NTNU
Norwegian University of
Science and Technology

Master's thesis

2019

Johannes Andersen

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Johannes Andersen

StarVR: Star Coordinates and kNN configuration in Virtual Reality

June 2019

NTNU
Norwegian University of
Science and Technology

StarVR: Star Coordinates and kNN configuration in Virtual Reality

Johannes Andersen

Computer Science
Submission date: June 2019
Supervisor: Theoharis Theoharis
Co-supervisor: Jo Skjermo

Norwegian University of Science and Technology
Department of Computer Science

Sammendrag

Denne masteroppgaven ønsker å undersøke muligheten til å bruke virtuell virkelighet som et verktøy for å visualisere og samhandle med flerdimensjonale data representert som Star Coordinates [1] og bruke den resulterende aksekonfigurasjonen til å sette parametere for kNN klasifisering. Ved å implementere Star Coordinates i Unity sammen med vanlige interaksjonsmetoder innen virtuell virkelighet, vises det at dette er mulig både fra et teknisk og brukersentrert perspektiv. StarVR kan visualisere et dataset som er stort nok til å illustrere at denne fremgangsmåten kan brukes professjonelt. Klassifiseringsnøyaktigheten når StarVR brukes som dimensjonsreduksjon for kNN er på linje med automatiserte metoder som PCA. Det konkluderes med at StarVR kan, ved hjelp av sanntids-integrasjoner og tilbakemeldinger, oppnå enda bedre resultater.

Abstract

This thesis aims to investigate the possibility of using virtual reality as a way of visualising and interacting with multidimensional data using Star Coordinates [1] and using the resulting axis configuration produced by this system as a method of setting parameters for kNN algorithms. By implementing Star Coordinates in Unity along with common VR interactions, this thesis shows that it is possible both from a system performance and usability perspective. StarVR can display a dataset that is large enough in both dimensionality and size to get accurate real-world results. The performance of StarVR as a dimensionality reduction technique in regards to kNN is shown to be similar to automated PCA, and conclude that even better results can be achieved by tighter, real-time integration's with common data analysis tools.

Preface

Data exploration and visualisation is an important part of *Big Data*. It plays an important role in converting data into meaningful results. Large dataset tends to be intimidating, both in dimensionality and size. It is therefore of great interest to create new ways of interacting with and understanding large datasets. This project is using Star Coordinates in Virtual Reality (VR) to create a more intuitive and immersive experience in early-stage data exploration. The unique interaction methods of VR provide a new way of looking at data interaction, and the project looks at how to improve existing interaction models with gestures and movement. The solution implements already existing multidimensional data transformations into 3D while extending them to fit the VR platform. The viability of visualisation of highly dimensional data is evaluated, both from a user and a technical perspective.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The availability of large, complex datasets has never been more abundant. This evolution of available data has huge implications and the potential to affect almost every aspect of our lives [2]. It also brings opportunities in regards to data analysis and visualisation. Being able to analyse data in a simple and intuitive way with powerful tools is going to give businesses the competing edge, save unnecessary costs and emissions, optimise processes and much more. The creation and evolution of these tools are therefore a valuable endeavour for all parts of society [3].

1.1 Acknowledgements

The author would like to thank the supervisors Theoharis Theoharis and Jo Skjermo for their excellent insights, helping hands and rewarding discussions during this thesis and the preliminary specialisation project. This is the culmination of both the specialisation project and the master's thesis itself and therefore builds on unpublished work done by the author previously in TDT4501. That work is included in this thesis, either in revised or original form.

1.2 Project goal

The concept of this thesis was initially proposed by Jo Skjermo. It can roughly be divided into two sections, each concerning slightly different, but related, concepts of data exploration and visualisation.

The first part of the project is to visualise multivariate data in virtual reality using Star Coordinates and complex user interaction in order to visually experiment with axis configurations. The second part concerns how Star Coordinates axis configurations can be used in conjunction with AI algorithms, specifically looking at kNN, to ease the generation of hyper-parameters and provide an initial starting point for further tweaks and optimisation for classification tasks.

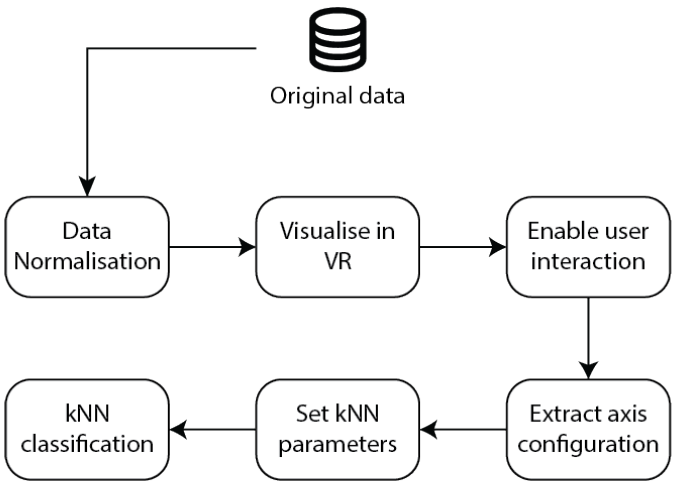


Figure 1.1: Project flow

In short, the main milestones of this project are to:

- Transform data into Star Coordinates for use in a game engine.
- Visualise the data in VR.
- Allow the user to transform the visualisation in VR.
- Use the resulting axes as parameters for kNN.

The general flow of the project, and how all the pieces fit together, can be seen in Figure 1.1.

1.3 Overview of relevant concepts

The following section will give a brief overview of some of the main concepts in this project. Those that are of the highest relevance to the project are also described further in chapter 2.

Multidimensional and multivariate data

The dimensionality, or more specifically domain dimensionality, of data, can be described as the number of attributes associated with a single data point. In this project, the multivariate Iris dataset [4] will be used for demonstration purposes. It consists of 150 instances of Iris flowers. Each flower has 4 attributes and a species name associated with them. This dataset is primarily used for classification tasks. An excerpt of the Iris dataset can be seen in Table 1.1.

Sepal length	Sepal width	Petal length	Petal width	Species
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa

Table 1.1: An excerpt from the Iris dataset. All measurements are given in cm.

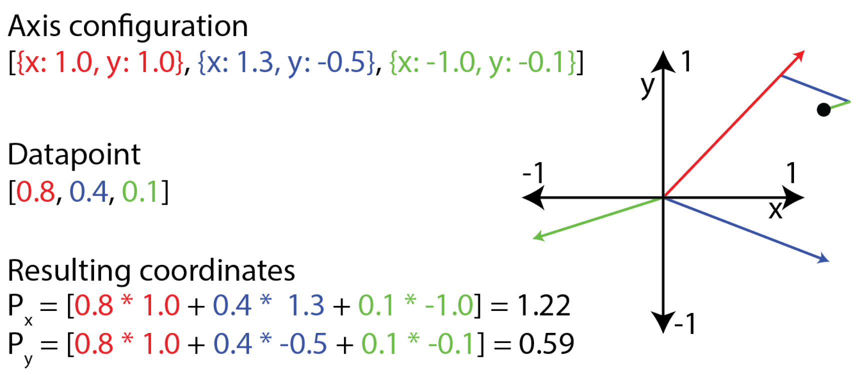

Axis configuration
[{x: 1.0, y: 1.0}, {x: 1.3, y: -0.5}, {x: -1.0, y: -0.1}]
Datapoint
[0.8, 0.4, 0.1]
Resulting coordinates
P_x = [0.8 * 1.0 + 0.4 * 1.3 + 0.1 * -1.0] = 1.22
P_y = [0.8 * 1.0 + 0.4 * -0.5 + 0.1 * -0.1] = 0.59
y
1
-1
1
x
-1

Figure 1.2: Simple star coordinate plotted for a single data point

The dataset has 4 dimensions, but a typical 2D visualisation only has 2. In order to be able to visualise it in 2D, we must first reduce the 4 dimensions down to 2. This is where Star Coordinates comes in.

Star coordinates

Star Coordinates is a solution for reducing the number of dimensions when visualising multivariate data. This approach was proposed by Eser Kandogan in 2000 [1]. The purpose of Star Coordinates is to enhance the user's insight into the dataset. However, by reducing the number of dimensions, some of the original information is lost. The method is therefore proposed as an early stage method for data exploration. A simple example of plotting a data point in Star Coordinates can be seen in Figure 1.2. A more detailed explanation can be found in section 2.1.

Virtual Reality

Virtual Reality is often associated only with the head-mounted displays that have become more mainstream and accessible in the last years [5]. It is important to think of them as more than a collection of hardware. The immersion, user experience and realism are all important part of a good VR system [6]. But in order to provide a good user experience, the hardware aspect does play an important role. Oculus Rift [7] and HTC Vive [8] are the two most popular high-end VR headsets on the market today. They are however rather expensive for normal consumers, and other alternatives, such as Google Cardboard [9] do exist. Google Cardboard allows the user to use their phone as a VR device. When performing scientific data analysis with a large dataset, a more powerful setup is often required [10].

Game engines

Creating realistic virtual environments is an expensive and time-consuming process [11]. Luckily, there exists game engines like Unity[12], Unreal Engine [13] and CryEngine [14]. Taking advantage of these engines, together with external libraries, allows for rapid prototyping that provides state of the art results with minimal effort. Most of these engines do come at a cost in the form of subscription, royalty and licensing fees. Their pricing model does vary, and most of the available engines only charge based on the value of the product that you create with their engine. Out of the ones already mentioned, CryEngine is free to use, while the others require a small royalty fee on commercial products. For the software produced in this project, these fees are of no concern as commercialisation is not an important factor in the project at this point in time.

kNN

k-Nearest Neighbour is an instance-based learning algorithm for classification and regression. This category of machine learning algorithms is easy to reason about at a conceptual level, but they do not come without drawbacks [15]. kNN base the classification on similar data points that are *close* to the data point that is to be classified. The *k* is the number of neighbours to look at to determine the class of our new data point. One of the drawbacks to kNN is the fact that almost all computation is performed when classifying and that the entire training dataset must be available [15].

Chapter 2

Background

2.1 Star Coordinates

Star Coordinates was presented by Eser Kandogan in 2000[1]. The inspiration for Star Coordinates comes primarily from Permutation Matrices[16] which allows for rearrangement of rows and columns and Parallel Coordinates[17] which utilises a uniform treatment of dimensions. Combining these concepts ultimately resulted in Star Coordinates. Star Coordinates arrange all uniform axis in a circle with the same length. The original proposition only did this in 2D, but Cooprider and Burton extended this to 3D in 2007[18]. A visualisation of a dataset with 4 dimensions plotted in 2D can be seen in Figure 1.2. Note that this reduction of dimensions, and the introduction of nonorthogonal coordinate axes, introduce ambiguity as several combinations of axis values can produce the same point in 3D space.

Calculating Star Coordinates

The first step to visualise a dataset D in Star Coordinates in 3 dimensions is to choose an origin

$$O_n(x, y, z) = (o_x, o_y, o_z)$$

where n is the dimensionality of the dataset. The axes are then defined. They should all initially be of equal length and start at the origin and can be defined as n three-dimensional vectors

$$A = \langle \vec{a_1}, \vec{a_2}, ..., \vec{a_i}, ..., \vec{a_n} \rangle$$

We can then map a point to Star Coordinates as follows:

$$P_j(x, y, z) = (o_x + \sum_{i=1}^{n} \hat{u_{ix}} \cdot (d_j i - min_i), o_y + \sum_{i=1}^{n} \hat{u_{iy}} \cdot (d_j i - min_i), o_z + \sum_{i=1}^{n} \hat{u_{iz}} \cdot (d_j i - min_i))$$

where the datapoints are defined as

$$D_j = (d_{j0}, d_{j1}, ..., d_{jn})$$

the length of the normalized axis vectors are

$$|\vec{u_i}| = \frac{|\vec{a_i}|}{max_i - min_i}$$

and the minimum and maximum of an axis i is

$$min_i = min\{d_{ji}, 0 \leq j < |D|\}, max_i = max\{d_{ji}, 0 \leq j < |D|\}$$

An intuitive understanding of the formulas above is that the point in Star Coordinates is the sum of all the normalised axes multiplied with their original data points components plus the origin.

Operations and features

Kandogan introduces a few key operations that the user can perform on Star Coordinates in [1]. One year later, he also published [19] where he introduces more user interactions and visualisations, such as marking, range selection, histograms, footprints and sticks. This section will describe the operations and why they are useful.

Scaling

Scaling changes the length of an axis and therefore changes its contribution to the visualisation. The axis can also be disabled by setting its length to 0. To change the scale of an axis, the user simply grabs the end of the axis marker and drags it to its desired position.

Rotation

Rotation changes the direction of the axis vector. This changes how this axis contribution is relative to the other axis. Rotation helps solve ambiguities and it can separate overlapping clusters [19]. To perform a rotation, the user selects one or more axes and drags them in the desired direction.

Data queries

Seeing the original data for a point can be useful. To do this, the user hovers over the point and the information is shown.

Axis range selection

Sometimes it is of interest just to examine a subset of the data. By selecting a range on one or more axis, the user can filter or mark the points to better understand how they are affected by a change in the axis configuration. The user can also perform logical operations on the different axes to further enhance the selection.

Marking

The user can mark points by selecting them. This is either done by clicking the points or dragging a rectangle around them. They can then be marked and filtered in order for the user to see how the transformations affect them.

Histogram

Histograms allow the user to compare selected clusters. They can then see how they differ in a selection of the attributes and determine the axes that make this cluster unique.

Footprints

Footprints track the position of the data points. These positional changes are then represented as a line, and the user can see how all the points changed under a particular axis transformation. By changing specific axes and tracking their change, the user can infer the axis contribution and how it relates to the other axes [19].

Sticks

The last visualisation options discussed by Kandogan is Sticks. It has been pointed out earlier that the points in Star Coordinates do not have a unique mapping to the original dataset. Different data points might show up on the same point in Star Coordinates. Sticks is an attempt at detecting this. In normal mode, the points are represented by dots. When stick-mode is active, all the dots are replaced by a collection of lines. Each line has a length relative to its corresponding axis. The user can then decide which axis to encode to examine. If the cluster has similar looking sticks and lengths, they have a similar value for those attributes. If not, the cluster might be comprised of more than one cluster that lies on top of each other due to the nonorthogonality of the coordinate system.

2.2 k-Means

The k-means clustering algorithm was first published in 1955 [20]. It is based on centroids, the arithmetic mean position of all points, and it contains two steps.

Before the iteration begins, initialise the centroids $\langle \mu_1, \mu_2, ..., \mu_k \rangle \in \mathbb{R}^n$.

The first step is to assign each point x^i to its closest centroid class given by

$$c^i = \underset{j}{\operatorname{argmin}}\, dist(x^i, \mu_j)^2$$

The second step is to update the centroids based on the points that belong to that centroid. For centroid μ_j this is done by

$$\mu_j = \frac{\sum_{i=1}^{m} 1\{c^i = j\} x^i}{\sum_{i=1}^{m} 1\{c^i = j\}}$$

These two steps are then performed until some criteria are met [21].

2.3 k-Nearest Neighbours

kNN is the most basic instance-based learning method. It assumes that every instance, or datapoint, corresponds to a point in n-dimensional space $\mathbb{R}^n$ [15]. The closest neighbours are defined by the Euclidean distance. Given a feature vector

$$\langle a_1(x), a_2(x), ..., a_n(x)\rangle$$

where $a_r(x)$ is the rth value of instance x, the distance between two instances, x_i and x_j, is

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2}$$

The k in kNN refers to how many neighbours to look at when classifying. There are two scenarios to consider, depending on whether we are approximating discrete or continuous-valued functions. In the case of a discrete function, we want to find the most common value. The hypothesis function $\hat{f}$ is given by

$$\hat{f}(x_q) \leftarrow \underset{v\in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where

$$\delta(a, b) = 1 \text{ if } a = b \text{ and } \delta(a, b) = 0 \text{ otherwise}$$

When approximating continuous-valued functions, we instead find the mean value of the neighbours. $\hat{f}$ is then given by

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

There are also modified versions of kNN that can be of interest. One of these is Distance-Weighted kNN. This version values the contributions of the neighbours based on how close they are to the point x_q. The closer the point, the higher the weight. The exact formulas are described in Machine Learning [15, Chapter 8].

kNN is not without its flaws. The most notorious problem is the *Curse of dimensionality* [22]. When the data contains a lot of dimensions, it is not unlikely that some of them are irrelevant to the task at hand. If a dataset contains 10 attributes, but the classification is only based on three of them, the remaining seven attributes can still be far apart and thus misleading the classification. One solution to this problem is to weight the attributes differently or even remove them. This weighting of attributes is where the axis configuration of Star Coordinates comes into play.

2.4 Principal Component Analysis

Principal Component Analysis is a multivariate dimensionality reduction technique dating back to Karl Person in 1901 [23]. PCA takes a set of possibly correlated variables and converts them into another set of linearly uncorrelated variables. These are the principal components. The goal of the transformation is to preserve as much data as possible in a lower dimension. Correlated variables contain overlapping information and by optimising for the largest variance between variables the resulting variables contain more of the interesting aspects of the dataset. All principal components must be orthogonal to each other and the variables are a linear combination of the original variables [23].

The fact that PCA is both orthogonal and utilises a linear combination of variables, like Star Coordinates, makes it a perfect comparison to make, as to how effective the Star Coordinates method is at optimising information retention in lower dimensional spaces. When StarVR is compared to PCA with 3 principal components, the result should show if a human can find a better linear combination of the variables than an automated method purely optimising for variance.

2.5 iViz

What seems to be the closest existing application to what this project wants to achieve is called iViz [24]. The goal of the iViz project is to create an interactive and collaborative Virtual Reality based visualisation tool for data visualisation and exploration.

Implementation

The implementation of iViz was originally made in vCaltech [25]. The vCaltech project is made by *Caltech Astroinformatics group*. It is an immersive Virtual Reality environment based on OpenSIM [26]. They did eventually move over to Unity, as the vCaltech engine was not optimised to the same degree. Unity did also provide a lot of convenient, state-of-the-art features that were lacking in the older engine. One of the important features was the ability to run the application in a web browser. Unity, being an engine also used for games, support multiplayer features which allowed for real-time collaboration.

Data representation

iViz aims to visualise large multidimensional multivariate dataset. In order to visualise more than 3 axes in 3D, some transformations need to take place. The axes chosen for iViz include, but are not limited to

- XYZ coordinates
- Colour
- Shape
- Size
- Transparency
- Texture
- Animation

which result in the ability to visualise at least 9 attributes.

Interactivity and collaboration

The iViz application supports the LeapMotion sensor [27] and Kinect. The user can then move around and select data points in the visualisation. When the user selects the data point, the original data is shown in a modal. In order to change the visualisation, the user can change which data dimensions map to which visualisation dimensions. If used on the iris dataset, the configuration could look like shown in Table 2.1.

Data dimension	Visualisation dimension
Sepal length	x-axis
Sepal width	y-axis
Petal length	z-axis
Petal width	size
Species	colour

Table 2.1: Possible axis configuration in iViz

2.6 iStar

One of the major drawbacks to Star Coordinates is that it does not handle large numbers of attributes very well. iStar(interactive Star Coordinates) [28] propose methods for dealing with high dimensional datasets through attribute clustering and dynamic level of detail representations.

Method

The method proposed consists of three parts. The first part is called *Linear Mapping* and deals with the fundamental math of Star Coordinates and how the data points are a linear combination of the axes. The other parts are Attribute Clustering and Reordering as discussed below.

Attribute Clustering

The similarity of the axes can be used to group axes together in order to prevent information overload. They propose three methods: Variance, Principal Component Analysis [29] and centroids.

Variance for the jth attribute can be defined as

$$\sigma_j^2 = \frac{\sum_{i=1}^{m}(p_{ij} - \mu_j)^2}{m}$$

where m is the number of instances μ_j is the average of the jth attribute. They then use k-Means to group similar attributes that can be merged in the visualisation.

The Principal Component Analysis approach sees each attribute as a point in m-dimensional space and maps the points to the space generated by the first two principal components. The attributes that map close to each other are similar and are then grouped using k-Means.

In order to use the centroids method, the dataset need to have classes. The centroid $\bar{p_{c_i}}$ is given by

$$\bar{p_{c_i}} = \frac{1}{N_{C_i}} \sum_{p \in C_i} p$$

where N_{C_i} is the number of instances with class C_i. A matrix M is then constructed with centroids as column vectors. Finally, k-Means is applied to the rows of M to group similar attributes.

All the methods mentioned above reduce multiple axes into one. The axis length is set to 1 and the contribution of each grouped axis to a point is found by averaging the values of the grouped axes with respect to the point.

Reordering

The positioning of the axes is important to give a good starting point for data exploration. There are two methods that are being evaluated: *Combinatorial optimisation* and *brute force*.

The combinatorial optimisation depends on a dissimilarity matrix M of size (k x k) where k is the number of axis after clustering. M is defined as

$$M_{ij} = \frac{1}{m} \sum_{s=1}^{m} \left| \frac{p_{si} - min_i}{max_i - min_i} - \frac{p_{sj} - min_j}{max_j - min_j} \right|$$

The closer this metric is to 0, the greater the similarity between the attributes. They then represent the matrix as a complete graph, and a Genetic Algorithm is used to find the optimal closed path connecting all the nodes. Then the axes are rearranged in the same order.

The brute force approach simply swaps axes until it has found a satisfactory solution that minimises a quality metric. They have used *topology preservation* and *Dunn index* as quality measures.

Interaction and tools

As a result of the combination of axes, several new tools have been proposed to help visualise the more complex nature of the dataset.

The node preview magnifies the selected axis and shows only the combined axes.

The node explorer is showing the same information as the node preview, but here the axis can be changed individually and the interaction is mirrored to the main visualisation.

The quality visualiser shows the history of the quality metrics when the user changes the visualisation.

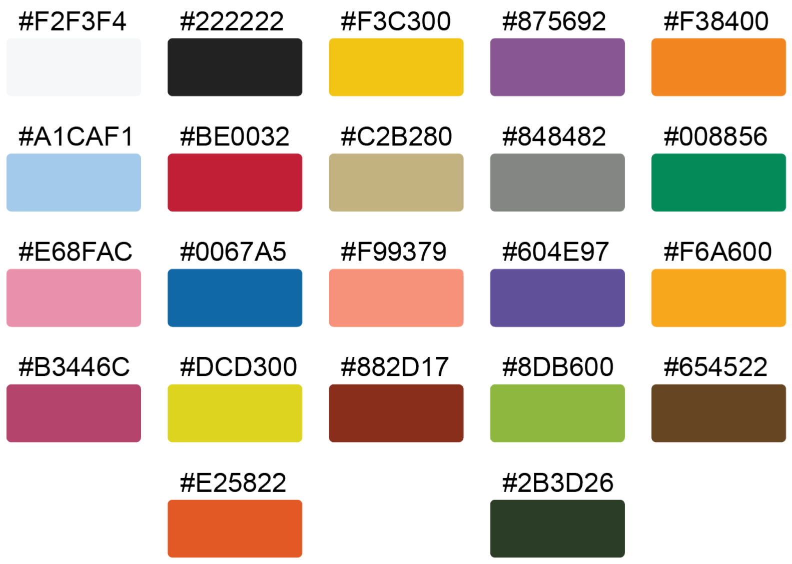


Figure 2.1: The Kelly colours and their corresponding hex values

Results and evaluation

The full results can be seen in [28]. The overall results were that iStar outperformed both Star Coordinates and radViz with statistically significant margins.

2.7 Colours

Choosing colours for user interfaces is a difficult task, which is often left to the discretion of a designer or user experience expert. In order to make the distinction of different colours easier, KL Kelly introduced the *Twenty-two colours of maximum contrast* [30]. They are aimed at being easy to differentiate and can be seen in Figure 2.1 along with their hexadecimal colour value.

2.8 Unity

A Unity application is mostly built using a few fundamental building blocks. These building blocks dictate the architectural decisions and best practises, and it is important to understand them in order to have a friction-less experience building high-quality Unity applications. Figure 2.2 shows these blocks in the context of StarVR.

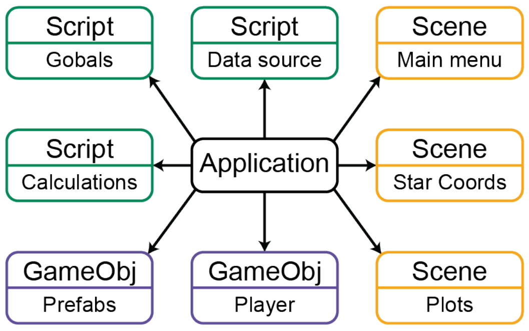


Figure 2.2: Overview of the artefacts in a Unity application

Scenes

A scene in Unity is the highest hierarchical level of the application. It can be thought of like a scene in a play. It contains its own set of GameObjects which allows the game to be designed in decoupled pieces that are mostly independent of each other. It is possible to have multiple scenes loaded simultaneously, but the application described in this section only relies on completely decoupled scenes.

GameObjects

A GameObject is a container for everything in a Unity application. Everything from primitives, meshes, lights, players to special effects. The GameObjects themselves cannot do anything apart from existing, so in order to give them properties a Component is added. A Component can do a multitude of things. To make an object show up in the application, a renderer component is required. To be able to move the GameObject, a transform component is applied by default. If custom behaviours are required, a script component is added. An overview of these is seen in Figure 2.3.

Transforms

A transform component dictates the GameObjects' rotation, position and scale in the scene. These can be set, updated and animated thought script components. Another important feature is the ability to set a transforms parent. The child will inherit the parents transform and then apply it's own on top of that. When transforming the parent, the children will respond accordingly. The benefit of this is that no scripts are required to individually move the children when the parent is either moved, rotated or scaled.

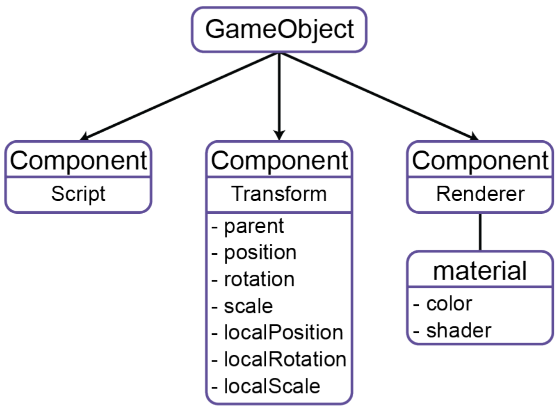


Figure 2.3: Some of the properties of the GameObject

Scripts

In order to make the GameObjects behave in the desired manner, scripts are used to modify their components in a programmatic manner. They can also be used to instantiate, hide/show and destroy GameObjects. The scripts are written in C#, and all supported runtimes (.NET) and libraries in the C# ecosystem can be used. The scripts can read from the filesystem, access the internet, access databases and use complex libraries for machine learning and data manipulation.

Chapter 3

Requirements

This section describes the functional and non-functional requirements of the StarVR application. The project has been developed in an iterative process with continuous feedback from supervisors and test users, so this section will mostly describe the requirements as they stand today. The secondary goal is also to give an introduction to the application so that the following chapters are easier to follow without experience using the application first-hand. The instructions for running the application can be found in Appendix C.

3.1 Functional requirements

The features of StarVR are all implemented according to common virtual reality principles and aims to be as intuitive to use as possible and provide tools that allow the user to extract valuable information in a faster and easier manner. The features are inspired by the original Star Coordinates [1] as well as extensions of this discussed in the previous chapter.

Dataset selection

The first scene that the user finds themselves in is the dataset selection menu as can be seen in Figure 3.2. Here they have to choose which dataset they want to work on. This is at the moment limited to *.csv* files that are bundled at build time, but it can easily be extended to integrate with any offline or online data source specific to the user's needs. The user uses the right-hand pointer and trigger to select a dataset, which is a common way of interacting with menus in VR applications.

Axis selection and representation

The next menu is the axis selection and representation menu as seen in Figure 3.3. Here, the user can select which axes to be visible when they start the Star Coordinates visualisation. They can also select how the datapoints are distributed along the axes. The

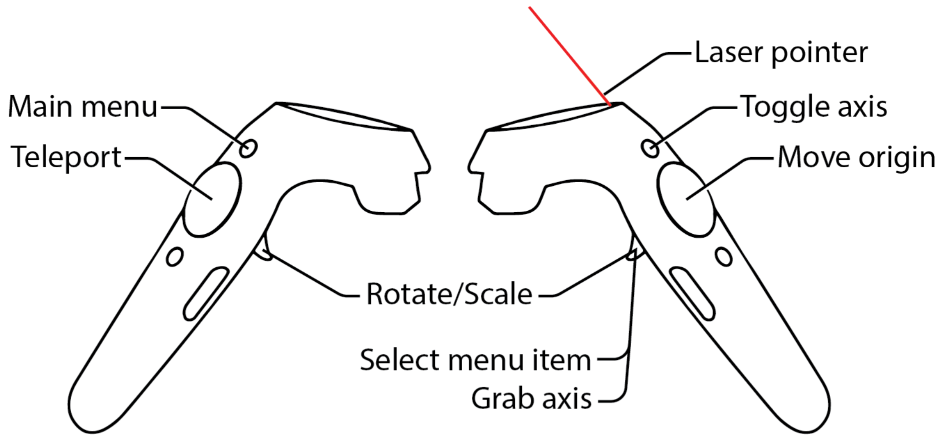


Figure 3.1: Controller scheme for SC application

options are listed below along with their respective functions. The ranges and domains of all functions are $[0, 1]$. They can also be seen in Figure 3.4.

- linear $f(x) = x$
- inverse linear $f(x) = -x$
- sigmoid $f(x) = \frac{1}{1+e^{-x*12+6}}$
- inverse sigmoid $f(x) = 1 - \frac{1}{1+e^{-x*12+6}}$
- exponent $f(x) = x^5$
- inverse exponent $f(x) = 1 - x^5$

Translation

There are three ways to change the position of the coordinate system relative to the user. The simplest is to use the d-pad, which is marked *Move Origin* in Figure 3.1. Another option is to hover over the origin, press *Grab axis* and drag it around. The last option is to use the left-hand d-pad, marked *Teleport* in Figure 3.1. This is the SteamVR default teleport behaviour.

Rotation

Rotation of the coordinate system is performed by pressing both triggers, marked *Rotate/Scale* in Figure 3.1 and move the controllers relative to each other. The rotation of the controllers is applied to the coordinate system. If the user holds one controller still while moving the other from the left of it to the top of it, the coordinate system rotates $90\deg$ on the corresponding axis.

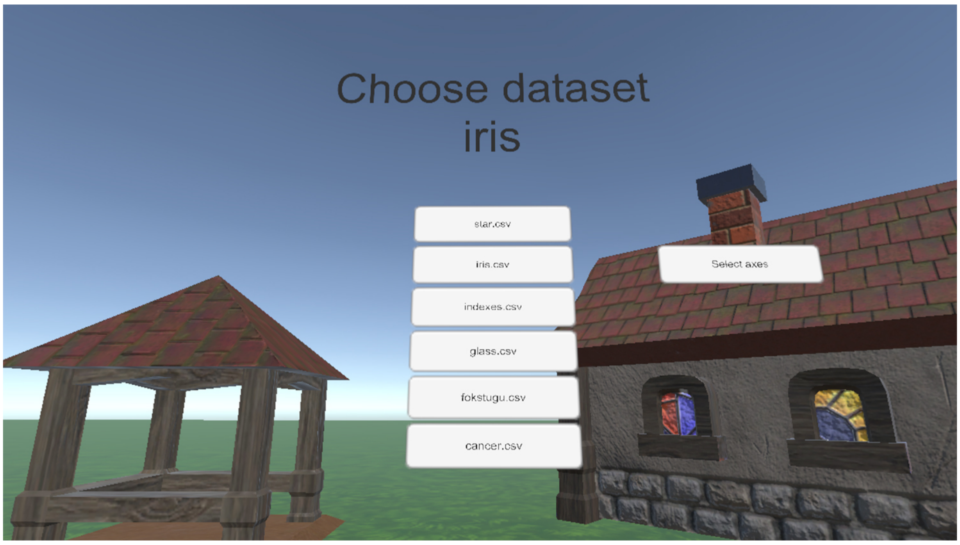


Figure 3.2: Dataset selection in the SC application.

Scaling

Scaling is done in a similar way to the rotation, but instead of measuring the relative rotation, it measures the distance between the controllers. Pressing the triggers and doubling the distance between the controllers increases the coordinate system scale by a factor of 2.

Toggle axes

Toggling the axes changes whether they are included in the SC calculation. To disable an axis, hover over the axis points, press the right menu button marked *Toggle axis* in Figure 3.1 and the axis is disabled. The operation can be seen in Figure 3.5. On the left, the axis point is being hovered over. By clicking the right menu button, the axis disappears. Although hard to see from the perspective, the datapoints have moved slightly and the axis name is now listed above the origin in the background. In order to get the axis back, hover over the name above the origin and click *Toggle axis* again.

Merging of axes

In cases where there are many axes to keep track of, the user can choose to merge them so that they can be moved together. Moving one axis on top of another merges the two axes, and the result can be seen in Figure 3.6. In order to separate the axis, hover over the axis name and click the *Toggle axis* button in Figure 3.1.

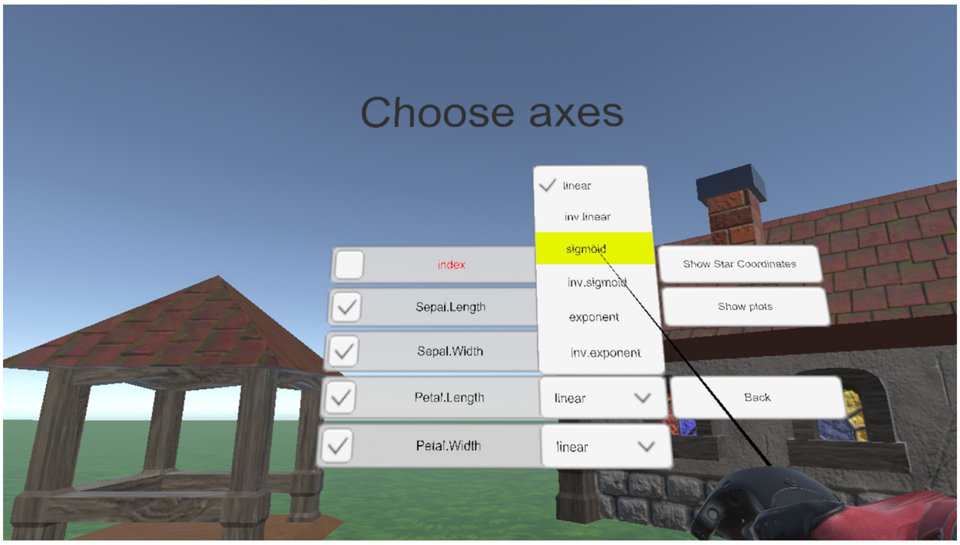


Figure 3.3: Axis selection and representation in the SC application.

Star Coordinate path

Another interesting and useful feature of Star Coordinates is the ability to see the *path* a datapoint took to get to its current position. This path consists of the individual vectors that make up the linear combination that is Star Coordinates. The path is shown when a datapoint is hovered over. The result can be seen in Figure 3.7.

Plots

An additional feature was proposed by Jo Skjermo during the development of the StarVR application. This was a series of 2D plots of the data arranged in 3D space. It can often be useful to compare the attributes of a dataset pairwise in the search for interesting correlations. Having $360\,\text{deg}$ of vision allows these axes to be laid out in such a manner that it is easy to quickly spot interesting attribute pairs. An example of this is shown in Figure 3.8. The user can teleport and rotate the circle of plots to quickly gain an oversight of the selected axes and their pairs.

Save configuration

When the user is satisfied with their axis configuration, they can export this for use in their continued data exploration. The configuration is simply the 3D vector representing each attribute in the dataset. This is saved to a text file, which can be used to transform the entire dataset into three dimensions using the Star Coordinate algorithm. The dataset of reduced dimensionality can then be used for further processing and analysis.

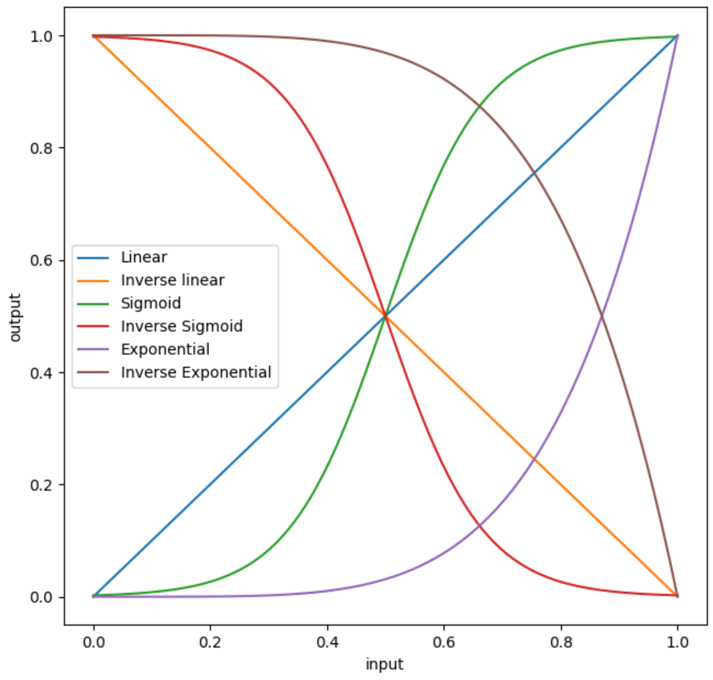


Figure 3.4: Comparison of axis transformation functions.

3.2 Nonfunctional requirements

Having nice and useful features are all well and good, but without a functioning application, they are worthless. StarVR is an MVP, and the functional requirements have taken some precedence. The main goal is to be able to test new hypotheses in a quick and easy way, while still be able to run it on datasets that are fairly similar to what one would expect to see in day-to-day use by an industry professional.

Performance

Performance in VR applications has many similarities with other real-time programs, such as computer games. There is only a set amount of time to perform the calculations for each frame, and that is a hard deadline. The primary goal has been to make a functional application, with the features mentioned, and still be able to use it with reasonably large datasets. The performance measurements can be seen in section 5.3, a discussion of them in section 6.2 and further improvements in section 7.1.

Extensibility

When making an MVP, the ability to quickly try out new things is important to find the best solutions. The StarVR application has been through numerous versions during the

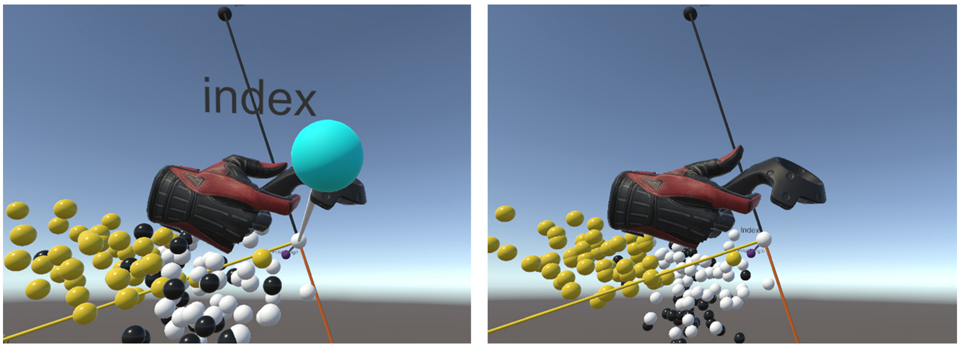


Figure 3.5: Enable and disable axes in the SC application.

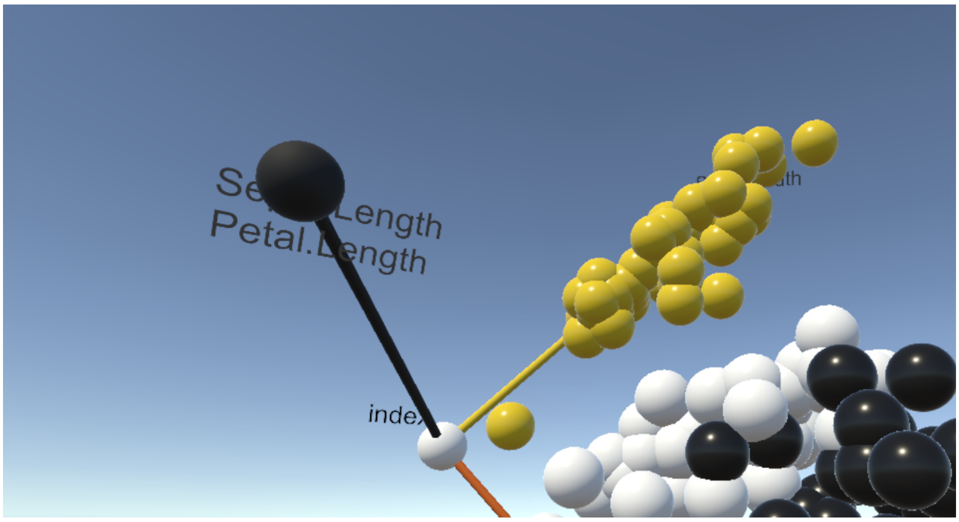


Figure 3.6: Merging axes in the SC application.

last year, some of which almost required a complete rewrite to account for performance optimisations and architectural changes. The resulting software is as a result more easily changed and extended, which has allowed the testing of many features, where some are included in the current implementation of StarVR.

Usability

Although the testing has been done internally in our organisation, following standard patterns for VR applications is important to create a usable product building on the experience of others. If the user is familiar with VR in general, picking up StarVR should be a breeze. It also allows the developer to leverage existing technologies that provide default interactions and functionality so that the focus can be on developing the StarVR concept.

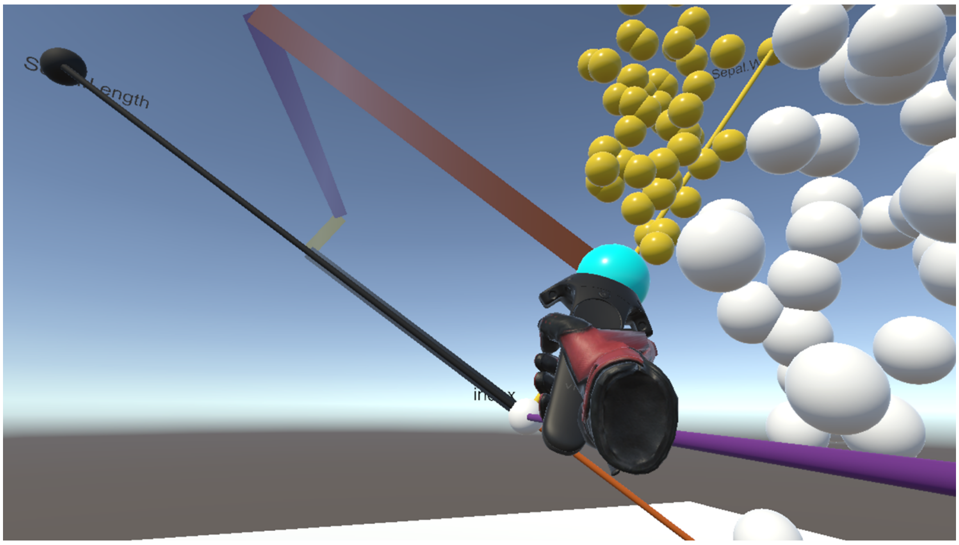


Figure 3.7: Enable and disable axes in the SC application.

Availability

Using Unity allows the developer to build the StarVR application for many target platforms. Given the targeted use-cases and audience, the most viable are desktop computers running either Windows or Linux, but there is flexibility to extend to others if this seems beneficial.

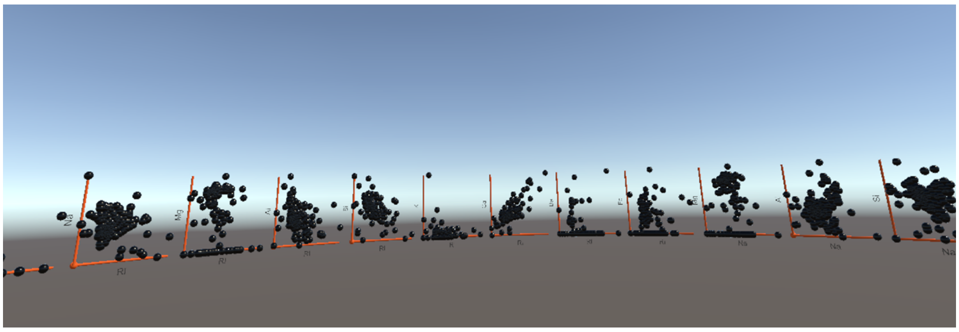


Figure 3.8: SC 2D plots of attributes.

Chapter 4

Implementation and Tools

There is a multitude of available game engines and VR systems out there. For this project, however, it was important that the chosen technology allowed fast prototyping, state-of-the-art results and that is was reasonably simple to use and learn. This section will go through how the solution was implemented, what technologies were used and what some other options are.

4.1 Game engine

The game engine used in this project is Unity [12]. There are multiple commercially available engines available, as discussed in section 1.3, so choosing Unity was based on a few criteria:

- Features: Provide state-of-the-art features and integrations.
- Graphics: Provide good graphics.
- Language: Has support for a familiar language.
- Learning curve: Has a lot of tutorials and examples.
- Plug-and-play: Works more or less out of the box with VR.
- Performance: High performance compared to competitors.

Most of the modern game engines fulfil all of the requirements mentioned above, so the choice of Unity was ultimately made due to the C# programming language, ease-of-use and the abundance of available tutorials. The results achieved here would be possible to recreate in every major game engine, so the choice is more of a personal preference of the author.

The other major commercially available game engine is Unreal Engine. It uses C++ as a scripting language. The graphics of Unreal Engine is generally seen as higher quality than Unity when developing complex games and visualisations. Unreal does have a steeper learning curve and their asset store is smaller than the Unity counterpart.

4.2 VR device

The choice of a VR device was made on the availability of hardware at the university. It is worth to mention that all other serious competitors provide similar features and performance and that the application support most available hardware due to the OpenVR SDK.

The main competitors in VR hardware are HTC Vive and Oculus Rift. Disregarding the new HTC Vive pro, the Vive and the Oculus have the same resolution and framerate. They both support full room setups, but the Oculus is more geared towards a less expansive setup. The Vive supports full room setups out of the box with a max diagonal of 5m while the Oculus only supports 3.5m. The new HTC Vive Pro has an increased resolution of 2,880 x 1,600 while the original Vive and the Oculus have 2,160 x 1,200 pixels. The screen has also changed from an OLED to AMOLED. In Oculus defence, it is the cheaper option, and it provides almost all of the features of the Vive counterparts at a lower price.

4.3 OpenVR SDK

When developing a VR application, you need to use the devices proprietary API's to communicate with the devices. This introduces a large overhead related to development time. One solution to this problem is OpenVR SDK [31] made by the game developer Valve [32]. Together with the OpenVR plugin for Unity, it is possible to develop one application for all OpenVR SDK supported devices. It also has support for HTC Vive and Oculus Rift, as they are most commonly used in enterprise and education settings.

4.4 System architecture

When developing an application in a game engine, there are a lot of choices that have already been made for you by the creators [33]. These choices and abstractions are what makes the game engine such an appealing choice for developers who do not know enough about the underlying mechanics to build an entire 3D application or those who do not wish to do so due to the complexity and resources involved [34].

Unity has some concepts and artefacts that are important to understand in order to build an application. The first one is the *GameObject* [35], which is the base class for all entities in a Unity scene. Unity is built with an object-oriented architecture, so all other entities will in some way inherit from *GameObject*. The other artefact is scripts [36]. Scripts are custom code that can be attached to *GameObjects* to provide functionality though *Components*. These scripts can access and alter any part of the scene that they have a reference to. All scripts are derived from the base class *MonoBehaviour*, and the scripts can choose to implement a set of methods that are called on certain events by the base class. The most used ones are *Start* which is called when the script is activated and *Update* which is called on every frame.

In this project, there are a few *GameObjects*. Their role is to act as containers for dynamically instantiated *GameObjects*. There are also a few *GameObjects* that act as a *prefab*, a template for dynamic content. When the application loads, the *plotter GameObject* calls the *Start* method. This loads the dataset from the filesystem, read it into memory and instantiates new *GameObjects* into the scene. Then, on every frame, the *update* method checks if any button is pressed. If it is, it will change the associated variable and update the position of all instantiated *GameObjects*. Then it simply repeats until the application ends. A diagram of this can be seen in Figure 4.1. Pseudocode of the *Render* function can be seen in Listing 4.1.

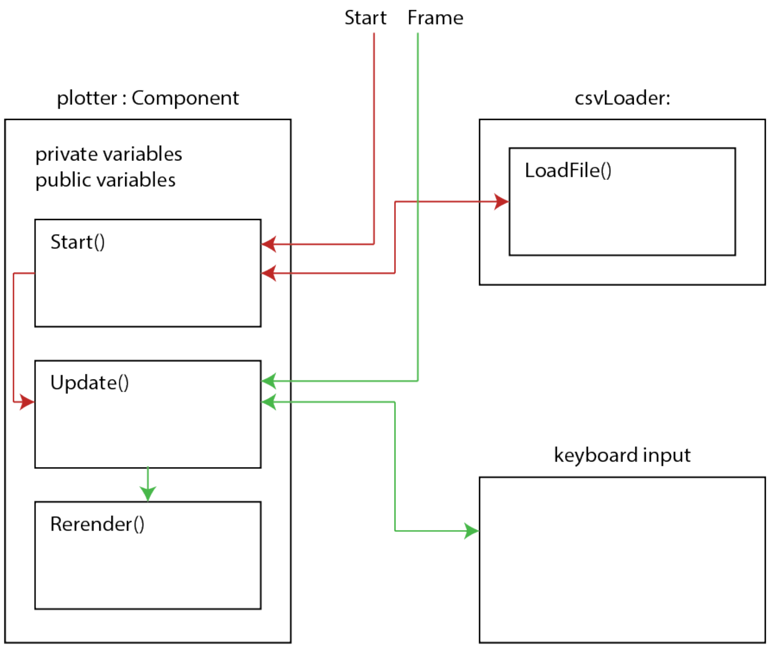


Figure 4.1: Unity update flow. The red lines indicate what happens once the script is activated and the green lines indicate what happens on every frame. The call to Rerender is only performed if a keypress is detected.

Listing 4.1: Pseudocode for updating of Star Coordinate visualisation in Unity

```
def Update:
    # Update points
    for instance, index in pointsHolder:
        position := origin

        for axis, axisIndex in axes:
            poisition += axis*data[index][axisIndex]

        instance.position = position

    # Update axis handles
    for instance, index in axesHolder:
        position := origin + axes[index]

        instance.position = position

    # Update axis lines
    for instance, index in axisLineHolder:
        instance.start = origin
        instance.end = origin + axes[index]
```

4.5 Application Architecture and Rationale

In order to create a high-performance data visualisation application in Unity, it is important to follow their guidelines and try to not work against the framework. The first implementation positioned the datapoints in world space, and therefore required a re-position on every change to the coordinate system. The newest version uses parents/children transform hierarchies to offload the computational expense of repositioning the GameObjects and relies on the built-in GPU implementation of transforms to do this automatically. This improvement also resulted in cleaner, easier to read and resulted in less code.

4.5.1 Architectural principles

When hearing the word global variables, a lot of people instantly regards this as a huge red flag and a code smell. They might be on to something. If all parts of the application have the ability to read and write to common variables, it is harder to reason about how a particular variable has changed. It is also harder to make sure all parts on the software that depends on these variables updates correctly. To mitigate this, the implementation in this project is inspired by the flux architecture [37] by Facebook. This results in a unidirectional data-flow, where actions triggered by the user results in state changes, which again results in all dependent entities update based on this state change. This does not prevent changes to global variables but simply creates a rigid structure to adhere to. By encapsulating the global variables, as described by Martin Fowler in Refactoring [38], a layer of abstraction is achieved which gives greater insight and control of a variable. When combining these two approaches, the result is a dataflow that works well for this kind of dynamic data visualisation application.

4.5.2 Star Coordinates

StarVR's implementation of Star Coordinates and how all the elements interact with each other can be seen in Figure 4.2. The main hub of all changes are the Data Plotter Script. This script updates the Star Coordinate configuration and changes the application state based on input from VR interaction. The update part of the Script can be seen in algorithm 1 which also calls algorithm 2. The remaining functionality is mostly wrappers for changing the application state.

4.5.3 Parenting

By leveraging the power of parent/child transforms, the complexity of the application went down and the performance went up. The performance benefit was only visible when not transforming the Star Coordinates system. The hierarchy can be seen in Figure 4.3.

4.5.4 Event system and updates

The fist implementation of StarVR relied solely on a polling mechanism for detecting user interaction and physics collisions. Using version 2 of SteamVR, a lot of the polling could be replaced with callbacks instead, lessening the overhead of constantly checking if certain criteria are met. There are still some parts of the application that uses polling, but only in performance insensitive areas. While Figure 4.2 show all the elements of the application, Figure 4.4 shows a higher level abstraction of how the updates are performed. The Scene registers listeners for relevant interaction actions and starts checking for update criteria. When a listener is triggered or a certain criteria is met, the accompanying state change is performed through the Data Plotter Script. The changes then result in a change to the Star Coordinate system. Then the loop repeats until the Scene ends.

4.5.5 Data structures

The data structures used in the application are native to the C# programming language, Unity and the .NET framework. The datapoints from the dataset are stored in a static variable of type

List <Dictionary<string, object>>pointList

This is a two-dimensional datastructure where the first layer is a List, where any datapoint can be looked up by its index. The returning value is a Dictionary. This is a mapping from an attribute, which is a string, to its corresponding value, which can be of any type. This object type does cause some annoyances when using the data, but the value could be of any datatype and it has to be dealt with at some point in the pipeline. The time complexity of the retrieval of any points attribute is constant time.

The attributes are also stored in a List, which gives them an order. This is used to store the mapping of attributes to groups in another List. The indexes of these two lists correspond to the same attribute/group pair. This is visualised in Figure 4.5. This is done as it is required to get all attributes in a group, as well as getting a certain attributes group. This is then easier than using a Dictionary, which would be preferable if only one of the lookups were required.

4.5.6 Source of truth

In a complex application, it is important to have a clear mental model of the current state of the system. Alongside this, having a single source of truth is helpful when debugging and it minimises the risk of unforeseen side-effects and discrepancies.

There are two entities in the application which are directly modifiable by the player using the built-in physics collision detection system: The origin and the axis points. When moved, they retain their position unless moved by a script. These two entities were therefore chosen to be the source of truth of the application state. It is also very convenient that a GameObject has a localPosition attribute, which is the position relative to its parent. In this case, the parent is the origin. This means that the origin can be transformed at will, and it will have no effect on the position or rotation of the axis configuration. Mathematically, these properties are also not important for the calculations and can be discarded when the axis configuration is exported from the system. This export is then simply the localPostition of all axisPoints which is then mapped to the attribute names.

4.6 Input and output

The input to the system is in the form of a *.csv* file that conforms to the schema in Table 4.1. If the problem at hand is a classification task, the class column will dictate the colours of the datapoints. If there is no class attribute, all the datapoints will have the same colour. The system only accepts numbers as attribute values, but can easily be extended to include discrete values as well.

The output from the system is a *.txt* file that conforms to the schema in Table 4.2. This configuration can then be used to transform the whole dataset into the 3D star coordinate version. An example of how to do that is found in Appendix B.

Algorithm 1: Main update loop

```
Input  : Datapoints(gameObjects) $D$
         Axes(gameObjects) $A$
         AxisLines(gameObjects) $L$
         Paths(gameObjects) $P$
         Origin scale $scale$
         Active datapoint $ad$
         hasMoved(axis index) $hasMoved$
/* If the origin scale has changed, scale the children down
   accordingly                                                    */
if previousScale != currentScale then
    for each datapoint $d \in D$ do
        $d.scale \leftarrow 1/scale$
    end
    for each axisPoint $a \in A$ do
        $a.scale \leftarrow 1/scale$
    end
    for each axisLine $l \in L$ do
        $l.scale \leftarrow (1/scale, l.scale.y, 1/scale)$
    end
end
/* If a datapoint is selected, draw the path from origin          */
if $ad >= 0$ then
    $P.setActive \leftarrow true$
    $paths \leftarrow calculatePath(ad)$
    for each $p, i \in paths$ do
        $p.getChild(i).setPosition \leftarrow p$
    end
end
else
    $P.setActive \leftarrow true$
end
/* If an axis has moved, update axes                              */
if $hasMoved$ != $-1$ then
    for each $axis \in axisHolder$ do
        $axis.setActive \leftarrow checkIfActive(axis)$
        if $axis.position.magnitude < 0.7$ then
            $disableAxis(axis)$
        end
        else
            if $axis$ is close to other axes then
                $mergeAxes(axis, otherAxes)$
            end
        end
        UpdateAxisData($axis$)
        RepositionAxisLine($axis$)
        UpdateAxisName($axis$)
        SetActive($axis$)
    end
    /* Update all datapoints in local coordinates                 */
    for each datapoint $d \in D$ do
        $d.position \leftarrow$ calculateGroupedStarCoordinates($d$)
    end
end
```

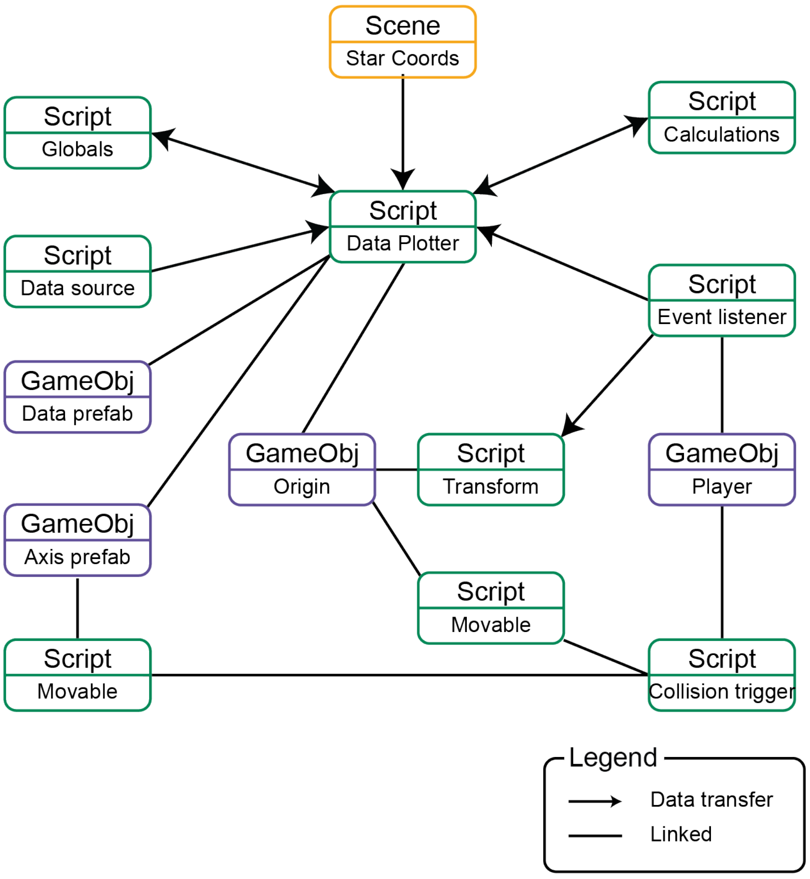


Figure 4.2: Star coordinates implementation overview

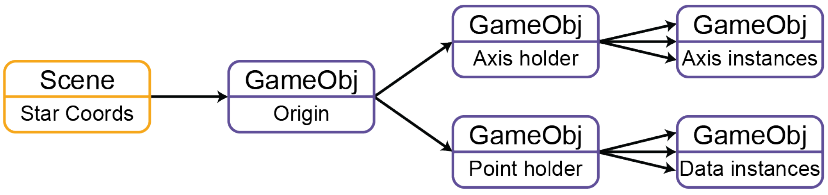


Figure 4.3: Application hierarchy

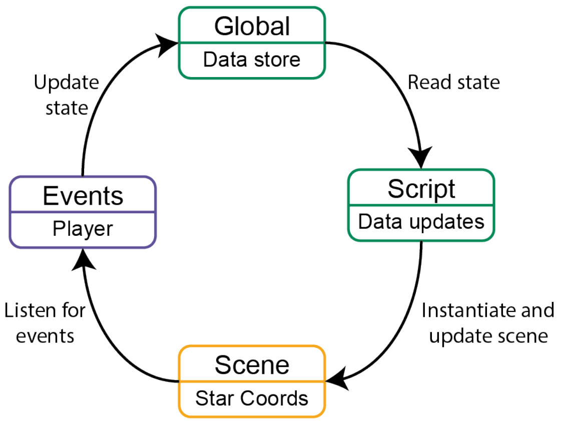


Figure 4.4: Application update loop

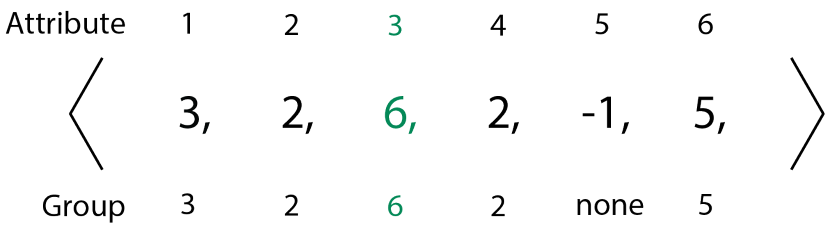


Figure 4.5: Grouping array. The green numbers highlight attribute 3 who is a part of group 6.

Algorithm 2: Computation of grouped Star Coordinates

```
Input  : Datapoint $D_j$
         Axis group configuration vectors $A$
         Mapping of attributes to groups $G$
Output: 3D vector $P_j$ representing position in 3D space
1 $P_j \leftarrow$ Zero vector
2 for each group $g \in G$ do
3     $groupValue \leftarrow 0$
4     for each attribute $i \in g$ do
5         $groupValue \leftarrow groupValue + D_{ji}$
6     end
7     $P_j \leftarrow P_j + A_g * (groupValue/|g|)$
8 end
9 return $P_j$
```

attribute 1	···	attribute N	class(optional)
4	···	1.0	Fish
1	···	1.1	Fish
6	···	0.7	Mammal
7	···	1.2	Fish
⋮	⋮	⋮	⋮

Table 4.1: Input data structure for SC application

attribute	x	y	z
attribute 1	1.4	-0.3	3.2
⋮	⋮	⋮	⋮
attribute N	1.4	-0.3	3.6

Table 4.2: Output data structure from SC application

Chapter 5

Results and discussion

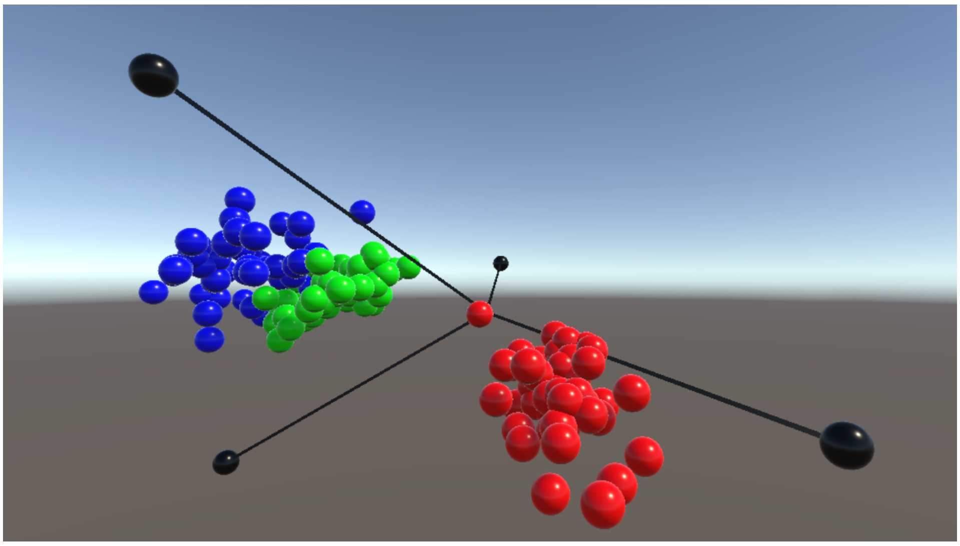


Figure 5.1: Screenshot from the project showing the Iris dataset.

Portraying a 3D immersive experience on a two-dimensional static piece of paper leaves a lot to be desired. A screenshot from the application can be seen in Figure 5.1. The black lines and spheres are the axes of the dataset and the coloured spheres are the data points. The dataset is the Iris Dataset as described earlier, and the colours represent the species. Even though this is a two-dimensional representation for the purpose of being presented on paper, some three-dimensionality can be inferred from the size of the spheres. The axes are distributed evenly in 3D space, and we can easily see how the red data points have a much stronger dependency on the axis pointing towards the camera. In the following section, a synthesised random dataset was created to measure performance and give an indication of cognitive overload. Keep in mind that a random dataset is a worst case scenario, as there is no correlation between data points.

5.1 Number of axes

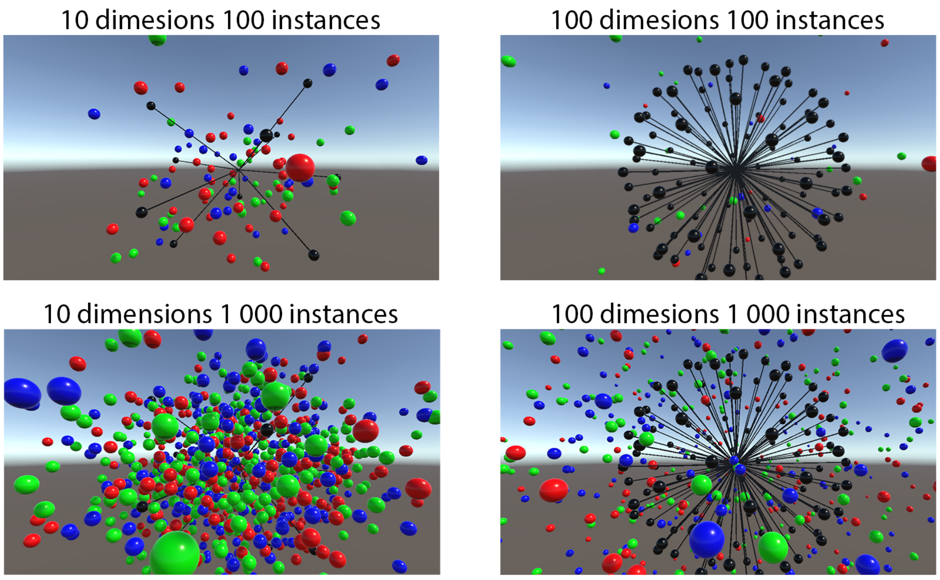

10 dimesions 100 instances
100 dimesions 100 instances
10 dimensions 1 000 instances
100 dimesions 1 000 instances

Figure 5.2: Comparison of different number of axis

The number of axes is given by the number of attributes in the dataset. Figure 5.2 shows the difference between 10 and 100 axes on the same dataset with 100 and 1000 instances. Even though complexity is a subjective matter, using this system with 100 axes is simply not feasible. Based on experience, the sweet spot for the numbers of axes is less than 10.

If the system were to be used with a dataset with high dimensionality, possible solutions include selecting a subset of the attributes or to combine the attributes.

5.2 Number of instances

The number of instances, or data points, is given by the size of the dataset. Figure 5.3 shows a comparison of 100, 250, 1000, 10000 instances. 250 instances were found to be the sweet spot for the randomised dataset, and 600 was sufficient for real-world datasets. A possible feature would be to allow the user to select the number of instances to show. For Parallel Coordinates, which has a lot in common with Star Coordinates, selecting a random subset of the dataset has proven to be the most accurate solution to solving cluttering of the visualisation[39].

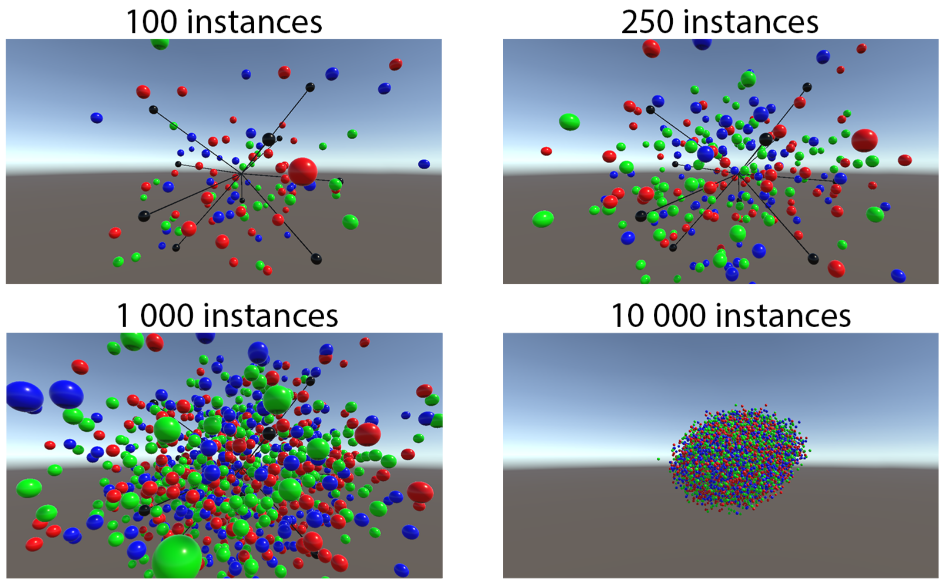


Figure 5.3: Comparison of different number of instances

5.3 Performance

Performance in a VR application is very important to prevent motion sickness and nausea. In Table 5.1 are the measured frames per second for the different random datasets. These are the lowest consistent framerates (ignoring short spikes) that are recorded while transforming the Star Coordinates axis configuration. This section will focus on CPU performance, as this is shown to be the bottleneck in this system by a large margin. GPU's are made to handle complex scenes with large amounts of geometry, like those found in modern videogames. Rendering a few thousand spheres is a simple task for a decent GPU, and the amount of video and system memory needed is also rather small. The specifications of the computer used to run the tests are found in Table 5.2. The refresh rate of the Vive is 90FPS, which is required in order to prevent VR motion sickness. In order to maintain a 90FPS framerate, all calculations must be performed faster than the time a frame stays on the screen. This can be seen as

$$timePerFrame = \frac{1}{framerate}$$

so in order to maintain 90 frames per second, the calculation can take no more than 11 milliseconds.

The colours found in Figure 5.5, Figure 5.6 and Figure 5.7 correspond to the labels in Figure 5.4.

An analysis of Table 5.1 shows that the only fully usable result from the three tests is found in Figure 5.5(10 axes x 100 instances). The HTC Vive used to develop the software runs at 90FPS, which means that 10 attributes and 100 datapoints will provide a smooth

	100 instances	1 000 instances
10 dimensions	140 fps	20 fps
100 dimensions	10 fps	unusable

Table 5.1: Worst case performance metrics on synthesised datasets.

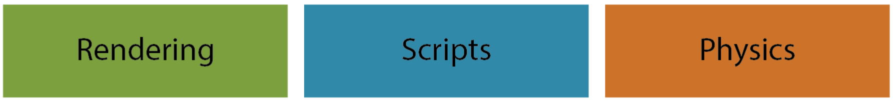
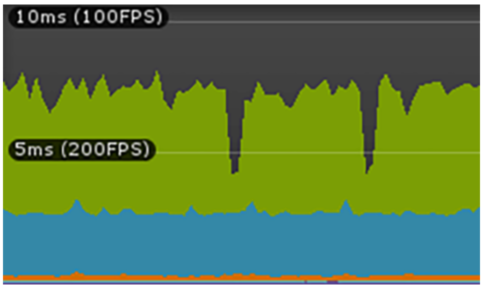


Figure 5.4: Colour map for performance graphs.

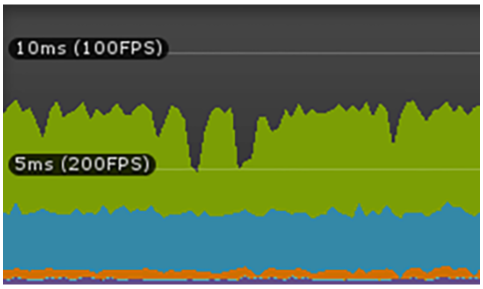
user experience without any lag, tearing or stuttering. The performance while idle, as seen in Figure 5.5a, shows a consistent framerate of about 140FPS which is well above the required 90. Most of the time between frames is used to render objects to the screen, and some script evaluation to detect events such as user interactions. In Figure 5.5b, the framerate while rotating the coordinate system is mostly similar. The most interesting difference is the work required to perform physics evaluations while rotating, whose job is to check for collisions between the data and axis points and the user controllers. There is however a huge change in performance when one of the axes are moved. This can be seen in Figure 5.5c. When the axes are moved, all the points positions in local space must be recalculated and updated. This is a rather large task that requires the multiplications of all axes and all datapoint attributes. This is all done on the CPU, and is therefore substantially contributing to the framerate reduction.

The next test increases the number of datapoints to 1000. The performance can be seen in Figure 5.6. Both idle, Figure 5.6a, and rotation, Figure 5.6b, show negligible framerate reductions from 100 datapoints.This indicates that roughly 140FPS is the baseline framerate for any scene. The amount of physics calculation has increased, as there are an order of magnitude more object to calculate interactions between. As expected, and confirmed by Figure 5.6c, the amount of CPU calculations required for the coordinate system has increased substantially. A framerate of 30FPS is usable but very tiring to the user. Script evaluation went from about $3ms$ to $30ms$.

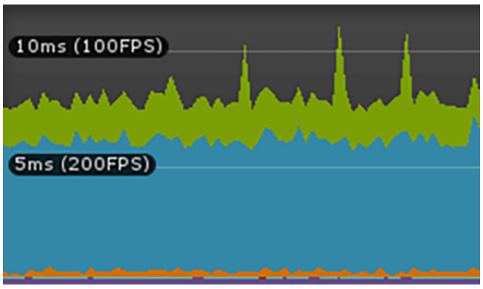
The last test set the number of attributes to 100 and the number of datapoints to 100. The result can be seen in Figure 5.7. Even when idle, Figure 5.7a, and when rotating, Figure 5.7b, the framerate is inconsistent but usable. The axes have a lot more functionality related to them than the datapoints, and the complexity of the calculations increases. When transforming the performance is abysmal, as can be seen in Figure 5.7b, and absolutely unusable. Using the system at 15FPS for even a minute causes nausea.



(a) Performance while idle.



(b) Performance while rotating



(c) Performance while transforming

Figure 5.5: Profiling. 10 axes and 100 datapoints.

5.4 Comparison to iViz

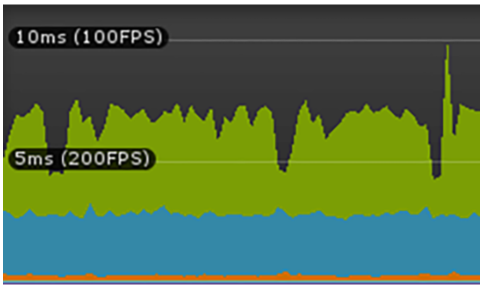
In this section, a comparison to iViz is performed, highlighting the key differences in the methods and tools used.

Interactivity and collaboration

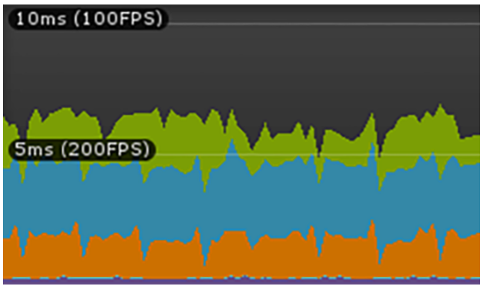
The interaction is one of the main differences between this project and iViz. The interaction model of iViz, described in section 2.5, is very static, and based on their publication[24], does not allow for any other transformations than translation and rotation. The only parameter available to the user is the dimension mapping.

Operating System	Microsoft Windows 10 Pro
CPU	Intel i7-5820K 6-core @ 3.30GHz
RAM	32GB
GPU	NVIDIA GeForce GTX 980 Ti
Game engine	Unity 2017-2-0f3
VR system	HTC Vive

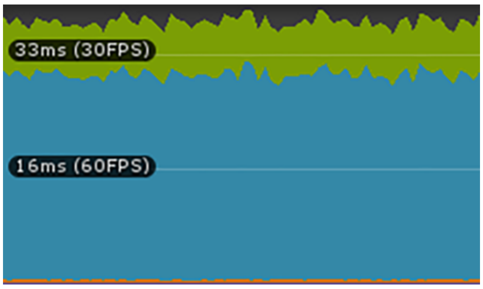
Table 5.2: Software and hardware specifications on test machine



(a) Performance while idle.



(b) Performance while rotating



(c) Performance while transforming

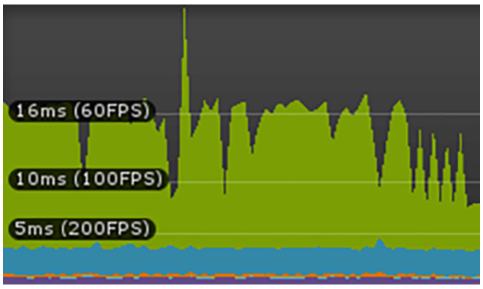
Figure 5.6: Profiling. 10 axes and 1000 datapoints.

Data representation

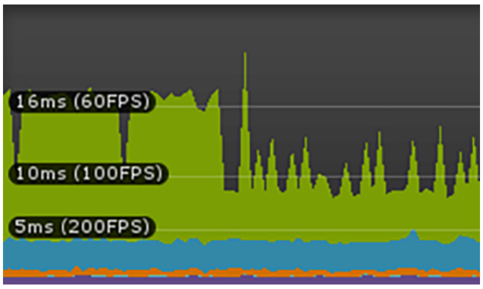
The other big difference between the projects is how they represent the data in 3D space. iViz represents data orthogonally and unambiguously by mapping attributes directly to many dimensions. Star Coordinates uses a non-orthogonal space, and the points become ambiguous but contained in three dimensions. A comparison can be made as in Table 5.3.

Data dimension	iViz	Star Coordinates
Sepal length	x-axis	x, y, z axis
Sepal width	y-axis	x, y, z axis
Petal length	z-axis	x, y, z axis
Petal width	size	x, y, z axis
attribute1	shape	x, y, z axis
attribute2	texture	x, y, z axis
attribute3	transparency	x, y, z axis
Species	color	color

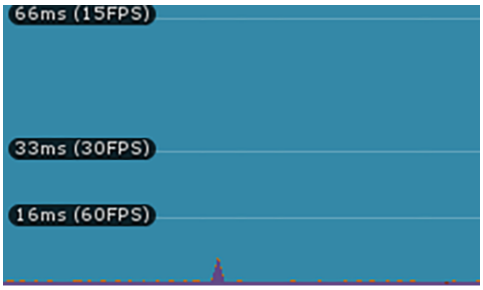
Table 5.3: Difference is axis mapping between iViz and Star Coordinates.



(a) Performance while idle.



(b) Performance while rotating



(c) Performance while transforming

Figure 5.7: Profiling. 100 axes and 100 datapoints.

Patents

The team behind iViz has also filed a patent for *Systems and methods for data visualization using three-dimensional displays* with patent number *US9665988B2*. At the surface, this patent seems very broad and generic. It describes their system in meticulous detail and seeing as many of the main components differ between our approaches, the patent is highly unlikely to cover this project.

5.5 Good vs bad configuration

When moving the Star Coordinate axes around, the goal is to achieve the maximum level of separation between the classes. To illustrate this, a dataset containing votes from the Senate in the USA has been used as a simple example with two classes. The class is either *Republican* or *Democrat*. The attributes are questions categories, and the datapoint is a person's votes on these categories. In Figure 5.8, the results of 3 minutes of work are shown. Most of the datapoints are situated along with their respective class, while a few are lost in the middle. These in the middle are the cause of the non-perfect classification score and should be minimised to the best of ones ability. It is however not always possible, as outliers do exist in most real-world datasets. When ran through kNN with $k = 3$, the accuracy was at 94.8%.

In Figure 5.9, we see an axis configuration that separates the classes to a lesser extent than the previous example. How difficult it is to separate the classes comes down to the properties of the dataset, like the degree of correlation between attributes, number of classes etc. When ran through the same kNN configuration, the accuracy was at 84.4%, which shows the importance of a good axis configuration.

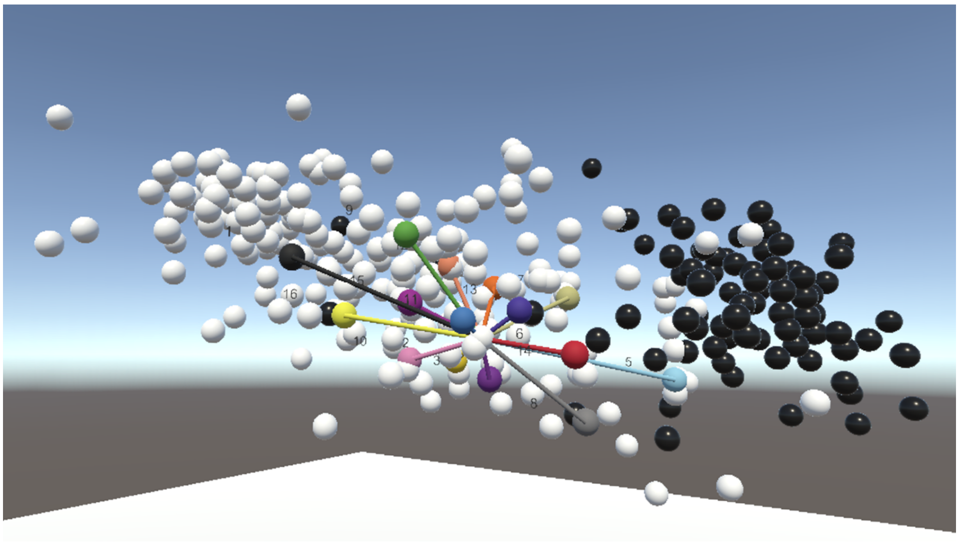


Figure 5.8: Good axis configuration with separation between the classes.

5.6 kNN accuracy and size reduction

For the Iris dataset, all methods provided good results. This is not a particularly difficult dataset to work with, apart from some datapoints that are in between two classes. These are hard to separate without any continuous feedback on the current configuration.

With the HTRU2 dataset, the goal is to binary classify pulsars, which is a type of neutron star. The dataset contains different scientific metrics of the candidates. Here, StarVR and kNN alone provided better results than PCA. This dataset was easy to work with in StarVR, and the separation of classes was easier to find than in a lot of the other datasets.

The Congressional Voting Records data set contains the votes of members of Congress concerning different topics (the attributes). The goal is to classify the member to his/her party based on these votes. This dataset was fairly easy to work with, but had some of the problems that the Iris dataset had with rouge datapoints between the clusters. StarVR performed marginally better than PCA, and both were worse than kNN alone.

The glass identification data set contains the mineral composition of glass samples and the goal is to classify the type of glass. This dataset has 7 classes, which were hard to keep track of when using StarVR. Even pure kNN struggled with this one. PCA managed a perfect score. This is an indication that implementing PCA into StarVR could have huge

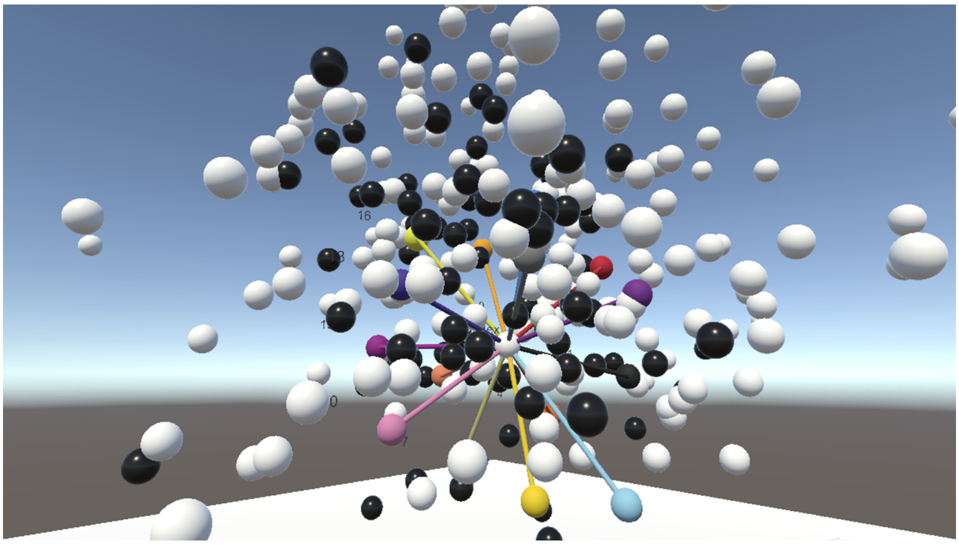


Figure 5.9: Bad axis configuration with little separation with classes.

benefits both in terms of accuracy and time saved configuring axes and is discussed further in section 6.3.

The last, and most difficult, dataset was the Breast Cancer Wisconsin (Diagnostic) dataset. This dataset has 14 axes. It was difficult for the user to understand how each axis affects the plot as a whole, and moving them did not seem to do provide better segregation of the classes. Both kNN and PCA provided good results, and this further backs up the idea of including PCA into StarVR to be able to handle more complex datasets.

Dataset	dims	points	classes	kNN acc	PCA acc	StarVR acc
iris [4]	4	150	3	98.0%	100.0%	96.0%
HTRU2 [40][41]	10	17898	2	97.1%	94.5%	97.0%
votes [42]	16	408	2	97.0%	93.3%	94.8%
glass[43]	10	214	7	70.4%	100.0%	67.6%
cancer [44]	14	569	2	94.7%	92.0%	N/A

Table 5.4: kNN performance results comparing StarVR and PCA. k has been set to 4 for both methods and all datasets.

Chapter 6

Conclusion

This project has enlightened both theoretical and practical aspects of using Star Coordinates in Virtual Reality. From a theoretical and mathematical perspective, the extension of Star Coordinates into three dimensions is fairly straightforward and provides a larger space of operation. The dataset then retains more of its original information. From a practical perspective, the usefulness and the practicalities of the application have been considered. Using a game engine to develop data visualisation software for VR has been a pleasant experience, for the most part. The setup was mostly painless, apart from some configuration and version mismatching of the plugins used in Unity.

6.1 Usability

One of the main selling points of VR is that the user can interact with the application as if it were real life. The interactions have their real-life counterparts, like grabbing, moving, stretching and turning. By leveraging these human motor functions and mapping them to application tasks, the usage of the application quickly becomes second nature.

It is important to talk about the target audience when discussing usability. The application has been tested mostly on people who are somewhat technically focused, and the results might not be representative of the population at large. It does not require any technical experience to use, but in order to understand what to do and why the datapoints behave as they do, a certain amount of knowledge is required. The application is more catered towards data scientists and students who have an interest in the field of data science.

6.2 Performance

While the performance of the system is good enough for displaying more elements than a user would need, there is no reason to stop there. The main bottleneck of the application showed to be the single-threaded, synchronous execution of the star coordinates calculation. The task is performed on the same thread as the application itself, and is the main factor for lowered performance while configuring the axes. The computer used for this project has a lot of CPU cores, but the single thread performance is not great at 3GHz. A CPU with fewer cores and higher clock speed would probably yield better results. This is however not a scalable solution, but merely a band-aid. In order to further enhance the application performance, a re-architecture is required. The most obvious solution is to move the slow code to an asynchronous thread. This would result in the datapoints positions not updating as smoothly, but the overall frame-rate would increase immensely. This would complicate the application further, which was found not beneficial for this prototype.

The important take-away is not the absolute performance, but the fact that anyone which some technical knowledge and programming experience can make this application work. It is therefore what I would consider a lower threshold of performance, and if someone were to make this a commercial product, they would have the know-how and resources to achieve much greater performance. StarVR serves as a proof that implementing Star Coordinates in Virtual Reality using a commercial game engine is doable from a performance perspective.

6.3 Dimensionality reduction

Based on the results discussed earlier, the results of dimensionality reduction for kNN classification show that StarVR has comparable results to automated methods. The main benefit over alternative methods is the ability to tweak the axis configuration to optimise for the dataset at hand. Not all datasets will yield the best results when optimised for variance or other metrics. It can also use these methods as a starting point for further manual optimisation to improve results further. There are however some limitations of the dimensionality of the dataset, and some datasets lend themselves better to Star Coordinates than others. For datasets with less than 10 attributes, the results were comparable and sometimes better, than automated PCA. Some datasets were also harder to make sense of than others, which is probably due to how the attributes are related to each other. If the dataset has some attributes that separate the classes more distinctly than others, these are easier to find and use as a starting point. Without these starting points, it becomes harder to find a good configuration in a short time. A possible solution to this is to use existing data reduction techniques as a starting point for StarVR. This would in many situations resolve this issue, or at least remedy it. Star Coordinates is therefore not a perfect solution to all dataset, but as many other techniques, a tool in a data analysts toolbox.

6.4 Commercial exploitation

Making a proof of concept is useful for validating a hypothesis, but in order to reach real-life users, a more commercial approach should be considered. The 3D data exploration space is not crowded with competitors, and those who exist fail to impress industry professionals. One of the more serious competitors, who have a similar scope to this project, is DatavizVR[45]. They launched in 2016, and have not had many updates since. The initial public reaction was positive to the use of data visualisation in VR, but the implementations and features were not good enough for data visualisation experts and business users.

If the star coordinates virtual reality application were to be expanded with some of the most basic features, like import and export, real-time analysis and integration with common data manipulation tools and frameworks, the possibility of commercial success is real.

6.5 Other applications

While the scope of this project has been evaluated above, some new applications of the project have emerged during its development. While it performs well at its intended tasks, it could also provide value to other parts of eScience and society as a whole.

The StarVR applications primary goal is to manipulate data and perform dimensionality reduction. There is however nothing in the way of using it as a static, immersive data visualisation tool.

Another area to consider is eLearing and gamification. The StarVR application could be used at high school and higher level to provide a more hands-on approach to eScience and mathematics. Using digital learning tools is becoming more main-stream, and using Star Coordinates in VR might have a motivating effect on students who are more visual learners rather than simply stating formulas and theorems. It could even implement gamification elements, like performing the best configuration for certain data analysis tasks.

Chapter 7

Future work

In this thesis, it has been shown that a tool like StarVR has a place in the world of data science. However, this does not mean that StarVR in its current form is ready for the commercial market. It is merely a prototype, meant to showcase the possibilities of interactive VR in data science. This section discusses the implications and steps necessary to take the ideas and mechanics of StarVR to the next level, while still being able to add new and useful features and integrations to be able to provide a versatile tool for professional data science and visualisation.

7.1 Performace

The main bottleneck of StarVR is real-time performance. While using a high-level framework for prototype use speeds up prototyping immensely, it also comes with its fair share of drawbacks. Unity is mostly used for game development, and the architectural decisions of the platform reflect that. Performing complex data analysis tasks is not the norm. In order to provide considerable performance improvements, more than simply optimising the current system would be to start building from a lower abstraction level. It is much more resource intensive in terms of manpower and expertise required but comes with fewer of Unity's drawbacks.

The main problem regarding performance is the calculation of the Star Coordinates on all datapoints. The CPU is excellent at performing a sequence of tasks rapidly, but it is not optimised for complex parallel workloads. It would be of interest to experiment with offloading the computation of Star Coordinates to a GPU. If successful, it will to a large degree remedy the main performance concern of StarVR.

While it might seem like leaving Unity only has benefits, it is far from the truth. Implementing a VR application on a range of operating systems, architecture, hardware and software is a massive undertaking. It could be of interest to investigate other VR compatible solutions, which are lower level than Unity and allows for more fine-grained control over how the calculations and visuals are carried out.

7.2 Integrations

In order to make the application more useful for the working professional, it is important to make the workflow as seamless as possible in conjunction with other industry standard tools, libraries and systems.

The next logical step would be to incorporate StarVR with common data storage strategies. Loading files from the filesystem as well as network locations are the most basic functionality while connecting to local databases and being able to query for data might be more advanced use cases. It would also be of interest to be able to export data back to the sources for storage, as well as any produced metadata, such as axis configurations and analytic scores. Another possibility is to allow streaming of real-time data, and visualise and configure the axes in a sliding time window where datapoints appear and disappear after a certain amount of time. Then, one would be able to see the evolution of the kNN score, for instance, over time and look for how time affects the incoming datapoints.

Integrating common data analysis methods is a two-edged sword. It is very nice to have if you need it, but quite useless and bloated if you don't. Some sort of pluggable system, using common data structures and interfaces could be of interest. Creating data structures compatible with scikit-learn, tensorflow and pandas, just to mention a few, could be very useful.

7.3 Dimensionality reduction

The dimensionality reduction of StarVR works very well when it does, and quite poorly when it doesn't. Using PCA as a good starting point should be investigated. Together with real-time feedback through integrations, the user could be able to consistently perform on par or better than PCA when performing kNN. Real-time classification comes with a lot of the problems of the calculation of Star Coordinates, but the penalty of waiting for a few frames is far less severe. Integrating the kNN classifier more deeply with Star Coordinates could also provide a performance increase, but it will decrease the modularity and pluggability of the system. This is a trade-off that must be evaluated based on business and user needs.

7.4 User testing and evaluation

The system is of no value if the users cannot perform their task in an effective and accurate manner. It is important to investigate common interaction models and evaluate their usefulness in an interactive 3D application. The aim is to provide a more intuitive and easy-to-use solution, and testing and evaluation on real users are necessary to validate this claim.

Bibliography

[1] E. Kandogan, "Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions," in *Proceedings of the IEEE Information Visualization Symposium*, vol. 650, p. 22, Citeseer, 2000.

[2] S. John Walker, "Big data: A revolution that will transform how we live, work, and think," 2014.

[3] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: from big data to big impact," *MIS quarterly*, pp. 1165–1188, 2012.

[4] R. Fisher, "Iris data set," 1936. `https://archive.ics.uci.edu/ml/datasets/iris` [Accessed 08.10.2018].

[5] J. Steuer, "Defining virtual reality: Dimensions determining telepresence," *Journal of communication*, vol. 42, no. 4, pp. 73–93, 1992.

[6] D. A. Bowman and R. P. McMahan, "Virtual reality: how much immersion is enough?," *Computer*, vol. 40, no. 7, 2007.

[7] OculusVR, "Oculus rift," 2019. `https://www.oculus.com/rift`.

[8] "Htc vive." `https://www.vive.com` [Accessed 05.12.2018].

[9] "Google cardboard." `https://vr.google.com/cardboard/` [Accessed 05.12.2018].

[10] N. M. Papachristos, I. Vrellis, and T. A. Mikropoulos, "A comparison between oculus rift and a low-cost smartphone vr headset: immersive user experience and learning," in *Advanced Learning Technologies (ICALT), 2017 IEEE 17th International Conference on*, pp. 477–481, IEEE, 2017.

[11] J. Laird, "Research in human-level ai using computer games," *Commun. ACM*, vol. 45, pp. 32–35, 01 2002.

[12] "Unity." `https://unity3d.com/` [Accessed 05.12.2018].

[13] “Unreal engine.” `https://www.unrealengine.com` [Accessed 05.12.2018].

[14] “Cryengine.” `https://www.cryengine.com/` [Accessed 05.12.2018].

[15] T. M. Mitchell, “Machine learning,” 1997.

[16] J. Bertin, *Graphics and graphic information processing*. Walter de Gruyter, 2011.

[17] A. Inselberg, “Parallel coordinates: A guide for the perplexed,” in *Proc. of IEEE Visualization, 1996*, 1996.

[18] N. D. Cooprider and R. P. Burton, “Extension of star coordinates into three dimensions,” in *Visualization and Data Analysis 2007*, vol. 6495, p. 64950Q, International Society for Optics and Photonics, 2007.

[19] E. Kandogan, “Visualizing multi-dimensional clusters, trends, and outliers using star coordinates,” in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 107–116, ACM, 2001.

[20] A. K. Jain, “Data clustering: 50 years beyond k-means,” *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.

[21] J. A. Hartigan and M. A. Wong, “Algorithm as 136: A k-means clustering algorithm,” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[22] E. Keogh and A. Mueen, “Curse of dimensionality,” in *Encyclopedia of machine learning*, pp. 257–258, Springer, 2011.

[23] J. E. Jackson, *A user’s guide to principal components*, vol. 587. John Wiley & Sons, 2005.

[24] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, *et al.*, “Immersive and collaborative data visualization using virtual reality platforms,” in *Big Data (Big Data), 2014 IEEE International Conference on*, pp. 609–614, IEEE, 2014.

[25] “vcaltech.” `http://www.astro.caltech.edu/~george/vcaltech/` [Accessed 09.12.2018].

[26] “Opensim.” `http://opensim.stanford.edu/` [Accessed 09.12.2018].

[27] “Leap motion sensor.” `https://www.leapmotion.com/` [Accessed 09.12.2018].

[28] G. G. Zanabria, L. G. Nonato, and E. Gomez-Nieto, “istar (i*): An interactive star coordinates approach for high-dimensional data exploration,” *Computers & Graphics*, vol. 60, pp. 107–118, 2016.

[29] H. Abdi and L. J. Williams, “Principal component analysis,” *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[30] K. L. Kelly, “Twenty-two colors of maximum contrast,” *Color Engineering*, vol. 3, no. 26, pp. 26–27, 1965.

[31] “Openvr sdk.” `https://github.com/ValveSoftware/openvr` [Accessed 05.12.2018].

[32] “Valve software.” `https://www.valvesoftware.com` [Accessed 05.12.2018].

[33] D. Jackson, *Software abstractions*, vol. 2. MIT press Cambridge, 2006.

[34] W. Goldstone, *Unity game development essentials*. Packt Publishing Ltd, 2009.

[35] “Unity gameobject.” `https://docs.unity3d.com/ScriptReference/GameObject.html` [Accessed 05.12.2018].

[36] “Unity scripting api.” `https://docs.unity3d.com/ScriptReference/` [Accessed 05.12.2018].

[37] “Flux application architecture.” `https://github.com/facebook/flux` [Accessed 03.06.2019].

[38] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[39] G. Ellis and A. Dix, “Enabling automatic clutter reduction in parallel coordinate plots,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 717–724, 2006.

[40] D. R. Lyon, “Htru2 data set.” `https://archive.ics.uci.edu/ml/datasets/HTRU2` [Accessed 03.06.2019].

[41] S. C. J. M. B. J. D. K. R. J. Lyon, B. W. Stappers, “Fifty years of pulsar candidate selection: From simple filters to a new principled real-time classification approach,” pp. 1104–1123.

[42] J. Schlimmer, “Congressional voting records.” `https://archive.ics.uci.edu/ml/datasets/congressional+voting+records` [Accessed 03.06.2019].

[43] B. German, “Glass identification data set.” `https://archive.ics.uci.edu/ml/datasets/glass+identification` [Accessed 03.06.2019].

[44] O. L. M. Dr. William H. Wolberg, W. Nick Street, “Breast cancer wisconsin(diagnostic).” `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)` [Accessed 03.06.2019].

[45] “Dataviz vr.” `https://www.oculus.com/experiences/rift/1251660221572429` [Accessed 04.06.2019].

Appendix A

Python data normalisation

```
from pathlib import Path
from pandas import pandas

def normalize(df):
    result = df.copy()
    # Normalize all but the last feature (species name)
    for feature_name in df.columns[0:-1]:
        max_value = df[feature_name].max()
        min_value = df[feature_name].min()
        difference = max_value - min_value
        value = (df[feature_name] - min_value)/difference

        result[feature_name] = value
    return result

df = pandas.read_csv(Path('dataset.csv')).drop('index', 1)
normalize(df).round(4).to_csv(Path('normalized.csv'))
```

Appendix B

Apply SC transformation

```
import pandas as pd

transform = {}

with open('./glass/starConfig.txt', 'r') as t:
    for line in t:
        [attr, vector] = line.split(':')

        transform[attr] = [float(x.strip())
                           for x in vector.strip()[1:-1].split(',')]

dataset = pd.read_csv('./glass/original.csv')

def trans(row):
    res = [0, 0, 0]
    for key in transform:
        res[0] += row[key] * transform[key][0]
        res[1] += row[key] * transform[key][1]
        res[2] += row[key] * transform[key][2]

    return res


dataset = normalize(dataset)

# Apply the transformation function
dataset['coord'] = dataset.apply(trans, axis=1)

# Split the 'coord' column into 'x', 'y', 'z'
```

```
dataset[['x', 'y', 'z']] = pd.DataFrame(
    dataset.coord.values.tolist(), index=dataset.index)

dataset.to_csv('./glass/starCoordinates.csv', columns=[
                'index', 'x', 'y', 'z', 'class'], index=False)
```

Appendix C

Running the StarVR application

In order to run the StarVR application, a Htc Vive headset and a fairly decent computer with a graphics card is required. One can either run the pre-built binary for Windows or build the project from source-code.

Run prebuilt binary

The easiest approach, if it works, is to simply start the executable that accompanies this thesis. This is dependent on system architecture and OS, so it might not work for everyone.

Build from source

Don't do this unless you are in for some debugging, as it can be tricky to set up. In order to build StarVR from source-code and use your own datasets, the follow these steps:

- Download accompanying source-code and unzip.
- Download Unity along drivers and plugins and initialise the project.
- Build the project and add your own datasets in the resources folder.
- Run the build.

If neither of these works, you can contact the author in order to get access to the online Unity code repository.