extra columns were implemented this way, such as the *Index* column, which simply contains the row-index of the data point.

5.4.2 Central logic and the DotLoader class

As mentioned in the previous chapter, the *DotLoader* class is a monolithic beast that has many duties that overlap with other components in the program. It works as a central nexus for most of communication within the application, handling much of the user interaction code and providing the data set to different parts of the program.

The much of the data set lifetime is handled inside the DotLoader class. Querying the user for data set location, reading the data, performing relevant pre-processing and updating the data set all are responsibilities of the class, and they are performed either directly inside the class or with the aid of a utility class.

The DotLoader class also includes the components required to display the entire user interface and the data visualisation itself. Runtime generating widgets, constructing and managing resources involved in the user interface components, moving and manipulating the displayed data visualisation and positioning some auxiliary display hints are all tasks implemented in the class.

5.4.3 Multithreading implementation

As the development of this thesis work proceeded, it became apparent that the application would be very demanding CPU performance wise, especially when running t-SNE algorithm or when building the correlation coefficient matrix. It is natural for these algorithms to be quite slow and performance intensive; the problem arises from these tasks interfering with the real time VR experience the application must provide. In Unreal Engine, the user interaction and logic of the application are run in the same thread known as the *game thread*. Running even a single step of t-SNE in the same thread alongside the VR simulation proved to be too slow to maintain adequate performance. The solution is to separate the heavy processing tasks into separate threads.

As multithreading can be confusing and dangerous (See Chapter 4.4), a few helper classes were built to contain the lower level usage of multithreading primitives. These include thread safe containers and scope-based locks utilizing C++ RAII that minimize the possibility of race conditions and deadlocks. A generic C++ container class *Mailbox* is an implemented manifestation of the thread safety techniques described. The Mailbox class contains only a single method *Acquire*. The Acquire method returns a *MailboxLock* instance, which is the only way the resource inside the container can be accessed. The class uses RAII principles to ensure thread safety: when MailboxLock is instantiated, a lock is acquired and when the lifetime of a MailboxLock instance ends, the lock is automatically released. These behaviours ensure

that there can exist only one MailboxLock for each Mailbox at a time, meaning that the resource inside the Mailbox is accessible to only one thread at a time.

Multithreaded tasks are handled in *MessagingRunner* class, a sub-class of Unreal Engine *FRunnable* class. *FRunnable* is the simplest interface in the engine for a task that can be run asynchronously, off-thread. The MessagingRunner class has some utilities designed around data processing and moving large quantities of data points. They are used for computationally heavy and long-lasting calculations, such as t-SNE or calculating correlations. They also are useful when loading large files from disk to avoid program freezing while operating large files or slow storage devices. *TSNERunner*, *DataSetRunner* and *StatisticsRunner* classes shown in the architecture diagram (See Appendix 2) are subclasses of MessagingRunner.

MessagingRunner class and its subclasses are designed to function in incremental and interruptible fashion. This is accomplished by dividing the task into smaller sections. In practice, this occurs in the virtual method *LoopStep* that executes one indivisible piece of work and then returns. After each call to this function the appropriate checks are performed whether to interrupt the thread or not.

5.5 Application sequence

In the most typical workflow of the program, at first the data set is queried from the user. This is triggered either by program start-up or with the user interface. This data set query occurs inside the DotLoader class. The result of this query is then forwarded to the DataSetRunner class that loads and parses the data set in another thread. The loaded data set is periodically requested by DotLoader in its tick function until it succeeds. After gaining access to the data set, a preliminary set of operations is performed on it, such as creating ColumnRedirector and ColumnCollection instances and spawning a DotRunner actor. The background threads for calculating correlation coefficients and t-SNE are launched.

After DotRunner is given access to the data set via the ColumnRedirector, it will generate the mesh on the next tick depending on the selected columns. The mesh is updated again every time the selections change, or if the data in the selected columns is updated, e.g. t-SNE advances to the next iteration.

All user interaction with the data starts from FirstPersonCharacter class. Input events are handled in the class Blueprints and forwarded to appropriate entities. The class is responsible for sending interaction events to the UI as well. Other than very simple algorithms, actual effects of the user interaction are implemented in C++ code inside the DotLoader class. There are a few cases when the UI event handlers directly communicate with the DotRunner instance behind the DotLoader. These exceptions relate directly to the graphical aspects of the visualisation such as the transparency of the data set.

The act of rotating, moving and scaling the data set with the motion controllers is handled by the DotLoader class. The class also handles the positioning of meshes and components that should appear glued to the data set. These include the graphical representation of spatial axes and the selected point indicator.

5.6 Virtual environment

In Unreal Engine, a displayed scene is contained in a *level*. Levels are built with Unreal Editor level editor that contains tools for designing the shape and behaviour of a scene. Placing assets and actors to the level is a matter of dragging them to the viewport. In this project, the level was built to be very simple: a flat world was built with the geometry tools included in the level editor. FirstPersonCharacter and DotRunner actors were instantiated directly in the level, along with a directional light entity to provide a lighting for the visualisation.

When designing the environment for VR use, methods were researched in avoiding motion sickness inside VR. Avoiding involuntary movement of the VR viewport and other objects was one design choice implemented. Techniques such as placing stationary props and adding a distant background inside the virtual world were tested. However, the actual advantages of these methods are debatable (Combating VR Sickness 2018). The decision was made to avoid visual cluttering as much as possible.

Turning a regular 3D project into a VR in Unreal Engine is a matter of setting up a few exceptions when VR is active. Camera movements should be disabled entirely in VR mode to ensure that the only thing moving the VR viewport is the user looking around. In the developed project, to move the virtual avatar around the VR world the

user must move physically, e.g. by walking. No other methods of movement exist in the developed application.

5.7 Loading data

Providing methods of loading data from a wide variety of sources was not a primary requirement of the application. For this reason, the data set loader only supports one data source: a *comma-separated values* file that is loaded from the computer's file system. CSV is a text-based data format where each row of data is represented by a row in the text file. Similarly, columns in the data set translate directly to columns in the text file. In CSV files, data row fields are separated by the comma symbol.

In the parser implemented in this project, the first row of the CSV file is assumed to be the header row, containing names or descriptions of the columns in the data set. When reading data rows, the only supported data type is decimal numbers. If a field is encountered with a non-numeric value, it is quietly replaced with a zero. The parser also supports two different separators for the CSV files: a comma and a semicolon.

The data set path is queried from the user via a typical file dialog UI found in most desktop operating systems. This file dialog halts all execution in the engine, which results in a sudden stop of the VR experience. Running the file dialog procedure in a separate thread would have been a solution to this possibly jarring problem; however, Unreal Engine does not allow the file dialog UI code to be executed outside of the game thread. After the user provides the path to the data set, another thread is launched. A file handle is opened, and the CSV data is read and parsed row by row. This procedure is contained wholly in a separate thread, in order to avoid possible input/output stalls, computational bottlenecks when parsing extremely large files or other errors depending on external factors. After the thread is finished in reading the data set, or failed in doing so, it will communicate back to the game thread with the new data set and possible error codes. The ownership of the loaded data is transmitted to the game thread and the loading thread is put to sleep.

5.8 Data processing

After the data is loaded, a small amount of pre-processing is performed to ease the usage of the data in other parts of the program. Simple per-column statistics such as minimums, maximums, means and standard deviations are computed. A variation of the data set is built where the numerical scale of every column is normalized between positive one and negative one, so the values can be visualised more easily. A correlation coefficient matrix is also calculated from the data.

Implementation of the algorithms for these calculations is a trivial matter when considering smaller data sets. In larger data sets care must be taken not to freeze the entire program during the calculations. Especially correlation coefficients can be very performance intensive. That is why a background thread is launched for correlation coefficient calculations. As with many other thread-based tasks, the background thread is designed to be interruptible to allow smoother operation when user wishes to cancel data processing, for example, when the user wishes to load a new data set. This is accomplished by splitting the correlation coefficient algorithm into smaller units instead of calculating the entire matrix in one go.

t-SNE was implemented by including a Barnes-Hut t-SNE C++ library directly into the project sources (See Barnes-Hut t-SNE). The included code was modified to support incremental stepping and easier memory management. The t-SNE is computed in a thread separate from the game thread because the algorithm can be very expensive computationally. Even though Barnes-Hut implementations of t-SNE can perform significantly better than regular t-SNE it can be extremely slow on large data sets. The result of t-SNE is shown continuously in the visualisation as the algorithm proceeds.

5.9 Motion controllers

Inside the virtual world of the developed application, the user can interact with the data set using the motion controllers. They can grab, rotate, pull, push or move the data set by using the trigger button on the controller. With two controllers, they can adjust the size of the data by moving the two controllers away or towards each other while the trigger buttons on both controllers are down. The motion controllers are

visible in the virtual world, so the user has visual feedback on the movement of the motion controllers (Figure 10).

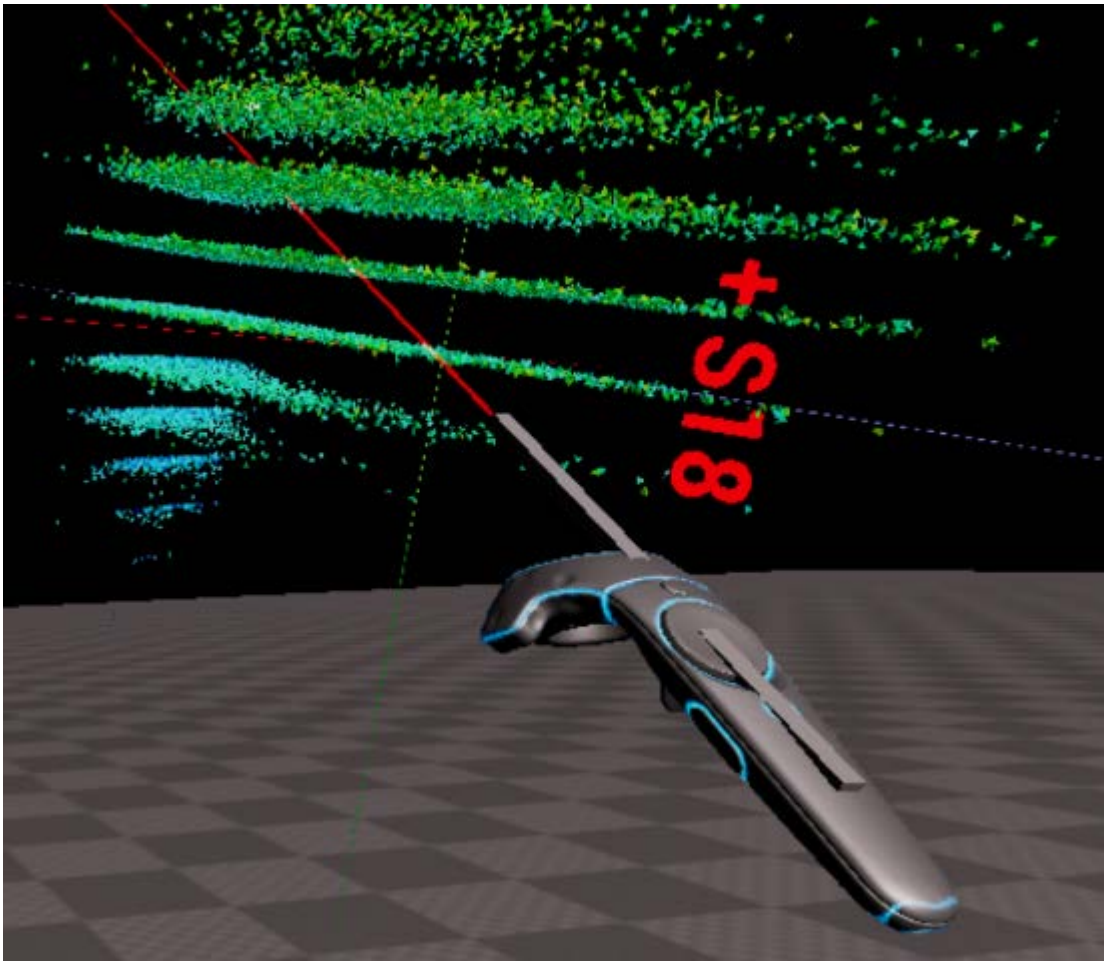


Figure 10. Interacting with the data set

With one controller, the movement of the controller is easy to utilize by simply taking the movement and rotation information of the controller and applying the information to the grabbed mesh. As the second controller enters the play, the algorithm must become more sophisticated to appear natural to the user. The specific solution that was developed for this thesis project is described here and it is by no means the only solution.

The application first finds a transformation that correctly transforms the old positions of the controller to their new positions. This transformation consists of movement, rotation and scaling. Movement through the space is simply the mean of the controllers' combined movement. Rotation is found by calculating the difference in angle between the two controllers. If the angle stays same, no rotation is performed.

The axis of rotation is the normal of the plane containing controllers' old and new relative positions. Scaling the data set is calculated by evaluating the difference in distances between the controllers. This transformation is then applied to the data set.

The translation of the data set is found via calculating the difference of controller centroids as shown in equation 1.

$$t = C_2 - C_1 \tag{1}$$

Where $t =$ translation vector

$C_1 =$ centroid of controllers' previous positions

$C_2 =$ centroid of controllers' new positions

Calculating the scale ratio of the dataset is found via the difference of distance between controls as illustrated in equation 2.

$$s = \frac{|A_2 - B_2|}{|A_1 - B_1|} \tag{2}$$

Where $s =$ scale factor

$A_1 =$ first controller's previous position

$B_1 =$ second controller's previous position

$A_2 =$ first controller's new position

$B_2 =$ second controller's new position

The rotating requires finding out the rotation axis. This can be done by finding the normal of a plane on which the un-translated movements of the controllers fall, which is shown in equation 3. Equation 4 is a method for acquiring the amount of rotation around this axis.

$$R = (A_1 - B_1) \times (A_2 - B_2) \tag{3}$$

Where $\times$ denotes cross product

$R =$ rotation axis

$$\Delta\alpha = \frac{A_1 - B_1}{|A_1 - B_1|} \cdot \frac{A_2 - B_2}{|A_2 - B_2|} \tag{4}$$

Where $\cdot$ denotes dot product

$\Delta\alpha$ = rotation in radians

The figure below (Figure 11) is a visualisation of the motion control algorithm. Points A_1 and B_1 denote the start points of controller one and two respectively, while points A_2 and B_2 are the end points of the two controllers. Points C_1 and C_2 are centroids of controller positions at both start and end of their travel.

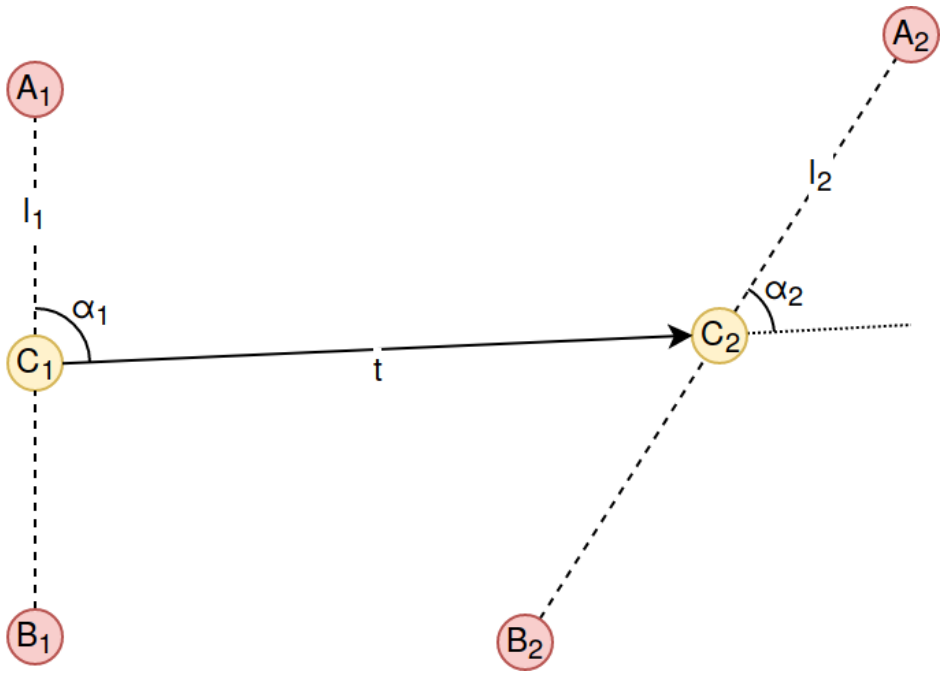


Figure 11. Motion controls algorithm

In Unreal Engine applying the rotation correctly requires adjusting component rotation pivot points. At the time of writing, there is no inbuilt way to simply rotate an actor or a component around a specified point. As a work around, an additional parent component can be created inside the actor, which then can be used as a pivot point by utilising component relative transformation and rotations.

The motion controller implementation inside Unreal Engine includes a feature known as *low latency update*. When enabled, this feature causes the known position of the motion controllers inside the game engine to be updated just before rendering the frame (UMotionControllerComponent). This minimizes the discrepancy between the real-world motion controllers and the motion controllers displayed on the display. As a side effect of this deferred update, there might be inconsistencies in the motion controller logic, as the position of the controllers are updated at the end of the frame

and the rest of the engine uses positional information gathered at the beginning of the frame. This might manifest as objects in the simulation lagging slightly behind the motion controllers.

5.10 User interaction

With the motion controllers, the user can move and rotate the visualisation. However, the question remains how the user might control what data is visualised and how the data is visualised.

5.10.1 Floating user interface

To provide the user the means of fine tuning, the settings of the visualisation, a floating user interface was included in the virtual world (Figure 12). One of the controllers acts as a pointer that can be used to interact with the UI. This pointer is visualised as a beam of laser emanating from the virtual reality controller. To utilize the pointer, the user points the laser beam at the target of the interaction and depresses one of the buttons on the controller.

The user interface is quite large and possibly distracting when the user wishes to focus on the data. For that reason, the user has the possibility to hide the user interface or move the user interface by pressing one of the buttons in the controller.

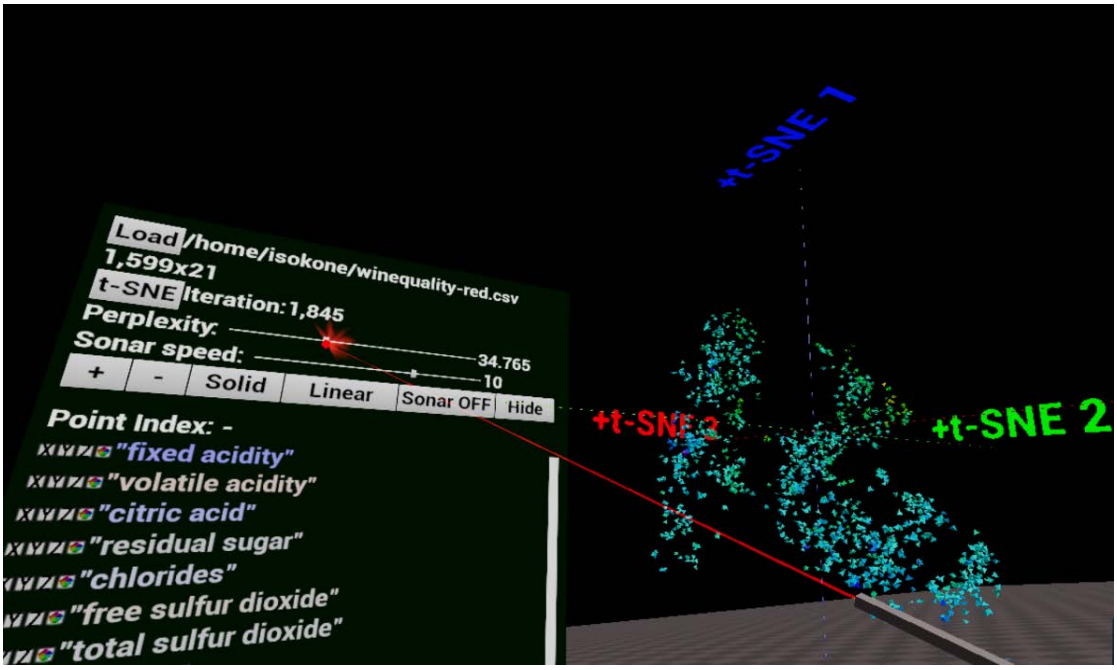


Figure 12. The floating user interface and the laser pointer

The floating user interface displays the variables of the data set in a list. An example of the variable list can be seen in the figure (Figure 12) in the lower left corner. The list of variables also contains a few extra items that do not exist in the original data set and are generated after loading the data set. Data point indices and t-SNE dimension reduction axes are represented with these new columns. The user can select a data point with the laser pointer and the values for that data point are displayed in the variable list, next to the corresponding variable names.

For each variable, there are four tick boxes that the user can tick, three for each spatial axis and one for colour. By changing these, the user can select which variables should be projected on what axes in the data visualisation. Projection type can also be changed from linear projection into a spherical projection. In spherical projection, two of the axes work as longitude and latitude and the third is the depth. This mapping is then overlaid on a spherical model of Earth (Figure 13).

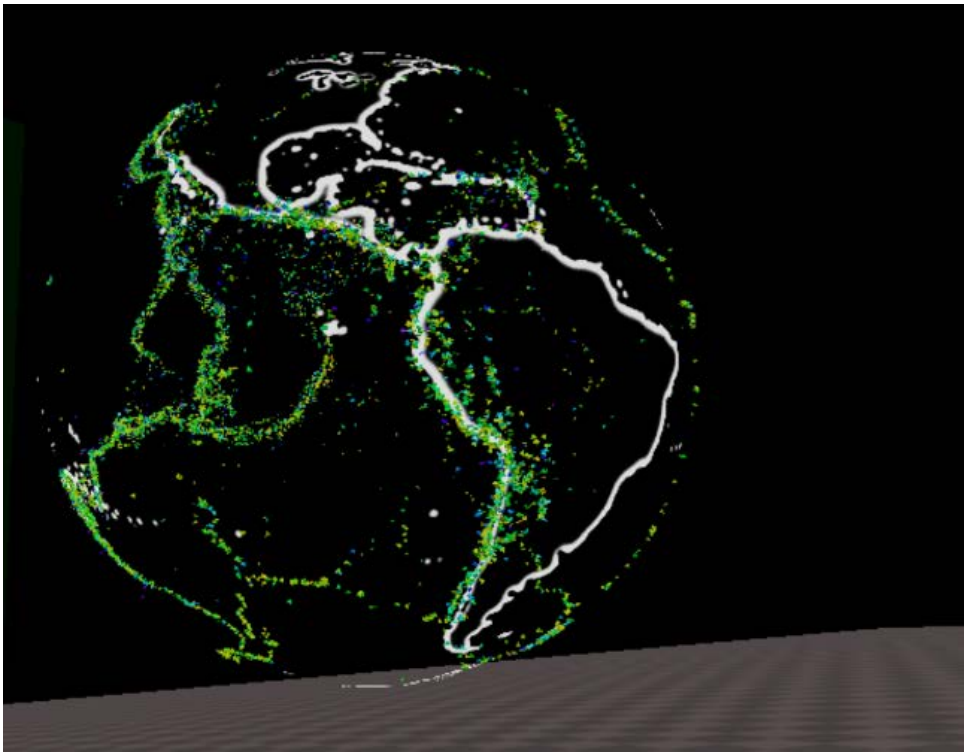


Figure 13. Data points mapped on a globe

5.10.2 Implementation

The flow of user interaction consists of multiple moving parts of the program working together. The laser pointer, which is the part closest to the user, is implemented in Blueprint scripts as it is a simple enough component. The laser pointer is attached to one of the motion controllers as a component.

The UI is interacted with an Unreal Engine inbuilt component, the Widget Interaction component. This component can simulate mouse clicks on user interfaces that exist in the game world. A Widget Interaction component is bound to the motion controls as another component, along with the laser pointer. As the user depresses the trigger, the Widget Interaction component is asked to perform a simulated mouse click on whatever thing it is pointed at. When selecting individual data points, Widget Interaction component is not used as the data mesh requires more sophisticated logic. After the trigger is pressed and the Laser Pointer is not pointing at the user interface, a method in the DotLoader instance is invoked, with the positional information of the laser pointer passed as arguments. This method then performs the necessary actions to check if a data point was selected.

Upon starting the program, the DotLoader class instance generates the required UWidgetComponent for the floating user interface. UWidgetComponent is a component used to render regular user interface widgets into the world as 3D planes. It is this component that is manipulated when moving the user interface or when completely hiding it.

The widgets displayed on the floating user interface are implemented in two parts: UI class and DataInspector class. UI class has all the interactable controls in the user interface, aside from the per-column widgets. DataInspector is used to display a list of data columns and the controls that are used to control which variables are visualised. This is accomplished with runtime widget generation each time a new data set is loaded.

The UI class is implemented with *Unreal Motion Graphics*, which is a user interface framework found in Unreal Engine. Unreal Editor includes a tool for editing and creating user interfaces with this framework (Figure 14). The tool integrates seamlessly with Blueprint scripts: scripts and event handlers for UI elements can be created very quickly. These were used to provide the desired functionality for the UI class.

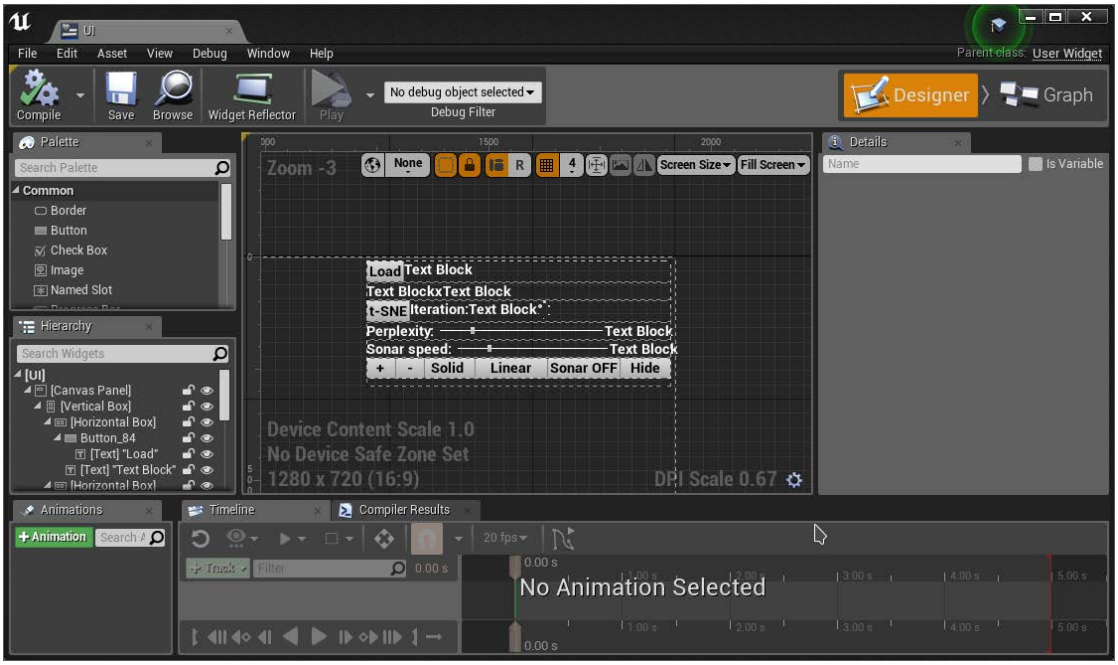


Figure 14. Unreal Motion Graphics UI designer

This runtime widget generation is achieved with C++ code that interacts with Unreal Motion Graphics UI and the Unreal Engine *Slate* UI framework. Slate UI is a lower

level framework for user interface development in Unreal Engine that has more comprehensive C++ support.

5.10.3 Dimensional sonar

The application includes a tool that can be used to measure the distance of data points along a certain axis. This feature can be activated from the primary user interface. In practice, the user first selects the sweep dimension from the user interface. After that the user can trigger the sonar by clicking on a data point which will immediately trigger a pulse originating from that point. The pulse will sweep through the selected dimension over time, and as other data points are encountered by the pulse, they are highlighted by a small flash in the visualisation (Figure 15). In other words, the sonar visualises distances along the selected dimension between points by highlighting the data points after a certain amount of time, proportional to the distance, has passed.

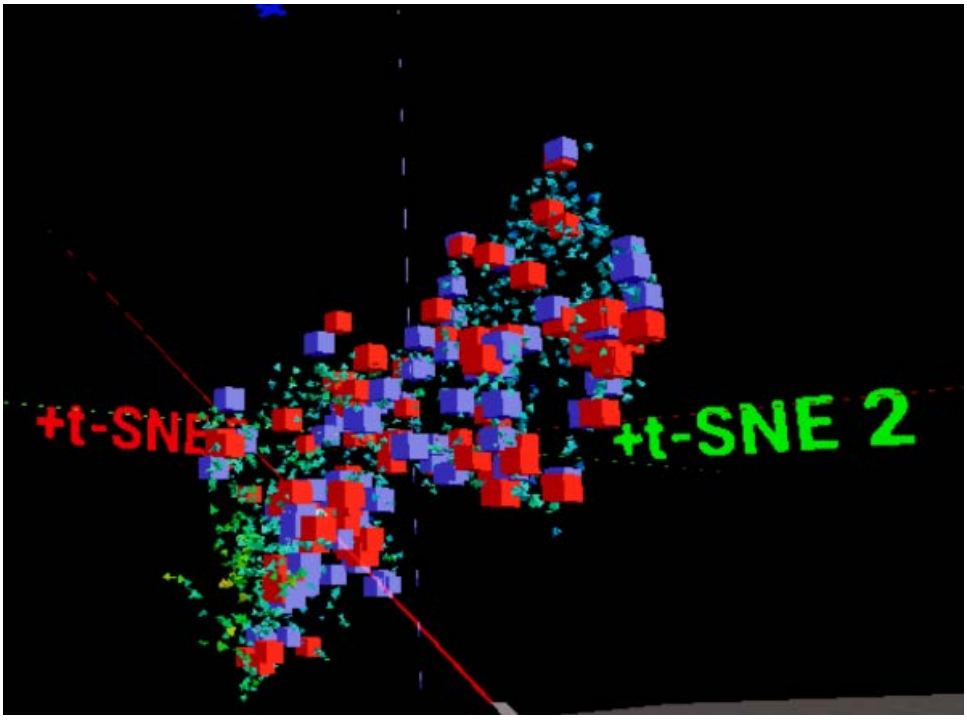


Figure 15. Data points highlighted by the dimensional sonar

This tool can be useful when exploring how a fourth variable is distributed among three other variables. The pulsing points can appear as waves that move through the data set. This may indicate a presence of a pattern in the data set.

5.11 Procedural mesh generation

To display the data points, there are many choices such as particle systems, specialized shaders or generated meshes. Mesh generation was utilised in this case.

The mesh generator's duty is to transform a list of data points into a three-dimensional mesh (Figure 16) consisting solely of triangles. In this case, a tetrahedron shape was generated with certain colour for each data point. The shape of tetrahedron was chosen for it being the simplest polyhedron there is, containing only four triangular faces and four vertices, and it has vaguely uniform silhouette regardless of viewing direction. The mesh generator iterates through the data set and generates a single tetrahedron for each data point. The tetrahedrons are positioned to the origins of the data points. Per-vertex colour and normal information is also generated along the positional information.

The generated tetrahedrons are randomly rotated as a part of the generation process. This is done to allow human eyes to distinguish between two data points more easily: uniformly rotated tetrahedrons tend to blend together as their faces are shaded with equal amounts of light. However, rotated tetrahedrons give a much rougher look, which may look less visually appealing.

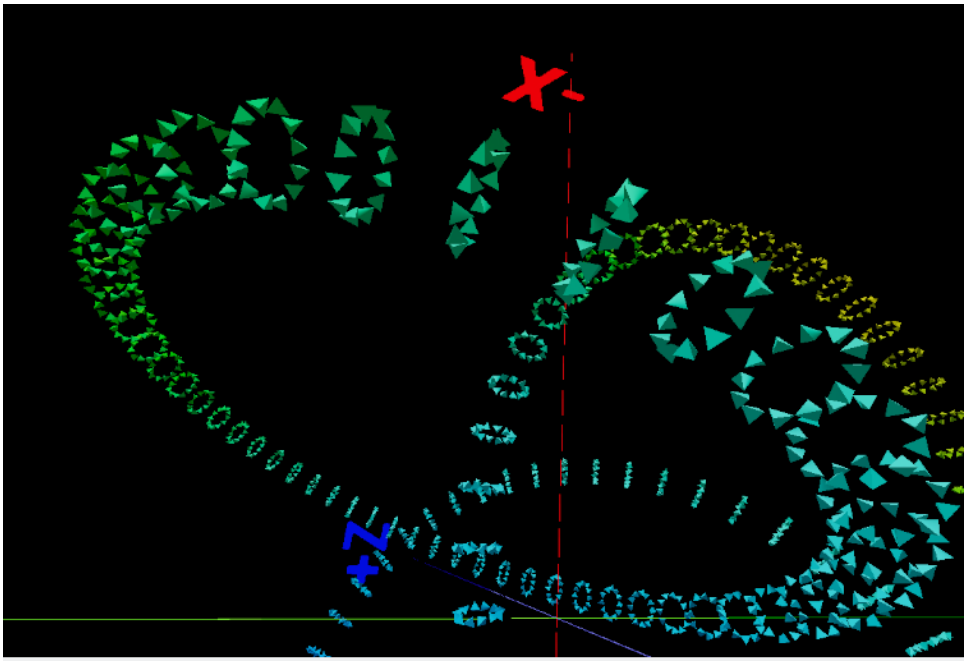


Figure 16. Close-up on the generated visualisation

There are many choices regarding the appearance of the points. The generated mesh can be given a slightly translucent appearance to allow the user to see through the points and better perceive when many points are situated very closely. Additive blending can also be used to make clustered parts of the mesh appear brighter, giving a sense of density to the viewer. On the other hand, translucent objects may be harder for eyes to focus on when looking through VR. A lightly shaded opaque material was chosen for the data points when running the default settings of the visualisation application.

5.11.1 In Unreal Engine

Unreal Engine provides graphical programming abstractions for many cases. Procedural mesh generation was supported with the *UProceduralMeshComponent*. This component provided an easy gateway for creating and updating a graphical model from nothing but a set of numerical values.

The mesh generator was designed to work performantly and in real time to accommodate possibilities for the visualisation to change in real time. This became especially useful when applying t-SNE to the data set, which mutates the data set to a new form at every step. Precomputing the rotated tetrahedron meshes into one large lookup table was utilized to improve performance. However, actual profiling results indicate much of the execution time occurs inside Unreal Engine internals for loading the mesh and calculating a collision shape for the mesh. When the size of data set exceeded tens of thousands of points, the mesh generator could not be run in real time with the test setup (See Chapter 6.1).

5.11.2 Alternatives

Single generated large mesh of tetrahedrons can hardly be considered optimal for data visualisation: the data points appear as rough and pointy polyhedra instead of more appealing perfectly round spheres. Many other choices exist that would be more performant, more visually appealing or both. These ideas were left to be implemented in future development as Unreal Engine 4 did not provide quick and easy ways to prototype the ideas.

Billboarding is the method of rendering a single image in the space, which always faces the camera regardless of the perspective it is viewed from. Billboards could be used to render the data points as small images. Ideally, this would be combined with *geometry shaders*. Geometry shaders can be used to build geometry on the GPU, instead of having to compute mesh vertices on CPU side. (Particle Billboarding with the Geometry Shader 2014.)

A technique known as *impostors* could be used to fake three-dimensional appearance with two-dimensional flat billboards. This could be achieved with *fragment shaders*, a type of GPU program that controls the colour and appearance of individual pixels. In combination with lighting information of the scene and the mathematical properties of a sphere, fragment shaders could be used to generate the appearance of a three-dimensional sphere on the billboard. (McKesson 2012.)

Level of detail systems and space-partitioning schemes could be used to improve efficiency. For example, if a cluster of points is viewed from far away, it could be rendered as a single large point instead of multiple small points. This would trade a negligible loss in visual quality for more performant rendering. However, some pre-processing must be done to achieve this. Space-partitioning trees must be rebuilt every time the display data points change, which could be a resource intensive operation especially for a real time application.

A way for rendering many billboard sprites efficiently in Unreal Engine 4 might exist, however, the writer of this thesis did not find any means to do so, other than particle systems which have no support for manually placing particles into desired positions. Geometry shaders remain an undocumented system in Unreal Engine 4. Another possible implementation for rendering data points would be instanced meshes. Unreal Engine 4 has a system for this; however, adjusting per-point colour and other properties might prove more difficult with this system.

6 Evaluation

6.1 Performance

When creating VR applications good performance is required. The application must maintain a steady frame rate of at least 90 frames per second to provide a good user experience. Lower frame rates can be unpleasant and even nauseating to the user. (The Importance of Frame Rates.)

This paragraph describes the machine used to test the application. The computer was ASUS ROG G20CB model machine with Intel® Core™ i5-6400 processor at 2.70 GHz. The random-access memory consisted of two 8GiB Samsung SODIMM DDR4 Synchronous 2133 MHz M471A1K43BB0-CPB memory modules. The GPU used was NVIDIA GP104 GeForce GTX 1070. The application and the data sets used were located on a Micron 1100 MTFD 256GB solid state drive. The test machine had Ubuntu LTS 18.04.1 installed and the application was built with Unreal Engine 4.20.2.

Application performance was measured with performance profiling tools included in Unreal Engine and SteamVR (See Chapter 4.3). When tested on the test machine, the program performed adequately, reaching 90 FPS, with most of the smaller data sets ranging from 50 to 2,000 data points and with 4 to 20 variables. When larger datasets up to 20,000 data points were introduced, updating the visualised data points to new relative positions caused frames to be skipped due to the mesh generation. With datasets larger than 20,000 points, the application fails to maintain steady frame rate, depending what exactly is visible on the screen. When a data set with a very large variable count, 500 or more is loaded, another performance bottleneck is encountered. The user interface of the application can become very sluggish and activating user interface elements cause notable hitches in the frame rate.

Three bottlenecks were discovered during the testing: rendering the data visualisation, generating the data visualisation mesh and Unreal Engine Slate UI. The rendering bottleneck comes from the nature of the visualised mesh and how it is drawn on the GPU. Generating the data visualisation mesh is closely related to the rendering bottleneck problem, as different rendering methods may not require mesh at all. However, the mesh generation also includes generation of the collision

information, which is vital when using the pointer tool to select data in the application. The Slate UI problem relates to the fact how the UI framework works and how the UI elements are generated. With a very large variable count, the UI becomes big enough to cause slowdowns.

The Slate UI problem requires more fundamental changes to the UI and how it operates, while the other two of the described bottlenecks could be solved by implementing more sophisticated rendering and collision detection algorithms. Many alternative rendering methods for large point mass exist (See Chapter 5.11.2), and evaluation of these methods could lead to a far more efficient rendering method. Collision detection for the visualised mesh could be implemented with a tree based algorithm (Collision Detection Part 6). These algorithms could be implemented to work in external threads in order to maximize game thread performance.

6.2 Future development

6.2.1 Embedding a scripting language

Embedding an interpreted scripting language into the data visualisation application is one possible way to allow wider usability. In such case, a savvy user could program and expand the utility for their own personal needs without the need to always recompile the program from scratch. Of course, the application is only as extensible as the scripting interface allows it to be: embedding a scripting language and providing an easy to user interface that encompasses the whole program are by no means simple tasks.

A system for adding custom data columns into the program with external scripts was planned. This was to replace the t-SNE and other auxiliary columns and to allow users to run their own column generation scripts. A user could write and run a piece of code that performs some analysis on the data set, and the results would be immediately available to be viewed and inspected with the VR headset.

Python is a very popular scripting language and it is used for many different purposes, including machine learning and data analysis. A user-friendly scripting language like Python is much less cumbersome compared to C++, not to mention the

vast ecosystem of machine learning and data analysis libraries available to Python programmers that could be utilized for much greater development efficiency. Python would be the prime contender for the embedded scripting language for these reasons alone. (Is Python the most popular language for data science?.)

UnrealEnginePython is an Unreal Engine plugin that embeds Python into the engine and provides Python interfaces for many of the engine features (UnrealEnginePython). However, the embedding of Python was scrapped as linking of Unreal Engine and Python proved difficult in the development environment. Crashes occurred inside Python interpreter for no apparent reason and they proved extremely elusive and hard to track down. The exact point of the crash depended on other factors, such as what memory allocator was used. The exact reason for this incompatibility is unknown, and it might be caused by variety of factors, such as Unreal Engine and Python version. For these reasons, it was deemed that Python integration should be dropped and left for future development.

6.2.2 Handling large data sets

When the user selects a very large data set, the program will not function properly. Various degrees of sluggishness, stuttering and freezing will occur depending on the size of the data set (See Chapter 6.1).

Many of these problems relate to the rendering of the data points. Instead of directly improving the performance of the rendering, one possible solution would be to allow the user to pick a smaller range of data from the data set to inspect, instead of displaying the whole set at once. This could be coupled with a system that monitors the program performance and automatically adjusts the displayed data set size when performance degrades.

The data set pre-processing is generally an unnoticeable procedure, lasting less than a millisecond, however, with a large enough data set it will become a performance problem. All operations on the whole data set should be offloaded into separate threads to avoid stalls. Generally, any operation that has time complexity dependent on the data set dimensions, row or column count, is prone to slowdowns and care should be taken to minimize possible stalls.

6.2.3 Improving user interface

The floating user interface, while functional, is a very large and possibly intimidating piece of user interface. This could be split into smaller and less intrusive windows that would categorize information more sensibly.

6.2.4 Additional user agency

As it stands, the application allows the user to adjust the visualisation in very few ways: the user has three spatial axes, one colour axis and a few adjustable options. Shape, size, text and graphical labels are visual traits that could be used when analysing data. The colour axis could be split into two new axes, hue and lightness. Categorical variables and other non-numeric data types, such as text or image data, should be visualisable as well.

In the current version of the application, axes are always scaled to fit in the visualisation. This can break the visual meaning between two variables if they are scaled out of proportion. Giving users the power to adjust the scales of the variables would give the users more room for creating their visualisation. Additionally, it might be useful to allow adjustments to the colour scheme of the colour axis.

6.2.5 Optimizing data loading

Loading a data set can be one of the most complex systems in the application. Data buffering, database connections and error handling must be considered when building robust data pipelines. This thesis project only contains a loader for simple CSV files. While this might be adequate, having more flexible data sources could expedite the visualisation process. Data bases, communications and networked transportation of data were subjects of research when designing the application. The embedding of a script language would be especially useful (See Chapter 6.2.1), as the data set loading could be delegated into these external scripts to which users would have full access.

The user interface associated with loading of the data would have to be improved as well. In its current form, it halts the whole application when querying a new data set, breaking the VR experience of the user. A virtual reality interface for querying a new

data set would be more optimal as the user could use it without having to detach from virtual reality.

6.2.6 Improvements to dimension reduction algorithms

Whenever visualising large data sets with no immediately visible correlations, it may be useful to use dimension reduction algorithms or machine learning to provide insights into the large data set. In the thesis work, only t-SNE algorithm is implemented in the visualisation application. If the user wishes to use some other dimension reduction algorithm such as principal component analysis, or if the user needs to save the results of the dimension reduction to some database, the user must rely on external tools to achieve this. Integrating these improvements to the application would streamline the experience of analysing data.

7 Conclusion

The data visualisation application built during this thesis work fulfils the primary requirements that it was given; the application provides a way to visualise data in virtual reality. Not all wishes of the assignors could be satisfied and many ideas and possible improvements were left for later development.

The choice of Unreal Engine 4 provided an easy starting point for VR data visualisation program. Using available VR visualisation frameworks, such as VR-Viz, might have been a more prudent choice for quickly developing a visualisation program. However, building the visualisation engine from scratch provided a valuable experience in graphics, mathematical and general-purpose programming.

The project proceeded as planned and the integration of VR into the application proved much easier than expected. The development moved fluidly and rarely did a problem prove insidious enough to halt the whole development process. One example of these problems was encountered when embedding Python scripting language into the application. In the end, Python support was left as a future improvement.

The developed application might not bring anything groundbreaking to the VR data visualisation scene; however, it gives the basic ground for a data analyst to view the data and possibly gain insight. The few special tools integrated in the program might

be useful for data analysis: t-SNE, the dimensional sonar, and spherical projection all give new ways to view the data. The real-world effectiveness of the application remains to be seen. If not for anything else, the application can transform data into a more visually pleasing form.

References

A-Frame. N.d. Accessed 14.11.2018. Retrieved from https://aframe.io/

Ali, Z. & Bhaskar, S. B. 2016. Basic statistical tools in research and data analysis. Indian Journal of Anaesthesia, 60, 9, 662-669. Accessed 12.11.2018. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5037948

Barnes-Hut t-SNE. N.d. Accessed 19.11.2018. Retrieved from https://github.com/lvdmaaten/bhtsne/

Avoiding Deadlock. N.d. Oracle – Multithreaded Programming Guide. Accessed 12.11.2018. Retrieved from https://docs.oracle.com/cd/E19455-01/806-5257/6je9h0347/index.html

Batallé, J. 2013. An Introduction to Positional Tracking and Degrees of Freedom (DOF). Accessed 1.11.2018. Retrieved from https://www.roadtovr.com/introduction-positional-tracking-degrees-freedom-dof/

Bertrand, F. 2017. Project NEO: Virtual Reality Data Visualization. Accessed 13.11.2018. Retrieved from https://www.youtube.com/watch?v=myI4P9C34A0

Cashorali, T. 2018. Data Visualization in Virtual Reality: Useless Hype or Practical Application?. Accessed 6.11.2018. Retrieved from https://www.iianalytics.com/research/data-visualization-in-virtual-reality-useless-hype-or-practical-application

Collision Detection Part 6. N.d. Accessed 13.11.2018. Advanced Methods in Computer Graphics. Retrieved from http://what-when-how.com/advanced-methods-in-computer-graphics/collision-detection-advanced-methods-in-computer-graphics-part-6/

Combating VR Sickness. 2018. ARVI Blog - Combating VR Sickness: Debunking Myths and Learning What Really Works. Accessed 9.11.2018. Retrieved from https://vr.arvilab.com/blog/combating-vr-sickness-debunking-myths-and-learning-what-really-works

Data visualization beginner’s guide. N.d. Tableau - Data visualization beginner's guide: a definition, examples, and learning resources. Accessed 6.11.2018. Retrieved from https://www.tableau.com/learn/articles/data-visualization

DatavizVR Demo. 2016. Accessed 13.11.2018. Retrieved from https://store.steampowered.com/app/551960/DatavizVR_Demo/

Description of race conditions and deadlocks. 2012. Accessed 11.11.2018. Retrieved from https://support.microsoft.com/en-ca/help/317723/description-of-race-conditions-and-deadlocks

Downloading Unreal Engine Source Code. N.d. Accessed 13.11.2018. Retrieved from https://docs.unrealengine.com/en-us/GettingStarted/DownloadingUnrealEngine

Few, S. N.d. The Encyclopedia of Human-Computer Interaction, 2nd Ed. 35. Data Visualization for Human Perception. Accessed 14.11.2018. Retrieved from

https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception

Hackathorn, R. 2015. Is Virtual Reality Useful for Data Visualization?. Accessed 14.11.2018. Retrieved from https://www.immersiveanalytics.com/2015/10/is-virtual-reality-useful-for-data-visualization/

Head-Mounted Displays. N.d. Accessed 2.11.2018. Retrieved from https://www.vrs.org.uk/virtual-reality-gear/head-mounted-displays/

History of Virtual Reality. N.d. Accessed 2.11.2018. Retrieved from https://www.vrs.org.uk/virtual-reality/history.html

Installing Unreal Engine. N.d. Accessed 13.11.2018. Retrieved from https://docs.unrealengine.com/en-US/GettingStarted/Installation

Is Python the most popular language for data science?. N.d. Maruti Techlabs blog. Accessed 13.11.2018. Retrieved from https://www.marutitech.com/python-data-science/

LaViola Jr., J. 2000. A Discussion of Cybersickness in Virtual Environments. SIGCHI Bulletin, 32, 1, 47-56. Accessed 9.11.2018. Retrieved from http://bulletin.sigchi.org/2000/january/articles/laviolapaper.pdf

Ludwig, J. 2018. Controllers Controllers Controllers: Introducing SteamVR Input. Accessed 13.12.2018. Retrieved from https://steamcommunity.com/games/250820/announcements/detail/3809361199426010680

McKesson, J. 2012. Learning Modern 3D Graphics Programming – Lies and Impostors. Accessed 1.11.2018. Retrieved from https://paroj.github.io/gltut/Illumination/Tutorial%2013.html

Noland, M. 2014. Unreal Property System (Reflection). Accessed 5.11.2018. Retrieved from https://www.unrealengine.com/en-US/blog/unreal-property-system-reflection

OpenVR SDK. N.d. Accessed 12.11.2018. Retrieved from https://github.com/ValveSoftware/openvr

OpenVR - API Documentation. N.d. Accessed 19.11.2018. Retrieved from https://github.com/ValveSoftware/openvr/wiki/API-Documentation

Overview of Positional Tracking Technologies for Virtual Reality. 2014. Accessed 1.11.2018. Retrieved from https://www.roadtovr.com/overview-of-positional-tracking-technologies-virtual-reality/

Particle Billboarding with the Geometry Shader. 2014. Accessed 1.11.2018. Retrieved from https://www.geeks3d.com/20140815/particle-billboarding-with-the-geometry-shader-glsl/

Pearson Correlations – Quick Introduction. N.d. Accessed 12.11.2018. Retrieved from https://www.spss-tutorials.com/pearson-correlation-coefficient/

RAII. 2017. C++ Reference. Accessed 2.12.2018. Retrieved from https://en.cppreference.com/w/cpp/language/raii

SteamVR Release Notes and Known Issues. 2018. Accessed 13.11.2018. Retrieved from https://github.com/ValveSoftware/SteamVR-for-Linux/blob/master/README.md

The Importance of Frame Rates. N.d. Iris VR article. Accessed 3.11.2018. Retrieved from https://help.irisvr.com/hc/en-us/articles/215884547-The-Importance-of-Frame-Rates

This is Unreal. 2016. Accessed 1.11.2018. Retrieved from http://hotgates.eu/this-is-unreal/

UMotionControllerComponent. N.d. Unreal Engine 4 Documentation. Accessed 5.11.2018. Retrieved from https://api.unrealengine.com/INT/API/Runtime/HeadMountedDisplay/UMotionControllerComponent/index.html

Unreal Engine – Actors and Geometry. N.d. Accessed 5.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Engine/Actors

Unreal Engine – Build Tools. N.d. Accessed 5.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Programming/BuildTools

Unreal Engine – Components. N.d. Accessed 5.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Engine/Components

Unreal Engine – Events. N.d. Accessed 5.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Engine/Blueprints/UserGuide/Events

Unreal Engine – How to Set up Qt Creator for UE4. N.d. Accessed 13.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Platforms/Linux/BeginnerLinuxDeveloper/SettingUpQtCreator

Unreal Engine – Oculus Rift Quick Start. N.d. Accessed 13.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Platforms/Oculus/QuickStart

Unreal Engine – Source Control. N.d. Accessed 07.11.2018. Retrieved from https://docs.unrealengine.com/en-US/Engine/UI/SourceControl

UnrealEnginePython. N.d. Accessed 15.11.2018. Retrieved from https://github.com/20tab/UnrealEnginePython

Van der Maaten, L. 2013. Barnes-Hut-SNE. Accessed 12.11.2018. Retrieved from https://arxiv.org/pdf/1301.3342.pdf

Van der Maaten, L. 2018. t-SNE. Accessed 5.11.2018. Retrieved from https://lvdmaaten.github.io/tsne/

Virtual Reality Best Practices. N.d. Accessed 9.11.2018. Retrieved from https://docs.unrealengine.com/en-us/Platforms/VR/ContentSetup

Vive VR System. N.d. Accessed 9.11.2018. Retrieved from https://www.vive.com/us/product/vive-virtual-reality-system/

VR-Viz. N.d. Accessed 14.11.2018. Retrieved from https://vr-viz.netlify.com/

Warfel, E. 2016. Everything Wrong with Traditional Data Visualization and How VR is Poised to Fix It. Accessed 14.11.2018. Retrieved from https://www.roadtovr.com/everything-wrong-with-traditional-data-visualization-and-how-vr-is-poised-to-fix-it/

WebVR. N.d. Accessed 14.11.2018. Retrieved from https://webvr.info/

What is Virtual Reality?. N.d. Accessed 2.11.2018. Retrieved from https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html

Wilke, C. N.d. Fundamentals of Data Visualization - Don’t go 3D. Accessed 12.11.2018. Retrieved from https://serialmentor.com/dataviz/no-3d.html

Appendices

Appendix 1. User guide

This documentation describes steps that must be taken to operate the VR data visualisation tool successfully and without much pain.

Running the program

1. Set-up your VR system
2. Install SteamVR if not already installed
3. Start SteamVR
4. Start the data visualisation program

Loading a data set

When the program launches, it automatically opens the load dataset file dialog. This file dialog is used to specify a path to a CSV file containing the data to be visualised. Note that this file dialog is not visible in the head-mounted display and is only shown on the normal desktop display of the computer. After specifying the path, the program is fully started, and the user may use the head-mounted display to look around in the virtual world.

Operating the program

Inside the virtual world, the data set is visible as a cluster of points surrounded by three axes. To manipulate the environments the user must use the motion controllers. The user can grab the data set using the trigger on the controllers. Subsequently, the user can also move, twist and scale the data set with two controllers by grabbing the data with both simultaneously. Upon pressing the grip button, the data set is immediately moved close to the controller.

Floating user interface

The floating user interface contains widgets the user can utilize in their data visualisation forays. If the user interface is not visible, the user can press the menu button on the controller to summon the user interface in front of the controller.

One of the motion controllers has a laser pointer, which may be used to interact with the UI elements by pressing the touchpad button on the controller.

Here follows a description of different buttons in the User Interface:

Load

This button triggers the load dataset dialog. When the user presses this button, the VR experience is halted until a new data set is chosen. Refer to section Loading a data set for instructions how to perform this.

Right of the load button the file name of currently selected data set is displayed.

t-SNE and Perplexity

The t-SNE button activates t-SNE with the perplexity the user has selected by using the Perplexity slider. After t-SNE is activated, it will display the current t-SNE iteration after the t-SNE button. Refer to section t-SNE for explanation and further instructions.

Sonar speed

The Sonar speed slider controls the sweep speed of the dimensional sonar in seconds. Refer to section Dimensional sonar for further instructions.

+ and **-**

These buttons respectively increment or decrement the size of individual dots in the data set visualisation

Solid/Alpha

This button controls the translucency of the data visualisation. The button may be labelled either Solid or Alpha depending on the current setting. When the button is labelled Solid, the data set is displayed as opaque points. When in Alpha mode, the data set is lightly translucent. Clicking the button changes between these two modes.

Linear/Spherical

This button controls the projection mode of the data visualisation. The button may be labelled either Linear or Spherical depending on the current setting. When the

button is labelled Linear, the data set is projected into the 3D-space with a linear projection. When in Spherical mode, the data set is projected on a surface of a globe displaying geographical features of Earth. In Spherical projection mode, the X axis represents point longitude, Y axis represents point latitude and Z axis is point depth.

Sonar

This button controls whether the dimensional sonar is active. The sonar is active when this button is labelled Sonar ON. Refer to section Dimensional sonar for further instructions.

Hide

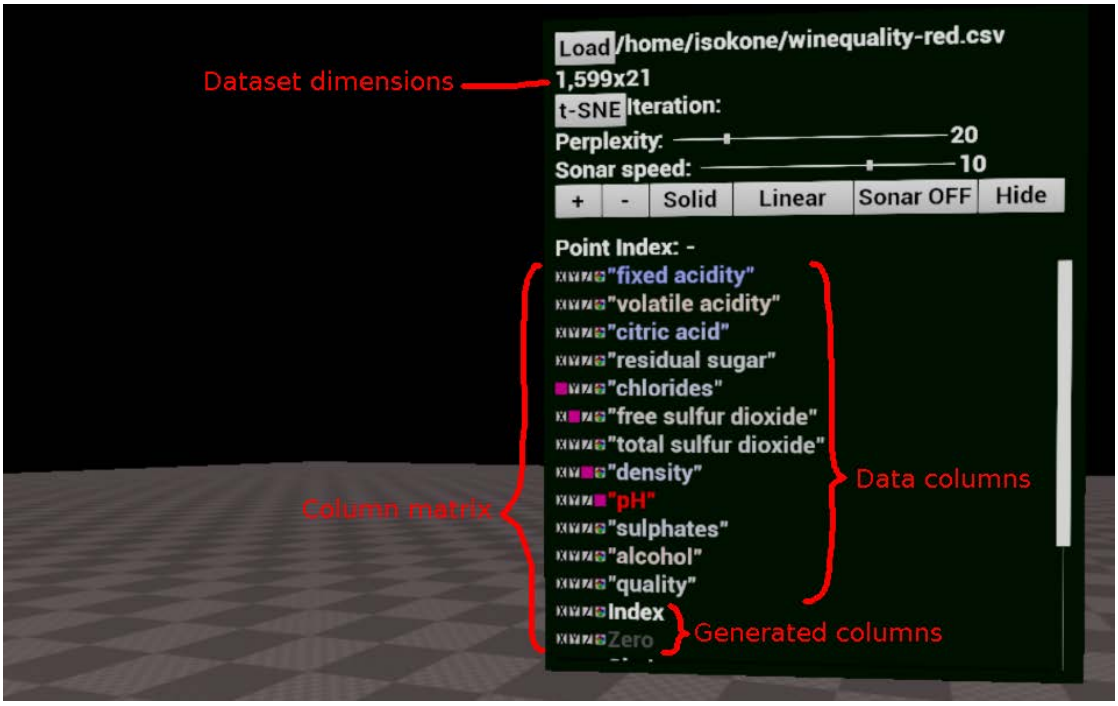
Clicking this button hides the UI, after which it must be summoned using the menu button on the motion controllers.

Point index

This text shows the index of the currently selected point in the data set.

Column matrix

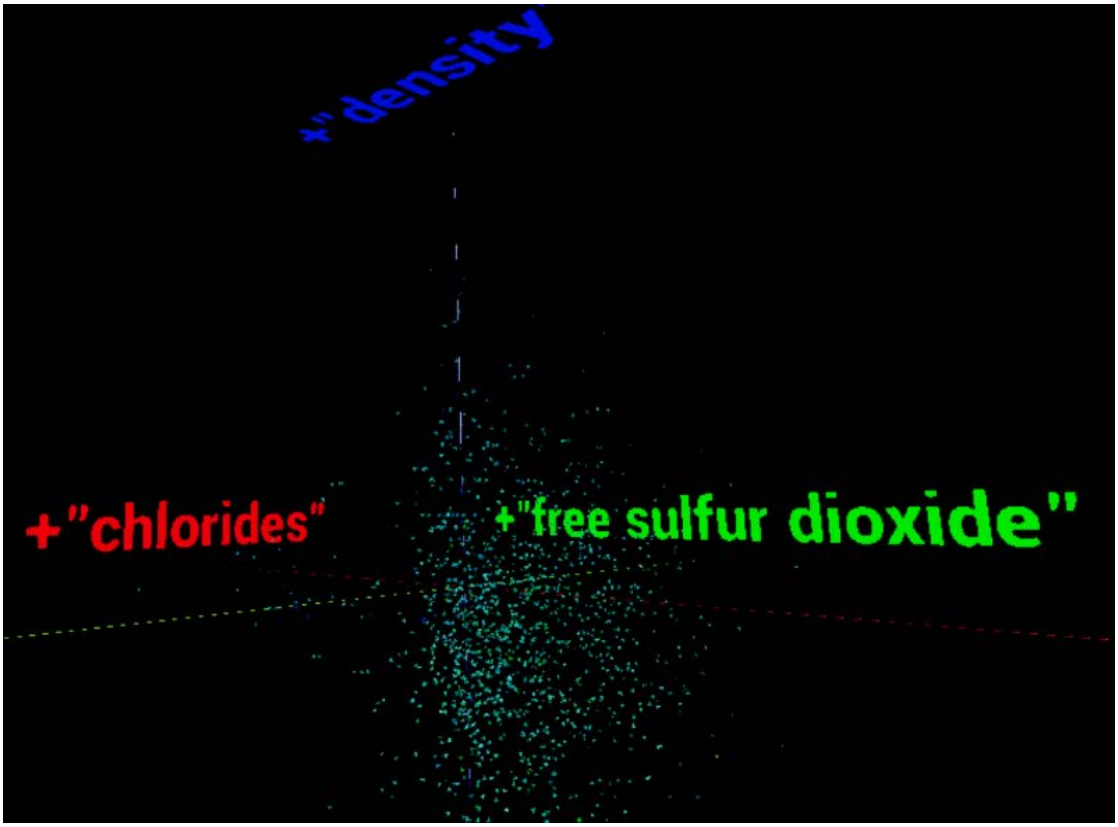
In the bottom of the floating user interface there is a list of all columns in the data set. To the left of the column names there are four buttons for each column. These buttons select which axis should be used to show the column. The axes are from left to right: X-axis, Y-axis, Z-axis and Colour.



In the above image, columns named "chlorides", "free sulfur dioxide", "density" and "pH" are assigned to X, Y, Z and Colour axes respectively.

The column matrix also displays the column names with varying red or blue to indicate the correlation coefficient between the column and the active column for the Colour axis. Vibrant red indicates strong positive correlation while strong blue indicates strong negative correlations. Grey indicates no correlation at all. Shades between the colours show a varying degree of correlation. A dark grey colour shows that the column is a constant column with zero variance.

In the image above, the column "pH" is selected on the Colour axis and therefore correlations between it and other columns are shown on the names. "pH" has some negative correlation with columns "fixed acidity" and "citric acid" because the names of these columns have a slight bluish tint. "pH" also strongly (and obviously) correlates with itself as the column label is in bright red letters. The column "Zero" is dark grey due to it being a constant and, zero-variance column.



This image shows how the selected data axes are displayed on the visualisation. The column labels are shown at the end of the axis lines, prepended with either a "+" or a "-" to indicate whether the axis is pointing into positive or negative direction. The coloration of the axes is used to distinguish them from each other: colour red is the X-axis, green is the Y-axis and blue is the Z-axis.

Generated columns

In addition, at the end of the list there are generated columns, such as row index, t-SNE and a "zero-column." These generated columns are not part of the loaded data set, but they are generated from the data set using some algorithm. The most useful of these are the t-SNE 1, 2 and 3 columns which represent the result of t-SNE that is run on the data set.

Operating t-SNE

t-SNE is a dimension reduction algorithm that may be used to find a representation of the data set in fewer dimensions. To activate t-SNE adjust the perplexity slider to desired perplexity and click the "t-SNE". These controls may be found in the floating

user interface. In addition, you must select the three t-SNE axes from the column matrix as display axes. The current t-SNE iteration is shown in the UI.

t-SNE can take a long time to start up on larger data sets. During the start-up period a small rotating symbol is shown after the t-SNE button. If the symbol does not appear and t-SNE does not seem to activate, you should try a smaller perplexity.

Appendix 2. Program architecture diagram