NTNU
Norwegian University of
Science and Technology

High-Dimensional Data Visualization

Mikal Nielsen

Master of Science in Cybernetics and Robotics
Submission date: May 2017
Supervisor: Sverre Hendseth, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

In modern research, high-dimensional data is omnipresent. These data are often complex and difficult to interpret. A proven method to make data more interpretable is to utilize the human vision, and visualize the data set. As display devices and our visual abilities limit dimensionality to three, methods to visualize higher-dimensional data have been massively researched. It is a field of high interest among academics, driven by ever-increasing computer power, improved data acquisition and needs for analyzing this complex data.

The structure of the problem is twofold. First, the task is to present a thorough literature research on the field, to unveil methodology relevant in the field, including both traditional and modern approaches.

Second, the task is to create a computer program that enables high-dimensional data visualization. The application should implement an interactive scene and involve user interaction, to allow for data exploration in several dimensions and grant visual interpretation of the high-dimensional data.

Summary

High-dimensional data is challenging to visualize. To gain some insight in the field, there has been conducted a literature research which is presented in this report. Visualization is done to accelerate and increase interpretation of data, and there are many approaches to visualize data that exceeds three dimensions. Some of these methods manipulate the numerical data, and make it presentable in classic two- or three-dimensional plots. Others focus on displaying multiple dimensions at once, using innovative graphing techniques.

An interactive visualization application has been implemented, through the use of a graphics rendering engine. Here, the data is presented in a traditional three-dimensional scatter plot, after having been projected onto a three-dimensional space. The user of the application can control rotations of the data using the computer's keyboard and mouse, and hopefully, gain more knowledge of the data.

Sammendrag

Høydimensjonell data er utfordrende å visualisere. For å få et innblikk i feltet har det blitt utført et litteraturstudie som presenteres i rapporten. Visualisering blir gjort for å raskere og bedre tolke data, og det finnes mange tilnærminger til å visualisere data som overskrider tre dimensjoner. Noen av disse metodene manipulerer den numeriske dataen, og gjør den presentabel i klassisk to- eller tredimensjonale plott. Andre fokuserer på å vise flere dimensjoner samtidig, ved hjelp av innovative plotteteknikker.

Et interaktivt visualiseringsprogram har blitt implementert, ved hjelp av en grafikkmotor. Her presenteres dataen i et tradisjonelt tredimensjonalt spredningsplott, etter å først ha blitt projisert ned i et tredimensjonalt rom. Brukeren av programmet kan kontrollere rotasjoner av dataen ved å bruke datamaskinens mus og tastatur, og gjennom dette forhåpentligvis få mer kunnskap om dataen.

Preface

This report is written in compliance with the Master of Science degree in Cybernetics and Robots at the Norwegian Univeristy of Science and Technology, during the spring semester of 2017.

I acknowledge the assistance offered by associate professor Sverre Hendseth at the Department of Engineering Cybernetics, Norwegian University of Science and Technology. For always being available, showing great interest in the task, and giving motivational talks to help keep up the spirit.

Mikal Nielsen

Trondheim, May 2017

Table of Contents

List of Figures

Abbreviations

OpenGL	=	Open Graphics Library
GLSL	=	OpenGL Shading Language
API	=	Applicaton Programming Interface
IDE	=	Integrated Development Environment
lwjgl	=	Lightweight Java Game Library
Java SE	=	Java Standard Edition
JVM	=	Java Virtual Machine
JRE	=	Java Runtime Environment
JDK	=	Java SE Development Kit
CPU	=	Central Processing Unit
GPU	=	Graphics Processing Unit
PCA	=	Principal Component Analysis
iPCA	=	interactive PCA
LDA	=	Linear Discriminate Analysis
t-SNE	=	t-Distributed Stochastic Neighbor Embedding
MDS	=	Multidimensional Scaling
CBF	=	Cell Based Clustering
VAO	=	Vertex Array Object
VBO	=	Vertex Buffer Object

Chapter 1

Introduction

For a long time, visualization has helped gain insight in data, by utilizing the fantastic human vision. Traditional visualization methods fall short in visualizing data of many dimensions. Developing new methods for doing this, is an ever-increasing field of research. From simple visualizations of bivariate relationships between dimensions, to modern data manipulation algorithms, that can handle hundreds of dimensions and discover relationships in the data automatically.

In this report, there is a presentation of methods to handle high dimensional data manipulation and visualization. Then, an implementation of such nature is suggested. It enables high-dimensional data visualization through an interactive application, which allows the user to explore the data in real-time.

1.1 Motivation

The purpose of this thesis is to seek knowledge about how high dimensional data is turned into graphics plotted on a two-dimensional display device, and still reveal a high amount of the information in the data. Meanwhile, it is interesting to test how to an implementation can be realized, through the use of a graphics rendering engine, and a portion of data manipulation.

1.2 Tools

As tools for the implementation of a graphic visualizer, the OpenGL API was chosen to enable hardware accelerated graphics rendering. Java is used as the programming language, along with the lightweight java game library, through the Java

Eclipse Neon IDE and Java DK 8u111. Through this, the program can render the scene in a window and introduce user interaction through the computer's keyboard and mouse.

The PC used throughout the project has the following specifications:

- **Processor** Intel(R) Core i7-6700 CPU @ 3.40 GHz 3.41 GHz
- **Memory** 32,00 GB
- **Operating System** Windows 10 Education, 64-bit
- **Graphics Card** Intel(R) HD Graphics 530

1.3 Limitations

Perhaps the most important limiting factor in this project was the fact that high dimensional data is hard to imagine. Our minds are used to a three-dimensional world, and the idea of increasing the number of dimensions is abstract. This, however, is the reason this field of research exist, and why working with it is of such interest.

1.4 Report Structure

This report starts with a literature review on the subject of data visualization, focusing mostly on high-dimensional data. Then, there is some theory presented, spotlighting methods used in the implementation. The specification of the implementation is described, and its structure outlined. Finally, the resulting application is presented and reviewed.

Chapter 2

Literature Review

This chapter is presenting information from relevant articles, books and publications regarding data visualization and graphics. The structure of the review is chronological at first, starting out with early graphics history. Following that, the structure is more based on topics in modern and state of the art high-dimensional data processing, manipulation and visualization techniques. Finally, some tools for visualization are mentioned and articles from certain fields of research are presented.

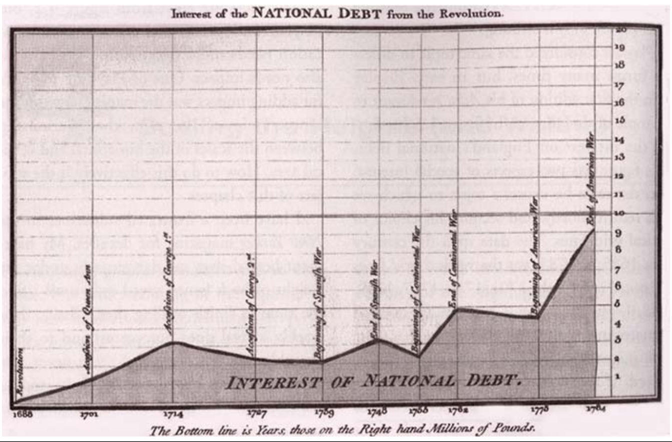
Charts and graphs are graphical representations of numeric data. The visualization of data makes it easier to interpret the information, and see how different parameters relate to one another. Mathematical and aesthetic rules form the grammar of graphics, which takes the limited set of characters into a vast realm of graphical forms.

Two- or three-dimensional data is graphically presented to us on a daily basis. The appearance of the graphics provides a more intuitively understanding of numerical values. However, the techniques of creating graphics does not easily scale to four dimensions and higher. Our minds are developed for a three dimensional space, and higher dimensional spaces are hard to imagine, let alone visualize.

2.1 Early History of Data Visualization

People using tables of rows and columns to arrange data, has been documented to exist at least as early as the 2nd century. Early graphics is mostly connected to the field of cartography. However, the idea of graphically represent quantitative and numerical data first arose in the 17th century. René Descartes developed his Cartesian coordinate system. Although intended as a graphical way of performing mathematical operations, the Scotsman William Playfair, exploited the two axes-system to create a graphical showing the change of values (“y”-axis) through time (“x”-axis). In his 1786 book *The Commercial and Political Atlas*, Playfair represented 18th century English economy by using statistical graphs, bar charts, line graphs, area charts and histograms. He is also credited the creation of pie charts. Below is an example of a Playfair graphics.

Figure 2.1 An early graphic, made by William Playfair

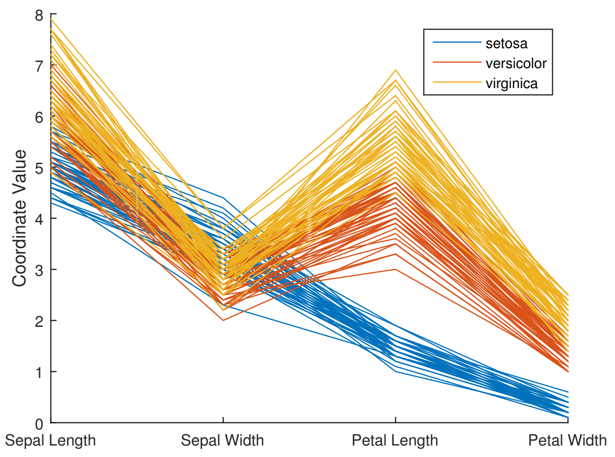


The plotting in Playfair’s graphics displayed one-dimensional data, or “1.5”- dimensional, that is changing over time. It was not until the introduction of scatter plots, often credited to the English statistician John Herschel, the first notion of two-dimensional visualization appeared. Now both axes of a Cartesian chart were representing a dimension each, of the data set values. Each data entry is plotted as a point, placed in the coordinate system based on its two data values, each given one axis.

An important historical figure in the field of data visualization is Florence Nightingale (1820-1910). Nightingale is by Wikipedia described as an English social reformer, statistician and the founder modern nursing [48]. She was a gifted mathematician. She used charts to describe different states in society. Through highly interpretable graphics, members of parliament got a description of the condition of medical care in the Crimean War, where traditional statistics might would have been more or less unreadable. Thus, the conditions were much better pointed out, making them more likely to be worked on.

During the time of Nightingale, a way of displaying higher-dimensional data was presented. A parallel coordinate plot, first introduced in 1885, allows several dimensions to be visualized at the same time.

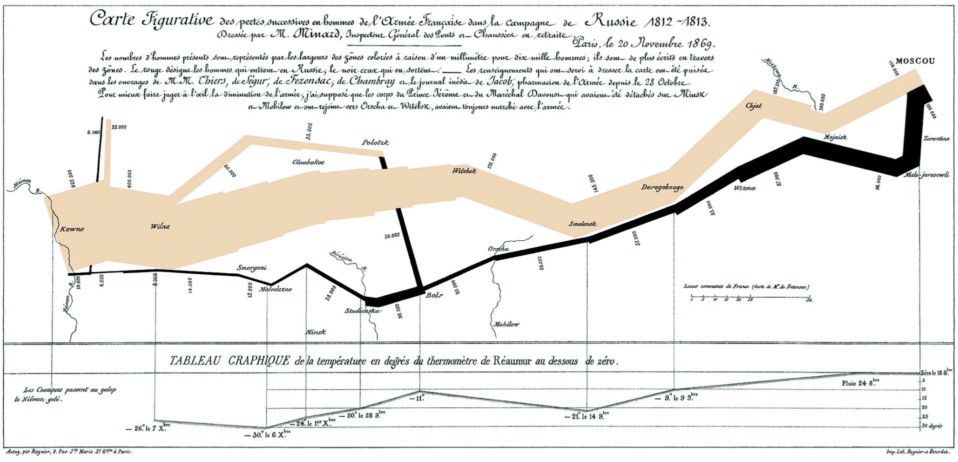
Figure 2.2 Parallel Coordinate Plot



Each increasation of dimensions adds one vertical axis; the points are then connected in series. The limitation is that a maximum of two other dimensions are directly compared to a feature, and not all patterns are revealed.

In the mid-20th century, there was put more focus on creating graphics that were highly interpretable for the perceivers. The American statistician Edward Tufte has published literature concerning quality of information visualization, which has been considered groundbreaking. Tufte is an American statistician, a pioneer in the field of data visualization. His work is regarded as an important accelerator of the popularity of graphics, by demonstrating how much information can be given in a graphic. Figure 2.3 is given by Tufte as an example of an extremely elegant and informative graphic.

Figure 2.3 War and Peace, by Minard



The French engineer Charles Joseph Minard drew the figure in 1869. It displays the French invasion of Russia, or attempt of invasion, in 1812. The horizontal tan stripe describes the march from the banks of Niemen River – with 420,000 French soldiers – from left. As the march goes on towards Moscow on the right side, the stripe narrows as soldiers die along the way. The black line, from the end of the tan one, describes the retreat. The black stripe is also narrowing; this towards left, as soldiers fall on their way home, often due to low temperature. Vertical lines in the bottom of the chart, points out significant low temperatures, and – as can be seen in the graphic - often leads to narrowing of the troops. At the end of the invasion, 10,000 French soldiers remain.

Elegant as the graphic is; it is describing the horrors of the Russian invasion, all in one drawing.

2.2 Computer Driven Data Visualization

Through modern computers, and the concepts of vector graphics and raster graphics, data graphics are now omnipresent. Visualization of data has become a familiar sight for most people. Computer programs provide a simple graph-creating interface, which allows high quality graphics to be generated by almost anyone. The power of modern computers allow for datasets that are more complex to be displayed, including data of high dimensionality.

The process of visualization can be divided into three transformation steps [19]: Data transformation, visual mapping, and view transformation. At each step, there again exist several methods, all working towards the common goal of highly interpretable visualization of data.

2.2.1 Data Transformation

In order to prepare the data for visualization, there is often some sort of data transformation. For data transformation, four different approaches are considered here: Dimension reduction, subspace clustering, regression analysis and topological data analysis.

Data Projection for Dimension Reduction

Dimension reduction is used to reduce high-dimensional data into (usually) two- or three-dimensional data for visualization. This, of course, comes with a cost. Some information is lost. Therefore, it is key to analyze how the reduction should occur. A common approach for reducing dimensionality is to project the data. The simplest way of doing this is by doing a linear projection. That is, a linear transformation to project the data from a high-dimensional space to a lower-dimensional space. Several well-known methods exist for doing this, including principal component analysis (PCA), and linear discriminate analysis (LDA).

Principal Components Analysis In PCA, linear projection depend on the singular decomposition of a real rectangular matrix of the form $\mathbf{X} = \mathbf{UDV}^{\mathrm{T}}$. The columns of $\mathbf{UD}$ are generally called the principal components of $\mathbf{X}$. The principal components represent where there data has the most variance. Plotting the first two or three principle components gives a linear projection onto a desired space. A paper by Jeong et al. [43] introduces interactive PCA (iPCA), where the user get presented, not only the projection after PCA, but also all eigenvectors in a parallel coordinate plot, the data in a parallel coordinate plot and, also, a scatter plot between each pair of variables. The user can use sliders to alter the amount of contribution of a dimension in the PCA calculation. The tool's main objective is to assist the user in better understanding and utilizing PCA for analysis.

Linear Discriminate Analysis LDA works similarly as PCA, but provides a linear projection that maximizes the class separation. The method finds a linear combination of features that characterizes or separates two or more classes of objects or events. The resulting combination may be used for dimensionality reduction before later classification.

Non-linear dimension reduction also exist. Non-linear techniques can be separated into metric and non-metric. If the input data is metric, graph-based techniques such as Isomap, Local Linear Embedding and Laplacian Eigenmap are used. In these methods, a neighborhood graph is used to capture local distance proximities and to build a data-driven model of the space.

t-Distributed Stochastic Neighbor Embedding t-SNE is a non-linear machine learning algorithm for dimension reduction. It was developed by Geoffrey Hinton and Laurens van der Maaten [24]. It is well suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. The t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects have a high probability of being picked, whilst dissimilar points have an extremely small probability of being picked. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence (a measure of how one probability distribution diverges from a second expected probability distribution [12]) between the two distributions with respect to the locations of the points in the map.

Non-metric problems use non-metric MDS (Multidimensional Scaling). The development of non-metric MDS was motivated by two main weaknesses in the metric MDS (Fahrmeir and Hamerle; 1984, Page 679):

1. the definition of an explicit functional connection between dissimilarities and distances in order to derive distances out of given dissimilarities, and
2. the restriction to Euclidean geometry in order to determine the object configurations.

Shepard-Kruskal algorithm A well-known method for non-metric MDS is the Shepard-Kruskal algorithm. A set of distances d_{ij} as a function of given dissimilarities δ_{ij} are calculated from the arbitrarily chosen initial configuration X_0. The second step – the non-metric phase – determines disparities $\hat{d}_{ij}^{(0)}$ from the distances $d_{ij}^{(0)}$ by constructing a monotone regression relationship between the $d_{ij}^{(0)}$'s and δ_{ij}'s. The third step has the spatial configuration of X_0 is altered to obtain X_1. From X_1 the new distances $d_{ij}^{(1)}$ can be obtained which are more closely related to the disparities $\hat{d}_{ij}^{(0)}$ from step two.

When datasets become very large and complex, traditional linear or non-linear projection methods become inefficient and expensive to compute. Control points based projections utilize a two-step approach, first projects control points. Based on the control points locations and local feature preservations, the rest of the points are projected. Such a method is much more scalable, and it opens up for the user to alter the outcome of the projection by modifying the properties of the control point's projection.

For a dimension reduction algorithm, a suitable distance metric is key for the computation outcome.

Dynamic Projections A different approach from common static projections is that of dynamic projections; that is, moving or animated projections. These can be metaphorically described as moving film cameras in high-dimensional Euclidean spaces that return movies consisting of dimensionally reduced views [27]. Dynamic projections are, basically, curves of projections. They exploit the human eye's natural ability to detect and recognize objects in motion.

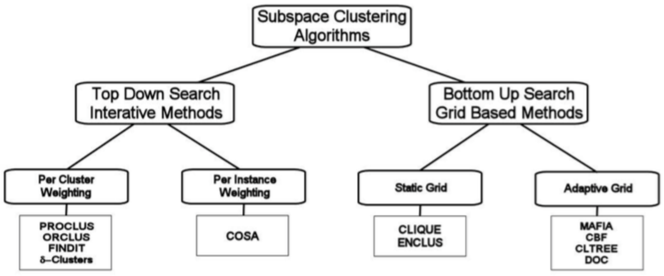
Subspace Clustering Methods

A different approach to transform high-dimensional data is subspace clustering. Where dimension reduction aims to compute one embedding to best describe the data, subspace clustering can output multiple embeddings of the data, either through clustering dimensions or data points. Subspace clustering is an extension of traditional clustering that seeks to find clusters in different subspaces within a dataset. Often in high-dimensional data, many dimensions are irrelevant and can mask existing clusters in noisy data. Feature selection removes irrelevant and redundant dimensions by analyzing the entire dataset. Subspace clustering algorithms localize the search for relevant dimensions allowing them to find clusters that exist in multiple, possibly overlapping subspaces.

A reason, stated by Parsons et al., that many clustering algorithms struggle with high dimensional data is the curse of dimensionality. As the number of dimensions in a dataset increases, distance measures become increasingly meaningless. Additional dimensions spread out the points until – in very high dimensions – they are almost equidistant from each other.

There are two major branches of subspace clustering based on their search strategy. Top-down algorithms find an initial clustering in the full set of dimensions and evaluate the subspaces of each cluster, iteratively improving the results. Bottom-up approaches find dense regions in low dimensional spaces and combine them to form clusters. Figure 2.4 by Parsons et al., presents a heirarchistic view of some methods.

Figure 2.4 Hierarchy of Subspace Clustering methods. Source: Parsons et al. [46]


Subspace Clustering Algorithms
Top Down Search Interative Methods
Bottom Up Search Grid Based Methods
Per Cluster Weighting
Per Instance Weighting
Static Grid
Adaptive Grid
PROCLUS ORCLUS FINDIT δ-Clusters
COSA
CLIQUE ENCLUS
MAFIA CBF CLTREE DOC

The bottom-up methods determine locality by creating bins for each dimension and using those bins to form a multidimensional grid. There are two main approaches to accomplishing this. The first group consists of CLIQUE and ENCLUS which both use a static sized grid to divide each dimension into bins. The second group contains MAFIA, Cell Based Clustering (CBF), CLTree, and DOC. These methods distinguish themselves by using data driven strategies to determine the cut-points for the bins on each dimension. MAFIA and CBF use histograms to analyze the density of data in each dimension. CLTree uses a decision tree based strategy to find the best cutpoint. DOC uses a maximum width and minimum number of instances per cluster to guide a random search.

The top-down methods start by finding an initial approximation of the clusters in the full feature space with equally weighted dimensions. COSA is unique in its approach, and uses k-nearest neighbors for each instance in the dataset to determine the weight for each instance. PROCLUS, ORCLUS, FINDIT, and δ-Clusters determine the weights of instances for each cluster.

Dimension Space Exploration A way of letting the user explore the different data embeddings in subspace clustering, is dimension space exploration. The user can interactively group relevant dimensions into subsets. Such explorations allows for better understanding the relationships of the subsets and to identify shared pattern among the dimensions.

Clustering Subsets of Dimensions When clustering subspaces of dimensions, related dimensions are automatically grouped into clusters. Interferences introduced by irrelevant dimensions are filtered out, allowing lower dimensional structures to be discovered. The methods originate from data mining community, and introduce some interesting exploration strategies for high-dimensional datasets. Methods prove to be particularly effective for data in which the dimensions are not tightly coupled. An interesting system is TripAdvisor, which employs a sight-seeing metaphor for high-dimensional space navigation and exploration. Subspace clustering is utilized to identify “sights” for the exploration.

Non-Axis-Aligned Subspaces A method which utilize that points are grouped for sharing similar linear subspaces, is called non-axis-aligned subspaces. This can be complex for very high-dimensional data, and is thus often combined with random projections. Instead of clustering the dimensions, the points are grouped together for sharing similar linear subspaces.

Regression Analysis and Topological Data Analysis

Regression analysis is a statistical procedure for estimating relationships among variables. It can be used in visualization to optimize and steer the design. In addition, regression has been used to summarize the structure of data via skeleton representations.

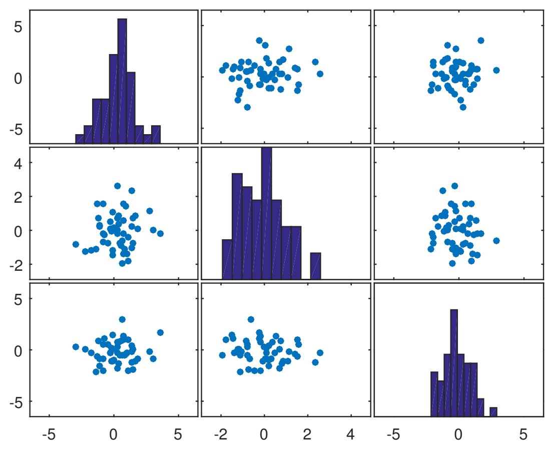
Topological data Analysis is also used. It is a relatively new field of study, and is used for feature abstraction, extraction and evaluation. This helps gain insight in large, complex high-dimensional data.

2.2.2 Visual Mapping of Data

The next step is visual mapping; the process of converting the (manipulated) data into visual structures. This is where the previously discussed methods like scatter plot matrices and parallel coordinates come into action. Both of these are so-called axis-based methods.

According to Firendly [41], scatter plots were not really introduced until scientist had need to examine bivariate data relationships between distinct variables directly. Herschel (1833) is credited the early introduction, but the term itself did not arrive until the early 1900’s. Scatter plots can be two- or three-dimensional, but the traditional way of presenting high-dimensional data is by using a scatter plot matrix.

Figure 2.5 Scatter Plot Matrix



A scatter plot matrix is a display of all pairwise scatter plots arranged in a dxd matrix for d variables. However, combined with dimension reduction, 3D scatterplots can be a nice way to display data. Three-dimensional scatter plots with computer systems allow direct rotation of a 3D view of high-dimensional data to any 2D orientation.

One popular modern method stems back from 1972 paper of D. F. Andrews; the Andrews plot. He proposed a method of plotting data, by mapping each data point $\mathbf{x} = (x_1, \ldots, x_k)$, onto a finite Fourier Series function, on the form

$$f_{\mathbf{x}}(t) = {}^{x_1}/\sqrt{2} + x_2 \sin t + x_3 \cos t + x_4 \sin 2t + x_5 \cos 2t + \cdots \tag{2.1}$$

where the function is plotted for $-\pi < t < \pi$. The resulting plots is good for detecting patterns and anomalies. Several common computer tools support Andrews Plotting, including Matlab[1]. The Andrews plot can be considered a smoothened version of parallel coordinates.

[1]Note that in Matlab, the data is plotted from 0 to 1, as seen in figure 2.6

Figure 2.6 Andrews Plot

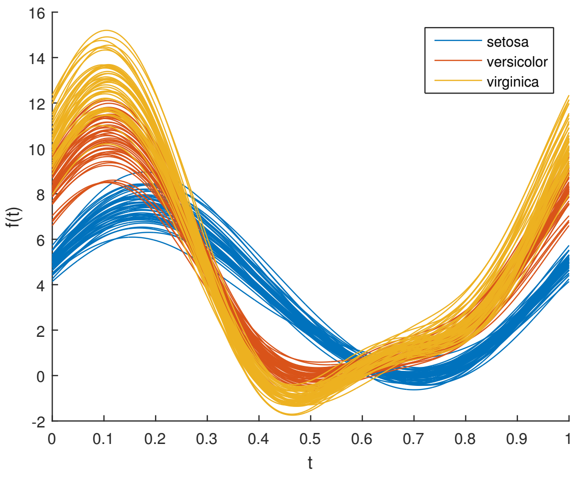
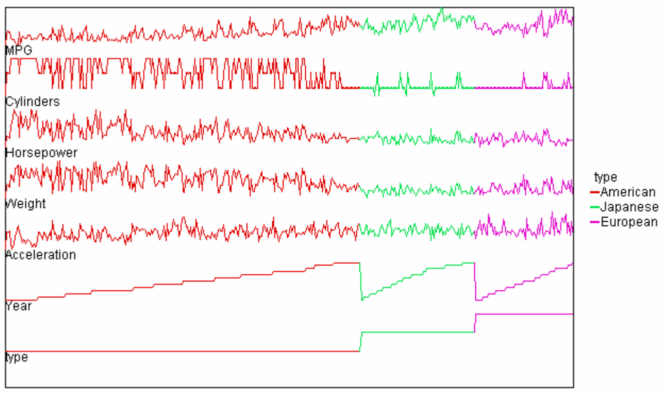


Figure 2.6 was generated by a Matlab example dataset, and visualize *The Iris flower data set* or *Fisher's Iris data set*; a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* [14].

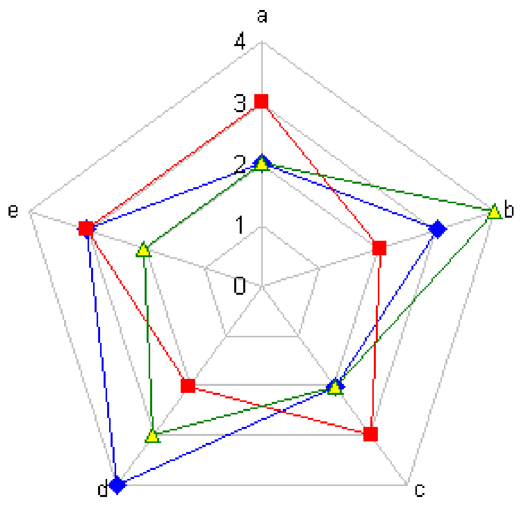
Multiple line graph is a line graph setup, in which multiple lines are displayed in the same window, to allow for several features to be visualized. Typically, the dimensions or features, are distinguished using different line-colours or other visual effect (dotted lines, solid lines, etc.)

Figure 2.7 Multiple Line Graph, Source: Grinstein et al., [6]



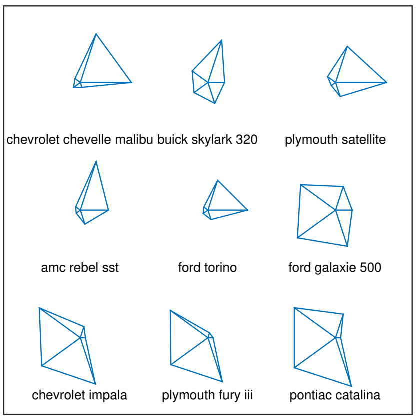
A radar chart places all features symmetrically around a circle. Each data point has its value marked for each dimension, and then lines are drawn between adjacent features.

Figure 2.8 Radar Chart



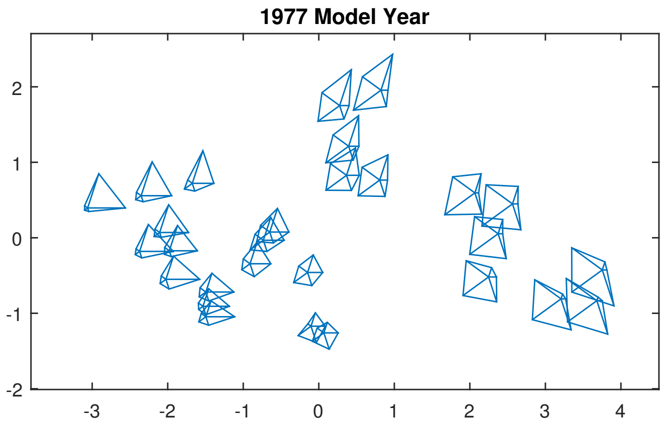
Glyphs were introduced first in 1973 Chernoff, and the objective is to map a high-dimensional point into one single glyph figure. It has been helpful in presenting trends in data.

Figure 2.9 Glyph Plot



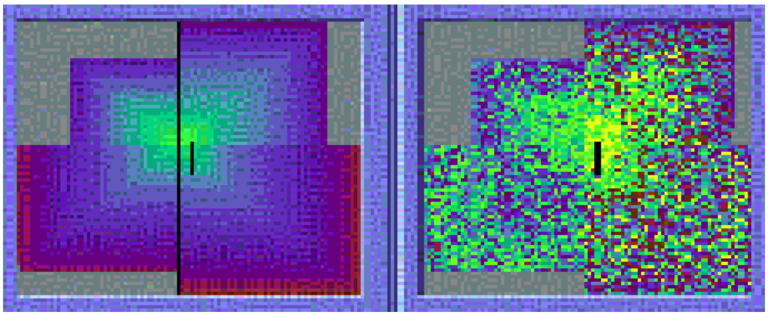
When a glyph plot is combined with a scatter plot, it is easy to find patterns among the entries.

Figure 2.10 Glyphs in Scatter Plot



One effort to encode a maximal amount of information is to encode data values as individual pixels and create separate displays, or subwindows, for each dimension. Relevant data can be placed in center, then spiraling outwards when less relevant (VisDB94). The goal (of VisDB) is to show similarities between the attributes of the data.

Figure 2.11 VisDB Figure, Source: Grinstein et al., [6]



To capture dimensional relationships, hierarchy based approaches are used. Dimension hierarchies explicitly reveal the dimension relationships, and helps in making the complexity of the dataset less costly. When exploring data set through low dimensional projections in a hierarchical way, one first constructs a top-level plot and then focus the attention on a local region of interest by recursively building the corresponding subprojections. The user can choose the regions of interest interactively. The data is organized, hierarchically, in a tree.

Animation transitions to enhance the perception of point and structure correspondences among multiple relevant plots. GGobi is such a tool. It calculates a continuous linear projection transition between any pair of linear projections based on the principal angles between them. In TripAdvisor, the user explores the neighborhood of a subplane by tilting the projection plane using a polygonal touchpad interface.

2.2.3 View Transformation

Finally, view transformation is what decides what appeared on-screen. Or, in Bertini et al., it is described as the rendering process.

Some of the rendering techniques include spline-based edge bundling, opacity-based hint to convey cluster density, and shading effects to illustrate a local line density. Illustrative rendering are also used for highlighting the focused areas.

For most high-dimensional visualization techniques, a discrete visual representation is assumed since each element corresponds to a data point. However, many applications prefer a continuous representation.

When rendering semi-transparent objects, colour blending methods have a significant impact on perception of order and structure.

2.2.4 Levels of User Interactions

The amount of user interaction in different approaches is divided as follows by Liu et al. from little to a lot:

Computation-Centric Approaches: Require only limited user input (setting a few parameters etc.). Center around algorithms for well-defined computational problems. Most concentrated in the data transformation step.

Interactive Exploration: Navigate, query and filter existing model interactively for more effective visual communication.

Model Manipulation: User manipulation is part of the algorithm.

2.2.5 Future Topics of Interest

Liu et al. suggest that some fields will prove interesting for future exploration. *Subspace clustering*, *model manipulation*, *uncertainty quantification* and *topological analysis and visualization* are all mentioned as hot topics.

2.3 Object-Oriented Graphic Design

To describe the process of graphics creation according to (Wilkinson, 2005), an object-oriented graphics system will be considered. An object can be regarded as an actor in a system. Objects are simple and *stupid*; they do only a few things and do them well. The objects of a system, communicate by sending messages. A good object-oriented design enables encapsulation – one object does not need to know how another works, it needs only to know how to interpret messages. Object-orientation also allows for attribute inheritance and polymorphism.

The reason object-oriented design is of such interest, is basically that *graphics are objects* (Hurley and Oldford, 1991). Objects which can be seen. Using an object-oriented paradigm in development, thus makes sense in terms of good design.

In (Wilkinson, 2005), the distinction between a *graph* and *graphics* is made. A graph is merely a set of points, an abstraction that cannot be seen. A graphic, however, is the physical representation of a graph. The translation from a graph to a graphic, is where the object-oriented design comes in. The aim is to create a generalized system based on a few objects and rules.

When a user requests a graphic, the object-oriented system goes through stages of specification, assembly, and display.

2.3.1 Specification

Specification involves the translation of user actions into a formal language. Graphics follow strict rules. They cannot be altered in whatever way, without lying about the data presented. Consequently, the core of any graphics system must rest on specification.

2.3.2 Assembly

In order to portray a scene, its geometry, layout and aesthetics must be coordinated for it to be rendered accurately. This process is called the assembly of a scene; the graphics computer program assembles a graphical scene from a specification.

2.3.3 Display

The final stage of the rendering process is to display the scene. It can be a simple step of simply drawing the scene as specified, or it could be a dynamic scene, allowing for things like zooming and rotation.

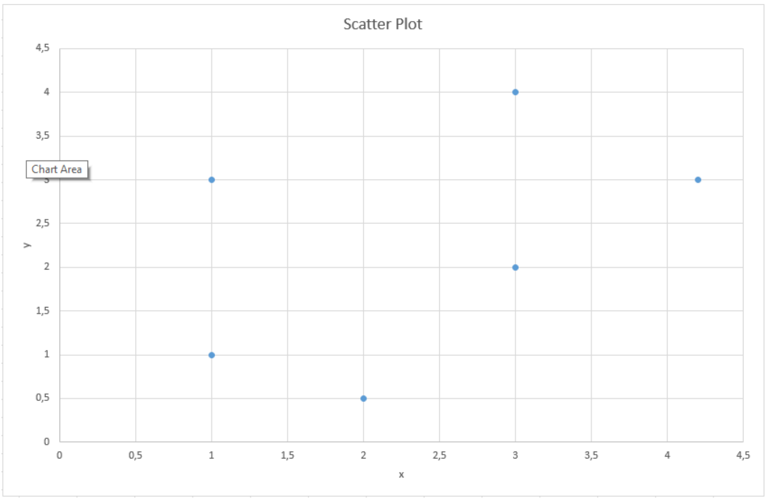
2.4 Graphic Rendering Computer Tools

This section looks at some common graphic rendering tools, and what features they provide.

2.4.1 Microsoft Excel

Microsoft Excel is a widely used mathematical spreadsheet, developed by Microsoft. Among its features are graphing tools, which consist of two-dimensional visualization methods, as well as a very limited three-dimensional graphical display. Excel supports charts, graphs, or histograms generated from specified groups of cells. The generated graphic component can either be embedded within the current sheet, or added as a separate object. These displays are dynamically updated if the content of cells change.

Figure 2.12 A simple scatter plot made with Microsft Excel



XLSTAT

XLSTAT is a statistical software for Excel. Through this package, high-dimensional visualization tools such as scatter plot matrix and parallel coordinate plot are realized.

2.4.2 Matlab

Matlab is a popular programming environment for mathematics and statistics, and contain several common ways to display high-dimensional data [13]. The traditional methods of scatter plot matrix, parallel coordinates, Andrews plot and glyphs are all available, well documented and easy to use. Figures 2.2, 2.5, 2.6, 2.9, and 2.10 were all created using Matlab.

2.4.3 Wolfram and Mathematica

Wolfram Mathematica has a large number of two- and three-dimensional visualization tools, and has recently added visualization of higher-dimensional data. This is limited, however, and is used to display a tesseract, the four dimensional analog to a cube.

2.4.4 gglpot2 - R package

ggplot2 is a package for the statistical programming language R. It is designed to accomplish tailored graphics for the relevant data. ggplot2 can render high dimensional data by letting the user for instance specify the different features, or dimensions, to affect a scatter plot differently. Two dimensions can be placed as traditional axis in the plot, one feature can determine point sizes, one colour of the point, one colour of the points border and more.

2.4.5 ggobi - Grand Tour

ggobi is a true high-dimensional visualization tool. It provides highly dynamic and interactive graphics such as tours, as well as familiar graphics such as the scatter plots, bar charts and parallel coordinate plots.

2.4.6 Graphics Engines

Graphics engines, like OpenGL and DirectX, can render two- or three-dimensional graphics of any sorts, specified by the programmer, and can thus be used for any visualization method. The obvious drawback is that the programmers must implement these methods themselves.

2.5 Existing Applications of High-Dimensional Visualization

In this section, a couple of interesting papers on high-dimensional data visualizations in certain fields of science are reviewed.

2.5.1 Biology

Yapeng Su, Qihui Shi, Wei Wei, Single cell proteomics in biomedicine: High-dimensional data acquisition, visualization, and analysis [16], presents methods for visualizing data, used in the field of biomedicine.

Recent research and new single cell omics[2] tools, provides high dimensional data of single cells. Thus, there is need for new tools for effective visualization. In this article, three categories of methods are mentioned; Clustering based analysis, dimension reduction algorithms and seriation-based analysis.

Clustering based algorithms are used to identifying biologically meaningful subsets. Methods - for example SPADE - clusters phenotypically similar cells together.

[2] neologism omics informally refers to a field of study in biology ending in -omics, such as genomics, proteomics or metabolomics [45]

Dimension reduction algorithms, such as PCA, help visualize the data, but do not explicitly identify and partition cells into subpopulations.

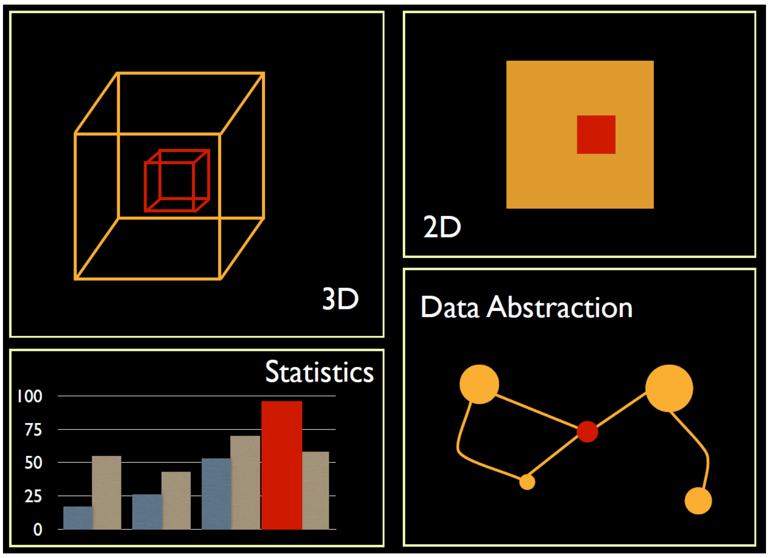
Seriation-based analysis, are used to study cellular transitions between cell states – a fundamental process in biology. The methods – including Wanderlust – address cell state progression with pseudo-temporal order.

2.5.2 Astronomy

Goodman, Alyssa, Principles of High-Dimensional Data Visualization in Astronomy [39], presents visualization in the field of astronomy.

In recent years, observing techniques has provided more data to explore, but the value of the “visual” to astronomers seems to have declined as a tool. This paper puts its focus on *linked view*, that is, multiple views of high-dimensional data. These systems update live as a researches selects, highlights or otherwise manipulates one of several views. The paper states that live linking views across display modes holds the key to effective visualization and analysis of high-dimensional datasets.

Figure 2.13 Linked view example. Figure created by M. Borkin, Source: [39]



The control of the views is done in real-time, with the possibility to save selections. There is put focus on making it easy for a researcher to use the system and learn from the data.

Chapter 3

Theory

3.1 High-Dimensional Data

High-dimensional data is, simply, a dataset with a large number of attributes or features [1]. In the case of visualization, high-dimensional data can be of any number of attributes above the traditional two or three. Technically, each datapoint can be viewed as an n-dimensional (row-)vector of numerals. Features could also be non-numerical values, but in the case of visualization it is common to map values onto numerals if that is the case. A set of points, or dataset, are arranged in a mathematical matrix. The number of columns in the matrix, is equal to the number of dimensions. Each row displays one entry of an n-dimensional point.

The number of features ensures that the data cannot be visualized traditionally in our three-dimensional reality. The visualization will ultimately end up on a two dimensional display surface, so even three-dimensional data must suffer somewhat and go through a view transformation to feel three-dimensional to the user. More on this in section 3.7.2.

3.1.1 The Curse of Dimensionality

The curse of dimensionality is a term used to describe several problems that concern data of high dimensionality, which do not affect two- or three-dimensional data. The expression was introduced by Richard E. Bellman [47].

The Euclidian distance in high-dimensional data, is not much different between different pairs of samples. The vastness of a high-dimensional Euclidean space can be illustrated by comparing the volumes of an inscribed hypersphere of radius

r and a hypercube of with edges of length $2r$. The volume of the sphere is $\frac{2r^d \pi^{d/2}}{d\Gamma(d/2)}$ (Γ is the gamma function, d is dimensionality), whilst the volume of the cube is $(2r)^d$. The ratio of these is $\frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)}$. As the dimensions d increase, the hypersphere becomes insignificant relative to the hypercube. This becomes clear when the dimension number d goes to infinity:

$$\lim_{d \to \infty} \frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)} \to 0$$

3.2 Scatter Plot

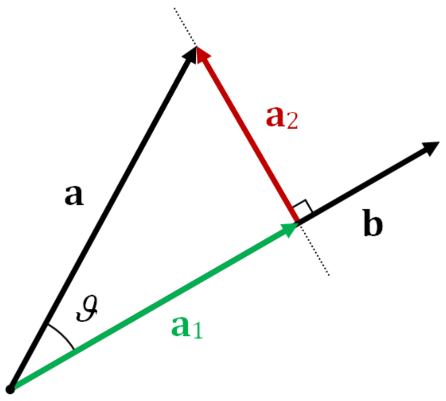
One of the most common ways to present data (two- or three-dimensional) is by using a scatter plot. As briefly presented in (2.2.2), a scatter plot is data points represented by placed them in a two- or three-dimensional Cartesian system; where each axis represent the value of a feature. Each data entry is represented as a point in the plane or the space, according to its feature values.

In order to represent high-dimensional data by using a single scatter plot – and not a scatter plot matrix – the dimensionality of the point has to be reduced.

3.3 Projection

To visualize high-dimensional data in a three-dimensional space, the datapoints must go through a process of dimension reduction; a projection. A projection is, in general, a mapping of a vector $\mathbf{a}$ onto the nonzero vector $\mathbf{b}$. The resulting vector $\mathbf{a}_1$ is a vector parallel to $\mathbf{b}$.

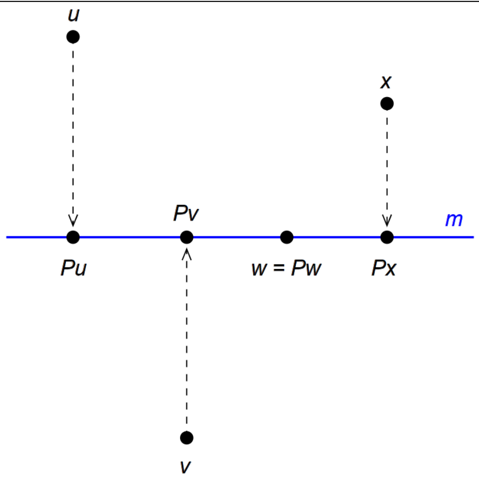
Figure 3.1 Simple Projection. Source: [49]



When a vector $\mathbf{a}$ is projected onto vector $\mathbf{b}$, the projection is the decomposed component $\mathbf{a}_1$, that is the result. The $\mathbf{a}_2$-component is called the rejection of $\mathbf{a}$ from $\mathbf{b}$.

So, a projection can be used to reduce dimensions, by projecting a point or vector in an n-dimensional Euclidean space, or Hilbert space, onto a lower dimensional space. Figure 3.2 is showing an orthogonal projection transformation from points in a two-dimensional plane onto a line.

Figure 3.2 Dimension reduction. Source: [50]



3.3.1 Orthogonal Projections

When the vector space W has an inner product and is complete (is a Hilbert space) the concept of orthogonality can be used. An orthogonal projection is a projection for which the range U and the null space V are orthogonal subspaces. Thus, for every x and y in W,$\langle Px, (y - Py)\rangle = \langle (x - Px), Py\rangle = 0$. Equivalently:

$$\langle x, Py\rangle = \langle Px, Py\rangle = \langle Px, y\rangle$$

A projection is orthogonal if and only if it is self-adjoint.

3.4 Rotation

A motion of a rigid body in a Euclidean space leaves the distance between any two points in the body unchanged after the transformation. Any direct Euclidean motion can be represented as a composition of a rotation about a fixed point (or axis or plane, depending on dimensionality) and a translation. In a rotation, all points have the same angular velocity, but the speed is proportional to the distance from the rotation point.

Rotations in three dimensions are generally not commutative, so the order in which rotations are applied is important even about the same point. Rotations about the origin have three degrees of freedom, the same as the number of dimensions.

A general rotation in four dimensions has only one fixed point, the center of rotation, and no axis of rotation. Instead the rotation has two mutually orthogonal planes of rotation. The rotation has two angles of rotation, one for each plane of rotation, through which points in the planes rotate.

3.4.1 Rotation Matrices

A rotation matrix is a matrix, which is used to mathematically perform a rotation in Euclidean space. The matrix is an orthogonal and quadratic matrix, with dimensions n by n in an n-dimensional space.

An n-dimensional rotation matrix $\mathbf{R}$ can be described by:

$$(\mathbf{I}+\mathbf{R})^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & & 0 & 0 & 0 \\ 0 & 0 & 1 & & 0 & 0 & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & 0 & 0 & & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & a & b & \cdots & c & d & e \\ -a & 0 & f & \cdots & g & h & i \\ -b & -f & 0 & & j & k & l \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ -c & -g & -i & & 0 & m & n \\ -d & -h & -k & \cdots & -m & 0 & o \\ -e & -i & -l & \cdots & -n & -o & 0 \end{bmatrix}$$

, and similarly in any other number of dimensions.

3.5 Projection and Rotation of a Point About an Axis

A joint calculation procedure for rotating a vector and projecting the resulting manipulated vector can be described. The projection of point $\mathbf{P}$ rotated about axis x_n by α degrees is calculated as follows:
First $\mathbf{P}$ is normalized, denoted $\mathbf{P}'$.

$$\mathbf{P}' = \frac{\mathbf{P}}{||\mathbf{P}||} \tag{3.1}$$

In the next step, axis x_n is described by vector $\mathbf{B}$, whose values are 1 for n'th element (the relevant axis) and zero otherwise.

For instance, axis x_2 in a five-dimensional Euclidean space is described by: $\mathbf{B}_{x_2} = [0\ 1\ 0\ 0\ 0]$.

The inner product between $\mathbf{P}'$ and $\mathbf{B}$ is multiplied by $\mathbf{P}'$ and subtracted from $\mathbf{B}$ to make up $\tilde{\mathbf{B}}$

$$\tilde{\mathbf{B}} = \mathbf{B} - \langle \mathbf{P}', \mathbf{B} \rangle \mathbf{P}' \tag{3.2}$$

By normalizing $\tilde{\mathbf{B}}$, we get $\mathbf{B}'$

$$\mathbf{B}' = \frac{\tilde{\mathbf{B}}}{||\tilde{\mathbf{B}}||} \tag{3.3}$$

And finally, the projected dataset is found through the operation

$$Proj = ||\mathbf{P}||((\cos\alpha)\mathbf{P}' + (\sin\alpha)\mathbf{B}') \tag{3.4}$$

3.6 Java Programming Language

Java is a general-purpose, object-oriented programming language. It is developed and maintained by Sun Microsystems. Java applications are typically compiled to bytecode that is run on a Java virtual machine (JVM), on any operating system with a JVM on it.

A “Hello World” program can look like:

```java
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

The `main()`-method is the entry point of the code. Each class variable, instance variable, or array component is initialized with a default value when it is created, unless specified by the user. For example, the type float the default value is positive zero, i.e. 0.0f.

3.6.1 Java Virtual Machine

The Java virtual machine (JVM) is an abstract computing machine on which a Java program is run. The instruction set of the JVM is Java bytecode, which is converted into machine language and executed.

3.6.2 Java Runtime Environment and Java Development Kit

Java Runtime Environment (JRE) is a software package that contains waht is necessary to run a Java application, such as the JVM. The Java Development Kit (JDK) is a superset of the JRE. JDK also includes tools for Java developers, such as a *javac* compiler. The Both JRE and JDK are provided free of charge from `http://www.oracle.com/technetwork/java/javase/downloads/index.html`[1].

3.6.3 lwjgl - Lightweight Java Game Library

The Lightweight Java Game Library (lwjgl) is an open-source Java software library. Its main purpose is video game developing, and allows for development using OpenGL and access to controllers such as mouse and keyboard. The primary goal of the lwjgl-project is to provide a way for Java developers to get access to resources that are otherwise unavailable or poorly implemented on the existing Java platform. The main philosophy is to expose underlying technology as a thin wrapper, thus creating an API close to the original.

3.6.4 Eclipse (Java Neon)

Eclipse is an integrated development environment. It is the most widely used IDE for Java development. It was developed by the Eclipse Foundation and the Eclipse software development kit is free and open-source software. In Eclipse, the developer can write code, run the program, and print to an in-built console. In order to develop Java applications in Eclipse, the JDK must have been installed in advance.

3.7 OpenGL

This section (3.7) is heavily copied from the project report [44].

Open Graphics Library is an API for rendering computer graphics. It is both cross-language and cross-platform, and offers interaction with a GPU. This results in hardware-accelerated rendering of 2D and 3D vector graphics. The system is managed by the non-profit consortium Khronos Group, and it is well maintained by the OpenGL Architectural Review Board (ARB). Updates come periodically.

OpenGL is royalty-free for software developers. However, hardware vendors do need to have a license to create an OpenGL implementation for their hardware.

[1]Note that JRE is included in the download of JDK

This license gain access to ARB-approved OpenGL applications and source code.

No other graphics API can match OpenGL's range of hardware platforms and software environments. It runs on every major operating system, including mobile/embedded systems through OpenGL ES (OpenGL for Embedded Systems). OpenGL is callable from Ada, C, C++, Fortran, Java, Perl and Python.

OpenGL is very well documented. There exist numerous literature on OpenGL and a vast amount of example code online.

Several APIs work alongside OpenGL, such as OpenAL (Open Audio Library) to add sound to one's projects.

3.7.1 Graphics Rendering through OpenGL

The rendering of graphics is the process of producing a raster image, that is a chunk of pixels with a certain RGB-colour value, from a dataset representing the polygon mesh of a model which is to be displayed. The colour value of each pixel is written by the GPU to the frame buffer of the graphics memory. The display reads the frame buffer, a process known as a raster scan. The reading is done row-wise, left to right, top to bottom; and the result of the scan is called a frame.

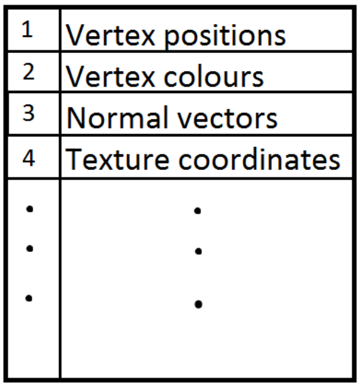
Vertices, Primitives and Meshes

A 3D model, or mesh, is made up of a collection of primitive geometric shapes, which again are defined by vertices. Each vertex is specified at a local position, and may also contain other information, such as colour information. A number of vertices are connected to form the primitives, which in OpenGL include single points (GL_POINTS), lines (GL_LINES) or closed polygons (GL_TRIANGLES). Triangles are the most common primitive used. Note that GL_QUADS, a quad primitive, was deprecated in OpenGL3.0, and later removed in 3.1.

Vertex Array Object

A vertex array object (VAO) is an object in which one store data about the vertices of a 3D mesh. The usage of VAOs is the orderly way of representing the vertex data. Figure 3.3 below is meant as an example illustration of a VAO.

Figure 3.3 Vertex Array Object example



1	Vertex positions
2	Vertex colours
3	Normal vectors
4	Texture coordinates
•	•
•	•
•	•

The VAO has slots in which different attribute lists can be stored. These lists of data are called vertex buffer objects (VBO). Typically, one VBO can contain vertex positions, another colours, normals or texture coordinates. A special attribute of the VAO, is the indices list. This is a list which specify the order of the vertices. This allows for vertices which are part of multiple primitives, only to be stored once in the VAO.

The VAO data is accessed through its ID, which is given upon creation. OpenGL can then use the information contained in the VAO to render models.

Shader Program - Communicating with the Graphics Processing Unit

In a game, processing is done by CPU, but rendering the scene is done by GPU. A GPU renders objects and determine how they are displayed. It may perform operations such as colouring, fogging, lighting, texturing and transformation. The reason these operations are outsourced to the GPU, is due to the highly parallel structure of the processor. Handling thousands of vertices, all of which are processed individually, is ideally performed on such a hardware component. The programmable rendering pipeline of OpenGL allows for parts of the rendering to be specified in code by the programmer. These OpenGL programs which can be run on the GPU are called shaders. OpenGL has its own programming language for this; GLSL.

GLSL - OpenGL Shading Language

GLSL is a C-like programming language developed by the ARB-group. A simple example of how a vertex shader, written in GLSL, may look is:

```
/******* Vertex shader *******/
#version 450 core

layout (location = 0) in vec3 position;
layout (location = 1) in vec3 colour;

uniform Mat4 modelMatrix;

out vec3 pass_colour;

void main(void) {
  gl_Position = modelMatrix * vec4(position.xyz, 1.0);
  pass_colour = colour;
}
```

As in C, the # precedes a pre-processing command – here specifying the version of GLSL – and main function defines the processor's entry point on execution. The **in** keyword is for input variables, and **out** for output. The **layout** qualifier specifies the storage of a VBO, here *location = 0* means VAO slot 0 and so on. In the example above, a simple vertex shader is implemented. The position of the vertex is transformed by a matrix and then passed to the built-in GLSL variable *gl_Position*, which is the the clip-space (3.7.1) output position of the current vertex. The colour of the vertex is passed to the fragment shader. GLSL shaders are not stand-alone applications; they require an application that uses the OpenGL API

Vertex Shader

The vertex shader executes once for each vertex in the VAO. It calculates the vertices positions on the screen. The positions may be altered by an object transformation or by camera movement. The vertex shader may also calculate some per vertex values which are passed to the fragment shader, such as vertex colours. Note that these values may be sent to a geometry shader instead, see the section on geometry shader.

Fragment Shader

The fragment shader is executed once per pixel. It uses information from the previous shader(s) and calculates the desired colour for each pixel on the screen. It uses linear interpolation of vertex colours, depending on vertices distances from pixel.

Geometry Shader

Optional. Execute once for each primitive. Inputs are arrays; one value per vertex. Have more information available than vertex shader, since we now get primitives, i.e. set of vertices.

Tessellation Control Shader and Tessellation Evaluation Shader

The tessellation control shader sits between the vertex shader and the tessellation evaluation shader. The shader controls how much tessellation is done. The main purpose is to feed these tessellation levels to the Tessellation primitive generator stage, as well as to feed patch data to the Tessellation Evaluation Shader stage.

The tessellation evaluation shader is a Shader program written in GLSL that takes the results of a Tessellation operation and computes the interpolated positions and other per-vertex data from them. These values are passed on to the next stage in the pipeline.

Uniform Variable

In GLSL, a uniform variable can communicate with a shader form the non-shader code. Without these operators, run-time alterations to shader components must be done before the shader-stages. That means, changing the VAO. Uniforms are very useful for transformation matrices so that calculations can be done by the shader, instead of changing the VAO data each frame. That way the GPU is utilized, instead of multiplying all vertices with the same matrix in CPU.

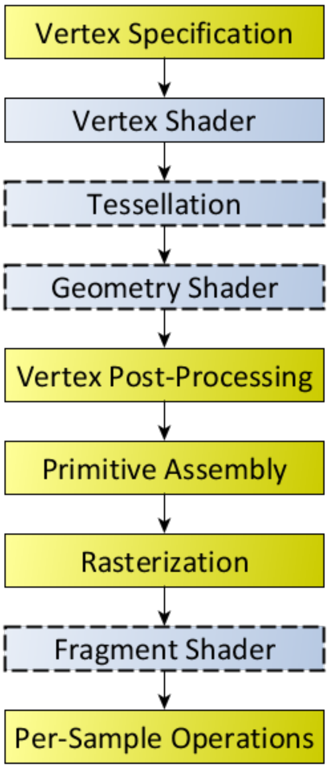
Sampler Object

A sampler object is used for textures in shader code. Instead of basing colours on vertex colour values, a texture image is sampled onto the object.

Rendering Pipeline

The rendering pipeline of OpenGL describes the sequential stages from model vertex data to the final frame. The attributes of each vertex in the model are in the first stage of the rendering pipeline. Each vertex is processed individually. Since a model may consist thousands of vertices, the per-vertex processing is highly parallelizable. Therefore, the operations are ideal for a many-core processor, as the GPU.

Figure 3.4 Rendering pipeline. The blue boxes are programmable shader stages.



Per-vertex operations are performed by the vertex shader (3.7.1), which specify vertex placement and outputs calculated per-vertex values

The tessellation stage is an optional stage, where patches of vertex data are subdivided into smaller primitives. With this, we achieve improved performance. Programmable parts include the tessellation control shader and the tessellation evaluation shader.

Optionally, the next step in the pipeline is the geometry shader, a per-primitive shader

Vertex post processing, vertices undergo a number of fixed-function processing steps, including clipping.

The next step is connecting the vertices to draw the corresponding primitives and create the mesh. At this step face culling is done, if requested (3.7.2).

Following is the rasterization step. This is the conversion of vector information (shapes/primitives) into a raster image. This includes the mapping from the OpenGL cube frustum into a 2D image, for displaying real-time 3D graphics. The primitives are broken down into fragments.

The fragment shader inputs are the fragment xyz-positions. It outputs pixel colours.

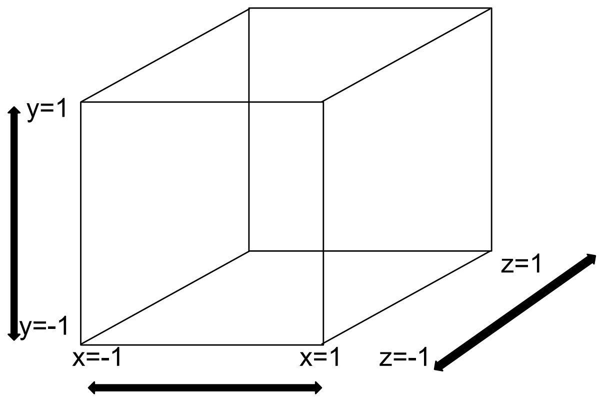
Before writing information to the frame buffer, per-sample operations are performed. This can be scissor test (cutting away certain fragment positions), or depth test (3.7.2).

Finally, the pixel data is written to the frame buffer.

The OpenGL Frustum

For a vertex - or any part of an object - to be shown on the display, the coordinates have to be located within the OpenGL cube frustum. The frustum is a 3D right hand system, with xyz all ranging from -1 to 1. Points outside of this frustum will be clipped.

Figure 3.5 The OpenGL cube frustum



When the frustum is mapped onto the 2D viewport, x is the horizontal component, whilst y is the vertical component. The z component of the object is not initially of any importance, other than that it has to be within the -1:1 boundaries to be rendered. Scaling and simulating the depth of the scene has to be simulated by manipulating the x and y components of the objects.

3.7.2 Spaces and Matrices

There are several different coordinate spaces to consider in graphics rendering. The transformation between spaces are done through matrix operations.

Local Mesh Space

Upon creation of a mesh, all vertices are specified according to a local mesh origin.

World Space

A mesh is put into a world space, through the use of a model matrix, or transformation matrix. The model matrix is composed of the following submatrices.

<u>Translation matrix</u>

$$\begin{bmatrix} 1 & 0 & 0 & translation.x \\ 0 & 1 & 0 & translation.y \\ 0 & 0 & 1 & translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

<u>Rotation about x-axis</u>

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

<u>Rotation about y-axis</u>

$$\begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

<u>Rotation about z-axis</u>

$$\begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale matrix

$$\begin{bmatrix} scale.x & 0 & 0 & 0 \\ 0 & scale.y & 0 & 0 \\ 0 & 0 & scale.z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying these yields the model matrix:

$$\begin{bmatrix} trnsfX.x & trnsfY.x & trnsfZ.x & trnsl.x \\ trnsfX.y & trnsfY.y & trnsfZ.y & trnsl.y \\ trnsfX.z & trnsfY.z & trnsfZ.z & trnsl.z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7.5}$$

The lwjgl-library has its own matrix operations for translation, rotation and scaling. The model matrix is created like this:

```
public static Matrix4f createModelMatrix(Vector3f translation, float rx,↩
     float ry, float rz, float scale) {
  // Create a new 4x4 matrix
  Matrix4f matrix = new Matrix4f();
  // The two final arguments of the Matrix4f functions denotes on which ↩
       matrix to apply function
  // and where to store the result.
  Matrix4f.translate(translation, matrix, matrix);
  Matrix4f.rotate((float)Math.toRadians(rx), new Vector3f(1,0,0), matrix↩
       , matrix);
  Matrix4f.rotate((float)Math.toRadians(ry), new Vector3f(0,1,0), matrix↩
       , matrix);
  Matrix4f.rotate((float)Math.toRadians(rz), new Vector3f(0,0,1), matrix↩
       , matrix);
  // Assuming uniform scaling
  Matrix4f.scale(new Vector3f(scale,scale,scale), matrix, matrix);
  return matrix;
}
```

Eye/Camera Space

The next step is to transform everything according to the camera position and view direction, so that when the camera moves, the objects move. This is done by applying a view matrix. Transforms every vertex opposite of camera movement. The view matrix appear similar to the model matrix, just negated.

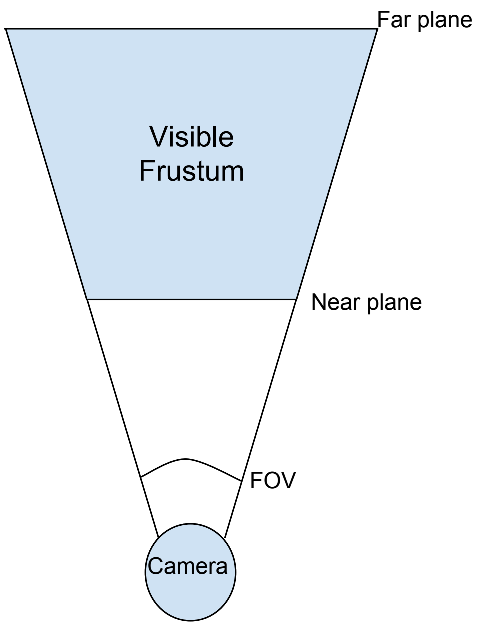
Perspective Projection Space

The Eye coordinates are truncated into a pyramid frustum, to simulate real vision, and mapped to the OpenGL cube frustum. The equation is as follows

$$\begin{bmatrix} \frac{\cot(\frac{FOVy}{2})}{aspectRatio} & 0 & 0 & 0 \\ 0 & \cot(\frac{FOVy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} & -\frac{2(Z_{far}Z_{near})}{Z_{far}-Z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{3.7.6}$$

where FOV (field of view), is often set to about 90 degrees in PC games. When the aspect ratio is above 1, FOVy = FOV. Z-near and Z-far describes at which distance the nearest and farthest object will be rendered, see Figure 3.6. Having a large Z-far, or small Z-near is not good, see section 3.7.2.

Figure 3.6 Camera projection



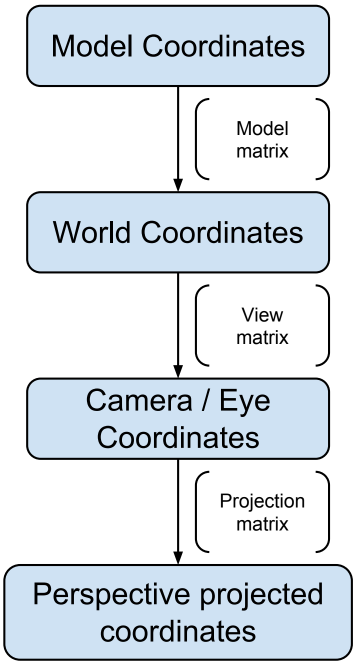
Viewport Space

The mapping of the OpenGL view frustum onto 2D screen raster image. Or, put another way, what is put on the display.

Transformation Pipeline

The transformation through the use of matrices, can be summarizes as follows:

Figure 3.7 Coordinate Transformation Pipeline

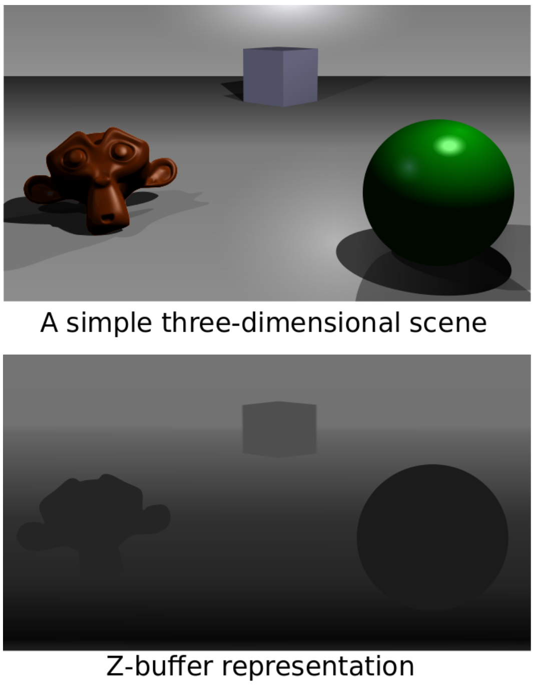


It is easy to go in the opposite direction, simply be doing inverse matrix calculations.

Z-buffering and Culling

Objects can be situated at the same display xy-positions, only at different depths. Thus, there is need for some techniques to determine what should be visible.

Figure 3.8 Z-buffer Example



The Z-buffer, or depth buffer, stores information about an object's depth. Figure 3.8 shows and representation of the Z-buffer, where darker colour is set for close objects. This can be used to determine which objects should be visible and which should be hidden. Without this, a phenomena called Z-fighting can appear; an object that is behind another object or terrain, and should thus be hidden, is visible. If the Z-far:Z-near ratio is set too high, the depth buffer might not be accurate enough, and Z-fighting might be a problem again.

Triangle primitives, after all transformation steps, have a particular facing. This is defined by the order of the three vertices that make up the triangle, as well as their apparent order on-screen. Triangles can be discarded based on their apparent facing, a process known as Face Culling. By default, OpenGL draws primitives in counter-clockwise order. So a primitive which, on screen, has been drawn apparently clockwise, it is the backside that is facing the viewer. To enable culling, the following OpenGL functions are called: `glEnable(GL_CULL_FACE)`; `glCullFace(GL_BACK)`. Face culling allows non-visible triangles of closed surfaces to be removed before expensive Rasterization and Fragment Shader operations.

Chapter 4

Specification and Implementation

For the purpose of implementing a high-dimensional data visualization application, it was chosen to use OpenGL for graphics rendering and build a system from scratch.

The visualization of the data points were realized using Java and Eclipse with OpenGL and the Lightweight Java Game Library. The datapoints are plotted in a three dimensional scatter plot, however the program enables the points to be rotated about all dimension axes.

4.1 Specification

The goal of the implementation was to display a projection of the data in a three-dimensional scatter plot. It should be possible for the user to rotate the data about any desired axis of the high-dimensional dataset.

To realize this wish, some calculation needed to occur, based on user interaction with the program. A setup was chosen, where the computer keyboard was to be used as rotation input. Selected keys affect certain angles of the dataset.

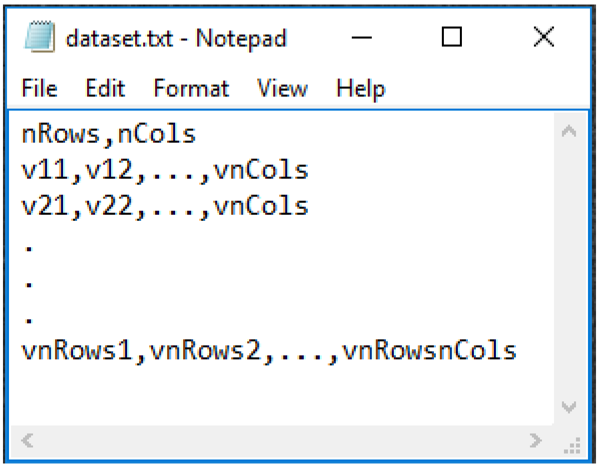
The user can also move through the space, using the keyboard's keypad, and explore the scene by rotating the camera with the mouse.

4.2 Design

4.2.1 Storing and Reading Data Points

The dataset is stored in a simple .txt-file. The first line of the file contains the number of rows and columns respectively – separated by a comma. The next lines each defines a row in the matrix, and each column value for that row is written in the respective line, separated by a comma. Hence, there is a readable matrix structure of the data point values. The numbers describing the dimensions of the matrix are integer values, whilst the data point values are floats. Below is a description of the setup:

Figure 4.1 Data points stored in file. nRows and nCols are integer values, vij are floating point values



```
nRows,nCols
v11,v12,...,vnCols
v21,v22,...,vnCols
.
.
.
vnRows1,vnRows2,...,vnRowsnCols
```

A simple file reader implemented in Java reads this data file, and the data is stored inside a matrix object. The specially implemented MNMatrix class, is designed to store M by N values. The values of the data points are put inside a Java ArrayList of floats. Class methods are specially designed for this system's requirements, and grants all necessary mathematical methods for the projection tasks.

The dimension-number of the dataset is used to create a transformation matrix, which is used to map the data into a three dimensional data set, that can be visualized in OpenGL.

4.2.2 Displaying Data Points

To display data points in a scatter plot, each point vector is mapped onto a three-dimensional space. OpenGL is designed to handle three dimensional data, so from here on the plotting is straight-forward.

Each row of the matrix dataset, after being manipulated as desired by the user, is put inside a VAO as a single point primitive; with one vertex position and colour. In the rendering process, the `GL_POINTS` primitive is used to easily display the data as desired.

4.2.3 Rotating the Data

The current design is capable of handling up to ten dimensions, but is made fairly scalable, so only small changes are needed to increase the number of dimensions that it can treat.

Following equation 3.4, any intention by the user to rotate an angle is handled accordingly. As the data is consisting several points, as opposed to one, the calculations of projections are altered to handle sets of vectors, that is, matrices, instead of one vector – or one data point – at a time. All of the matrices used are objects of the specially implemented MNMatrix class previously described.

Rotation about any axis is made possible by keyboard inputs from the user. Using a QWERTY-style keyboard, the specific angle of rotation, from about the first dimension to about the n'th, increase right-wise with the keys. lwjgl's keyboard API maps each specific key to an integer value, increasing by one in the QWERTY-style order from left to right. Thus, it is easy to transform key-pressing to axis-manipulation.

4.2.4 Axis Vectors

The axis vectors are created through one simple asset model, and rendered as a set of `GL_TRIANGLES` primitives, with a total number of seven vertex positions. The standard OpenGL x-axis, is set as default direction, and y-axis and z-axis vectors are rotated accordingly to create a fitting coordinate system.

The display of higher dimensional axes, are portrayed different from the first three. They are placed in the bottom left corner of the screen, by default pointing in x-direction. As their main objective is to help the user interpret the data in different angles, their placement is chosen to enhance the view of the changing vector mostly, thus, they all appear to be rotating about the z-axis of the OpenGL viewport, simply displaying the size of the angle of rotation.

4.3 Packages and Classes

4.3.1 Display Package

The display package contains one single class; the DisplayRenderer. Here, the display size, that is, width and height of the window, is set and the *frames per second* rate is specified. Furthermore, methods for updating and ultimately closing the display are implemented. The class is an interface for lwjgl-functionalities regarding display.

4.3.2 Models Package

The models package is consisting of the Model class. Here, any model – that is, object to be rendered – has its vertex data stored. The model object keeps track of the vertex array buffer ID and the number of vertices for a specific object.

4.3.3 Renderer Package

The renderer package handles the main task of using OpenGL to render graphics.

Loader Class

The Loader class handles loading model information into vertex buffer objects and into a vertex array object and contains the method in which Model objects are created. The Loader object keeps track of all VAOs and VBOs and has methods for deleting them on program termination. When an asset is created the Loader class method *loadToVAO()* is always used to store its data into a VAO, the method returns the model, which is created simultaneously. The methods starts by creating a VAO through the use of built-in OpenGL methods. Next, vertex and colour information, as well as indices, are stored as VBOs into the current working VAO. Finally, the VAO is unbound and the model returned.

MainRenderer Class

As different assets are to be rendered, or drawn, in different ways, the MainRenderer class works as a control class for rendering, to set the preliminary conditions and further push the specific renderers into action. It does no actual rendering on its own. It creates instances of the other renderers and the shader, and enables some basic OpenGL functionalities. The main renderer's *render()*-method is called once per frame. This method starts the shader program, calls the different rendering methods (who render the assets), and then stops the shader program afterwards.

PointRenderer Class

The PointRenderer, renders each point as a `GL_POINTS` element.

AxisVectorRenderer Class

The AxisVectorRenderer, renders the vectors as `GL_TRIANGLES` elements, in contrast to the points.

GridRenderer Class

The GridRenderer class draws a grid in the scene as a collection of `GL_LINES`↩ -elements. The grid consists of a number of squares, each drawn by twelve lines.

4.3.4 ShaderPrograms Package

The package of shaderPrograms consist of the general "super"-shader and the shader itself.

SuperShader Class

The superclass SuperShader handles the general creation of a shader program in OpenGL. It allows several special shader-classes to be implemented.

The class implements the shader file reader, and a method for creating vertex and fragment shaders and attaching them to the main shader program of OpenGL. The class also implements methods for deleting shaders.

Shader Class

The main task of the Shader class is to specify uniform variable locations, and loading them from the Java-code specifications to a location accessible from the shaders.

Vertex Shader

The vertex shader has two inputs; vertex position and vertex colour. Three uniform variables are associated; projection matrix, model matrix and view matrix, as specified in section 3.7.2. The shader places the vertices according to view projection and camera movement, and pass the colour to the fragment shader.

Fragment Shader

The fragment shader simply colourizes the fragments as specified by its only input variable; the colour passed from the vertex shader.

4.3.5 Assets Package

The assets package contains class definitions for the different asset elements that are displayed in the program, as well as the camera class.

Point Class

The Point class is a simple class keeping track of a point's position and its model. The position is given from the dataset after being transformed into three-dimensional, renderable coordinates.

AxisVector Class

The AxisVector class defines a vector pointing along a specific axis. The mesh model is by definition an arrow vector pointing in the x-direction. To keep track of which dimension the vector is representing, there is a MNMatrix object composed as a column vector, specifying the wanted direction.

ViewControl Class

The ViewControl class handles camera movement and also datapoint rotations, that is, all user interaction with the program via the keyboard. Once per frame, the class method *checkInputs()* detects if a key is pressed, and handles it according to specification.

Grid Class

The Grid class describes one square in a grid. It is defined by its eight corners, and lines drawn between them in the correct manner.

4.3.6 Utilities Package

The utilities package consists of two separate classes. The Utils class, contains helper methods for object transformations, whilst the MNMatrix class defines a general MxN-matrix which is used throughout the program for handling the datapoints.

Utils Class

The three functions of the Utils class, are performing the matrix transformations from local mesh space to screen coordinates, according to section 3.7.2. The *Projection Matrix* projects three dimensions onto the two dimensional display. The *Model Matrix* describes an assets position, rotation and scaling, related to the standard asset vertex positions. The *View Matrix* handles movement of the camera, and how that changes the position, rotation and scale of an asset. These matrices must not be confused with the matrix operations used to manipulate the high dimensional dataset; these matrix operations are part of the rendering of the already manipulated data.

MNMatrix Class

The MNMatrix class is designed to be a data type for holding matrix values for a matrix of arbitrary dimensionality, that is, a M by N matrix. The dimensions are stored in integer placeholders, whilst the data itself is stored as floating point values in a Java `ArrayList<Float>` container. Simple getters and setters exist for data values, so they can be easily manipulated or collected using Cartesian style coordinates.

Several functions are created to realize the necessary mathematical operations required to fulfill the transformation from high dimensional data to three-dimensional data. The functions are tailored for the purpose of this program.

4.3.7 Datasets Package

The datasets package consists of the file reader, the projection methodology and the dataset itself.

DataFileReader Class

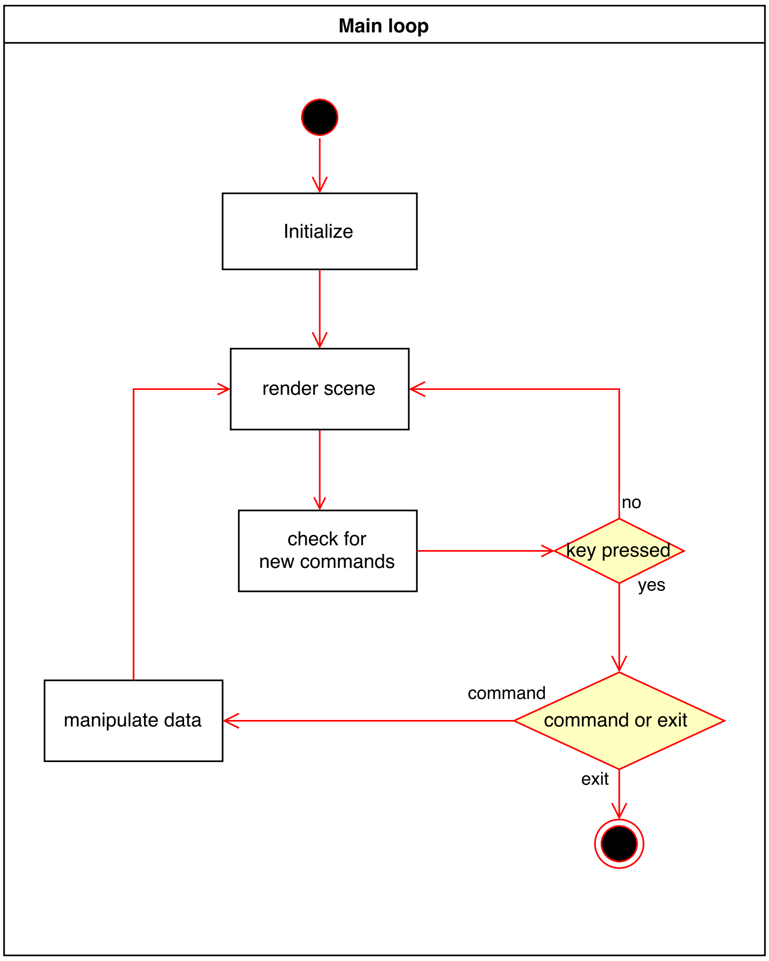
The DataFileReader class, reads the datapoint values from a file, and creates a matrix of it accordingly. The data is used to create a Dataset object.

Dataset Class

The Dataset class serves as the brain of the program; containing the data matrix and creating axis vectors and relevant transformation- and helper matrices to fit the dataset. The class's *udpate()*-function is called whenever keyboard action is registered, and therefrom transform the data as requested by the user.

4.4 Program Runtime Execution Pipeline

Figure 4.2 Flow Chart of the Program Execution



4.4.1 Startup

At startup, the display is initialized. The loader, view control, and renderer, are all made. The file containing the data is read, followed by storing its M datapoints of N dimensions in a MNMmatrix.

$$\mathbf{Dataset} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & & & \\ x_{M1} & x_{M2} & \cdots & x_{MN} \end{bmatrix}$$

The data is made to create a Dataset object. The matrix is stored, so that it can be used in further transformations. A transformation matrix is initialized as

$$\mathbf{Transf} = \begin{bmatrix} 1 & 0 & 0 & 0 & & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & & 0 \end{bmatrix}$$

efficiently project the points onto the three-dimensional xyz-space, through the calculation

$$\mathbf{Result} = \mathbf{Transf} \times \mathbf{Dataset}^T$$

, where **Result** is a 3 by M matrix, where each column represent a three dimensional point. The three-dimensional points are by initialization identical to the three first coordinates of the high dimensional point. The columns are made into Point objects, which are ready to be rendered by OpenGL.

Rendering

The rendering process starts when a Point object is created. Then, all point information is loaded into a Vertex Array Object. This information consists of vertex positions, colours and indices. Since a point only consist of one vertex, the indices array has size one. Still, the use of an index array is kept to keep the loader and mesh model general. The dataset of points is passed as an argument to the main renderer, and from there, the dataset's getPoints()-method pass the ArrayList of Point-objects to the PointRenderer. The points are then rendered as `GL_POINTS`.

4.4.2 Main Loop

In the main loop, the scene is rendered in the frequency specified at 60 frames per second. At each round, the keyboard input is checked. If a key is pressed, the event is handled according to specification.

Data Manipulation

To manipulate the data, the formulations presented in 3.5 are applied. The functions are designed to work as desired on matrices in the MNMatrix class definition. First, the points are normalized, that is, each row of the data matrix is normalized, according to $\mathbf{P}' = \frac{\mathbf{P}}{||\mathbf{P}||}$

Algorithm 1 For each row in matrix “this”, normalize elemets

```
1:  function NORMALIZEROWS($this$)
2:      $resultMatrix \leftarrow MNMatrix(this.rows, this.cols)$
3:      for all rows do
4:          sum ←0
5:          for all elements in row do
6:              $sum \leftarrow sum + this[i][j]^2$
7:          end for
8:          $sum \leftarrow sqrt(sum)$
9:          for all elements in row do
10:             $resultMatrix[i][j] \leftarrow this[i][j]/sum$
11:         end for
12:     end for
13:     return $resultMatrix$
14: end function
```

After this the next step is $\tilde{\mathbf{B}} = \mathbf{B} - \langle \mathbf{P}', \mathbf{B} \rangle \mathbf{P}'$. The sequence starts by finding the inner product $\langle \mathbf{P}', \mathbf{B} \rangle$:

Algorithm 2 Dot row by row of two matrices

```
function DOTROWS($this, rhs$)
    $resultMatrix \leftarrow MNMatrix(this.rows, 1)$
    if this.cols != rhs.cols then
        return null;
    end if
    for all rows do
        $sum \leftarrow 0$
        for all cols do
            $sum \leftarrow sum + this[r][c] * rhs[r][c]$
        end for
        $resultMatrix[r][0] \leftarrow sum$
    end for
    return $resultMatrix$
end function
```

Next, that inner product is multiplied to fulfill $\langle \mathbf{P}', \mathbf{B} \rangle \mathbf{P}'$:

Algorithm 3 Each element of column vector lhs defines a scalar value which each element in the corresponding row in rhs-matrix is to be multiplied by

```
function SCALARMULTIPLYVECTOR($lhs, this$)
    if $lhs.cols! = 1$ then
        return null;
    end if
    $resultMatrix \leftarrow MNMatrix(this.rows, this.cols)$
    for all rows in this do
        for all cols in this do
            $resultMatrix[r][c] \leftarrow this[r][c] * lhs[r][0]$
        end for
    end for
    return $resultMatrix$
end function
```

This step in the process is completed by subtracting the result from above steps from $\mathbf{B}$. The result is $\tilde{\mathbf{B}}$ which can be normalized to get $\mathbf{B}'$. The final step of the procedure is to calculate $||\mathbf{P}||((\cos\alpha)\mathbf{P}' + (\sin\alpha)\mathbf{B}')$. Getting $||\mathbf{P}||$ can be done by:

Algorithm 4 Calculate the norm of all row-vectors in the matrix

```
 1: function GETNORMS($this$)
 2:     $resultMatrix \leftarrow MNMatrix(this.rows, 1)$
 3:     for all rows in this do
 4:         $sum \leftarrow 0$
 5:         for all cols in this do
 6:             $sum \leftarrow sum + this[r][c]^2$
 7:         end for
 8:         $resultMatrix[r][0] \leftarrow sqrt(sum)$
 9:     end for
10:     return $resultMatrix$
11: end function
```

Multiplying matrices by cosine and sine is done by:

Algorithm 5 Multiply all matrix elements by a scalar

```
1: function SCALARMULTIPLY($this, value$)
2:     $resultMatrix \leftarrow MNMatrix(this.rows, this.cols)$
3:     for all rows do
4:         for all elements in row do
5:             $resultMatrix[r][c] \leftarrow this[r][c] * value$
6:         end for
7:     end for
8:     return $resultMatrix$
9: end function
```

These formulations combined make up the whole process:

Algorithm 6 The complete projection

```
1:  function PROJECT($P, angles, key$)
2:      $currentAxis \leftarrow key - 16$
3:      $axisMatrix \leftarrow MNMatrix(P.rows, angles.length)$
4:      $currentWorkingAngle = angles[currentAxis]$
5:      for all $rows$ in $P$ do
6:          $axisMatrix[r][currentAxis] \leftarrow 1$
7:      end for
8:      $normalizedMatrix \leftarrow normalizeRows(P)$
9:
10:     $tildeB \leftarrow dotRows(normaliezdMatrix, axisMatrix)$
11:     $tildeB \leftarrow scalarMultiplyVector(tildeB, normalizedMatrix)$
12:     $tildeB \leftarrow axisMatrix - tildeB$
13:     $markedB \leftarrow normalizeRows(tildeB)$
14:
15:     $cosine \leftarrow scalarMultiply(normalizedMatrix,$
16:         $\cos(currentWorkingAngle))$
17:     $sine \leftarrow scalarMultiply(markedB, \sin(currentWorkingAngle))$
18:     $projection \leftarrow cosine + sine$
19:     $projection \leftarrow scalarMultiplyVector(projection, getNorms(P))$
20:     $projection \leftarrow transpose(projection)$
21:     return $projection$
22: end function
```

The key pressed by the user, is passed as an argument to the update-method, which means there is knowledge of which angle is updated. After the projection process is done, the scene can be rendered.

4.4.3 Termination

Termination of the running program is prompt by closing the display window. Upon termination, shaders, and vertex array objects and their vertex buffer objects must be deleted.

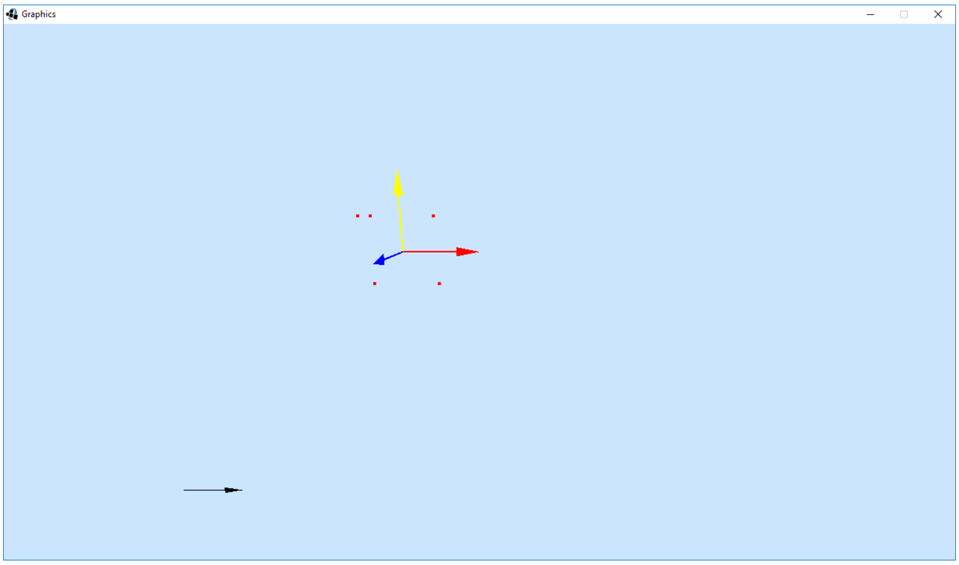
Chapter 5

Result

5.1 The Resulting Implementation

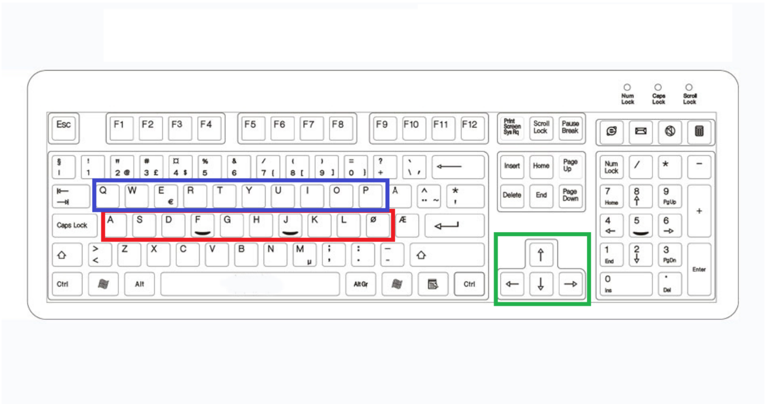
Figure 5.1 shows how a rendered scene may appear.

Figure 5.1 The projected data displayed. Here, the camera has been moved slightly to the left and rotated to unveil all axes.



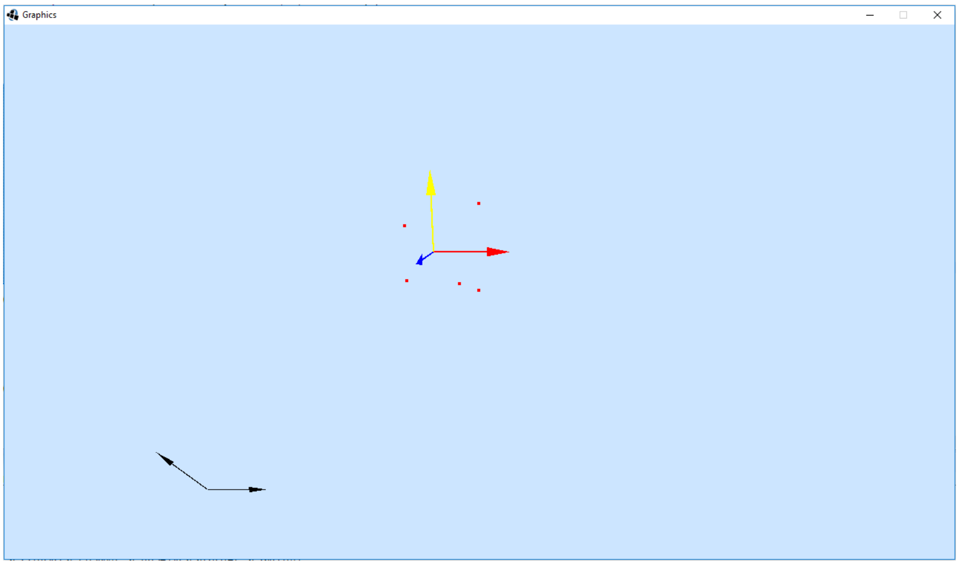
In this figure, five 10-dimensional points are portrayed. Using keyboard and mouse, the user can move around in this space to explore the data, which has been projected onto a three dimensional space.

Figure 5.2 Keyboard controls layout



To rotate the data, the keyboard is used. Figure 5.2 illustrates the controls of the program. The blue rectangle encapsulates the keys used to make a positive rotation, whilst the red rectangle represent negative rotation. In both key sets, the dimension of rotation increases from left to right. The keys in the green field control camera translation. Figure 5.3 is an example of the same data as in figure 5.1 having been rotated.

Figure 5.3 The data from figure 5.1 is rotated by approximately 135 degrees about the 4th axis

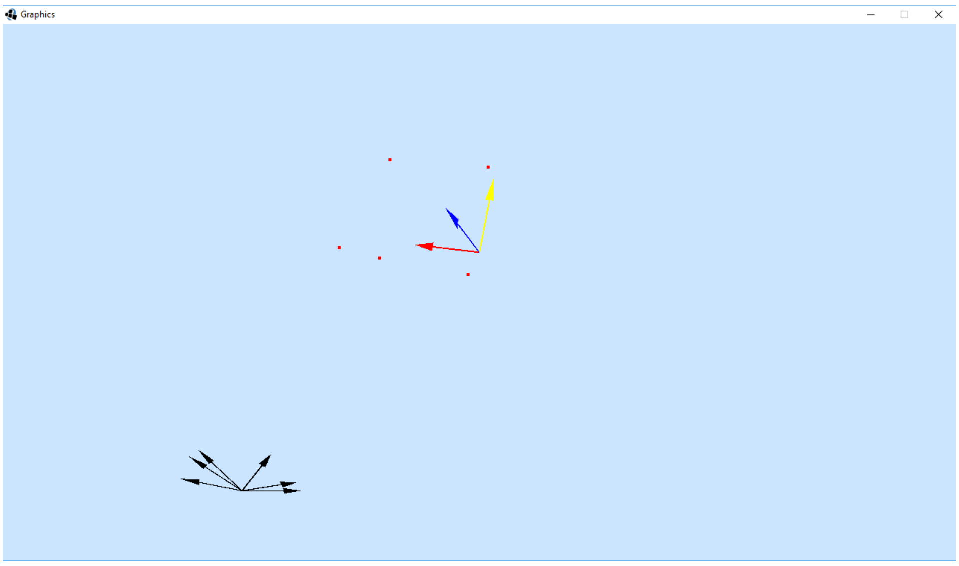


The black arrow in the bottom left corner indicates the magnitude of the rotation having been made on the relevant axis.

The different rotations work separatley on the dataset. That is, the placement of the points is only dependent on the current working angle. When pressing a new key, different from one used before, the points reset their position and only regard the new angle values. This is done due to rendering problems when keeping all angle affections in the data at once. When returning to a previously manipulated angle, the motion from that session is still kept.

Figure 5.4 shows a scene in which several angles have been manipulated. The data, though, is displayed as if only one angle has been changed from its initial value.

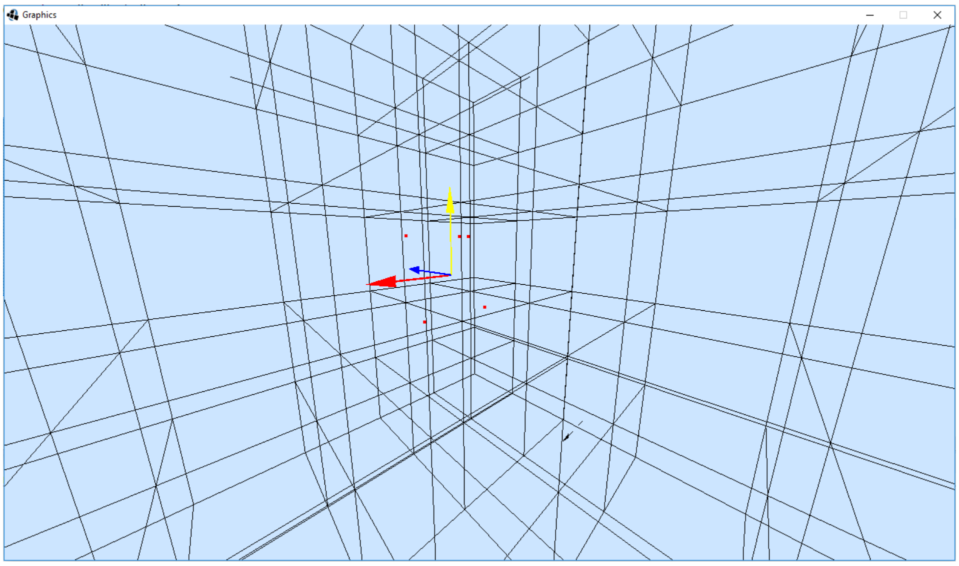
Figure 5.4 Several rotation commands having been applied to the dataset



5.2 Grid Rendering

A three-dimensional grid was implemented.

Figure 5.5 A grid to display different values. Each square has the size 1x1x1.



The aim of the grid was to help make the point positions more relatable in space. This did not work out satisfactory, so the grid is not kept in the default execution of the implementation.

5.3 Scalability of the System

The system is made fairly scalable. Due to the layout of the keyboard, a maximum of ten dimensions is implemented. However, features for extra dimensions could easily be added.

In terms of how many data points the system can handle at once, the performance is great for some thousands of data points. Figures 5.6 and 5.7 shows the rendering of 1500 points and 10,000 points, respectively.

Figure 5.6 Dataset of 1,500 random ten-dimensional datapoints

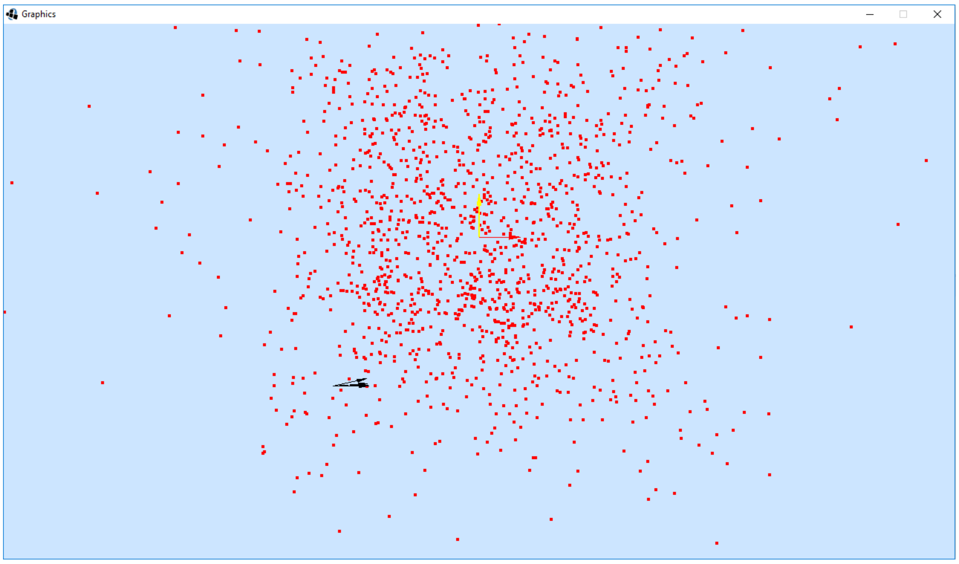
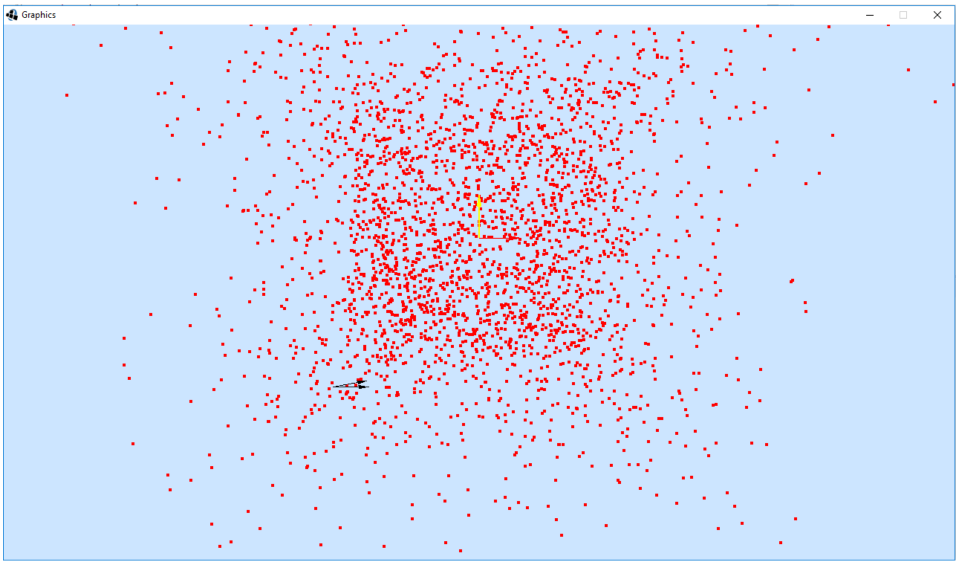
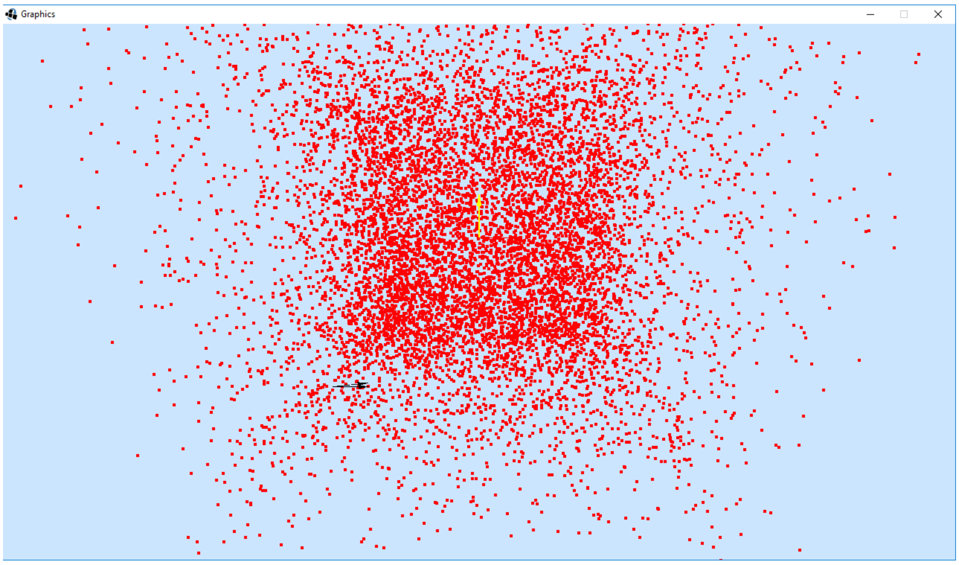


Figure 5.7 Dataset of 10,000 random ten-dimensional datapoints



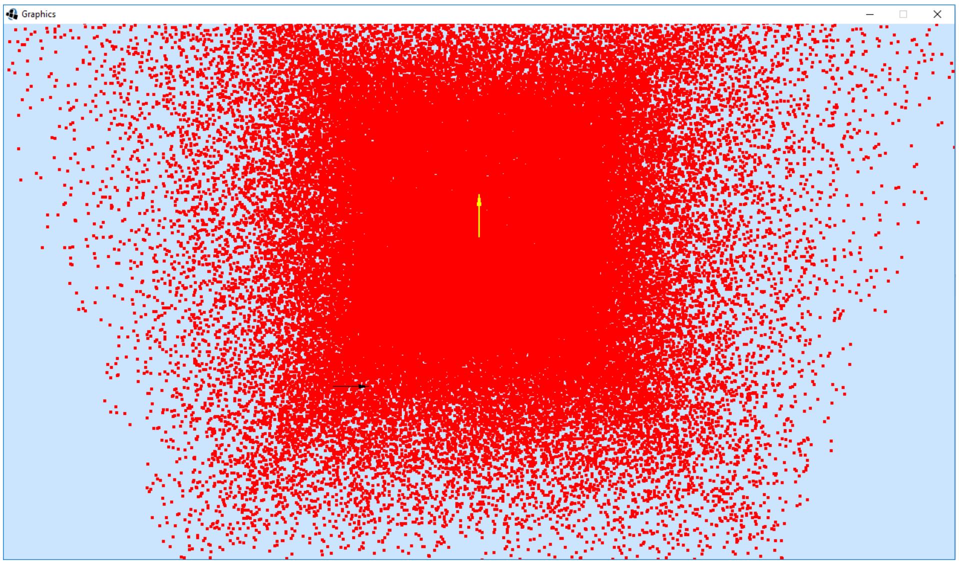
In figure 5.8, 100,000 data points are displayed. Now, the performance is punished noticeably and applying rotations to the data is not smooth. The visuality of each point is also suffering; it can be challenging to isolate points. The animation of points does help, but not a lot, as the motion of the data is choppy.

Figure 5.8 Dataset of 100,000 random ten-dimensional datapoints



When the dataset consists of one million points, as in figure 5.9, the performance is poor. It is now nearly impossible to rotate the data, as each movement is time consuming. The amount of points in the scene makes it more or less impossible to single out points; except for extreme outliers.

Figure 5.9 Dataset of one million random ten-dimensional datapoints



The challenges when increasing the amount of entries in the dataset aside; the program *is* able to handle this amount of points.

5.4 Interpreting the Graphics

Figure 5.10 Datapoints Before (Top) and After (Bottom) Rotation

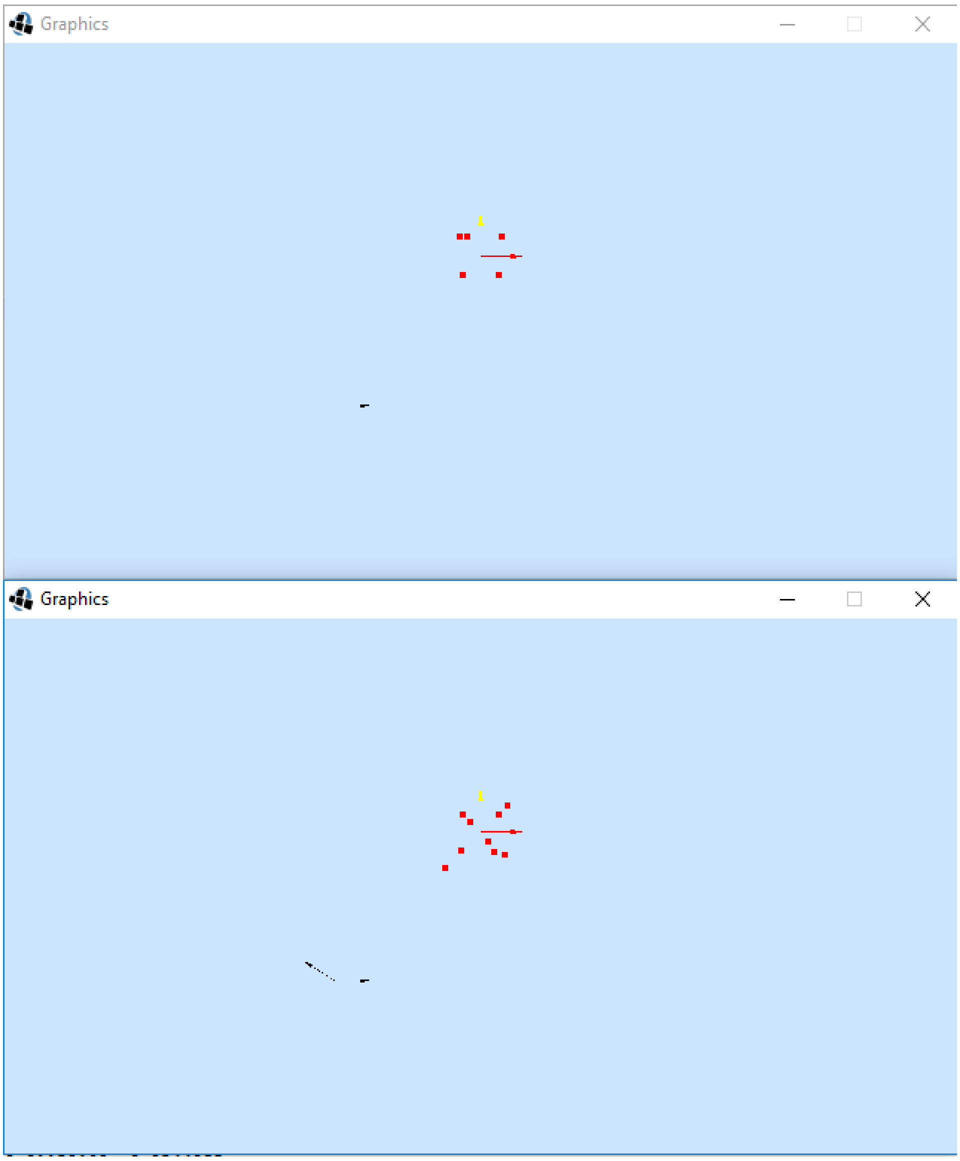


Figure 5.10 shows a set of datapoints before and after being affected by a rotation. This is done to illustrate how hidden dimensional information is revealed when rotation occurs. This is an important feature of the program, as it realizes the main objective of the implementation; the user of the program can interactively discover information about the data, which is not visible in the initial three-dimensional projection of the data.

Chapter 6

Discussion

6.1 Literature Review

Several literature exist on the field of high-dimensional data visualization, and also several algorithms are designed to deal with high dimensionality. To efficiently analyze such data, the aim is to utilize the human vision, by somehow visualizing the feature-rich dataset. As our minds are limited in this, different approaches are done to present data in a presentable way. This includes manipulating the data or using techniques to present all information at once. Data manipulation often offers some sort of pattern recognition; locating where there is interesting information to be seen or found. Another approach is to let the user herself explore the data and find interesting things in the set. High-dimensional data visualization is an active field of research and is more and more commonly used in modern fields like machine learning, but also classic fields like astronomy. Ever-increasing computing powers allow larger and larger datasets to be analyzed, and high-dimensional visualization will be a field of rising attention in the future.

The subject of visualization is vast, and the reviewing of literature can be taken into many different directions. There are many methods that can visualize the data through special plotting techniques, designed for high-dimensional data, such as scatter plot matrices or parallel coordinate plots. The limitation is: (1) only limited bivariate relations are found, and (2) they do not scale well for very complex data. Thus, the best approach to modern high-dimensional visualization is, arguably, through some sort of data manipulation. An interesting approach is subspace clustering methods. These algorithms can find more complicated multivariate structures in the dataset. Datasets are assumed to be composed of a mixture of low-dimensional linear spaces with mixed dimensions. These methods produce non-axis-aligned spaces, and work well for datasets that has different closely related dimensions.

Perhaps the most interesting approach though, visually, is exploration methods. Here, the data is projected onto a lower dimensional space, typically three-dimensional, so it can be visualized by a traditional scatter plot method. However, allowing real-time manipulations of the data, to explore what is hidden in the higher dimensions, these methods invite the user to explore and discover useful insight in the data. This was the approach chosen for the implementation made.

6.2 Development

The development of the program consisted of setting up an appropriate OpenGL environment and setting up data manipulation techniques. It was decided to use OpenGL, due to previous experience with it; and also the fact that this leaves the development stage very open, so that the design can be fully tailored for its purpose. Since OpenGL renders points at given Cartesian coordinates in the scene, the visualization itself is simple after the points have been projected and downscaled into three-dimensional points.

To invoke OpenGL functionalities was simple for this setup. Different renderers were used to enable different settings at rendering of different objects. The shader programs were simple, and used common matrix manipulation for projection, translation and rotation. It was decided to implement a common video game-like camera, so that the user not only can rotate the data, but also move around in the rendered scene.

The dataset was put through stages of reading, storing, rendering and manipulation. Manipulation was the time consuming part. Picturing a rotation of a four-dimensional space is quite hard. It does not get easier, as the number of dimensions increase. Hence, a lot of time was spent on the formulas for doing this.

6.3 Resulting Implementation

The implementation proved challenging. High-dimensionality is abstract and hard to grasp, but in the end, an approach was found.

As per specification, the data successfully rotates on demand. It is difficult to predict behaviour based on the fact that the data is high-dimensional, but the motion is a clearly periodical and rotation-like, about the origin. The user interface works as desired and gives the opportunity to both rotate the data, and also, move around and look around in space. The response time is satisfactory – unless there is a huge amount of data rendered – and the program runs smoothly.

The application can help gain insight in the data. Trends can be unveiled, and seemingly equal datapoints can be revealed to have different values in hidden dimensions. The challenge here is that it is abstract to think in high dimensions, and reading a data point is not directly revealing its features, in a way that is interpretable.

Another challenge is that OpenGL is designed to work in three dimensions (or two). Hence, the highly parallelizable task of affecting all datapoints with the same operation, cannot be done through GPU-powered shader programs. Therefore, the VAOs has to be updated each time the data is manipulated and the CPU has to do the work.

The grid was not helping; it probably did more damage than good. The points were almost hidden in the mess. If there is to be a grid, it should be placed in the background of the data, and not on top of it. Also, the points were all just rendered as points of a certain colour. The points are difficult to single out and keep track of and points far away are hidden. So, some shading techniques to help separate and differentiate the points would be useful.

A drawback with the design is that it does not allow for rotation about multiple angles in the same motion. Each time a different angle is manipulated, the points reset their position. When switching back to an already altered angle, the data returns to the position it was in the last time it was used. This is obviously not ideal. The preferred design would allow for seamless motion among all angles.

Even though this system can be up-scaled, it is not the ideal approach for very high-dimensional data. When the number of dimensions reach hundreds or thousands, using space clustering methods – at least in advance of the projection – is a better way to analyze the data. This also applies to a very large dataset. Manual exploration can gain insight and lead to interesting discoveries, but letting an algorithm do the work, is much less work for the researcher – and thus, a method more likely to be used by a researcher.

6.4 Future Work

There a three clear points regarding future work.

First, realize a design that allows all angles to be different from the initial value, simultaneously. This opens up for better understanding of the data and the relationships of the different dimensions.

To make it easier to interpret the data, it could be useful to use shading techniques to make points more distinct. For instance, using shader techniques to make nearby points transparent. This is quite complex. When making something transparent in OpenGL, the fragment shader mixes the colours of the transparent object in the front with the colours of the object behind. This way transparency is simulated. Thus, in order to make the colour of a nearby point resemble what is behind it; all points must be ordered by distance, and then rendered in that order, from farthest to closest. Otherwise it is not possible to show far behind points.

Finally, to maximize interpretation, it is very useful to have a good grid, legends and otherwise a good looking design. These things were not weighted in the implementation made in this project. Basically, the step of view transformaion has a lot of potential for improvement.

Chapter 7

Conclusion

To analyze data, visualization has been useful for many years. Graphics makes full utilization of the human vision, and helps make interpretation of data easier. Due to physical limitations, however, visualizing high-dimensional data is a challenge.

Methods like parallel plot, Andrews plot, glyph plot, scatter plot matrix and radar chart, all display the data more or less directly, without much need of manipulation. Other techniques depend on the data suffering some sort of dimension reduction or other data manipulation, before displaying the data in more traditional two- or three-dimensional graphics. These methods can involve different levels of user interaction.

Algorithms are often used to identify patterns and trends automatically. These are especially useful in big datasets where, manual work is highly time-consuming. Other methods do, however, include user interaction. Here, the user can manually explore the data.

Manual exploration of the data was the focus on the implementation presented in this thesis. In the final application, data is projected onto a three-dimensional space, rendered by the use of OpenGL. The program allowed the user to rotate the datapoint about any of the maximum ten dimension axes.

With the implementation, a researcher can display several thousand high-dimensional points, and interactively rotate them. Through this, the user can discover relations among points and detect outliers.

The design has clear points of improvement for the future. The current rotates the data about one axis at a time, and finding relationships between different dimensions cannot be done optimally. Neither is the design optimized; all points look the same, and only what is in front is shown. Nor does a good grid system or legends exist.

However, the implementation can serve as a starting point for further development, if exploration is the desired resulting application. The framework is set, and the rendering through OpenGL is initialized. The design is made scalable and should be easy to modify into a desired direction.

Bibliography

[1] Wikipedia article on data visualization.
URL: `https://en.wikipedia.org/wiki/Data_visualization`

[2] Historical aspect of visualization
URL: `https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception`

[3] List of methods for visualizing high-dimensional data
URL: `https://www.quora.com/Whats-the-best-way-to-visualize-high-dimensional-data`

[4] Introductional video to visualizaiton in machine learning.
URL: `https://aiexperiments.withgoogle.com/visualizing-high-dimensional-space`

[5] Buja, A., Cook, D., Swayne, D. F., Interactive high-dimesional datavisualization. Journal of Computational and Graphical Statistics, Vol. 5, No.1 (Mar., 1996), pp. 78-99.

[6] Grinstein, G., Trutschl, M., Cvek, U., 2001. High-dimensional visualizations.

[7] Wilkinson, L., 2005. The Grammar of Graphics, Second Edition. Springer.

[8] Ward, M., Grinstein, G., Keim, D., 2010. Interactive Data Visualization:Foundations, Techniques, and Applications. 360 Degree Business. CRC-Press.
URL: `https://books.google.no/books?id=Kk7NBQAAQBAJ`

[9] Post, F., Nielson, G., Bonneau, G., 2002. Data Visualization: The State of theArt. The Springer International Series in Engineering and Computer Science.Springer US.
URL: https://books.google.no/books?id=WAZYsfeMi4kC

[10] Wang, J., 2012. Geometric Structure of High-Dimensional Data and Dimensionality Reduction. Springer Berlin Heidelberg.
URL: https://books.google.no/books?id=0RmZRb2fLpgC

[11] Famili, A., 2005. Advances in Intelligent Data Analysis VI: 6th International Symposium on Intelligent Data Analysis, IDA 2005, Madrid, Spain, September 8-10, 2005, Proceedings. No. v. 6 in Information Systems and Applica-tions, incl. Internet/Web, and HCI. Springer.
URL: https://books.google.no/books?id=Ip8bxhQZMKYC

[12] Wikipedia article on Kullback-Liebler divergence.
URL: https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

[13] Matlab methods overview
URL: https://se.mathworks.com/help/stats/examples/visualizing-multivariate-data.html

[14] Iris flower dataset, wikipedia
URL: https://en.wikipedia.org/wiki/Iris_flower_data_set

[15] Ultsch, A., 2014. Maps for the visualization of high-dimensional data spaces.

[16] Single cell proteomics in biomedicine: High-dimensional data acquisition, visualization, and analysis
URL: http://onlinelibrary.wiley.com/doi/10.1002/pmic.201600267/full

[17] Contextual mapping: Visualization of high-dimensional spatial patterns in a single geo-map
URL: http://www.sciencedirect.com/science/article/pii/S0198971516301946

[18] Khot, T., 2016. Visualizing high-dimensional data.

[19] Liu, S., Maljovec, D., Wang, B., Bremer, P.-T., Pascucci, V., 2015. Visualizing High-Dimensional Data: Advances in the Past Decade

[20] Waddell, A., Oldford, W., 2014. RnavGraph: an R package to visualize high dimensional data using graphs as navigational infrastructure

[21] Producing high resolution animations of high-dimensional data with tour
URL: `http://www.calgorithms.com/blog/2014/07/11/producing-animations-of-high-dimensional-data-with-tourr/`

[22] Tagliaferri, R., Staiano, A., 2005. Visualization of High Dimensional Scientific Data

[23] Blog on high-dimimensional visualization
URL: `http://jamesxli.blogspot.no/`

[24] UCSD lecture on high-dimensional visualization
URL: `https://www.youtube.com/watch?v=EMD106bB2vY`

[25] How-to in wolfram
URL: `http://community.wolfram.com/groups/-/m/t/236139?p_p_auth=v83aljXm`

[26] Scatter plot article
URL: `http://link.springer.com/article/10.1007/s12650-014-0230-5`

[27] Buja, A., Cook, D., Asimov, D., Hurley, C., 2004. Theory of Dynamic Projections in High-Dimensional Data Visualization

[28] Buja, A., Cook, D., Asimov, D., Hurley, C., 2004. Computational Methods for High-Dimensional Rotations in Data Visualization

[29] Self-organizing map, wikipedia
URL: `https://en.wikipedia.org/wiki/Self-organizing_map`

[30] Scatter plot, wikipedia
URL: `https://en.wikipedia.org/wiki/Scatter_plot`

[31] Andrews Plot, wikipedia
URL: `https://en.wikipedia.org/wiki/Andrews_plot`

[32] Article on parallel coordinates
URL: `http://www.juiceanalytics.com/writing/writing/parallel-coordinates`

[33] Four-dimensional sphere
URL: `http://www.foundalis.com/phy/4Dsphere.htm`

[34] OpenGL visualization.
URL: `http://www.sciencegl.com/`

[35] High-dimensional OpenGL
URL: `https://cs.earlham.edu/~mccoyjo/project/`

[36] PCA, wikipedia
URL: `https://en.wikipedia.org/wiki/Principal_component_analysis`

[37] Multidimensional rotations
URL: `https://www.av8n.com/physics/rotations.htm`

[38] Intro to R and ggplot2
URL: `https://www.youtube.com/watch?v=Uwb1wjBu8xA&index=1&list=PLc7gSg5yVeC3dyQFSxIZ8wxLBsPTfmqDq`

[39] Goodman, A., 2012. Principles of High-Dimensional Data Visualization in Astronomy

[40] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-SNE. Journal of Machine Learning Research 9 (2008): 2579-2605.

[41] Friendly, M., Denis, D., 2005. The Early Origins and Developments of the Scatterplot

[42] Andrews, D.F., 1972. Plots of High-dimensional Data

[43] Jeong, D.H., Ziemkiewicz, C., Fisher, B., Ribarsky, W., Chang, R., 2009. iPCA: An Interactive System for PCA-based Visual Analytics.

[44] Nielsen, M., 2016. Rendering a 3D Game World Using a Programmable Graphics Pipeline and Implementing a Procedural and Infinite Game World Terrain.

[45] Omics, wikipedia
URL: `https://en.wikipedia.org/wiki/Omics`

[46] Parsons, L., Haque, E., Liu, H., Subspace Clustering for High Dimensional Data: A Review

[47] Bellman, R. E. (1957). Dynamic programming. Princeton, NJ: Princeton University Press.

[48] Florence Nightingale, wikipedia
URL: https://en.wikipedia.org/wiki/Florence_Nightingale

[49] Image: Paolo.dL (Own work) [CC BY-SA3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons. https://upload.wikimedia.org/wikipedia/commons/9/98/Projection_and_rejection.png

[50] Image: No machine-readable author provided. JitseNiesen assumed (based on copyright claims). [Public domain], via WikimediaCommons. https://upload.wikimedia.org/wikipedia/commons/e/eb/Orthogonal_projection.svg

Appendix

A Setting up the Project in Eclipse

This is a description of how to download necessary tools and setting up the project in Eclipse

1. First, the Java SE Development kit has to be downloaded and installed from `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`.
2. Download and install the Eclipse Neon IDE - `http://www.eclipse.org/neon/`
   In the installer, the option *Eclipse IDE for Java Developers* i chosen to be installed.
3. When Eclipse is installed, run it and create a new project.
4. In the eclipse workspace folder we want the following structure.

Folder / File	Description
`workspace`	The folder of the eclipse workspace
`└─projectname`	The folder of the relevant eclipse project.
`   ├─bin`	
`   ├─lib`	The folder containing the library files.
`   │  ├─jars`	The folder containing jars.
`   │  └─natives`	The folder containing natives.
`   └─src`	The folder containing the java packages.
`      ├─package1`	
`      │  ├─class11`	
`      │  ├─class12`	
`      │  └─class1N`	
`      ├─package2`	
`      └─packageN`	

That typically means that the lib-folder, and its subfolders has to be created manually.

5. In the *jars* folder, the jar-files from the lwjgl-folder are put. The appropriate OS native files are put into the *natives*-folder.
6. Right-click the project in the eclipse workspace viewer. Hover above *Build Path*, and from here select *Configure Build Path...*.
7. In the right column, click on the *Add JARs...* button. Navigate to the jars-folder, mark all .jar files and press OK.
8. Next, in the middle window (of the build path configurator), click on the arrow left of *JRE System Library*. Third from top, should be a *Native library location*. Click on that, then click on *Edit...* in the right column. Here, click on *Workspace...* and select the natives-folder. Then click OK-buttons until the properties window disappears.
9. The provided src-folder's contents can be copied to the new project's src-folder.
10. By right clicking the project in the eclipse workspace viewer again, one can press the *Refresh* button, and the classes should appear, and the code can be run.