Aggregated Parallel Coordinates

Multi-Dimensional Information Visualisation
in Race Car Engineering

Majda Osmić

Aggregated Parallel Coordinates

Multi-Dimensional Information Visualisation
in Race Car Engineering

Master's Thesis

at

Graz University of Technology

submitted by

Majda Osmić

Institute for Information Systems and Computer Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

21 May 2015

Advisor: Ao.Univ.-Prof. Dr. Keith Andrews

Aggregierte Parallelkoordinaten

Multidimensionale Informationsvisualisierung im Motorsport

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Majda Osmić

Institut für Informationssysteme und Computer Medien (IICM),
Technische Universität Graz
A-8010 Graz

21. Mai 2015

Diese Arbeit ist in englischer Sprache verfasst.

Begutachter: Ao.Univ.-Prof. Dr. Keith Andrews

Abstract

In race car engineering, specialised simulation software is used to derive a car setup and a driving strategy for optimal performance during a race. Such software simulations generate complex, high-dimensional datasets with a large number of records. To explore such datasets, a corresponding visualisation tool is required.

This thesis is the result of cooperation between the AVL's Racing department and Graz University of Technology. The main goal was to build a visualisation tool optimised for exploration of datasets produced by the AVL's race car simulation software. Since the complexity of these datasets emerges both from high-dimensionality, as well as the hierarchical structure of the dimensions, several techniques for visualising multi-dimensional and hierarchical data were explored. While some existing techniques are appropriate for visualising particular subsets of the produced data, none of them provide an appropriate mechanism for visualising the hierarchy of dataset dimensions.

"Aggregated Parallel Coordinates" is an extension of the standard parallel coordinates technique, which. aside from an effective way to visualise high-dimensional data, supports visualisation and exploration of hierarchies within the dataset dimensions. As part of this thesis, an aggregated parallel coordinates visualisation was implemented as a WPF user control library, which can be added to any WPF application. It provides a comprehensive set of interactions, consisting both of features found in many parallel coordinates implementations, and some unique features especially designed to enhance the process of visual exploration of race car simulation data. The software is already being used as a part of AVL's race car simulation data visualisation tool called SimBook.

Kurzfassung

Spezialisierte Simulationsprogramme werden im Motorsport verwendet, um die Fahrzeugeinstellung und Rennstrategie zu ermitteln, die während des Rennens am effizientesten sind. Derartige Softwaresimulationen generieren große, komplexe Datenmengen. Zur Verarbeitung dieser Datensätze ist eine entsprechende Visualisierung erforderlich.

Diese Diplomarbeit ist in Zusammenarbeit zwischen AVL Racing und der Technischen Universität Graz entstanden. Hauptziel war es, ein Visualisierungsprogramm zu entwickeln, welches für die Untersuchung, der aus Simulationen erhaltenen Datensätze, optimiert ist. Da sich die Komplexität solcher Datensätze aus der hohen Zahl an Dimensionen sowie der hierarchischen Struktur ebendieser ergibt, wurden verschiedene Techniken zur Visualisierung von mehrdimensionalen und hierarchischen Daten untersucht. Obwohl einige der bereits vorhandenen Methoden zur Abbildung bestimmter Bereiche der erhaltenen Daten geeignet sind, bietet keine eine passende Darstellung von den Hierarchien der Dimensionen dieser Datensätze.

Die "aggregierten Parallelkoordinaten" sind eine Erweiterung der Standard-Parallelkoordinaten, welche, neben einer effektiven Möglichkeit zur Visualisierung von hochdimensionalen Daten, auch die Darstellung und Untersuchung von Hierarchien innerhalb der Datendimensionen erlaubt. Als Teil dieser Arbeit wurde die entsprechende Visualisierung in Form einer WPF Programmbibliothek entwickelt. Diese bietet eine umfangreiche Palette an Methoden aus bereits bestehenden Parallelkoordinaten Implementierungen, sowie einzigartiger Funktionen, die speziell entwickelt wurden, um den Prozess der visuellen Untersuchung von Simulationsdaten aus dem Motorsport zu optimieren. Die Software, die im Rahmen dieser Diplomarbeit entstanden ist, kommt bereits als Teil einer Visualisierungssoftware namens SimBook bei AVL zum Einsatz.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

First and foremost, I would like to thank my parents and my sister for all their help and support over the years of my studies.

I would like to thank Gerhard Schagerl for providing me an opportunity to work and write my thesis at the AVL Racing. I especially wish to thank Catrin Schlager, whose advice and help during the time I spent working and writing this thesis have been invaluable to me. Furthermore, I would like to thank Jörg Schlager for his advice on performance optimisation of my software. For providing me with good literature on vehicle dynamics I have to thank Guillermo Pezzetto. I also have to mention my colleague, Thomas Gerstorfer, who has reviewed my chapter on race car simulation.

For all the help during the implementation of the practical part of my thesis, for providing me with a LaTeX skeleton for this thesis, as well as for the endless hours spent correcting the draft versions of this work, I would like to thank my advisor, Keith Andrews.

Special mention goes to my friends, Ilija Šimić and Aniko Toth, who have taken the time to review individual chapters of this work.

Majda Osmić
Graz, Austria, May 2015

Chapter 1

Introduction

"The beginning is the most important part of the work."

[Plato]

One of the main objectives in race car engineering is finding a vehicle configuration which can achieve optimal performance during a race. Since a large variety of factors influence the behaviour of a car on a race track, software simulations are used extensively to support this highly complex task. Chapter 2 provides a short overview of some of the car components which can be fine-tuned to correspond to the demands of the driver and the race track. Based on this overview, the complexity of the datasets produced by race car simulation software is explained.

Making sense of the data can be very difficult without an appropriate visualisation. In information visualisation, the capacity of the human visual perception system to notice changes in size, colour, and shape is utilised to convey abstract information. Finding a suitable visual representation depends highly on the type of data being visualised. Furthermore, interaction with the data plays an important role in the process of visual exploration. Chapter 3 provides an overview of interaction techniques commonly used in many information visualisation systems. Since the datasets produced by race car simulations are high-dimensional, and a certain hierarchical structure is present within the dimensions of these datasets, some of the techniques commonly used to visualise both of these data types are described.

Parallel coordinates are a multi-dimensional data visualisation technique. Each dimension in the dataset is represented by an axis, but, rather than placing the axes orthogonally, like in scatter plots, the axes are ordered in parallel to each other. Using this approach, an arbitrary number of dimensions can be visualised. The values of individual data points are mapped to a corresponding position on each axis. The data points which belong to the same record in the dataset are connected by polylines, which span across the axes. This visualisation technique is discussed in detail in Chapter 4. Special attention is given to the interactive features commonly implemented in parallel coordinates visualisations. When visualising datasets with a large number of records, the display tends to become cluttered by overlapping polylines. An overview of techniques for handling this problem is provided. This chapter also gives an example of an exploratory data analysis with parallel coordinates. Furthermore, several extensions of the standard parallel coordinates techniques are described, and an overview of four software applications containing parallel coordinates visualisation is given.

When supported by a set of appropriate interactive features, parallel coordinates provide an effective way to visualise and explore high-dimensional datasets. However, standard parallel coordinates cannot deal with hierarchical structure within the dimensions of the dataset. "Aggregated parallel coordinates" is an extension of the standard parallel coordinates technique, which is capable of visualising hierarchies within the dataset dimensions. Chapter 5 describes the implemented parallel coordinates visualisation, with a focus on "dimension aggregation", the feature which enables navigation through the hierarchical

structure of the dataset dimensions. An overview of how aggregated parallel coordinates are used in SimBook, AVL's tool for visual exploration of race car simulation data, is also provided.

The software was implemented in WPF, using Microsoft .NET framework version 4.0, and C♯ as the programming language. Since race car simulation datasets tend to contain a large number of records, the performance and responsiveness of the visualisation was of high importance. Chapter 6 explains how the initial implementation in WPF was done and what performance problems were met. The causes of these problems are described in detail, and the steps taken in order to solve them are explained. Other important details of the implementation are discussed in Chapter 7.

Chapter 8 gives an overview of some of the features which were not implemented due to the time limitations. Furthermore, several ideas for new interactive features which could enhance the process of visual data exploration with aggregated parallel coordinates are also presented.

Since aggregated parallel coordinates are implemented as a WPF user control library, a simple demonstration tool was also built as part of this thesis, in order to provide a way to demonstrate the capabilities of the visualisation. A user guide for this demo application is given in Appendix A. The developer guide in Appendix B explains how the user control library can be included into, and be used as a part of, other WPF applications. Additionally, a short overview of the code structure is also provided for those who wish to extend the current implementation.

Chapter 2

Race Car Simulation

"I was doing fine until about mid-corner when I ran out of talent."

[Unknown author]

Car racing is a challenging and competitive sport, requiring high physical and mental capabilities of the driver, and relying heavily on the design and setup of the vehicle. Based on the type of the vehicle and the track, car racing can be divided into many different categories, such as formula racing (Formula 1, GP2, etc.), touring car racing (WTCC, DTM, etc.), stock car racing (NASCAR racing series), rallying, off-road racing, and others. Independent of the category, the goal is to complete the pre-defined track in the fastest possible time.

Racing vehicles are designed and manufactured according to the rules and guidelines of the specific racing category. Fédération Internationale de l'Automobile [2015] (FIA), the governing body for motor sport worldwide, defines strict technical regulations regarding the dimensions of the vehicle, its weight, materials used in construction, fuel system, power unit, braking system, wheels, tyres, electronic and software systems, safety mechanisms, etc. Although all vehicles in the specific racing category have the same basic components and characteristics, the performance of the vehicle on the track varies based on the adjustments and fine-tuning of the individual car components.

> "The overall technical objective in racing is the achievement of a vehicle configuration, acceptable within the practical interpretation of the rules, which can traverse a given course in a minimum time (or at the highest average speed) when operated manually by a driver utilising techniques within his/her capabilities." [W. F. Milliken and D. L. Milliken, 1995, page 3]

Before every racing event, race car engineers adapt the setup of the car to the track, to the expected weather conditions, and to the driver's driving style [Khan, 2007] . Deriving a racing strategy and a car setup to achieve optimal performance for a given race track is not a simple task. Teams are allowed to spend only a very limited amount of time on the track before the main racing event, during which both the car's and driver's performance can be tested. In modern race car engineering, special software solutions which simulate the behaviour of the racing car on the circuit are used before the race to test how tuning different vehicle components affects the overall lap time. This allows teams to focus more on the driver's performance during the testing sessions.

In order to achieve a better understanding of the input and output parameters of a typical race car simulation, Section 2.1 provides a short overview of some of the vehicle parameters which can be fine-tuned and the effect of changing their values on the car's performance. This is followed, in Section 2.2, by a discussion of the complexity of the datasets produced by such simulations.

2.1 Race Car Vehicle Dynamics

Smith [1978] defines vehicle dynamics as

> "... the study of the forces which affect wheeled vehicles in motion and of the vehicle's responses, either natural or driver induced, to those forces."

The most important aspects of vehicle dynamics in car racing are acceleration, braking, cornering, top speed, and vehicle handling [Smith, 1978].

Acceleration is influenced at high speeds by the engine power, and at low speeds by traction limits on the drive wheels [Gillespie, 1992]. Power-limited acceleration, along with top speed, is pre-defined by the vehicle design. Traction-limited acceleration, on the other hand, can be controlled by setting-up tyres so that the optimal amount of friction between the tyre and the road is achieved. The term "vehicle handling" refers to the responsiveness of the vehicle to the drivers input (steering). The behaviour of a vehicle in curves (corners) is, among others, related to handling, and is usually measured by the understeer or over-steer gradients [Gillespie, 1992]. Handling and cornering performance can be adapted to the requirements of the track and the driver's preferences by fine-tuning tyre setup, suspension system, gearbox, differential, brakes, and aerodynamics [Khan, 2007]. The main components in more detail are:

- **Tyres**
  Tyres are the primary source of forces which influence the control and the stability of the vehicle [W. F. Milliken and D. L. Milliken, 1995]. The grip achieved by tyres together with the car balance dictate the speed at which the car is able to drive around a track corner. The contact area between the tyre and the road depends on the *camber angle*, that is the angle between the vertical axis of the wheels and the vertical axis of the vehicle, as seen in Figure 2.1. When the vehicle enters a curve, the camber angle decreases making the contact area between the tyre and the road larger. Changing the camber angle has a direct influence on the tyre grip. Similarly, changing the angle between the horizontal axis of the vehicle and the horizontal axis of the tyres (*toe*) influences the tyre grip and the slip angle (the angle between the direction in which the tyre is rolling and the direction in which the tyre is pointing). Changing the tyre pressure also affects the vehicle's cornering performance by affecting the tyre stiffness, friction, etc. [Khan, 2007].

- **Suspensions**
  The suspension system is the link between the vehicle's body and its tyres, and has an enormous influence on the motion behaviour of the vehicle. It provides vertical compliance, so the wheels can follow uneven roads, maintain the proper steer and camber attitudes, react to the forces produced by tyres, resist roll of the chassis, and keep the tyres in contact with the road [Gillespie, 1992]. The suspension system includes springs, dampers, anti-roll bars, and suspension geometry. Among others components, springs control the rate at which the suspensions deflect during vehicle cornering. Spring stiffness can be varied in order to control handling and the transmission of vibrations [Khan, 2007]. Anti-roll bars are springs added to the front and rear of the vehicle in order to provide resistance to roll movement. Their stiffness can be changed in order to adapt the amount of steering necessary to pass a corner. Dampers, along with the springs, control the amount of transferred oscillation, thus also influencing vehicle handling.

- **Brakes**
  Adjusting the amount of hydraulic pressure applied to the front and the rear brake components is necessary in order to control braking efficiency. *Brake bias* indicates the relative amount of pressure applied to the front brake components, and is usually expressed in percentages. A brake bias of 52 indicates that 52% of the brake pressure is applied to the front, and 48 % to the rear

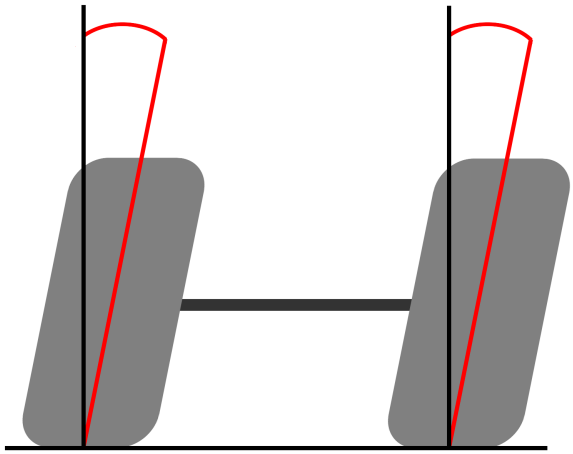


Figure 2.1: From the front: the negative camber angle of the left wheel makes it lean towards the axle. The right wheel has a positive camber angle, and it leans away from the axle. [Image created by the author.]

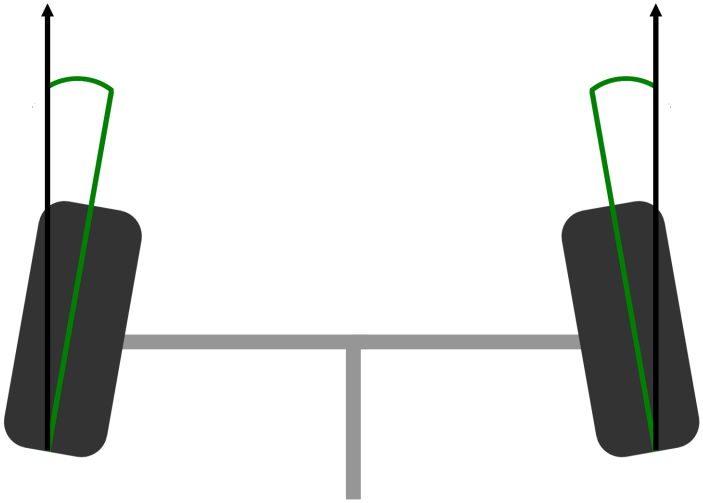


Figure 2.2: From above: both wheels have a positive toe ("toe in") since they point towards the centreline of the vehicle. [Image created by the author.]

brake components [iRacing, 2015]. Apart from the driver's personal preferences, the optimal brake balance also depends on the track, because brake efficiency changes depending on whether the car is moving on a flat surface, uphill, or downhill [Khan, 2007].

- **Aerodynamics**
  The downforce created by the aerodynamic properties of the car increases the loads on the tyres (which is normally limited by the vehicle's weight), pressing the car more towards the surface. This has a direct influence on the cornering performance of the vehicle [Katz, 1995]. The aerodynamic downforce also induces aerodynamic drag (friction) which decreases vehicle's maximum possible acceleration and speed. The aerodynamic forces exhibited on the vehicle have to be balanced depending the track type. On tracks with more sharp corners, downforce needs to be increased, whereas high-speed tracks with more straight segments call for less downforce in order to reduce drag. The downforce can be controlled by varying the front and rear wing angles, and the distance of the chassis from the road (*ride height*) [Khan, 2007].

When fine-tuning a racing vehicle, many more components, in addition to the ones described above, have to be considered. Describing all of them is out of the scope of this work. However, the overview above should suffice to grasp both the complexity of such an endeavour, as well as the complexity of a typical dataset produced by race car simulations.

2.2 The Complexity of the Simulation Dataset

In most racing categories, the track is a closed path consisting of straight segments, followed by curves, which are again followed by straight segments. The challenge of car racing on such a track is to:

> "... increase speed at maximum rate (accelerate) out of each turn and continue to the point where, with maximum braking (deceleration), the speed can just be brought down to the maximum speed of the next corner." [W. F. Milliken and D. L. Milliken, 1995]

As seen from the short overview of vehicle dynamics provided in Section 2.1, the number of variables influencing the car's behaviour both on straight segments of the track as well as the curves is enormous. When using simulation software to test different car setups and their influence on the overall lap time, the parameters of the setup and their variation range are given as input. Such input parameters include tyre camber angle, spring stiffness, position of the front and rear wings, etc. The number of records in a dataset produced by race car simulations depends on the number of input parameters, since the combinations of different values of input parameters (different setup variations) are simulated. The simulation output contains not only the time in which the car will complete the track with a given setup, but also the calculated influence of the setup variations on the huge number of parameters affecting vehicle performance, such as aerodynamic efficiency, tyre grip, and many others. Some of the calculated output parameters are more relevant for the straight segments of the track, whereas others are more relevant for vehicle cornering.

The corners of the track are usually divided into three segments: entry, mid, and exit. Since some of the parameters influencing the vehicle's cornering performance have different values in different segments of the corner, their values are calculated separately for every segment of every corner of the track. In addition, parameters which have different values at the front and back, as well as the left and right side of the vehicle, are also calculated separately. Thus, the complexity of datasets produced by simulating changes in the car's setup and their influence on the overall lap time is not only caused by high-dimensionality, but also by the fact that many dimensions cannot be observed independently, since a certain hierarchical dependency structure is present. Table 2.1 shows a typical dataset produced by race car simulations containing different types of parameters, both with and without hierarchy dependencies.

Depending on the number of input parameters, these datasets can also contain a large number of records, especially in cases where every possible variation of one parameter is simulated with every possible variation of all other parameters.

An essential requirement for a visualisation tool designed for use with the kind of datasets described above is identifying and grouping the dimensions (parameters) and presenting them in a meaningful way. The user has to be able to navigate easily through the list of parameters. Apart from separating input parameters (the setup of the car components) from output parameters, parameters specific to corners, and those specific to straight parts of the track have to be easily identifiable. A visualisation of the track, including its straights, corners, and corner segments (entry, mid, and exit) should be provided in order to give visual feedback as to which specific parts of the track are selected.

No.	...	CamberFL	CamberFR	CamberRL	CamberRR	ToeFL	ToeFR	ToeRL	ToeRR	Lap Time	...
1		1.6	3.1	1.8	3.2	2.7	1.5	5.3	-0.61	98.1238	
2		2.8	2.5	2.6	2.4	3.6	2.6	4.1	-0.32	99.3412	
...		...								...	
1000		1.3	4.2	3.1	3.8	5.1	3.8	2.6	-0.49	98.9615	

(a) Portion of a typical race car simulation dataset showing two input parameters (camber and toe) defined separately for all 4 vehicle wheels, as well as one output parameter (lap time).

No.	...	Top Speed	Top Gear	RideHeight C1	RideHeight C2	...	RideHeight C20	RideHeight C21	...
1		287	5	5.28	3.21		7.28	9.53	
2		292	7	6.31	9.30		3.56	4.81	
...		...					...		
1000		291	6	8.75	6.32		4.85	3.21	

(b) Portion of a race car simulation dataset showing two types of output parameters: general parameters (top speed and top gear), and a single parameter (ride hight) with separate table entries for every corner of the track.

No.	...	Handling C1Entry	Handling C1Mid	Handling C1Exit	...	Handling C21Mid	Handling C21Exit	...
1		-0.2	-0.3	-2.0		-0.3	-0.8	
2		-2.8	-1.1	-1.0		-0.1	-0.3	
...		...				...		
1000		-0.5	-0.7	-0.6		-0.9	-0.9	

(c) Portion of a race car simulation dataset showing an output parameter (handling) with table entries for all three segments (entry, mid, and exit) of every corner of the track.

Table 2.1: A typical race car simulation dataset has a few hundred dimensions. Here, three subsets of dimensions are shown for the same set of records. Table (a) shows a portion of a dataset with 8 columns for input parameters CamberFL, CamberFR, CamberRL, CamberRR, ToeFL, ToeFR, ToeRL, ToeRR, and one column for the output parameter LapTime. The input parameters Camber and Toe are aggragate parameters, since each is actually defined individually, once for each of the four wheels. Table (b) shows a portion of the dataset with two types of output parameters: parameters with values referring to the entire track (TopSpeed and TopGear), as well as one parameter (RideHeight) calculated for every corner of the track, meaning that RideHeight is an aggregate parameter (dimension), containing as many subdimensions as there are corners on the track. Table (c) shows yet another type of output parameter with even deeper dependency hierarchy, since the values of one parameter (handling) are calculated for every segment (entry, mid, and exit) of every corner of the track.

Chapter 3

Information Visualisation

> *"There are things known and there are things unknown, and in between are the doors of perception. "*
>
> [Aldous Huxley - The Doors of Perception]

Visual representation is a powerful way to communicate information. The human brain can process images much faster than text, because the speed of consumption of written information is limited by the sequential process of reading, while images can be consumed "as a whole" in one glance. Furthermore, visual information is often culturally independent and does not rely on an individual's knowledge of any particular language [Matthew Ward, Grinstein, and D. Keim, 2010, Chapter 1].

Throughout history, humans have used pictures to convey information. The earliest known examples are cave paintings. Hieroglyphics and maps also appeared relatively early in human history. The expanding knowledge of the world and the development of science contributed to the development of different types of charts and other kinds of visualisations. One example is the Charles Minard's map of Napoleon's march on Moscow, shown in Figure 3.1, which cleverly depicts the difference in the number of soldiers who started the march and those who returned. Matthew Ward, Grinstein, and D. Keim [2010, pages 6–15] and Andrews [2015, Chapter 3] provide an extensive overview of history of visualisation.

The appearance of computers has enabled not only more possibilities to visualise data, but also new ways of gaining knowledge by means of interaction. Computer-based visualisations of data help people carry out tasks more effectively [Munzner, 2014]. Card, J. D. Mackinlay, and Shneiderman [1999] define visualisation as:

> "The use of computer-supported, interactive, visual representation of data to amplify cognition."

Depending on the type of information that is being represented, the broad field of visualisation can be divided in three main sub-fields [Andrews, 2015]:

- scientific visualisation,
- geographic visualisation, and
- information visualisation.

Example of typical scientific visualisations would include a drawing of a human bone, or a 3D model of a car. Typical visualisations of geographical data are maps. In both of these visualisation sub-fields, the visual representation is inherent in the data itself. Information visualisation, on the other hand,

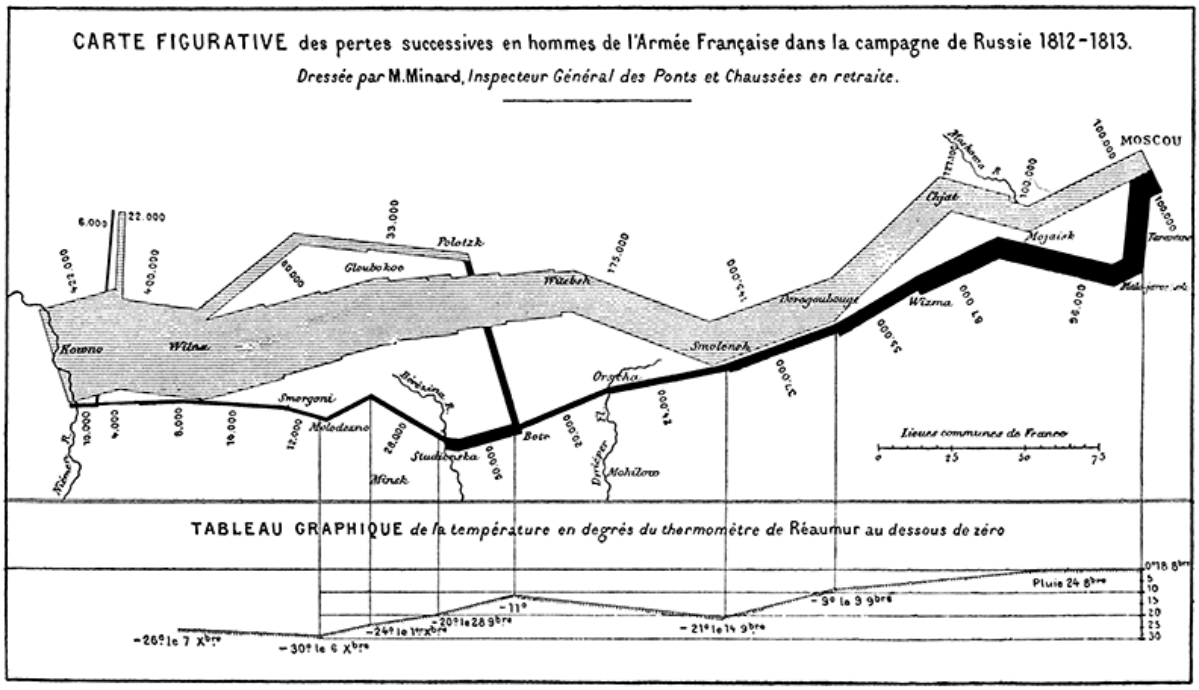


Figure 3.1: Visual representation of the army size during the Napoleon's march on Moscow. Line thickness encodes the number of soldiers, whereas the colour indicates the movement direction. The information about the temperature at specific locations is also present in the lower part of the map.
[Image scan used with kind permission of Keith Andrews.]

focuses on visualising abstract datasets, whose visual representation cannot be directly derived from the data. Such datasets include hierarchical structures, collections of documents, multivariate data, graphs, and graph structures such as social networks.

The development and improvement of different kinds of sensors, recording methods, as well as the de-creasing price of data storage, and increasing available computing power have lead to the appearance of a phenomenon called Big Data. To make use of either captured or created data, a palette of visualisation, exploration and analysis mechanisms are necessary. The use of such mechanisms in analytical reasoning is called visual analytics. D. A. Keim, Mansmann, et al. [2006] define visual analytics as:

> "iterative process that involves collecting information, data preprocessing, knowledge representation, interaction, and decision making".

There are two sides to visual analytics: data analysis and information visualisation. Data analysis refers to statistical methods and data mining algorithms which are applied "in background". Information visualisation provides visual representation and interaction in the user interface.

The two basic components of information visualisation are the visual representation and interaction. Visual representation depends on the type of the data that is being visualised. In Section 3.1 the importance of interaction with the data is shortly discussed. Since the focus of this thesis is visualisation of hierarchical and multi-dimensional data, an overview of some of the visualisation techniques for these data types is provided in Sections 3.2 and 3.3.

3.1 Interaction

The two essential parts of information visualisation are the visual representation and the interaction. Interaction enables navigating through the data, and manipulating the visual representation in order to gain more insight into the dataset. Interaction is a necessity when exploring large amounts of data. Shneiderman [1996] emphasises the importance of interaction with the data during the process of knowledge discovery in his Visual Information Seeking Mantra:

> "Overview first, zoom and filter, then details-on-demand."

Depending on the user's intent in performing a specific interaction, Yi et al. [2007] define the following categories of interaction techniques:

- **Select - marking something as interesting**
  Marking one or several data items in order to keep track of them through the process of data exploration is one of the most common interaction techniques. It is often used in combination with other interactions.

- **Explore - showing something else**
  Information visualisation systems often deal with large amounts of data which makes it almost impossible to explore the entire dataset as a whole. Users often focus on one subset of data at a time. Explore interactions allow transitions from one part of the dataset to another. One of the most common "explore" interactions is panning. "Panning" achieves transition to different parts of datasets by either moving the camera from one part of the scene to the next, or by moving the scene while the camera stays still. Scroll bars, which are available in most software nowadays, provide panning interaction by means of moving the camera.

- **Reconfigure - showing a different arrangement**
  This interaction techniques supports viewing a dataset from a different perspective. This is an important feature when it comes to exploring data in order to discover new patterns and gain new insights. Occlusion of single data items is a common side-effect when visualising large amounts of data. In this case, reconfiguring the data representation often makes previously occluded items visible. Occlusion of a part of the dataset is often present in 3D visualisations (independent of the size of the dataset). For this reason, most 3D visualisations allow users to reconfigure the representation by moving the camera to a different position.

- **Encode - showing a different representation**
  Encoding the information properly is another important aspect in visual data representation. Some of the most common properties used to encode information are colour, line length, 2D position, size, shape, and orientation. Encode interactions allow adjustment of the encoding according to the user's preferences and can help uncover different properties of the visualised dataset. Changing the opacity of the displayed data often uncovers the distribution of the dataset. Apart from changing visual properties of the displayed data items, one might change the way the data is represented. Switching from a pie chart to a bar chart representation is an example of an encode interaction which can change the user's perception of the data.

- **Abstract/Elaborate - showing more or less detail**
  These interactions allow viewing the data at different levels of abstraction. At specific stages of the data exploration process, the user might want to view the dataset either as a whole or to focus more on a particular subset. The most common abstract/elaborate interactions are simple zooming or displaying a tool-tip when a particular data item is hovered over.

- **Filter - showing something conditionally**
  Focusing only on the parts of the datasets which satisfy certain conditions is provided by the filter interaction. Items which are filtered out, are either displayed in a different way, or are not displayed at all. With this interaction, users change the conditions under which certain data items are displayed, rather than the perspective from which the data is viewed. Common filter interactions include the value range of the displayed data, or selecting a limited number of categories to visualise.

- **Connect - showing related items**
  Many information visualisation tools include simultaneous views of the same data set in different representations. The connect interaction helps users identify same single data item across several representations. Typically, when a visual property of one item changes in one view, it also changes in all other views. This interaction is also present in single views, for example when showing related nodes.

- **Undo/Redo**
  This interaction enables users to return the system into the state in which it was before applying a certain interaction, or to repeat one or more interactions. Undo/redo is a technique found im almost every software nowadays.

Information visualisation tools often deal with large amounts of data. Apart from implementing interaction techniques suitable for exploring the specific datasets, information visualisation systems must also provide reasonable responsiveness to user input. Depending on the visual representation, different techniques for reducing the number of displayed data items exist. Displaying fewer visual items, as well as employing multi-threading techniques to reduce the amount of time required to render the visualisation, can help in making the software more responsive.

3.2 Visualising Hierarchical Data

A hierarchy can be defined as a group of elements (nodes) with parent-child relationships. Each hierarchy consists of three types of elements:

- root: the single parentless element,
- leaf nodes: childless elements,
- inner nodes: elements with both a parent and one or more children.

Hierarchical data is very common, since hierarchies emerge in many different aspects of life. Examples of hierarchical data include family trees, organisational structures, file systems, etc. File system explorers are one of the most common hierarchical data visualisations.

Hans Jörg Schulz, Hadlak, and Heidruno Schumann [2011] divide visualisation techniques for hierarchical data based on whether the parent-child replationship is represented by nesting the child into the parent, or by exploiting vertical adjacency of the elements. The two categories differ in the following four design dimensions:

- dimensionality (2D or 3D),
- element (node) representation,
- edge representation,

- layout.

The elements of a hierarchy are often represented by rectangles (or cuboids in case of 3D visualisation), but common variations include circles and irregular convex polygons. Edges (the connections between the elements in the hierarchy) can be represented explicitly, by drawing a line between the elements, or implicitly, by means of inclusion, overlap or adjacency. The elements of the visualisation are layed-out using either subdivision or packing methods. In subdivided layouts, a certain amount of space is assigned to one element, and this space is then divided and designated to each of the element's children. Packing works in exactly the opposite way: based on specific attributes, the elements are designed with a size and a shape and packed, along with their siblings, into the parent's space.

Hans Jörg Schulz [2011] started a visual bibliography of known tree visualisation techniques. At the time of writing, the bibliography contained 283 techniques categorised by dimensionality (2D, 3D and hybrid), edge representation (explicit, implicit and hybrid) and alignment (axis-parallel, radial or free) [Hans Jörg Schulz, 2015]. Exploring all these techniques is out of the scope of this thesis.

Andrews [2015] categorises hierarchy visualisations by edge representation, differentiating between node-link (explicit) and space-filling (implicit) trees. Based on this categorisation, Andrews [2015, Chapter 5] provides a relatively extensive overview of the most important hierarchy visualisations. These are summarised in the following text using the same categorisation method.

3.2.1 Node-Link (Explicit) Trees

In node-link (or *explicit*) trees, the connections between nodes are explicitly visualised. Walker [1990] designed an algorithm to draw trees in node-link style, which runs in O(N) time, where N is the number of tree nodes. This algorithm was later optimised by Buchheim, Jünger, and Leipert [2002]. As shown in Figure 3.2, the root element is drawn on top of other elements, while children reside in layers beneath the root. The same amount of space is allocated for each child at the lowest level.

A well-known example of an explicit tree hierarchy visualisation is File System Navigator (FSN), shown in Figure 3.3. In FSN, the parent node is represented as a pedestal, and leaf nodes are drawn on top of it. All child nodes are explicitly connected to the parent node. The height of the pedestals conveys the overall size of the contained elements. US patent for this technique was filed on 23 March 1993, and issued on 18 June 1996 [Strasnick and Tesler, 1996]. FSN was made famous in 1993 through its appearance in the movie "Jurassic Park". Harmony Information Landscape (HIL) [Andrews, 1996] applies a similar visualisation approach, but provides some additional features, which are not available in the FSN. While in the FSN only forward and backward motion is possible, HIL supports free navigation through the landscape in all directions. Furthermore, HIL can visualise hyperlink relationships between single items (documents), in addition to hierarchical relationships.

Cone Trees [Robertson, J. D. Mackinlay, and Card, 1991] visualise hierarchies in 3D by ordering all items at the same level of hierarchy in a circle, and connecting them to their parent and their children by a line. This results in a cone-like structure, shown in Figure 3.4, which can be laid out either horizontally or vertically. In Cone Trees, as in many 3D visualisations, the items which reside in the background can be occluded by the items at the same level of hierarchy, which reside in the foreground. This problem is solved by enabling cone rotation, an interaction which can be used to bring items to the foreground. Shadows of cones and nodes are cast on the floor, to provide a 3D depth cue and convey additional structural information about the hierarchy. The US patent for the Cone Tree technique was filed on 21 May 1993, and issued on 15 March 1994 [Robertson, J. Mackinlay, and Card, 1994].

Another radial node-link hierarchy visualisation is the Hyperbolic Browser [J. Lamping and R. Rao, 1994]. Linked items in the Hyperbolic Browser are laid out on a hyperbolic plane, which is then mapped onto the unit disc for display. Such mapping results in more space being allocated to the focused portion of hierarchy, while still displaying the entire context, as shown in Figure 3.5. US patent was filed on 14

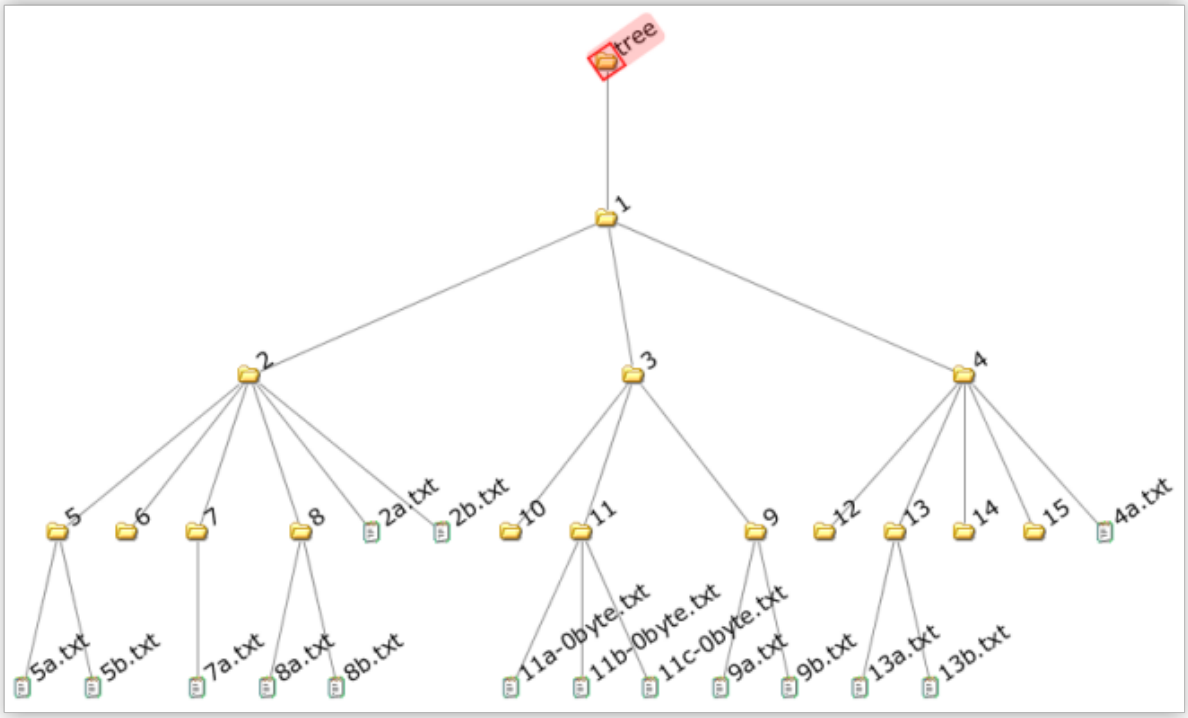


Figure 3.2: Walker Tree [Walker, 1990] layout generated by the Hierarchical Visualising System (HVS) [Andrews, Putz, and Nussbaumer, 2007]. The root element is drawn on top of other elements, while children reside in layers beneath the root. The same amount of space is allocated for each child at the lowest level. [Image used with kind permission of Keith Andrews].

September 1994, and issued on 08 April 1997 [J. O. Lamping and R. B. Rao, 1997]. It won the CHI'97 Great Browse Off.

Munzner and Burchard [1995] developed an algorithm for laying out hierarchy items in 3D hyperbolic space, based on which Walrus [Hughes, Hyung, and Liberles, 2004] was later developed. A similar visual effect is achieved by MagicEye [Kreuseler and Heidrun Schumann, 1999]. In this visualisation, a layout is generated using the Reingold and Tilford [1981] or Walker [1990], which ist then mapped to the suface of a hemisphere.

3.2.2 Space-Filling (Implicit) Trees

Space-filling (*implicit*), tree structures nest child nodes within the space allocated to their parent nodes. The parent-child relationship is implicit, since no explicit edges are drawn. The size of the nested nodes is often mapped to a metric such as size or weight.

A well-known space-filling hierarchy visualisation is the Tree Map, developed by Johnson and Shneiderman [1991]. In the original "slice and dice" tree map algorithm, the root rectangle is first sliced into smaller vertical rectangles whose number corresponds to the number of children in each specific element, after which alternating horizontal and vertical slicing is applied to child nodes. All child rectangles have the same height, but a different width. The width of each element encodes its weight. This space allocation technique supports the ordering of elements, so that child nodes can be ordered alphabetically, for example. The disadvantage of this approach, however, is that at deep levels of the hierarchy the child nodes often degenerate to narrow strips. This problem is solved in a variation of Tree Maps called Squarified Tree Maps and exemplified by Market Maps [Shneiderman and Wattenberg, 2001]. In Sqarified Tree Maps, the aspect ratio of the rectangles is reduced, resulting in child rectangles having differing height and width, as shown in Figure 3.6. This approach is aesthetically more pleasing, but does not

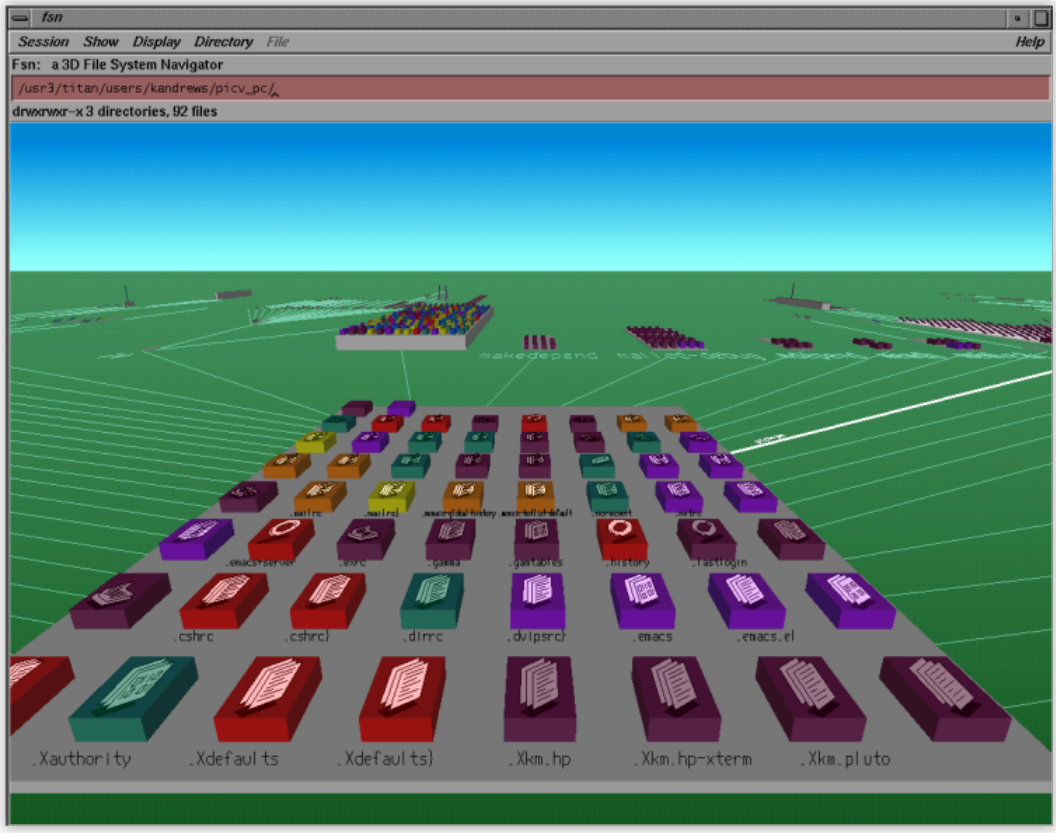


Figure 3.3: File System Navigator (FSN) [Strasnick and Tesler, 1996] providing a landscape visualisation of file system hierarchy. The parent node is represented as a pedestal, and leaf nodes are drawn on top of it. All child nodes are explicitly connected to the parent node. The height of the pedestals conveys the overall size of the contained elements. [Image used with kind permission of Keith Andrews.]

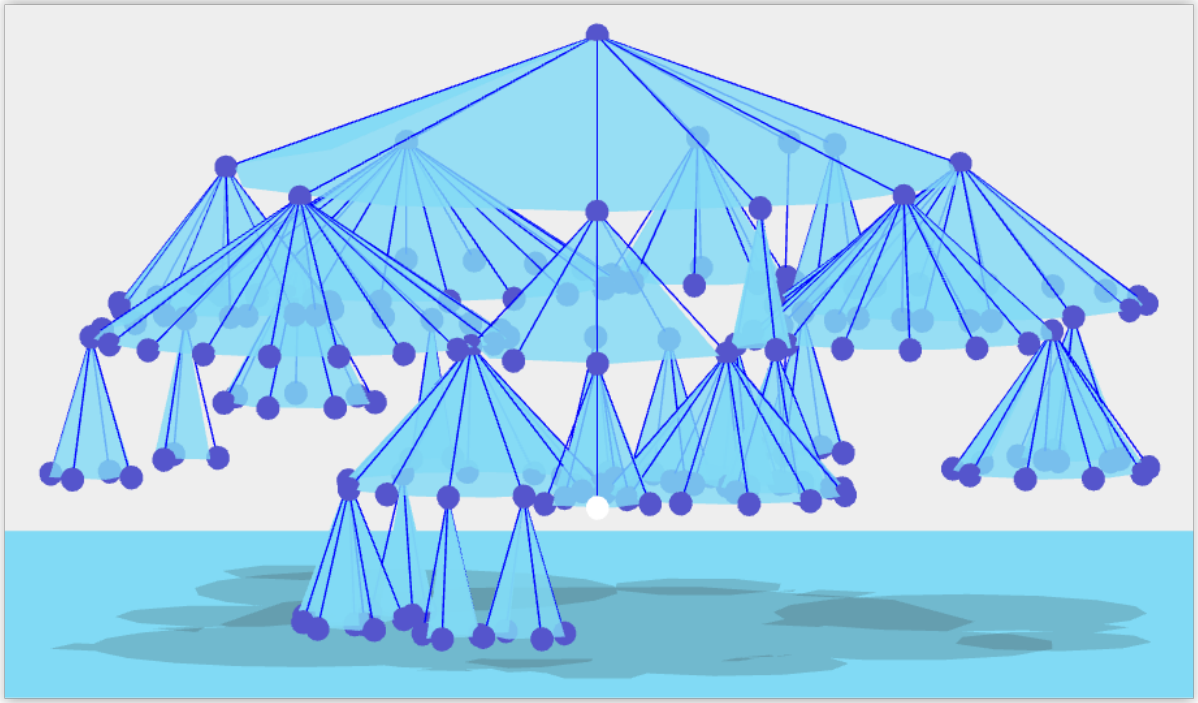


Figure 3.4: Cone Tree [Robertson, J. D. Mackinlay, and Card, 1991] hierarchy 3D visualisation. All items at the same level of hierarchy are ordered in a circle, and connected to their parent and their children by a line, which results in a cone-like structure. Each cone can be interactively rotated in order to bring occluded items to the front. Shadows cast on the floor provide a 3D depth cue and convey additional structural information about the hierarchy. [Screenshot of Fluid Diagrams [Andrews, 2014] Cone Tree visualisation created by the author.]

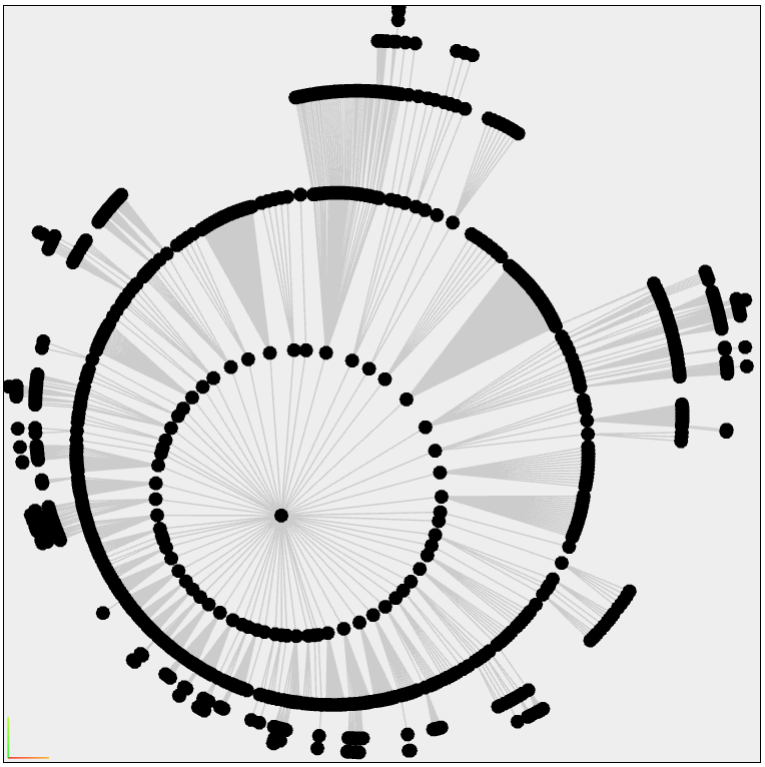


Figure 3.5: Hyperbolic Tree [J. Lamping and R. Rao, 1994] hierarchy visualisation with 3418 nodes. Linked items in the are laid out on a hyperbolic plane. The entire hierarchy is displayed at all times. More space is allocated to the items in focus, while still displaying the entire context. [Screenshot of the Hyperbolic Tree implementation in Fluid Diagrams [Andrews, 2014] created by the author.]

support any ordering of the child items.

In space-filling tree visualisations, space is a valuable asset. Cushion Tree Maps [van Wijk and van de Wetering, 1999] effectively increase the amount of available space, by simply not drawing borders between rectangles. Shading is applied to the margins of the rectangle and the middle is kept bright, which achieves the "cushion" effect.

Information Pyramids [Andrews, Wolte, and Pichler, 1997] is a 3D variation of the tree map technique. In Information Pyramids, the root element is displayed as a plateau upon which smaller plateaus (child elements) reside. This approach results in a pyramid-like structure, which grows upwards as the hierarchy is descended. The size of each plateau is proportional to the weight of its children. Leaf elements are represented by icons in order to encode the type of the element, for example, files or documents.

In the above examples, the tree maps all use rectangles to represent nodes. Info Sky Cobweb Browser [Andrews, Kienreich, et al., 2002], and later Voronoi Tree Map [Balzer, Deussen, and Lewerentz, 2005], are two variations of the tree map technique, which do not divide the available space into rectangles, but use Voronoi subdivision to assign space to child polygons, as shown in Figure 3.7.

Cheops [Beaudoin, Parent, and Vroomen, 1996] uses triangles to represent nodes at a given level of hierarchy. This technique effectively reduces the amount of space necessary to display a portion of the hierarchy by overlapping the nodes. Selecting one of them brings its parent and child nodes into focus.

Radial methods for implicit hierarchical data visualisation have also been proposed. In Information Slices [Andrews and Heidegger, 1998], semi-circular discs are used to represent one or more levels of hierarchy. The number of hierarchy levels depicted on a single disc can be changed interactively. The amount of space allocated to each element at a given level of hierarchy corresponds to its weight. While an overview of a part of the hierarchy is displayed in the left view, the content of the selected segment is displayed in the right view, as shown in Figure 3.8. When a segment in the right view is selected, the disc in left view is displayed as an icon, the disc from the right view moves to the left view, and a new disc opens in the right view.

Sun Burst [Stasko and Zhang, 2000] is an extension of the Information Slices technique which uses full discs instead of semi-discs. The root node is displayed in the centre of the disc, and is surrounded by concentric circles, each of which represents one level of hierarchy. For each element at a given level of hierarchy, a portion of the concentric circle is allocated. The angle of the allocated circle portion, as well as its colour, can be mapped to attributes of the item, such as size or type. Three interaction techniques were developed in order to support examination of smaller, peripheral items, which can appear when one hierarchy level contains many items:

- Angular Detail: animated fan-out interaction causes the disc to shrink and move aside, leaving space to display the contents of the selected item.
- Detail Outside: causes the disc to shrink but stay in the middle, while the elements of the selected item are displayed in a ring around the inner disc.
- Detail Inside: causes the selected item to move to the centre of the disc, displaying it as a root item, with its children displayed in concentric circles around it.

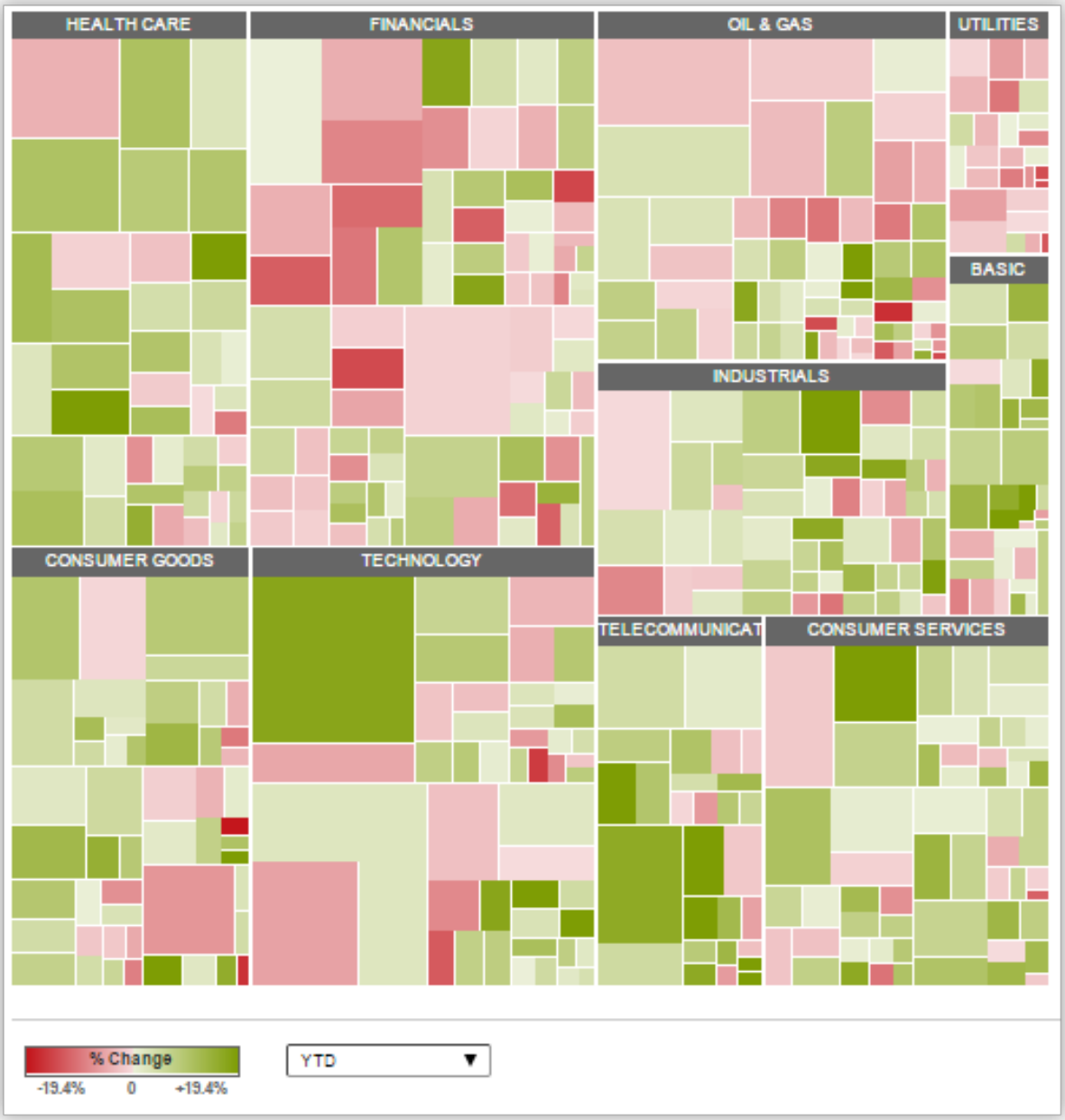


Figure 3.6: Market Map [Shneiderman and Wattenberg, 2001], an example of Squarified Tree Maps. The rectangles representing the hierarchy elements have differing widths and heights. This techniques supports efficient space allocation for deep hierarchies avoiding narrow slices, but items cannot be ordered.
[Screenshot of the MarketWatch [2015] created by the author.]

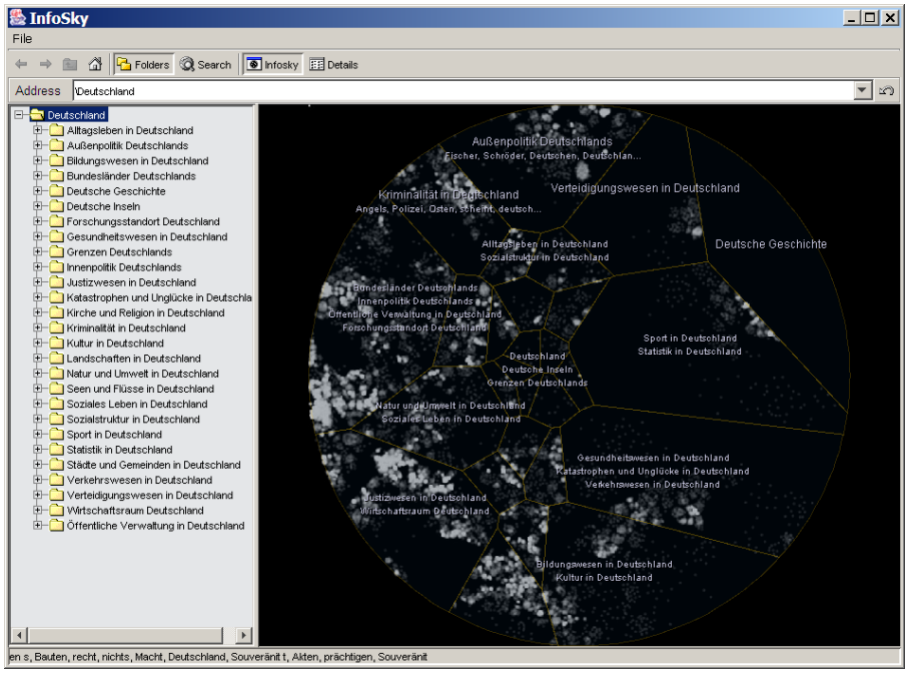


Figure 3.7: Info Sky Cobweb Browser [Andrews, Kienreich, et al., 2002] visualisation of a file system hierarchy. Hierarchy elements are represented by polygons instead of rectangles. The space for child polygons is allocated using recursive Voronoi subdivision. [Image used with kind permission of Keith Andrews.]

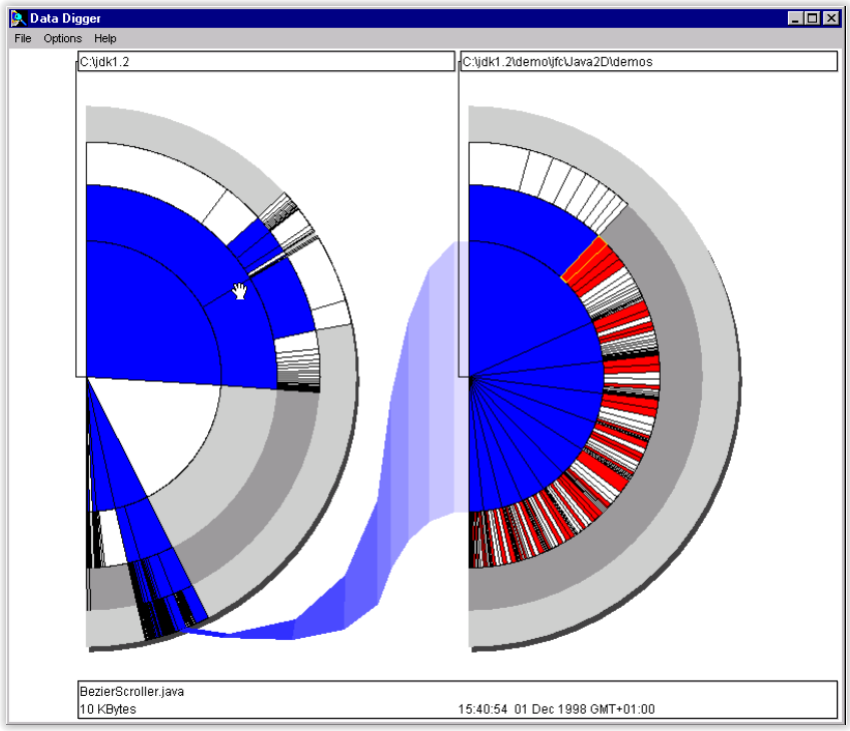


Figure 3.8: In Information Slices [Andrews and Heidegger, 1998], a user-defined number of hierarchy levels is displayed on a semi-circular disc. Selecting a disc segment in the left view displays its content in the right view.
[Image used with kind permission of Keith Andrews.]

3.3 Visualising Multi-Dimensional Data

Multi-dimensional datasets usually come in the form of tables. Each column in a table represents one data dimension, and each row, a data record. Matthew Ward, Grinstein, and D. Keim [2010, Chapter 7] categorise visualisation techniques for multivariate data according to the graphical primitive used in the rendering: points, lines, and regions.

In point-based visualisation techniques, n-dimensional data records are represented as points in k-dimensional space. These techniques include:

- scatter plots and scatter plot matrices,
- force-based methods.

Line-based visualisation techniques join together points which belong to particular records in selected dimensions using straight or curved lines. These techniques include:

- line graphs,
- parallel coordinates and Andrews curves (curved parallel coordinates),
- radial axis techniques (star plots, circular bar charts, circular area graphs, and circular bar graphs).

Region-based techniques rely on filled polygons to communicate the data values through their size, shape, colour and other similar features. Some of these techniques are:

- bar charts or histograms,
- tabular displays (heatmaps, permutations or reorderable matrices, survey plots, and table lens),
- dimensional stacking.

Some features of the above mentioned techniques can be combined into hybrid techniques such as:

- glyphs and icons,
- dense pixel displays (hybrid between point-based and region-based methods).

This section provides an overview of some commonly used techniques: scatter plots, table lens, Chernoff faces (glyphs), and star plots. Scatter plot matrices are given as an example of the small multiples technique. Parallel coordinates are described in more detail in Chapter 4.

3.3.1 Scatter Plots

Scatter plots, or simply XY plots, are used to visualise dataset values in two dimensions, whereby one dimension is mapped to the horizontal axis and the other to the vertical axis. The patterns which arise by plotting the records in two-dimensional space reveal the relationship between the given dimensions [Yale, 1997]. As can been be seen from Figure 3.9a, an upward trend of data points in the scatter plot indicates positive association, whereas a downward trend indicates negative association between the dimensions (3.9b). Another common pattern which can be easily revealed by the scatter plot is clusters of data records (3.9c). If there is no association between the records in two dimensions, the data points will be scattered across the plot, as shown in Figure 3.9d.

When there are a large number of records, the patterns described above usually do not occur in the entire dataset, but are apparent only in specific subsets and are not immediately visible in the display.

Interactive features implemented in most scatter plot visualisations allow exploration of the specific parts of datasets. These features usually include zooming and axis scaling. Zooming allows dynamic selection of a portion of the displayed data points and the display of the enlarged view with both axes automatically scaled to conform to the value range of the displayed data. Axis scaling, on the other hand, enables the selection of scaling of one or both axes. In order to make scatter plots usable with a large number of records, Buering, Gerken, and Reiterer [2006] propose using fisheye distortion which allows focusing on one part of the dataset while viewing the remaining data points in context.

Overlapping points is a common side-effect of visualising large amounts of data with scatter plots. A general approach to solving this problem is making the data points semi-transparent in order to enable the exploration of dataset density. Places in the graph where points overlap can also be marked by drawing one larger point instead of several overlapping ones. A similar effect can be achieved by using different shapes and/or colours to convey information about the density of the data at the given point in the scatter plot [D. A. Keim, Hao, et al., 2010]. To overcome the problem of overlapping points, Ming et al. [2010] propose the combination of binning and zooming. In their variation of scatter plots, the entire plot area is divided into bins based on value ranges of x and y coordinates. In order to increase visibility, bins cluttered by large numbers of overlapping data points are zoomed-in. Additionally, colour is used to visualise the data density in each bin.

Matthew Ward, Grinstein, and D. Keim [2010, pages 238–239] list the following choices for visual analysis of more than two dimensions using scatter plots:

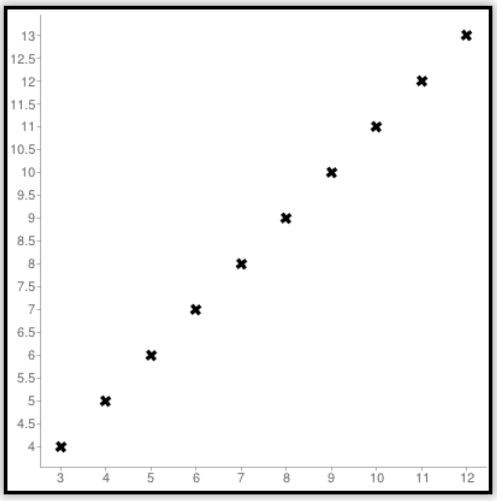
- **dimension subsetting**: allowing selection of dimensions to display or implementing algorithms to identify dimensions containing the most useful information,
- **dimension reduction**: transforming a dataset with high number of dimensions to a dataset with a low number of dimensions using techniques such as principal component analysis (PCA)or multi-dimensional scaling (MCS),
- **dimension embedding**: mapping dimensions to attributes such as colour, size, or shape of the data points,
- **multiple displays**: showing several scatter plots, each of which contains some of the dimensions.

An example of dimension embedding can be seen in Figure 3.10. Multiple displays are discussed in Section 3.3.5.

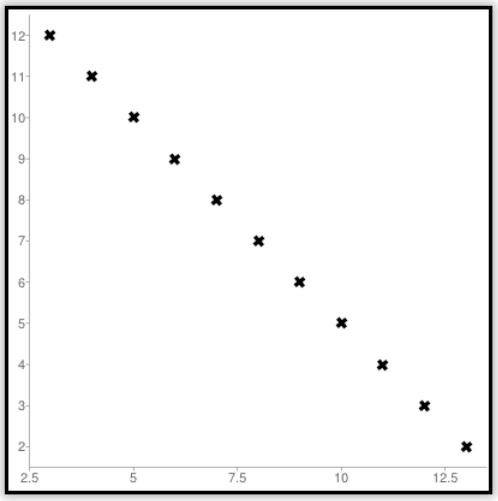
Even with these approaches, scatter plots can visualise only a limited number of dimensions. Owing to this property, they are often used in combination with other visualisation methods as a synchronised view providing more detailed insight into the data being explored. As such, scatter plots are an essential part of many visual data exploration tools.

3.3.2 Table Lens

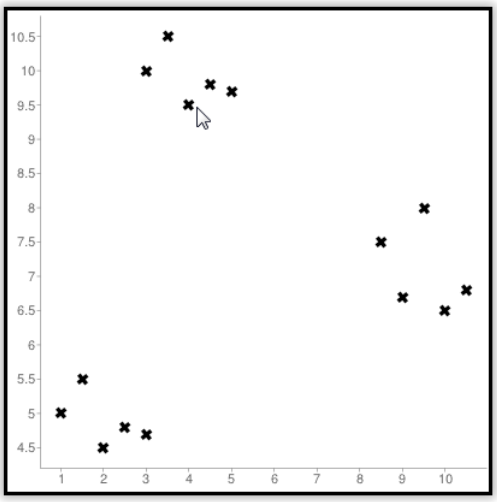
Table lens is a graphical extension to the common table or spreadsheet representation of multivariate data. This visualisation technique was developed in 1994 [R. Rao and Card, 1995] by Xerox PARC. The US patent was filed on 5 March 1996, and issued on 20 May 1997 [R. B. Rao and Card, 1997]. Table lens makes use of a focus-and-context technique to manipulate the display of large tables, enabling users to view more data records in a single screen than is possible with traditional tables. Depending on the number of dimensions and records in the dataset, both columns and rows of the table can be focused or displayed in context. As a result of the view distortion approach implemented in table lens, each cell can be displayed in one of the four states (see Figure 3.11):



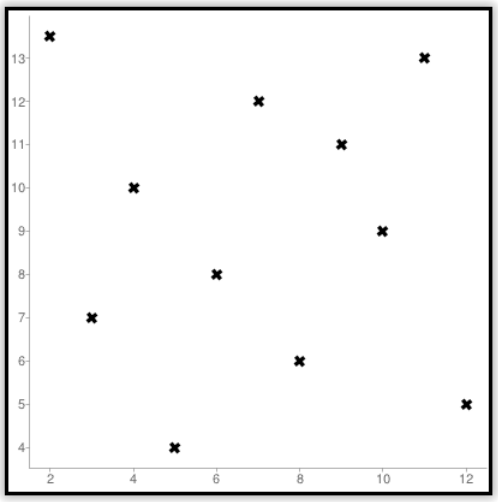
(a) An upward trend in data points indicates a positive association between the dimensions.



(b) A downward trend in data points indicates a negative association between the dimensions.



(c) Clusters of data points indicate groups of records with similar values in these two dimensions.



(d) Scattered data points indicate no association between dimensions.

Figure 3.9: Scatter Plots Patterns

[Images created by the author using the Online Scatter Plot Generator [Alcula, 2009]]

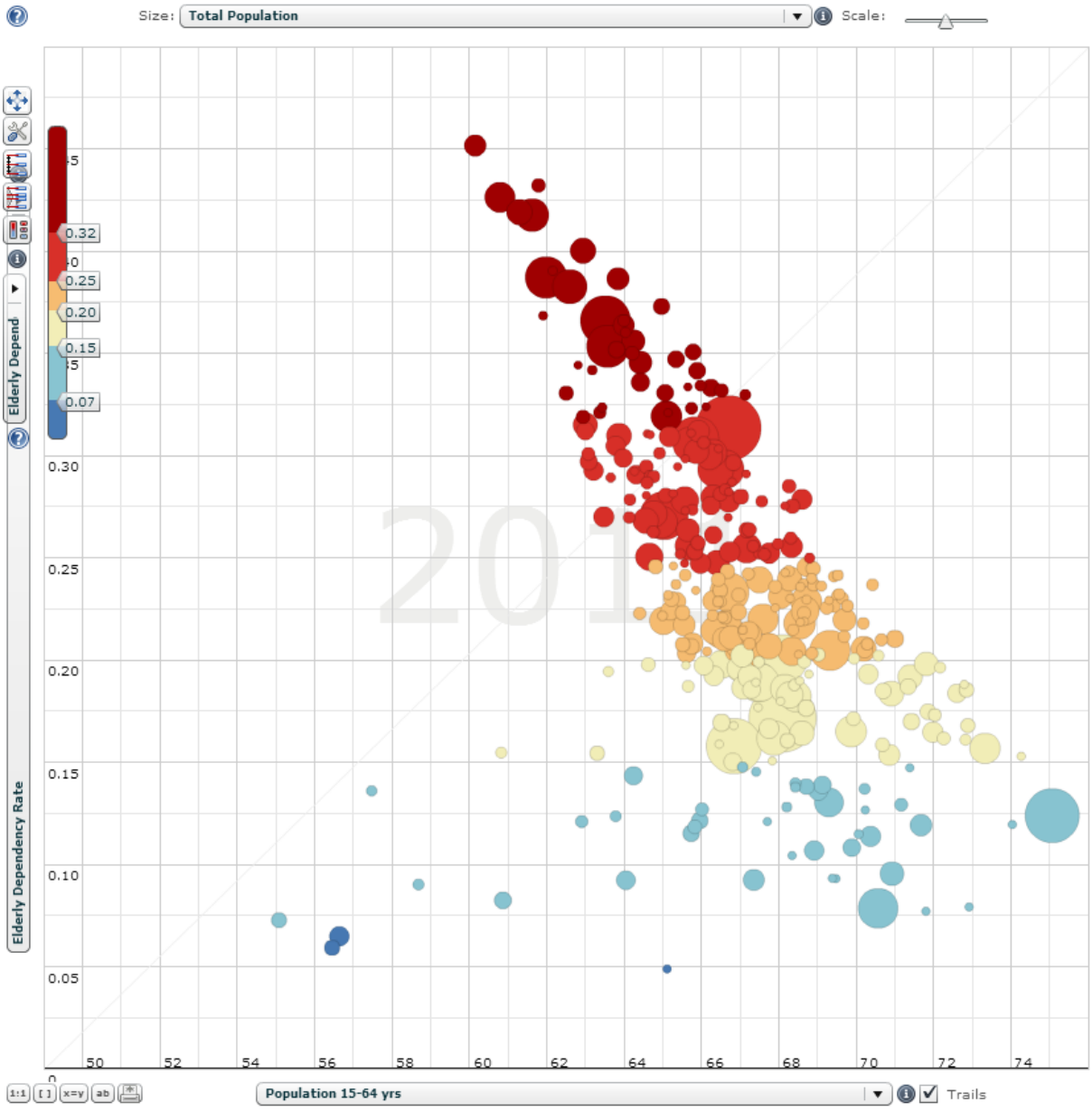


Figure 3.10: Scatter plot implementation in the OECD eXplorer [NComVA, 2014] showing the relationship between the elderly dependency rate and the population of age 15–24 in 2012. Each data point in the scatter plot represents a country in the world. The size of the data point is mapped to an additional dimension (total population), and the colour is mapped to the selected percentile values of the elderly dependency rate. All dimension mappings (both axes, the size, and the colouring) can be selected dynamically by the user. [Screenshot of the scatter plot in the OECD eXplorer [NComVA, 2014] taken by the author.]

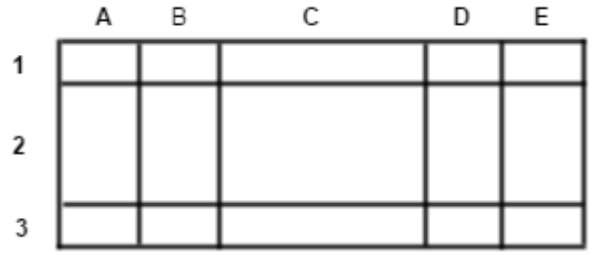


Figure 3.11: The four table lens focal states: focal (C2), column-focal (C1 and C3), row-focal (A2, B2, D2, E2), and non-focal (A1, B1, D1, E1, A3, B3, D3, E3). [Diagram created by the author using Gliffy [Kohlhardt and Dickson, 2015]]

- **focal:** the cell is in focus along both axes,
- **column-focal:** the cell is in focus along vertical axis,
- **row-focal:** the cell is in focus along horizontal axis,
- **non-focal:** the entire cell is in context.

When the cell is not in focus, its value is represented graphically. The graphical representation depends on the value type. Quantitative values are represented by a bar, where the length of the bar indicates the record value in a given dimension. Categorical records, on the other hand, are represented by a shaded, coloured and/or positioned swatch. In column-focal state, the graphical representation of the cell is more detailed then in the non-focal state. If the cell is in focal state, the value of the cell is displayed both graphically and textually.

Rao and Card [R. Rao and Card, 1995] discuss the following three interactive features of the table lens visualisation:

- **Zoom**: increases the entire focus area without changing the number of focused cells,
- **Adjust**: changes the number of focused cells without changing the size of focus area,
- **Slide**: changes the location of the focus area.

An important feature, which any table lens implementation should provide, is record sorting based on the values of the records in a given dimension [Pirolli and R. Rao, 1996]. In order to directly compare the values of records in two (or more) dimensions, the users need to be able to rearrange the columns. These two features along with the creation of new columns based on given formulas are inherited from traditional spreadsheets. Figure 3.12 shows a table lens in which the values of approximately 400 records are visualised in a relatively small area. In this case, focus is enabled only for rows.

3.3.3 Chernoff Faces

Based on the fact that people easily observe changes in facial expressions, Herman Chernoff [Chernoff, 1973] presented a method for representing multivariate data in a k-dimensional space ($k \leq 18$) in which each dimension of the dataset is mapped to an individual part of a human face. As shown in Figure 3.13, eyes, eyebrows, nose, mouth, ears, and other parts of a human face, are used in Chernoff faces to represent dataset dimensions. Different shapes, sizes, placements, and orientation of individual face parts represent different values of records in given dimensions. For each data record, a new face is drawn, as shown in Figure 3.14. Chernoff argued that given a sequence of faces, the user could easily recognise clusters

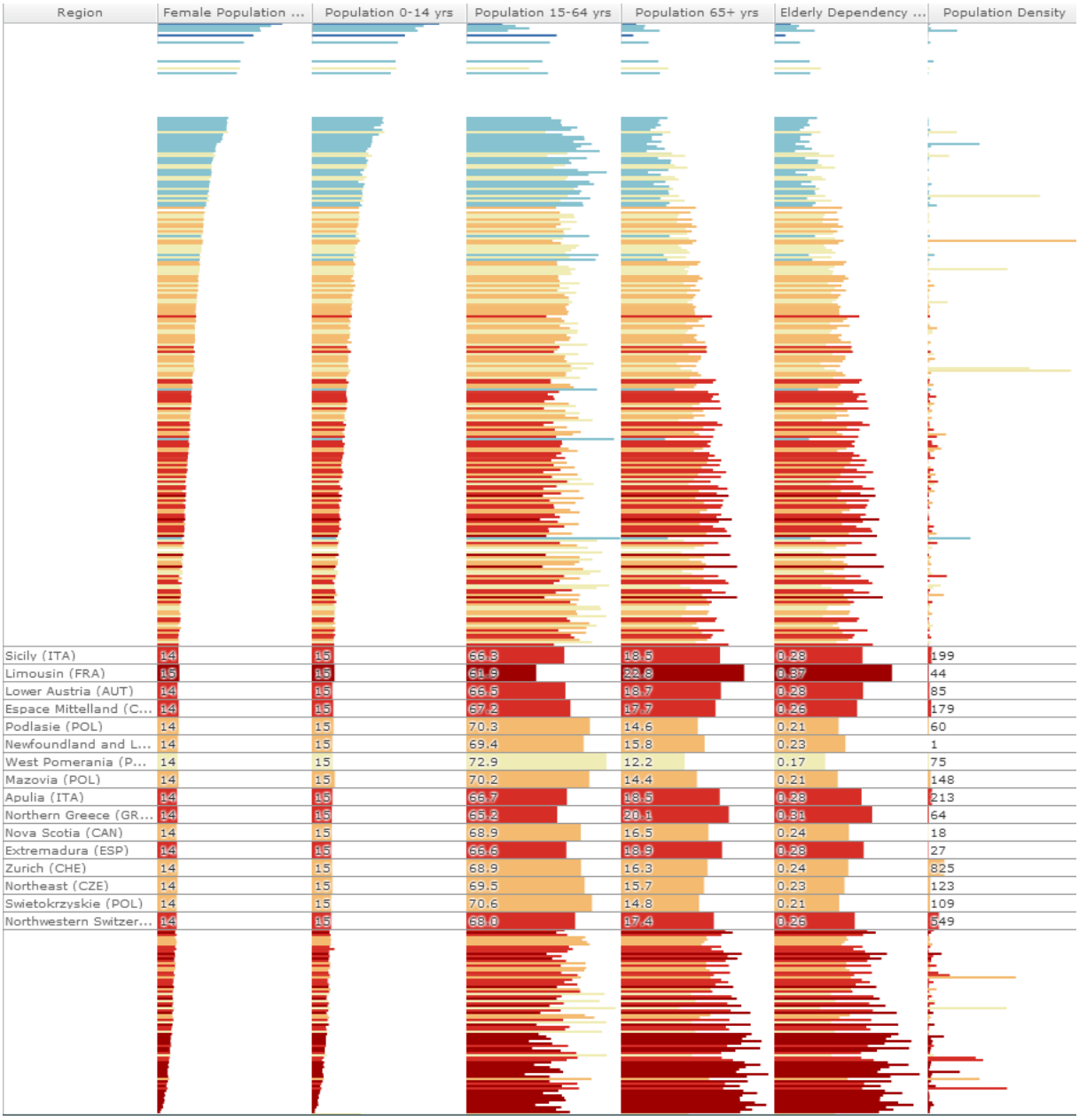


Figure 3.12: The table lens implementation in the OECD eXplorer [NComVA, 2014], showing 7 dimensions of statistical data for approximately 400 different countries in the world. The values of 16 records, which are in focus, are shown as common table cells with a bar in background, whose length provides additional information about the record value in a given dimension. The values of all other records, which are not in focus, are represented only by short bars. The colour of bars is a mapping to percentile values of the elderly dependency rate. This mapping can be changed by the user in a separate, connected view (not shown in this image). In this implementation, the rows can be in either focal, or non-focal state, while the columns are always in focal state. [Screenshot of the table lens implementation in the OECD eXplorer [NComVA, 2014] taken by the author.]

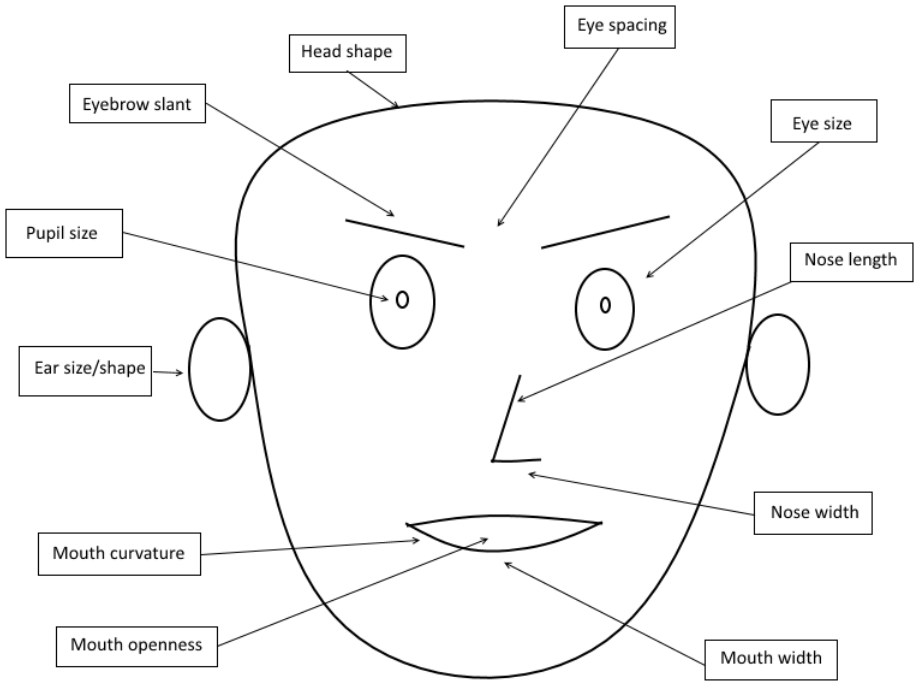


Figure 3.13: Mapping data to different parts of a human face. [Image created by the author.]

Figure 3.14: Three data records represented as Chernoff faces. The dimensions of the dataset are mapped to eyebrow slant, length of nose, and mouth openness. By observing changes in these facial attributes, the user can see how the values of the data records change in each dimension. [Diagram created by the author using paint.net [dotPDN LLC, 2015].]

of points by grouping together similar faces. Apart from recognising clusters, Chernoff suggestged that given a sequence of faces representing different variable values in time, users could easily recognise significant changes in variable values based on eye-catching facial transformations.

In order to test the effectiveness of features in Chernoff faces, Morris, Ebert, and Rheingans [2000] conducted a user study in which they tested faces with the following four features:

- small eyes,
- a specific face,
- inwardly slanting eyebrows, and
- a combination of small eyes and inwardly slanting eyebrows.

The study investigated how easily faces with these distinctive features are recognised by users when placed among 5, 10, 25 or 50 other faces with different features. None of these features were perceived

pre-attentively. However, for longer viewing time (two seconds), eyebrow slant and the eye size were perceived more easily than the other features.

Since different parts of a human face are perceived differently, one should carefully select which dimensions are mapped to which facial features when visualising multivariate data with Charnoff faces. Longer viewing time is necessary to recognise patterns even in small collections of Chernoff faces, suggesting that this technique might not always be effective.

3.3.4 Star Plots

A star plot, also known as a radar chart, spider chart, cobweb chart, irregular polygon, polar chart, or kiviat diagram [Wikipedia, 2014] is a multivariate data visualisation technique similar to parallel coordinates. Instead of placing the axes in parallel, a minimum of three axes are ordered radially at an equi-distant angle with the same point of origin. The records in the dataset are drawn as polylines connecting each axis. The point where the segment of a polyline meets an axis is the value of the record in that dimension. The closed polygonal line connecting all axes sometimes looks like a star, hence the name "star plot".

In order to accurately compare the values of records across different dimensions, the user needs to compare the lengths of the lines, which are originating from the centre of the plot and ending at the point where the record's value is mapped on the axis. However, since the axes are ordered radially and not placed in parallel to one other, comparing the lengths of these lines is not easy. Furthermore, connecting the polyline segments on radially ordered axes creates polygonal shapes, which encourages comparison of areas enclosed by these shapes, rather than the line lengths. Larger areas in star plots indicate higher data record values. However, the size and the shape of the areas depend on the scaling of the axes. Most datasets do not have the same units and value range on every dimension, so visualising such datasets with star plots can make it difficult to accurately read the values of some of the dimensions, as shown in Figure 3.15. Another problem with star plots is that the overall visual impression is highly dependent on the ordering of the dimensions, because different dimension orderings form different polygonal shapes.

A common variation of the star plot is created by shading or filling the polygonal area. In this case, the user is supposed to compare the areas. However, accurately comparing areas is much more difficult than comparing line lengths. In Figure 3.16, a star plot, filled star plot, and bar chart are used to visualise the same dataset. This example shows that comparing lengths of bars in bar charts is much easier than comparing lengths of lines in a star plot, or areas in a filled star plot.

3.3.5 Small Multiples

Small multiples is a technique in which several small visualisations of data are displayed side by side at the same time in order to allow direct comparison of value changes in different slices of the dataset. Tufte [2007, page 170] compares small multiples to frames of a film, describing them as:

> "...a series of graphics, showing the same combination of variables, indexed by changes in another variable".

The essential part of the small multiples idea is consistency, which is achieved by using the same context in each representation. The representation itself can be any form of map, chart, or graphic but its size and scaling (that is, the context) must remain the same.

Tufte [2007, page 93] suggests that a large share of ink in a graphic should represent data and that the changes in ink should correspond to the changes in data. This concept is called "data-ink". Tufte defines properties of well-designed small multiples as [Tufte, 2007, Page 175]:

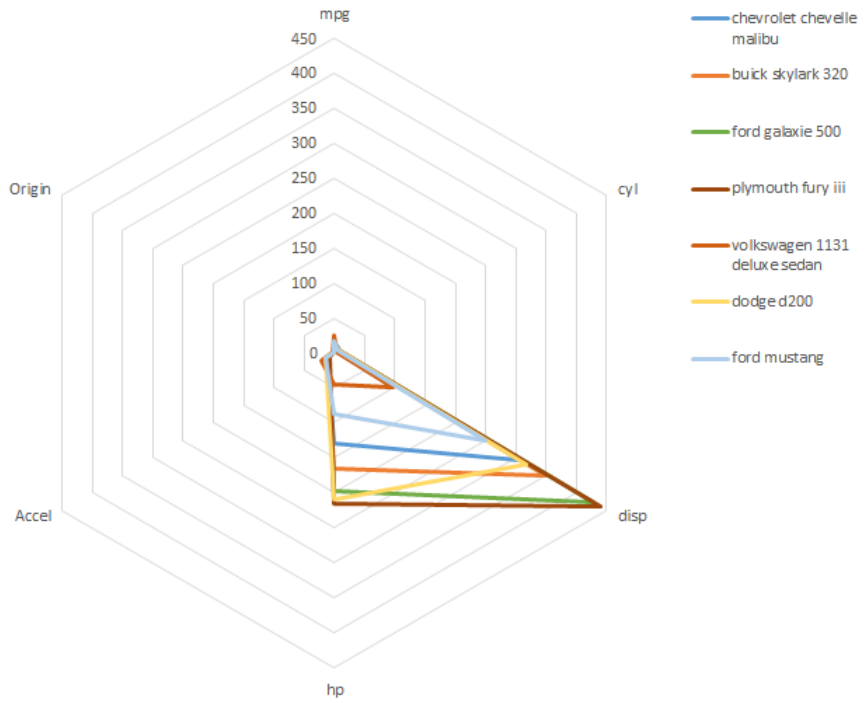
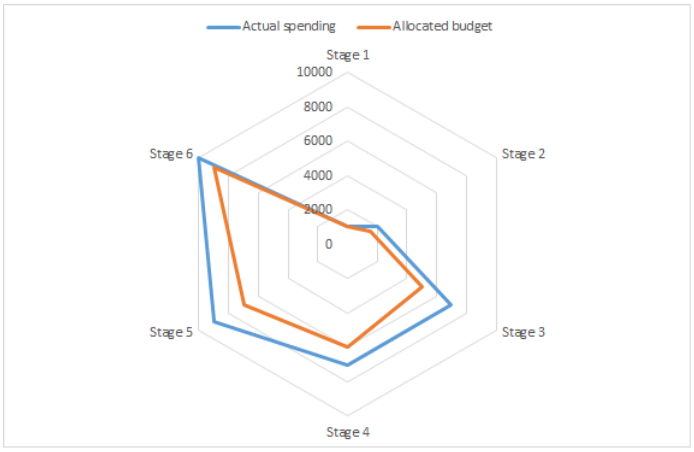
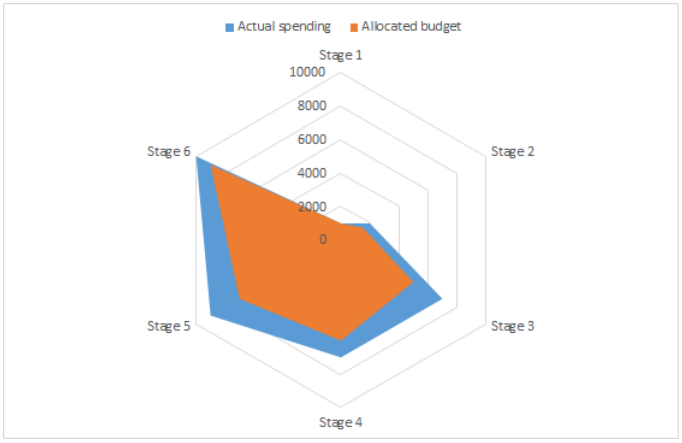


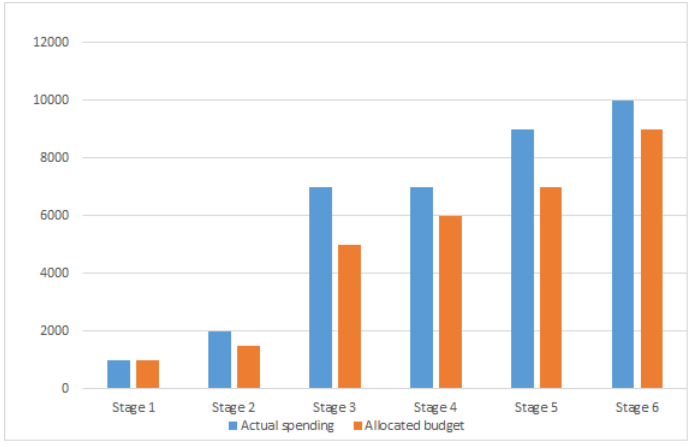
Figure 3.15: Star plot visualisation showing 6 dimensions of data (mileage per gallon, number of cylinders, displacement, horse power, acceleration, and origin) for 7 different cars. All axes have the same scaling, which is in this case dictated by the "displacement" dimension, whose maximum record value is 450. This makes it almost impossible to accurately read the values of other dimensions, such as the number of cylinders, whose maximum value is 8. Furthermore, closed polygonal lines formed by the radial axis ordering encourage users to compare the areas instead of line lengths on each axis, which might lead to an inaccurate interpretation of the plot.
[Chart created by the author using Microsoft Excel 2013 [Microsoft, 2015b] based on Auto MPG Data Set [Lichman, 2013].]



(a) Star plot.



(b) Filled star plot.



(c) Bar chart.

Figure 3.16: Three visualisations of allocated budget and actual spending in seven project implementation stages. [Images created by the author using Microsoft Excel 2013 [Microsoft, 2015b].]

- inevitable comparative,
- deftly multivariate,
- shrunken, high-density graphics,
- usually based on a large data matrix,
- drawn almost entirely with data-ink,
- efficient in interpretation, and
- often narrative in content, showing shifts in the relationship between variables as the index variable changes.

A common usage of small multiples are scatter plot matrices. By always displaying the same two dimensions on one horizontal and vertical axes on all scatter plots in the matrix, while changing the dimensions on the other axis, data values in different dimensions can be directly compared. Similarly, if more than two dimensions are displayed in a scatter plot (by mapping their values to colour, size of data points and other visual attributes) by keeping the same dimensions on all scatter plots the effects of changing the values of an additional dimension can be observed. An example of such scatter plot matrix is shown in Figure 3.17.

Scatter plot matrices are only one example of a small multiples view. Almost any kind of visualisation can be displayed in the form of small multiples. However, the detail in such displays should be limited, in order to make changes in the dataset visible.

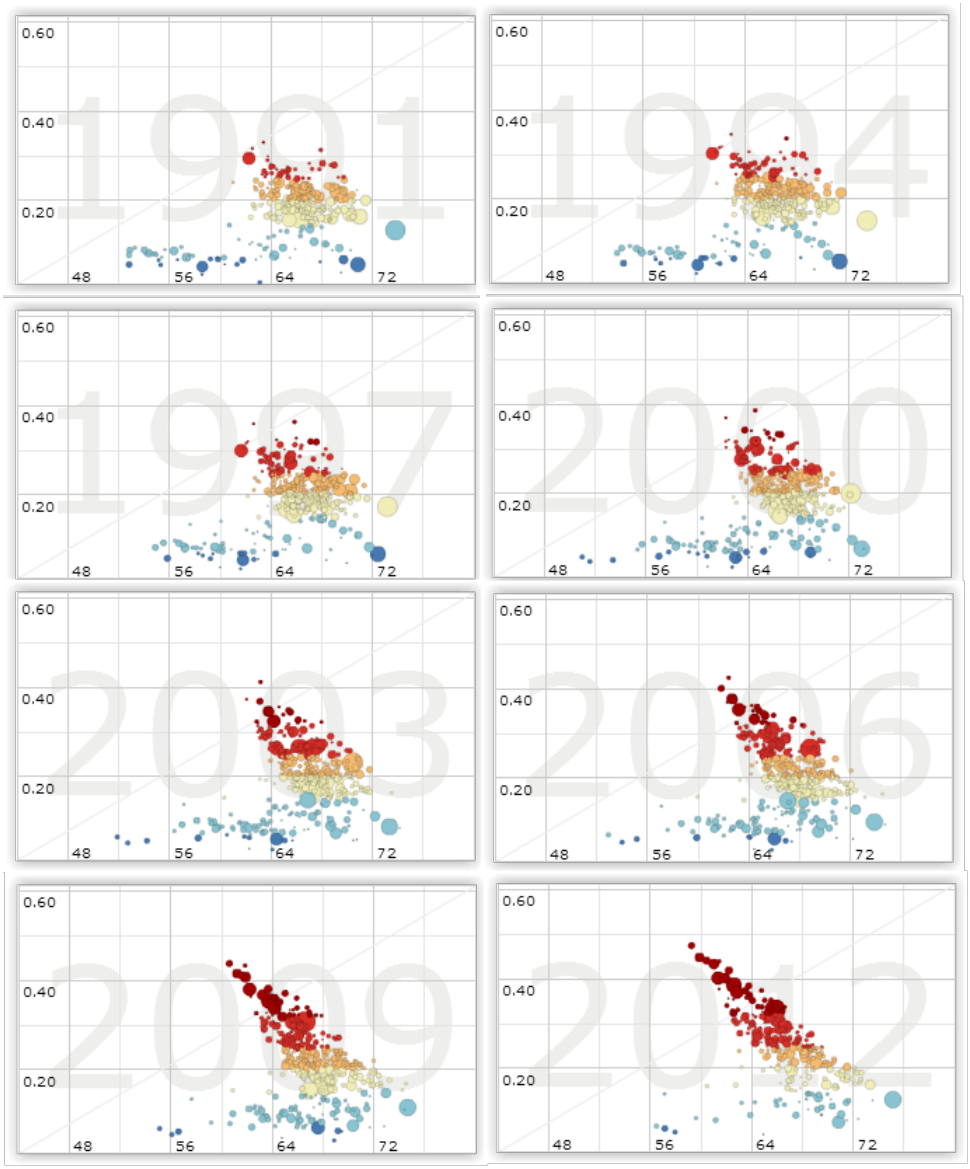


Figure 3.17: Scatter plot matrix showing the changes in the elderly dependency rate (y axis) of the population at ages 15–64 (x axis) over time. The data is available every three years in the time span between 1991 and 2012. The context in all scatter plots (dimensions mapped to the x and y axes, as well as colour and size of the data points) remains the same, only the year varies. [Image created by the author based on screenshots taken from OECD Regional eXplorer [NComVA, 2014]]

Chapter 4

Parallel Coordinates

"If you torture the data long enough, it will confess."

[Ronald Coase, Economist]

The first known use of parallel instead of orthogonal axes stems from the French mathematician d'Ocagne [1885] in his book "Parallel and Axial coordinates. A geometric transformation method and a new graphical calculation procedure derived from the consideration of parallel coordinates". However, d'Ocagne discussed only the mathematical and theoretical properties of parallel coordinates. As a technique for visualising multi-dimensional data, parallel coordinates were first suggested by Alfred Inselberg in 1959 [Inselberg, 2010, page xxiii], but the first research on this visualisation technique was not done until 1978.

In parallel coordinates, dimensions of a dataset are visualised as axes placed in parallel to one other, whereas records are visualised as polygonal lines (polylines) whose segments span between the axes. The position of a polygonal line on the axis maps the value of a record in a given dimension. Since the axes are placed in parallel to each other, an arbitrary number of dimensions can be visualised.

Positive correlations between two neighbouring dimensions in a dataset visualised with parallel coordinates cause polyline segments to be drawn between the axes with no (or few) crossings. Negative correlations, on the other hand, cause polyline segments to cross in the middle. This can be seen in Figure 4.1. Note that dimensions can only be compared if they are placed next to each other.

4.1 Common Interactive Features and Extensions

Interaction with the visualisation is an essential part of every exploratory analysis. This section provides an overview of the most common ways to filter, compare, and explore the details of a dataset visualised with parallel coordinates. Furthermore, common extensions to the standard implementations are described:

- **Record Selection (Brushing)**
  Record selection interactions enable users to select a single record, or a specific subset of records in the dataset. The terminology regarding such interactions is somewhat inconsistent in the literature. Inselberg [2010] calls such interactions "querying". The term "brushing" is also used [Hauser, Ledermann, and Doleisch, 2002]. For consistency purposes, in the following text, the term "record selection", or simply "selection" will be used to refer to these interactions.

  In most parallel coordinates implementations, one record in the dataset can be highlighted (temporarily selected) simply by hovering over it with the mouse. When a single data record is highlighted, a label is usually displayed on each axis indicating the value of the selected record in that

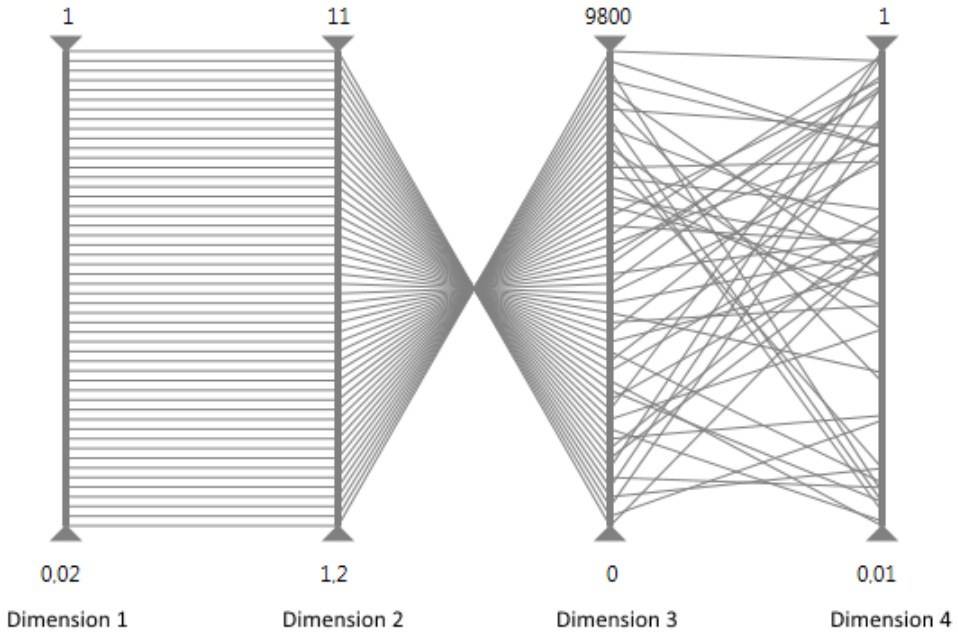


Figure 4.1: Line segments which are largely parallel indicate a positive correlation between two neighbouring dimensions, such as here between the first and the second axis. Crossings, such as those between the second and the third axis, indicate negative correlation between the two dimensions. Both parallel and crossing segments are present between the third and fourth dimension indicating no obvious correlation, although subsets of the data may be correlated. [Image created by the author.]

dimension. In most intuitive implementations, several records can be selected by left-dragging a rectangle with the mouse over the area containing the records of interest. A single left-click selects a single record and deselects all other records. A control-left-click toggles the selection of a single record, adding it to, or removing it from the set of the currently selected records. A shift-left-click adds a single record to the current set of selected records. The set of selected records is usually displayed in a different, prominent colour. Sometimes, sets of records can be named, assigned a colour, and saved for later use.

Hauser, Ledermann, and Doleisch [2002] present two additional techniques: angular selection and smooth selection. With angular selection, the user selects the records of interest by specifying the slope of the polyline segments between two axes. Smooth selection enables selection of records in datasets based on the degree of interest. The user explicitly defines a record in a dataset which is the centre of interest and this record is displayed in a specific colour. Based on spatial distance to this record, other data records are drawn using the same base colour while applying different alpha values. The closer a record is to the centre of interest, the higher its alpha value.

- **Record Filtering**
  Filtering interactions enable users to filter out dataset records which are currently not of interest. A common way of implementing this interaction is adding sliders at the top and bottom of each axis. By moving these sliders, the user specifies the value range of records which are of interest. All other records are usually deactivated or disabled. Disabled (deactivated) records are not removed from the display, but are grayed out and displayed in the background, as shown in Figure 4.2.

  The records in the dataset can also be filtered by changing the axis scaling. In most implementations, the axes are initially scaled so that one end is mapped to the maximum record value in that dimension and the other one to the minimum. By increasing or decreasing the scaling of an axis, the records are either removed or added to the display. A common way of implementing this

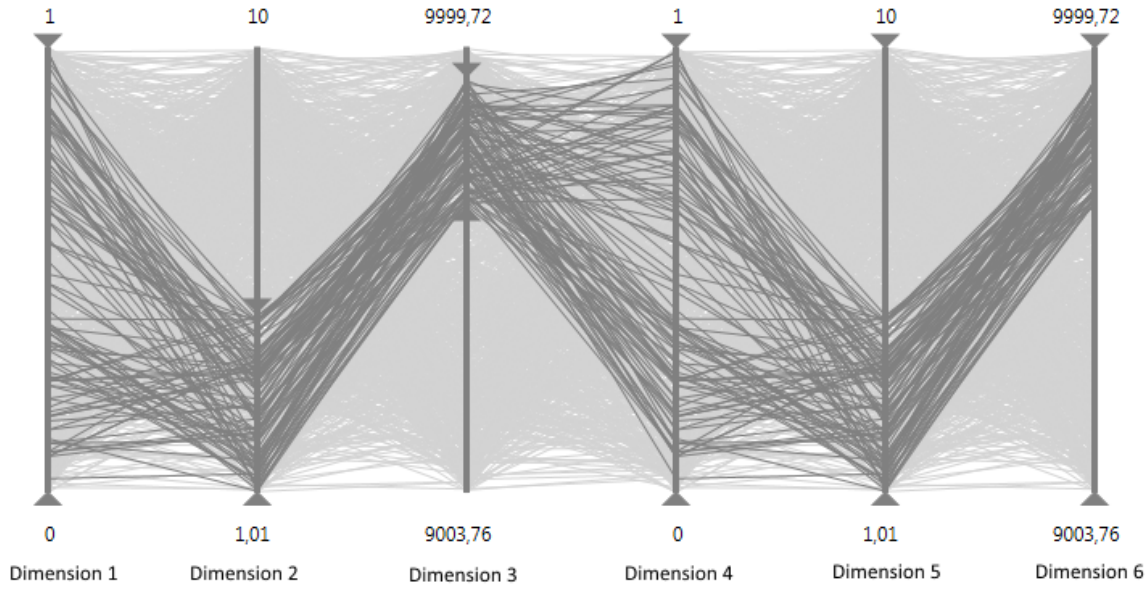


Figure 4.2: A parallel coordinates visualisation of a dataset with 6 dimensions and 1000 records filtered using sliders at the top and bottom of the axes to display the records within the specified value range. The disabled records are still visible, but are greyed out and in the background. [Image created by the author.]

feature is to enable manual editing of the axis range by directly editing the range labels.

- **Inverting an Axis**
  In Figure 4.1, the bottom of the axes map the minimum value of the dataset, and the top of the axes map the maximum value. This ordering is arbitrary, and in most implementations each axis can be inverted to reverse the order of records. This feature can be useful for dimensions where smaller numerical values are considered "good". For example, if car acceleration is expressed in the number of seconds until a car reaches a certain speed, a smaller number of seconds means higher acceleration. Inverting such axes provides consistency in interpretation, with "good" values at the top of each axis.

- **Re-Ordering Axes**
  Comparison between dimensions is only possible on neighbouring axes. For this reason, most parallel coordinates implementations give the users the possibility to select their own axis ordering. Intuitive implementations allow grabbing an axis from one position and then dragging and dropping it to a new position.

  All permutations of axis ordering, in which every axis is adjacent to every other axis, can be computed by representing the axes as vertices of a graph, and finding all distinct Hamiltonian paths [Math Images, 2015], as shown in Figure 4.3. For an even number of axes ($N = 2M$), the minimal number of permutations containing adjacencies for all pairs of axes is M. For an odd number of axes ($N = 2M + 1$), the number of such permutations is $M + 1$ [Inselberg, 2010, page 389]. Some tools are able to compute these permutations and display them.

- **Adding and Removing Axes**
  Dynamically adding and removing axes is another common feature in parallel coordinates, which is usually implemented by adding or removing a dimension from a list of displayed axes and refreshing the parallel coordinates view. Some implementations even allow the addition of the same axis several times, enabling users to compare one dimension with more than two other dimensions in the same view. Furthermore, new axes can be added to the view by combining two or more

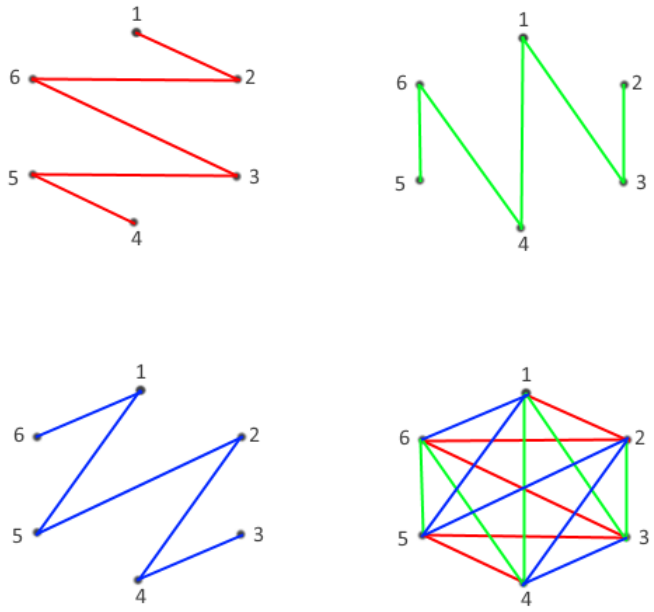


Figure 4.3: Six axes of a parallel coordinates plot are enumerated and represented as vertices in a graph. There are three distinct Hamiltonian paths in such graph. The union of all distinct Hamiltonian paths is a complete graph, in which every vertex is connected to every other vertex. For 6 axes, the following three axis orderings (distinct Hamiltonian paths) allow direct comparison between every axis in the plot: 126354, 231465, and 342516. [Image created by the author, based on figure 10.8 in Inselberg [2010].]

currently displayed dimensions. For example, dimensions A and B can be combined by computing $C = \sqrt{A + B}$, thus yielding a new dimension.

- **Displaying Histograms on Axes**
  As shown in Figure 4.4, histograms can be dynamically added to the axes in order to convey information about the frequency of records in particular parts of the dataset. The number of histogram bins can usually be selected by the user.

- **Zooming**
  An important interactive feature is zooming in and out of specific parts of the parallel coordinates visualisation. This allows detailed exploration of particular parts of the dataset and enables easier record selection in larger datasets.

- **Mean Line**
  To visualise the general data trend a mean value is calculated for each dimension. The mean line is the polyline connecting the mean values of each dimension. It is usually made visually distinctive from other polylines and can be dynamically turned on or off.

4.2 Handling Large Datasets

The usefulness of a parallel coordinates visualisation decreases with the increasing number of records in the dataset, because the view tends to become cluttered by overlapping polylines. A simple approach to solving this problem is to make the polylines semi-transparent. As can be seen in Figure 4.5, decreased polyline opacity makes areas with many overlapping segments appear darker, thus forming visual clusters, which reveal patterns in the dataset. However, this approach does not work for an arbitrary number

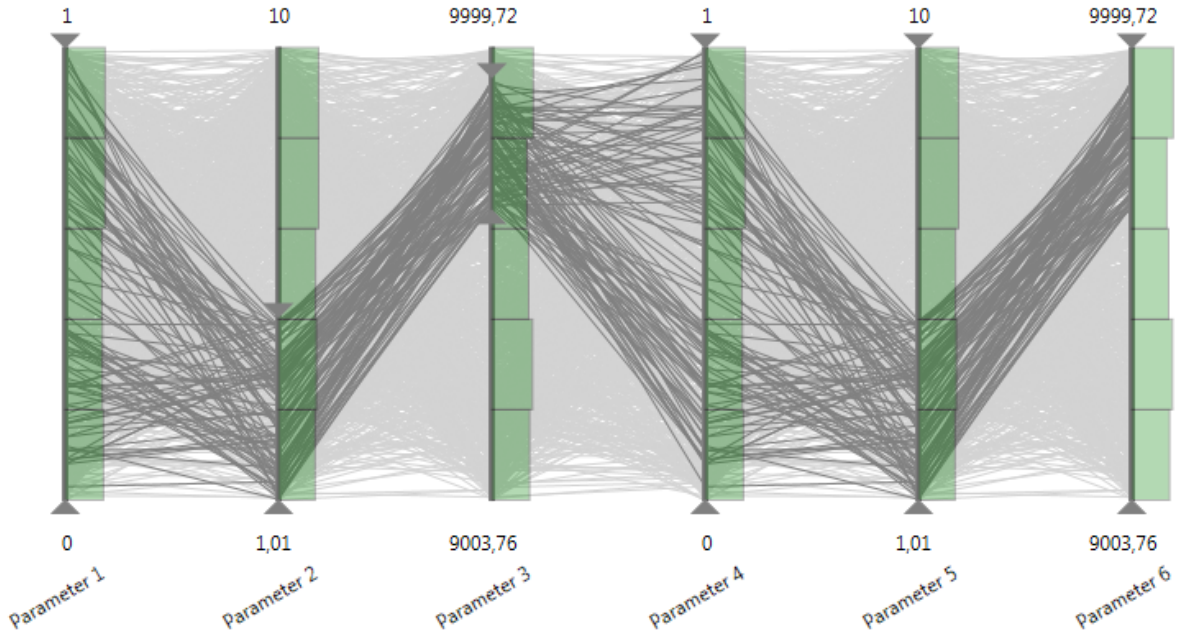


Figure 4.4: A parallel coordinates with histograms added to each axis to show the distribution of record values in each dimension. [Image created by the author.]

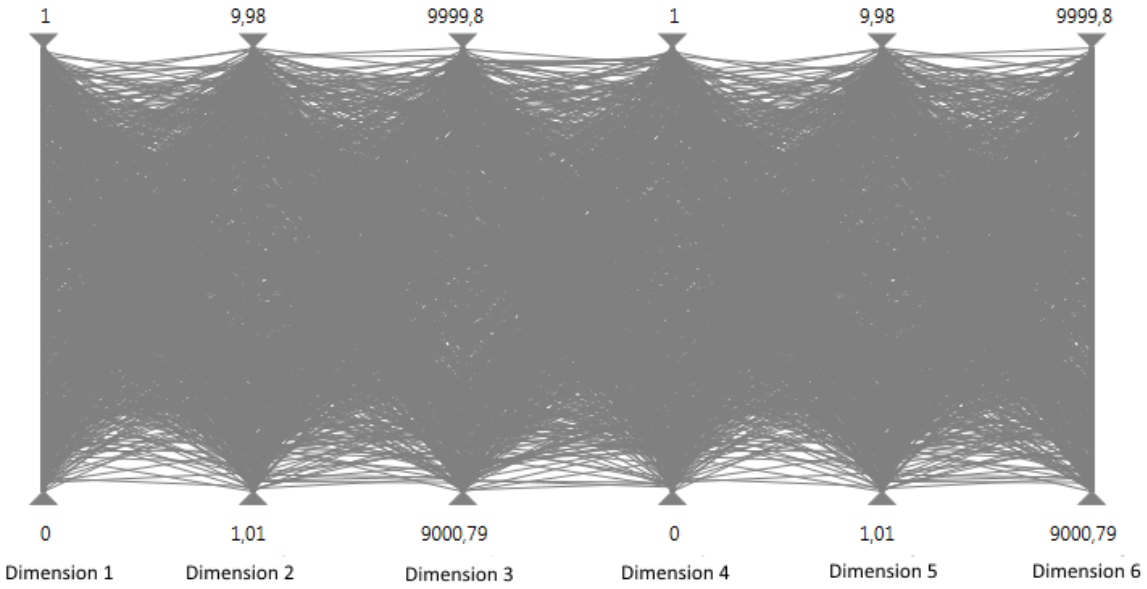
of records, and the view will still become cluttered given a large enough dataset. One might argue that the size of the dataset which can be effectively represented with parallel coordinates depends on the available screen size, but even on larger screens the view can become cluttered when displaying a few thousand records. Furthermore, large datasets tend to cause performance problems making filtering and interaction with the data very difficult.

One way of dealing with large datasets is drawing only a part of the dataset. Ellis and Dix [2006] discuss techniques for selecting random samples of the input data. However, a more common approach is reducing clutter by grouping similar records together either visually or numerically (in the background). Based on this idea, several different solutions have been proposed over the years. They differ not only in the algorithms applied to group similar records together, but also in the way the grouped records are displayed.

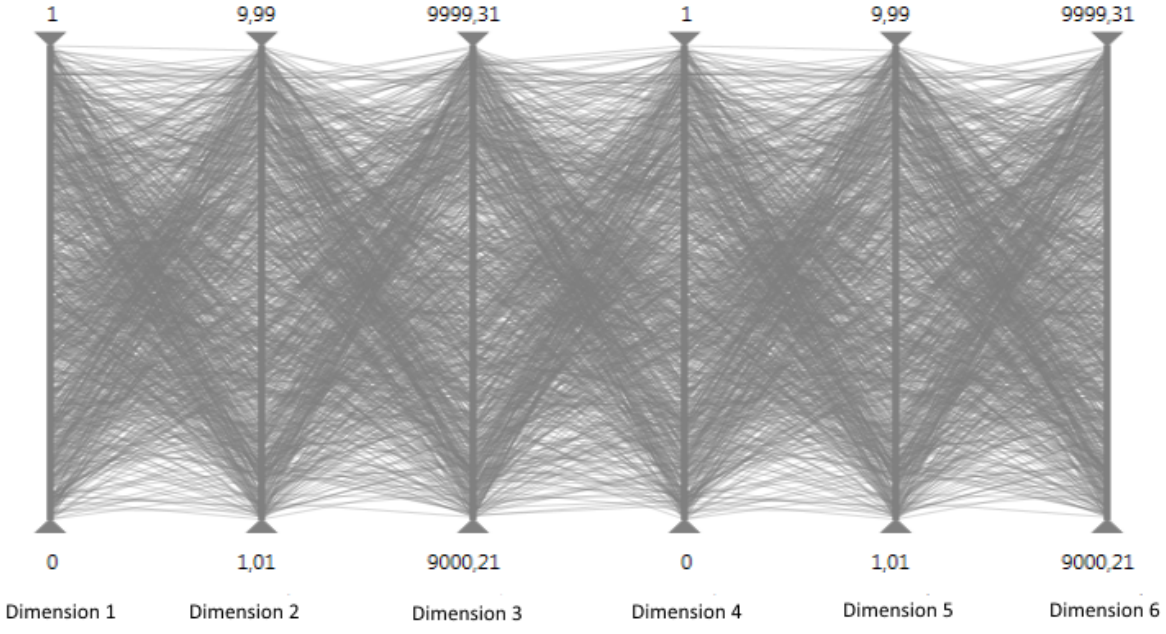
Ying–Huey, M.O. Ward, and Rundensteiner [1999] propose deriving a hierarchy of nested clusters, based on which records are visualised in different levels of abstraction. The similarity between records is measured by calculating the proximity between pairs of objects, after which a tree of nested clusters of objects is built. Each depth level contains a single level of abstraction. The clusters are represented in parallel coordinates as bands with high transparency at the top and bottom edges, which decreases toward the middle of the band, as shown in Figure 4.6. The width of the band indicates the number of records in the cluster for a given dimension.

Records in parallel coordinates do not necessarily have to be represented as polylines. Edge bundling is the concept of drawing similar edges curved and bundled together in order to reduce visual clutter. Zhou, Panpan, et al. [2013] divide the edge bundling techniques into three categories: cost-based, geometry-based, and image-based. One of the cost-based techniques is energy minimisation, which works by reducing the overall energy of a mathematical system. Zhou, Yuan, et al. [2008] applied this technique to parallel coordinates by modeling the polylines as flexible springs and applying attractive forces between them. Similar records are clustered together based on geometrical rather than numerical similarity, so polylines which are visually parallel and close to each other are curved and grouped together, as shown in Figure 4.7.

Apart from edge bundling, McDonnell and Mueller [2008] discuss a variety of different rendering



(a) Overlapping polylines with full opacity clutter the display.



(b) Polylines with 30% opacity make the display appear less cluttered.

Figure 4.5: Making polylines semi-transparent is an effective way to visualise the density of data in a parallel coordinate visualisation. [Image created by the author.]

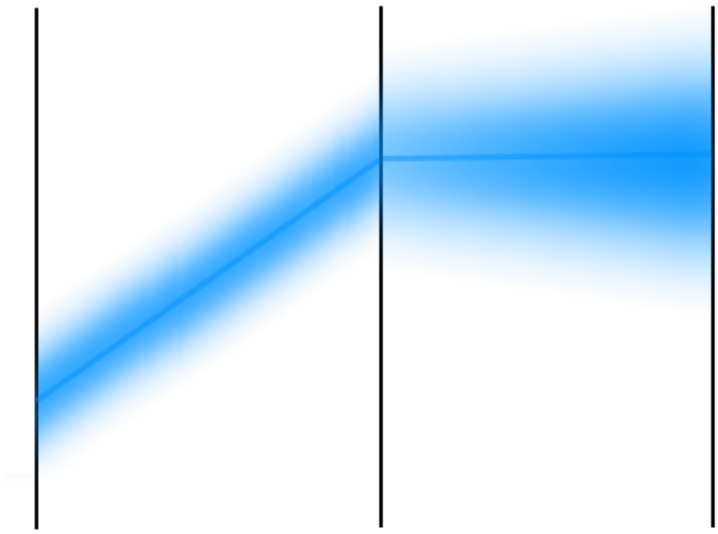


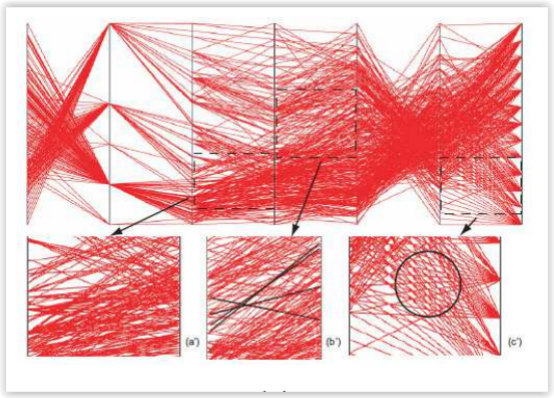
Figure 4.6: A cluster in hierarchical parallel coordinates is displayed as a band with high transparency at the top and bottom. The width of the band indicates the number of records in the cluster for a given dimension. [Image created by the author, using paint.net [dotPDN LLC, 2015], based on Figure 3 in Ying–Huey, M.O. Ward, and Rundensteiner [1999].]

techniques to improve the parallel coordinate visualisations, such as: spline-based cluster rendering, branched clusters, and using silhouettes, shadows and halos, as well as density plots to convey the distribution of the data within the clusters. Artero, de Oliveira, and Levkowitz [2004] discuss using density and frequency plots to uncover clusters in crowded parallel coordinates visualisations.

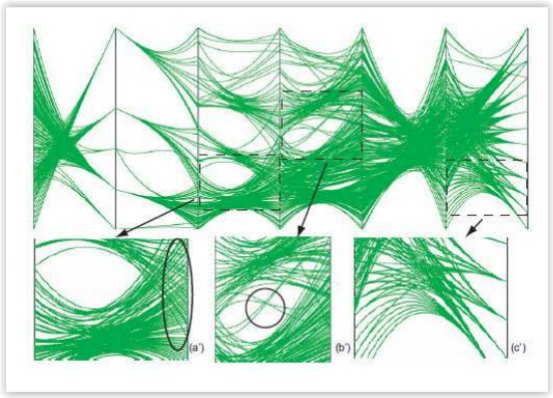
G. Andrienko and N. Andrienko [2004] propose enhancing visual exploration of subsets of records by displaying either class envelopes (semi-transparent bands), or ellipse plots on axes which provide all necessary information about the distribution of attributes both in the entire dataset, and in the specific classes. This approach can also be used to visualise clusters of data. An example of using semi-transparent bands for visualising data which results from cluster analysis is shown in Figure 4.8.

Allowing users to select the level of detail is a concept which has been used by Zhou, Cui, et al. [2009]. Their approach includes users observing the drawing and clustering process, which is executed over a specific period of time. This process consists of two major components: polyline splatting and segment splatting. In polyline splatting, whenever a new line is drawn, the neighbouring ones are enhanced and irrelevant ones are suppressed. Segment splatting consists of representing each polyline as segments which are splatted with different speeds, colours and lengths. The user may pause and resume the animation. The concept of using animation to convey statistical properties has also been explored by Johansson, Ljung, et al. [2005].

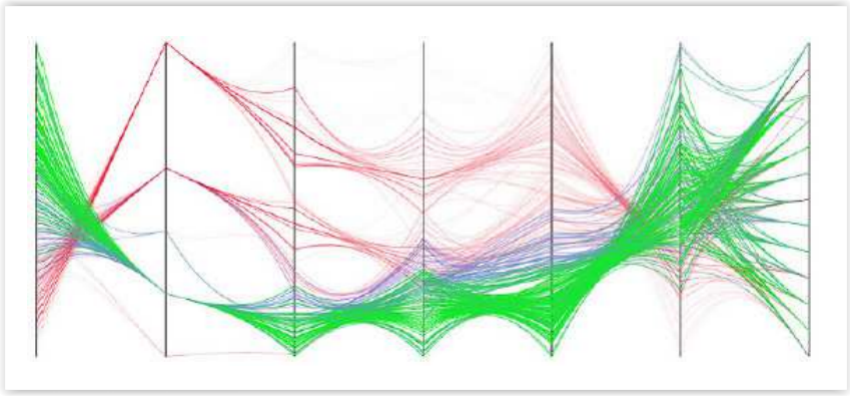
As can be seen from the short overview given above, many different clustering algorithms can be used to group together similar records in large datasets. In addition to the standard classification algorithms such as k-nearest neighbours, approaches such as self-organising maps, fuzzy-logic, and special approaches such as hierarchical and visual clustering (classifying based on geometrical properties) may also be taken. The most trivial approach to displaying clustered records is colour-coding, but other methods, such as edge bundling and displaying class envelopes, may be more effective at de-cluttering the view and conveying the information about the number of records in particular clusters. Interactive animation of the rendering process can also be applied to help users discover patterns in large datasets. Combinations of some of these techniques may also yield interesting results.



(a) The initial parallel coordinates display.



(b) Parallel coordinates display after applying the edge bundling technique.



(c) Parallel coordinates with bundled edges and cluster colouring.

Figure 4.7: The clutter in the initial parallel coordinates display (a) is reduced after applying the edge bundling technique (b). In (c), polyline clusters are additionally coloured. [Images extracted from Zhou, Yuan, et al. [2008], with kind permission of Huamin Qu.]

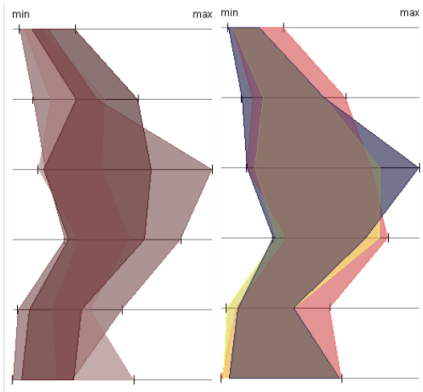


Figure 4.8: Transparent colour bands, or envelopes in parallel coordinates. On the left, the ranges of object characteristics for the clusters of countries according to population density is shown, while the right plot shows clusters according to age structure. [Image extracted from G. Andrienko and N. Andrienko [2004], with kind permission of Gennady Andrienko.]

4.3 Data Exploration and Analysis with Parallel Coordinates

The success of the data exploration and analysis process often depends on how the user interacts with the data. Certain exploratory skills are required, in order to reveal meaningful insights from a given dataset. Inselberg [2010] provides several examples of data exploration and analysis with parallel coordinates. This section describes the process of exploratory analysis of a financial dataset conducted by Inselberg and four financial experts, as described in his book [Inselberg, 2010, Chapter 10].

The specific financial dataset contains the following dimensions:

- WEEK - quotes for 54 weeks in a year,
- MONTH - the 12 months of a year,
- YEAR - 9 years (1985 - 1993),
- BPS - British Pound Sterling quotes,
- GDM - German Dmark quotes,
- YEN - yen quotes,
- TB3M - interest rates for three months (expressed in percentages),
- TB30Y - interest rates for 30 years bonds (expressed in percentages),
- GOLD - price of gold (expressed in $/ounce), and
- SP500 - the American Standard and Poor's 500 stock market index [Investopedia, 2015].

Figure 4.9 shows the parallel coordinates visualisation of the financial dataset in ParallAX [T. Avidan and S. Avidan, 1999]. The available records for the years 1986 and 1992 are selected and compared. The visualisation reveals that, in the year 1986, the yen had the greatest volatility, which can be seen by comparing the value ranges among the three currencies in this year. Interest rates varied in the mid range, and there was a gap in the price of gold. In the year 1992, the yen was stable, while the British pound (BPS) was highly volatile. Both interest rates and the gold price were very low. Next, the data for the year 1986 is isolated, and the low range of the prices of gold are selected, as shown in Figure 4.10. It is immediately discovered that the price of gold varied in the lower range until the second week of august, when it suddenly jumped and stayed high.

An interesting pattern shown by visualising the dataset in parallel coordinates is the relation between the YEN and the TB3M dimension. As shown in Figure 4.11, there are many crossings in the area between the sixth and seven axis, which is caused by the negative correlation between the two dimensions. By varying the values of other dimensions, it is discovered that there is also a positive correlation between YEN and the TB3M in the year 1990. This positive correlation is shown in Figure 4.12. The price of gold in this year was in the low and mid range.

Based on the discovered pattern, the exploration was continued by examining the portion of the dataset where the value of the yen is in the upper range. By isolating this portion of the dataset, further interesting patterns were discovered. As shown in Figure 4.13, the negative correlation between the value of the yen and interest rates matches the low price of gold, while the positive correlation between YEN and TB3M matches the high price of gold. In other words, movements in currency exchange rates appear to be related to the price of gold. To test this assumption, the upper range of gold price was isolated. This time, the correlation between currencies BPS and GDM, is examined using the scatter plot visualisation. As shown in Figure 4.14, the correlation between BPS and GDM forms an almost perfect straight line. The slope of the line represents the exchange rate between GDM and BPS, which is constant when the

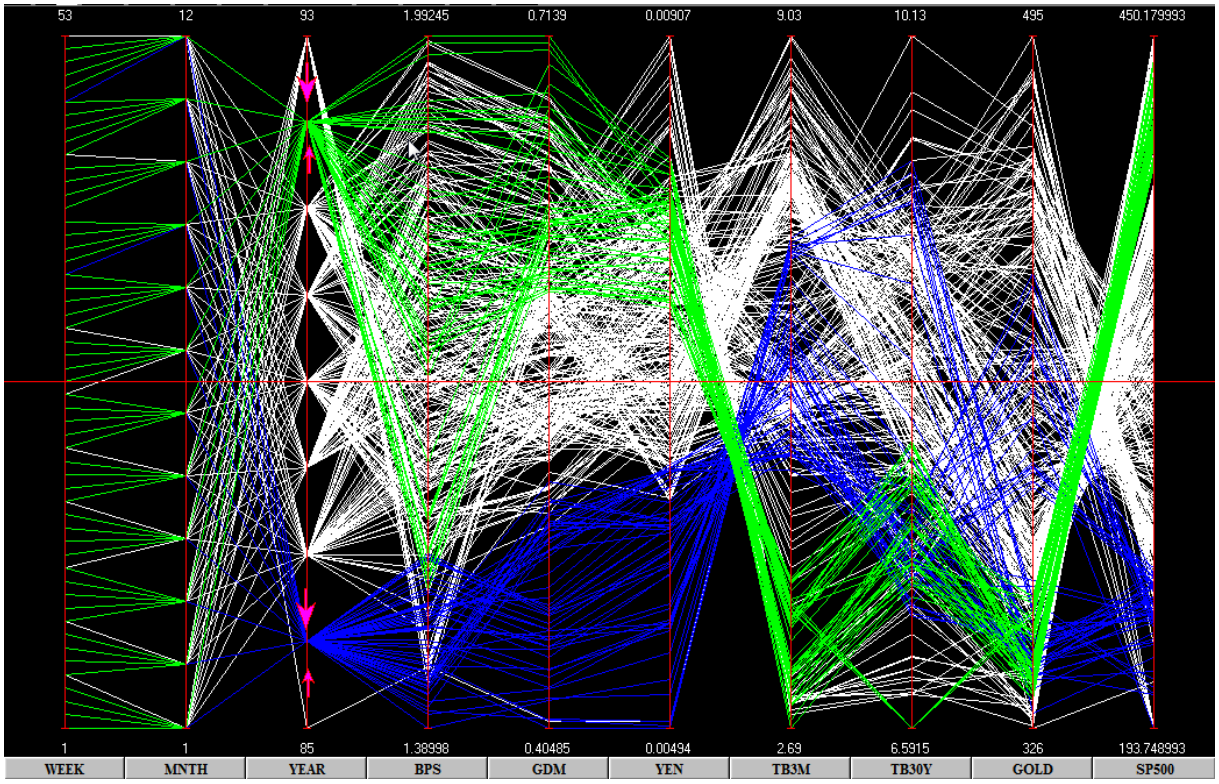


Figure 4.9: Parallel coordinates visualisation of financial data for the years 1985-1993. The data for the years 1986 and 1992 is selected and represent in green and blue. In 1986, the yen had the greatest volatility, interest rates varied in the mid range, and there was a gap in the price of gold. In 1992, the yen was stable, while BPS was highly volatile. Both interest rates and the price of gold were very low. [Screenshot of the parallel coordinates visualisation in ParallAX [T. Avidan and S. Avidan, 1999] created by the author.]

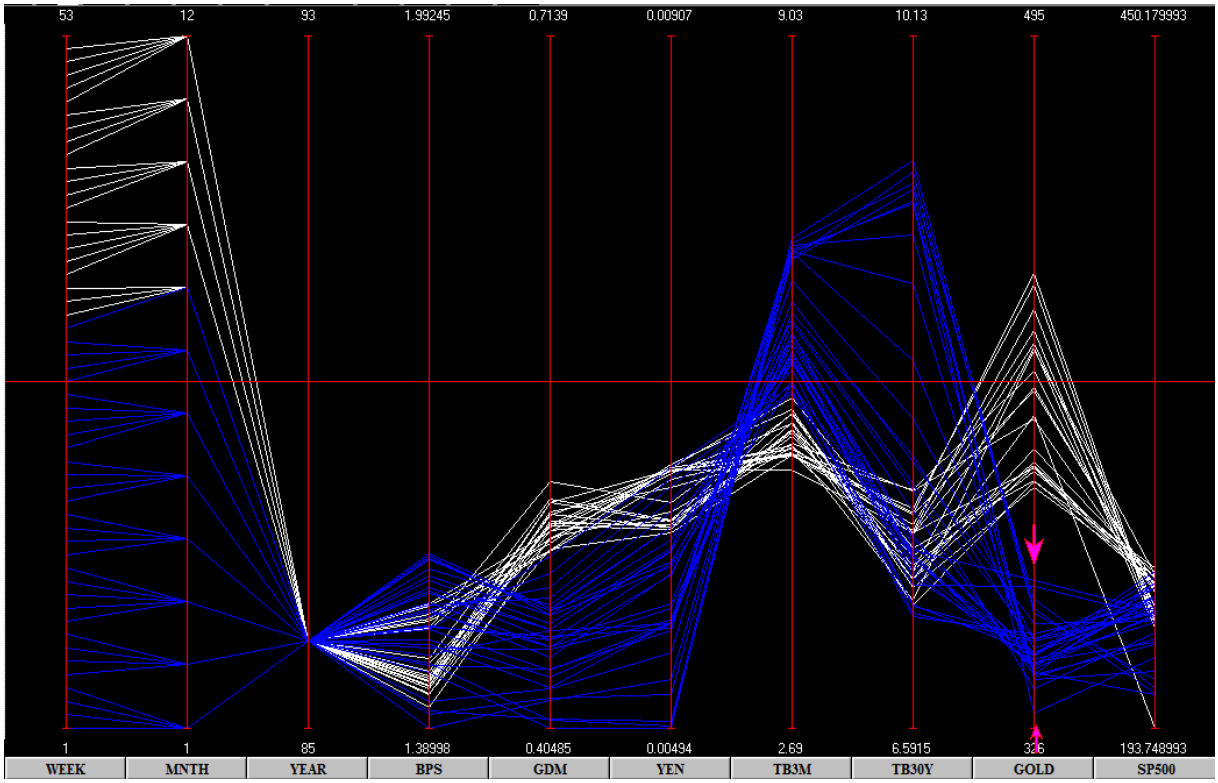


Figure 4.10: The data for the year 1986 is isolated. The price of gold in 1986 was low until the second week of August, when it suddenly jumped and stayed high. [Screenshot of the parallel coordinates visualisation in ParallAX [T. Avidan and S. Avidan, 1999] created by the author.]

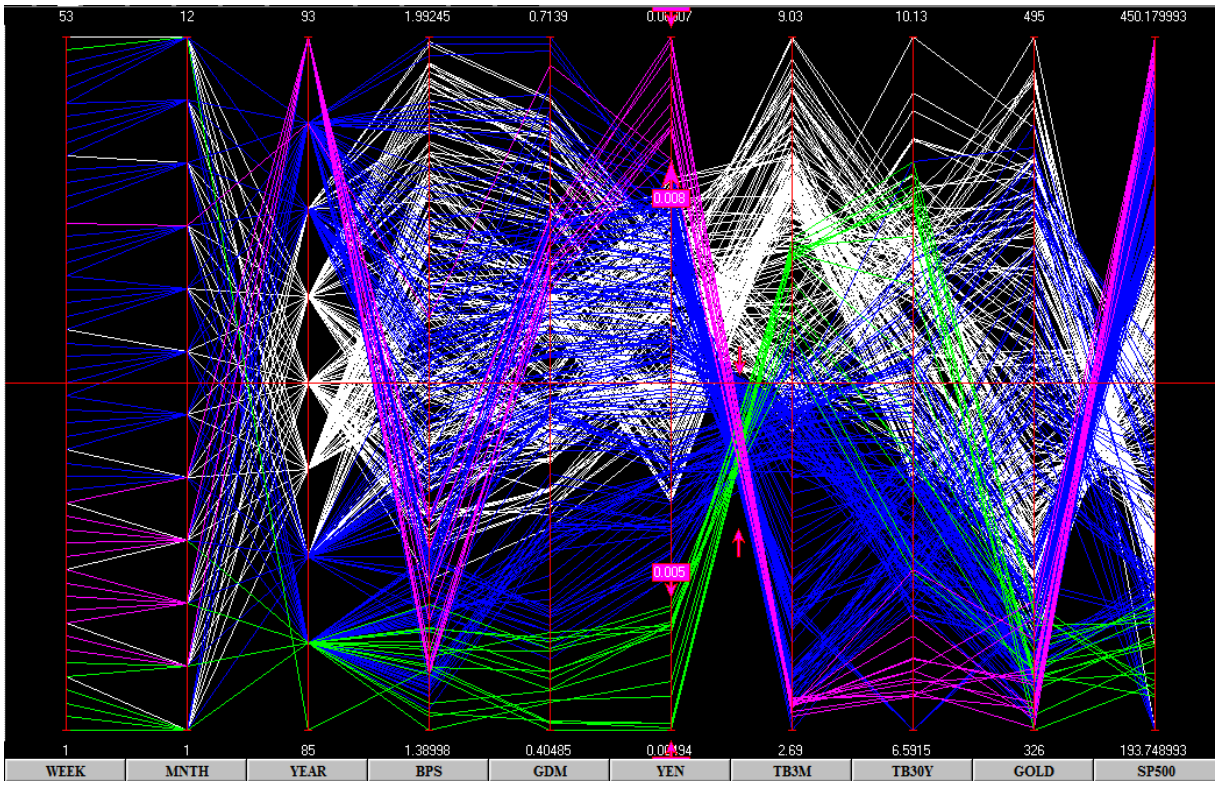


Figure 4.11: The negative correlation between the sixth and seventh axis is examined. When the value of the yen was low, interest rates were high, and vice versa. [Screenshot of the parallel coordinates visualisation in ParallAX [T. Avidan and S. Avidan, 1999] created by the author.]

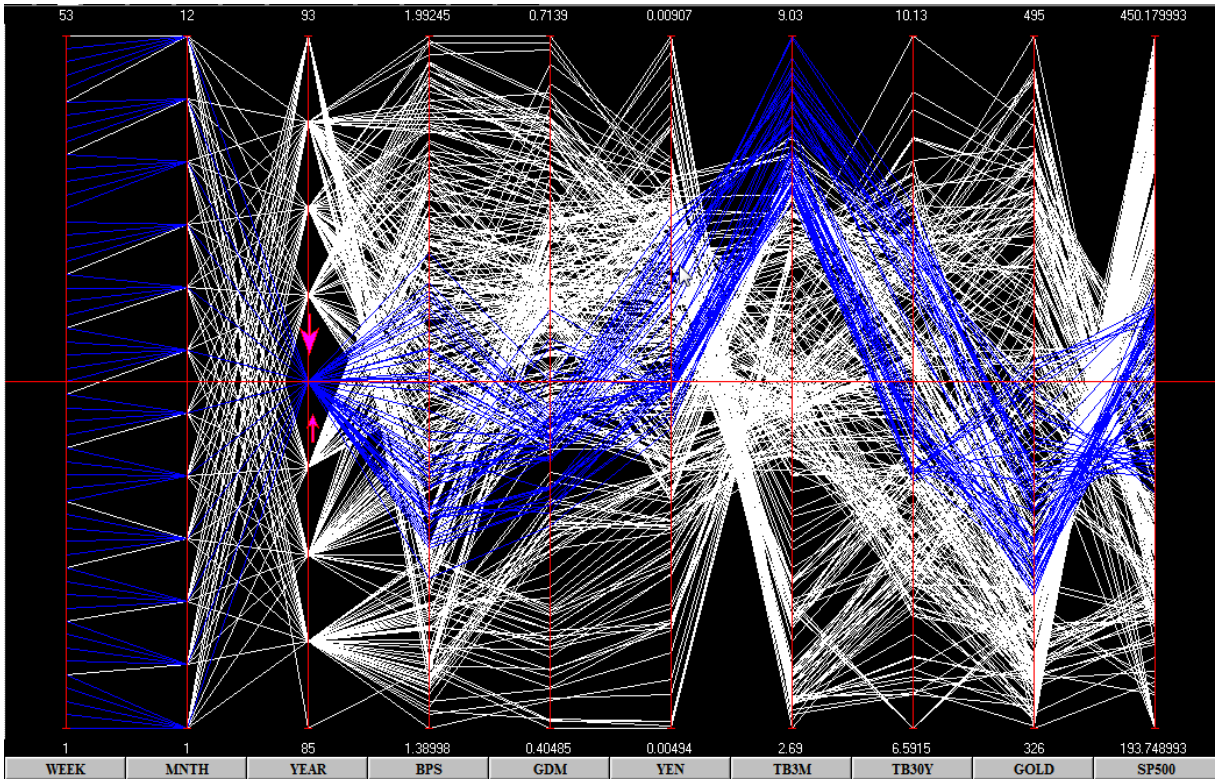


Figure 4.12: Positive correlation between YEN and TB3M is present in the year 1990. [Screenshot of the parallel coordinates visualisation in ParallAX [T. Avidan and S. Avidan, 1999] created by the author.]

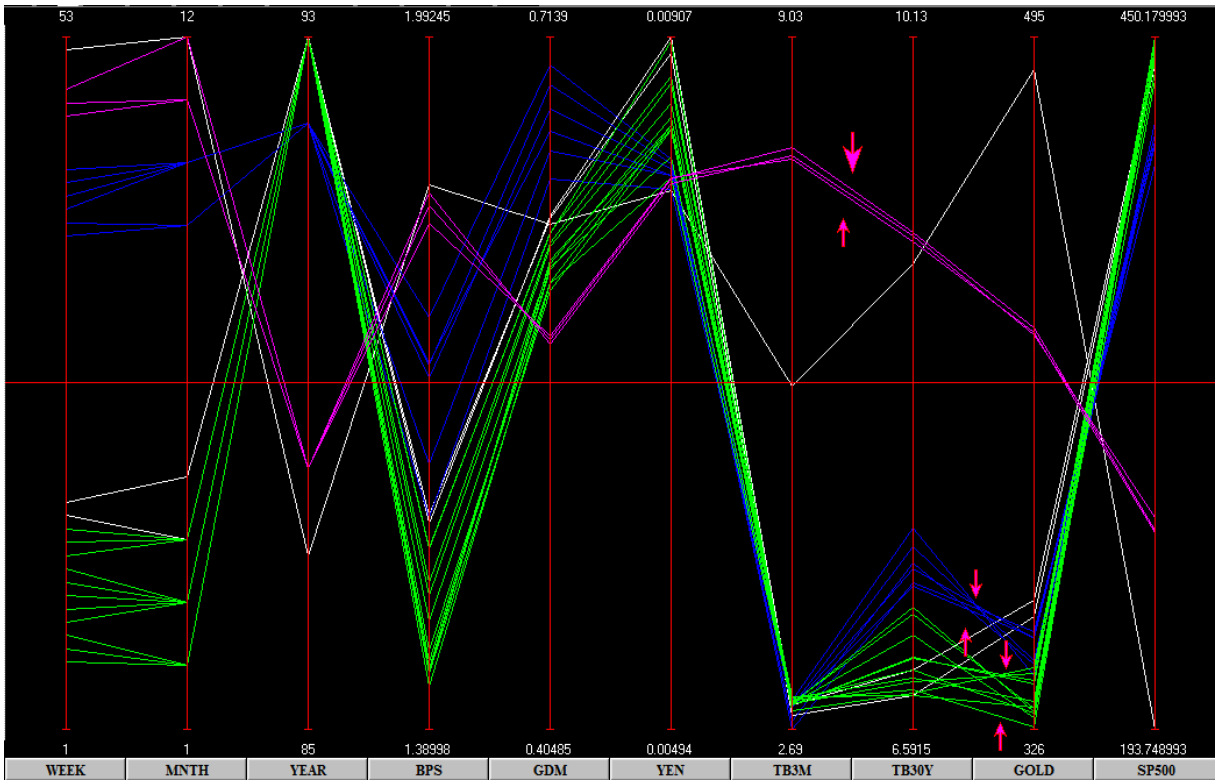


Figure 4.13: The high value range of the YEN is isolated. The negative correlation between the value of yen and interest rates goes together with a low price of gold, while the positive correlation between YEN and TB3M goes together with a high price of gold. [Screenshot of the parallel coordinates visualisation in ParallAX [T. Avidan and S. Avidan, 1999] created by the author.]

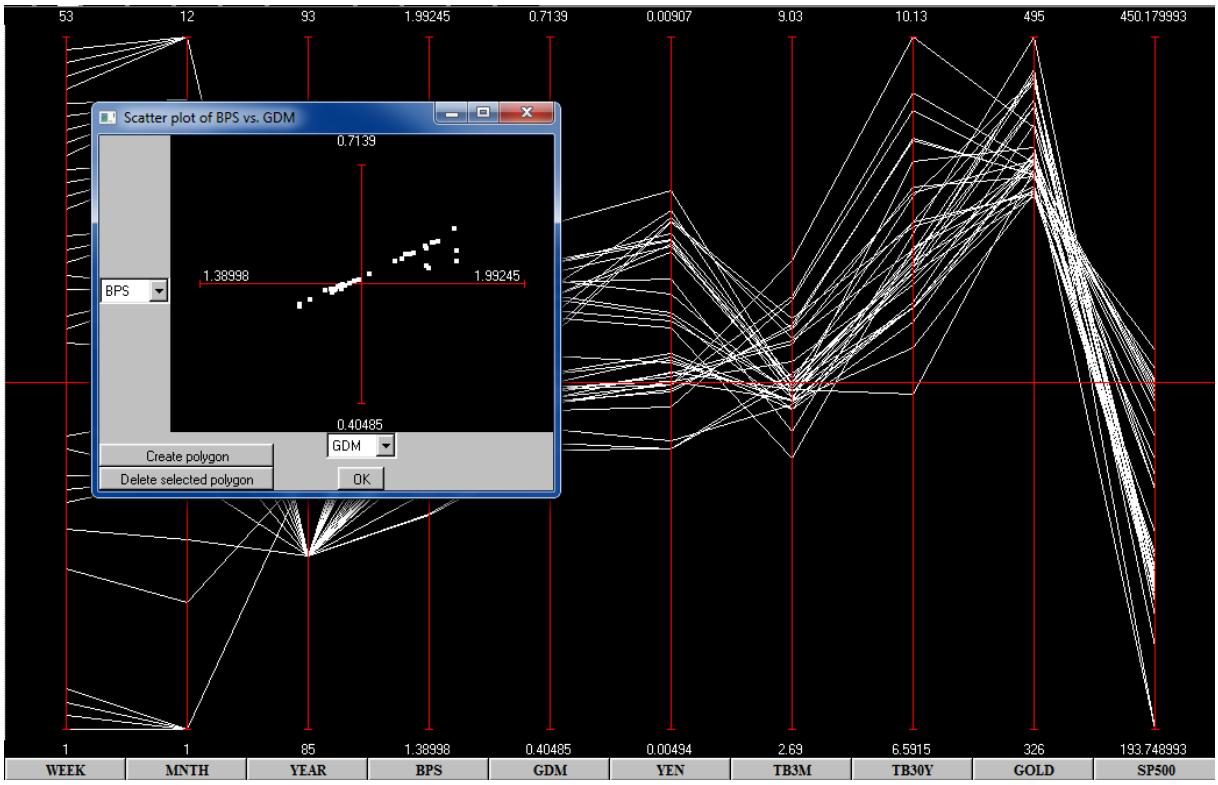


Figure 4.14: The high value range of gold prices are isolated, and the correlation between BPS and GDM is shown in scatter plot. The slope of the almost perfect straight line formed in the scatter plot indicates an almost constant exchange rate between GDM and BPS when the price of gold is high. This establishes a trading guideline and suggests a "behind-the-scenes manipulation of the gold market" [Inselberg, 2010, page 396]. [Screenshot of the parallel coordinates and scatter plot visualisation in ParallAX [T. Avidan and S. Avidan, 1999] created by the author.]

price of gold is high. This establishes a trading guideline and suggests a "behind-the-scenes manipulation of the gold market" [Inselberg, 2010, page 396].

The process of exploratory analysis described above demonstrates how a parallel coordinates visualisation can be used to gain new knowledge and insight. Although such patterns are usually not obvious at first glance, knowledge of the geometry of parallel coordinates, attention to detail, and the use of interactive features may lead to stunning discoveries. Based on this, and the analysis of four other datasets, Inselberg [2010, Chapter 10] derives the following guidelines for visual data exploration:

1. Do not let the picture intimidate you.
2. Understand the objectives.
3. Carefully scrutinise the data display for clues and patterns.
4. Good choices may be worth repeating.
5. Vary the value of one of the variables, while watching for interesting changes in the other variables.
6. Be sceptical about the quality of datasets with a large number of dimensions.
7. Test the assumptions and especially the "I am really sure of"s.

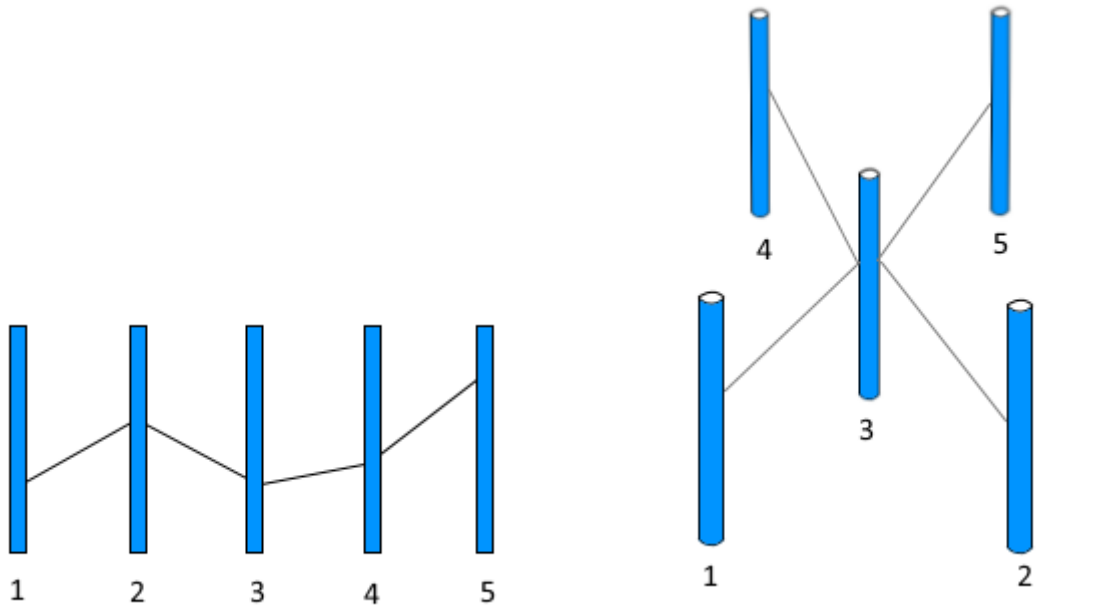
The sixth guideline is derived from an exploration of a dataset with 400 variables. After visualisation with parallel coordinates, it was immediately discovered that many dimensions contained invalid data. Furthermore, the repetitive visual patterns showed that many dimensions were simply repeated several times in the dataset under different names. Although these guidelines were derived using parallel coordinates, they can be also be relevant when exploring data with (a combination of) other methods.

4.4 Variations of Parallel Coordinates

The original idea of parallel coordinates, as introduced by Inselberg [2010], has been extended in many ways. Extensions introduced in order to handle the problem of occlusion when visualising large datasets were previously discussed in Section 4.2. The focus of this section are variations of parallel coordinates which solve other problems associated with parallel coordinates. An overview of the most common parallel coordinates variations was also provided by Hackl [2011, Chapter 3].

4.4.1 Three-Dimensional Displays

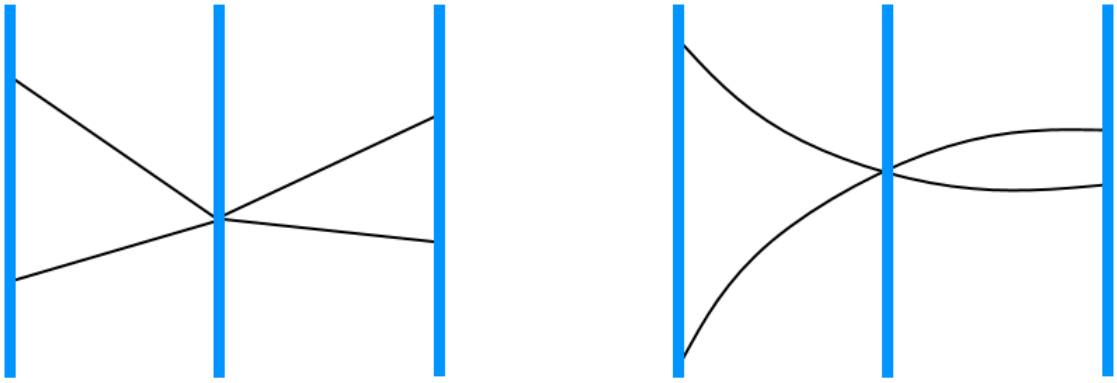
Johansson, Cooper, and Jern [2005] proposed using a three-dimensional variation of the standard parallel coordinate implementation. In this variation, one axis is positioned in the centre of the view, while all other axes are placed in three dimensional space around the central axis. Every axis is connected only to the central axis. This approach enables easy exploration of relations between the axes, as shown in Figure 4.15, since the user is able to simultaneously compare several dimensions to a single dimension of interest. The dimension (axis) of interest can be changed using interactive features. Although an arbitrary number of dimensions can be visualised using this approach, Johansson, Cooper, and Jern [2005] argue that the upper limit for the number of dimensions which can easily be perceived and distinguished is between 15 and 20.



(a) Two-dimensional parallel coordinates.

(b) Three-dimensional parallel coordinates.

Figure 4.15: In two-dimensional parallel coordinates, only adjacent axes can be compared. In (a), the third axis can be directly compared only with the second and fourth axis. In three-dimensional parallel coordinates (b), one axis is displayed in the centre of the view, and all other axes are connected to it. Here, axis 3 can be directly compared to all other axes. [Image created by the author based on Figure 1 in Johansson, Cooper, and Jern [2005], using paint.net [dotPDN LLC, 2015].]



(a) Parallel coordinates with polylines.

(b) Parallel coordinates with polycurves.

Figure 4.16: Using polylines to represent records in parallel coordinates visulisations can cause polylines to cross on an axis. When polylines cross, it is almost impossible to tell which segment belongs to which polyline (a). This problem can be solved using polycurves instead of polylines. In (b), crossing polycurves are visually kept apart simply based on their curvature. [Image created by the author based on Figure 8 in Graham and Kennedy [2003].]

4.4.2 Curves

The use of polycurves instead of polylines in parallel coordinates was previously mentioned in the context of edge bundling, as a method for handling large datasets. Graham and Kennedy [2003] suggest using polycurves to solve yet another problem associated with the standard parallel coordinates implementation. If polylines are used to represent the records, and two or more records have the same value in one of the dimensions, the polylines will cross on the axis which represents that dimension. When polylines cross on one axis, it is impossible to tell which segments belongs to which polylines. As shown in Figure 4.16, this problem can be avoided by using polycurves. With this simple approach, the records can be visually kept apart simply based on the curvature of the record representation.

4.4.3 Parallel Sets

Data categorisation was discussed as a method of dealing with large datasets in parallel coordinates along with different methods of presenting the categorised data. In such parallel coordinates implementations, the categories are used primarily to find an interesting subset of the dataset whose numerical values are then explored in more detail. Real-world datasets often contain categorical data in which the number of items belonging to a category plays a more important role than the numerical values of the items. Parallel sets is a visualisation technique optimised for such datasets. Like in parallel coordinates, each dimension is treated separately and is represented by an axis. The categories themselves are represented by boxes [Kosara, Bendix, and Hauser, 2006]. The size of the boxes are scaled to the number of records belonging to the category. The relations between the categories are represented by parallelograms between the axes. The numerical values of individual records are not displayed. An example of a parallel sets visualisation can be seen in Figure 4.17.

Large datasets can be displayed very effectively with parallel sets, because using the frequency information reduces the number of objects which have to be displayed. Both performance and of cluttering depend only the number of categories, which is generally small relative to the number of records in the dataset. In their implementation of parallel sets, Kosara, Bendix, and Hauser [2006] provided the following interactive features: selection, highlighting, interactive querying, filtering, and reordering of dimensions and categories.

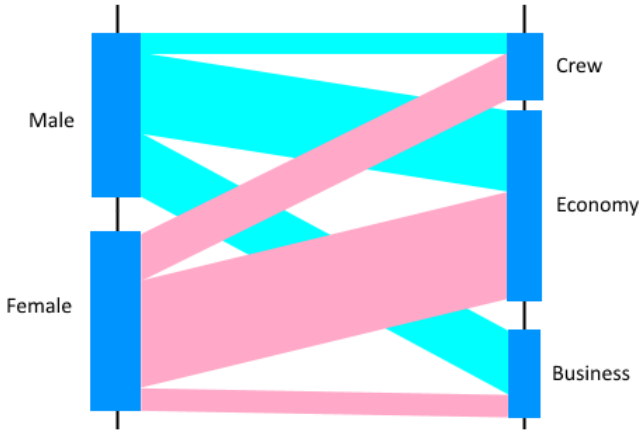


Figure 4.17: Categorical data visualised with parallel sets. Boxes on the axes represent the categories. Adjacent axes are connected with parallelograms. This (fictional) dataset contains data about the gender of the passengers and the class in which they travel. The parallelograms are colour-coded based on the categories represented on the first axis, thus enabling better visual distinction between these two categories on the second axis. [Image created by the author using paint.net [dotPDN LLC, 2015], based on Figure 3 in Kosara, Bendix, and Hauser [2006].]

4.5 Software Applications

Since the invention of the parallel coordinates, this visualisation technique has become a popular component of many visual data exploration tools. While the basic representation of the dataset in parallel coordinates is similar across many tools, many of these tools differ in terms of interactivity and other features. In this section, an overview of four commonly used parallel coordinates tools is provided: Ggobi, XDAT, OECD eXplorer, and ParallAX. In addition to these three tools, an example implementation of parallel coordinates in WPF is also described.

4.5.1 GGobi

GGobi is a free, open-source, general-purpose high-dimensional data visualisation tool [Cook and Swayne, 2007], [Swayne et al., 2006]. It is implemented in C. The GTK toolkit [The GTK+ Team, 2014] is used to build the graphical user interface (GUI). Datasets can be imported either from `.xml` or `.csv` files. Several multi-dimensional information visualisation techniques are supported in GGobi: scatter plot, scatter plot matrix, parallel coordinates, bar chart, and time series. Each of these visualisations is displayed in a separate view. The views are interconnected, which means that changes made in one visualisation are automatically applied to all other visualisations.

The dimensions in GGobi parallel coordinates can be ordered either horizontally or vertically. As shown in Figure 4.18, only the polylines and individual data points are displayed. The axes are not drawn explicitly. The currently selected dimension is marked with a white rectangle surrounding the data points. The dimensions can be re-ordered by dragging this rectangle to a new position. Record selection is possible, but has to be explicitly activated using a menu option, which is available in a separate view. After this interaction has been activated, only the records within the active dimension can be selected by drawing a rectangle with a mouse over the data points. Selection of records by drawing a rectangle over the polylines is not possible. When a different dimension is selected, the currently selected records are automatically unselected, unless the “Persistent” option is active. In order to show the record values, the “Identify” interaction has to be explicitly activated. Labels with the record values are then displayed while hovering over a data point within the currently selected dimension.

GGobi provides only a limited set of interactions. Each interaction has to be explicitly activated

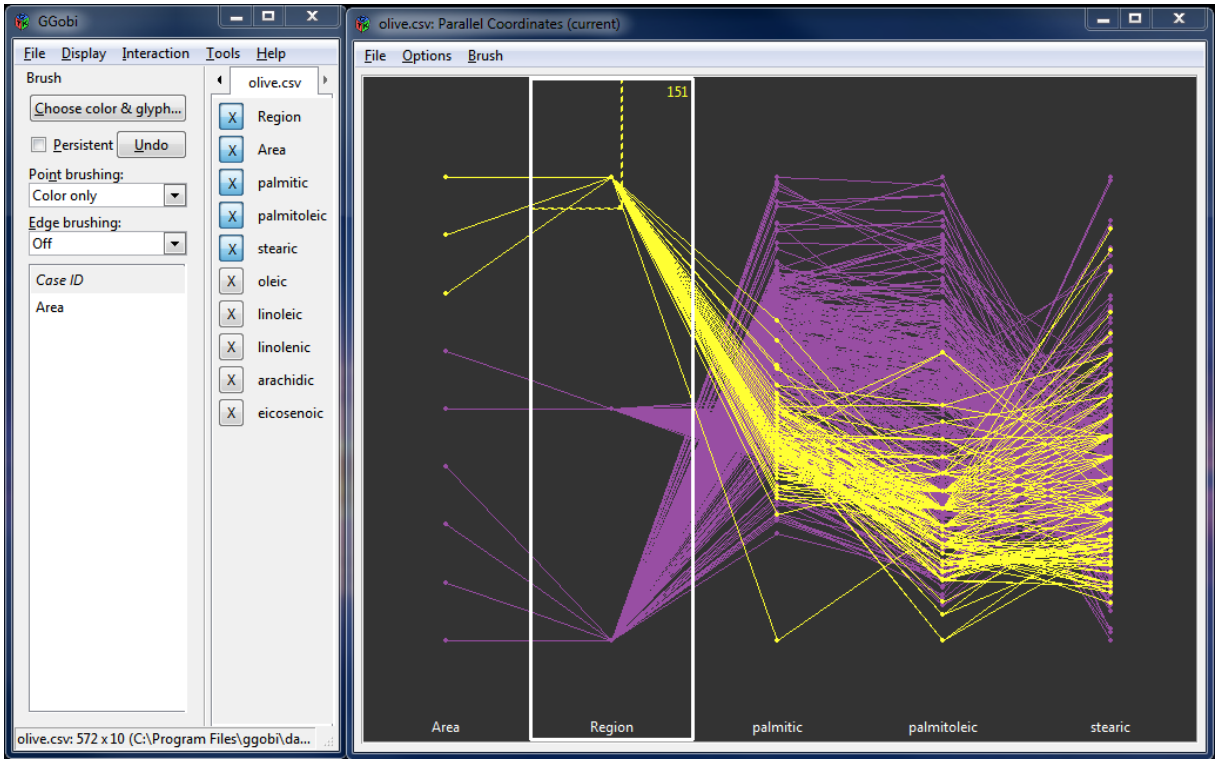


Figure 4.18: GGobi [Cook and Swayne, 2007] parallel coordinates visualisation of a dataset with 573 records and 5 dimensions. Each unique data point is visualised, along with polylines connecting the data points to represent a single record of the dataset. The axes are not drawn explicitly. The active dimension is marked by a white the rectangle which surrounds the data points. Record selection, as well as the order implemented interactions, are activated explicitly in a separate view (shown on the left). [Screenshot created by the author.]

using a special menu, which is available in a separate view. While one interaction is activated, all other interactions are disabled. This means, for example, that the axes cannot be reordered while the record selection or identification is enabled, which is rather unintuitive.

4.5.2 XDAT

XDAT is a general-purpose, free, and open source visual exploration tool for multivariate data [Enguerrand de Rochefort, 2015]. It is implemented in Java [Oracle, 2015]. Datasets can be imported from text files, and visualised in form of a table, scatter plot, or parallel coordinates.

The XDAT parallel coordinates visualisation is shown in Figure 4.19. Dimensions can be added or removed from the view using the “Parameters” option, which is provided in the toolbar. The axes can be re-ordered using drag-and-drop. Sliders provided at the top and the bottom of each axis can be used to filter the records. Records which are filtered out, are simply removed from the view. The context menu can be activated for each axis individually and provides a comprehensive set of features for axis manipulation, such as axis inversion, scaling, hiding, filter resetting, etc. Single records can be highlighted simply by hovering over the individual polylines. A selectable number of “ticks” can be drawn on each axis, so users can identify value ranges.

A distinctive feature implemented in XDAT is manual record clustering. The set of currently active records can be marked as a single cluster, for which a colour can be selected. By removing the current portion of the dataset from the view using the sliders, and adding a new one, a new cluster can be defined. An arbitrary number of clusters can be added. Single clusters can then be dynamically added or removed from the view, thus enabling users to selectively explore different parts of the dataset.

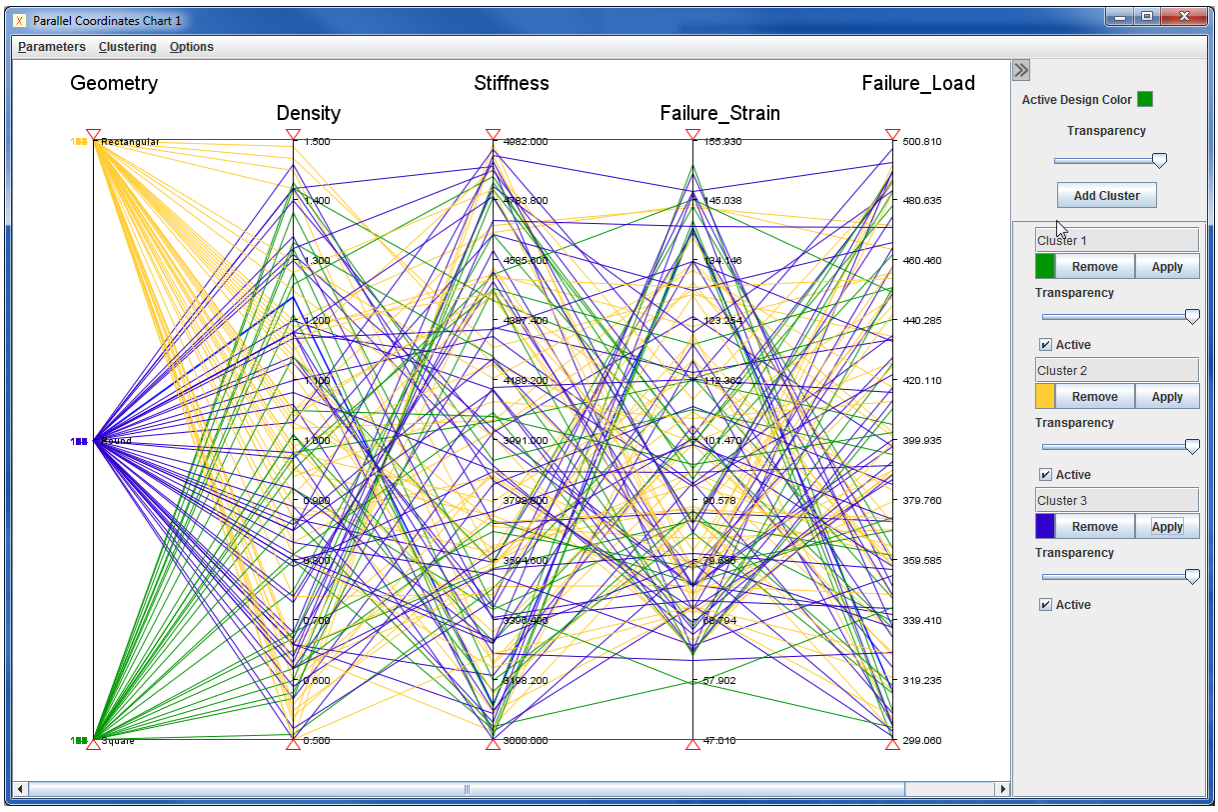


Figure 4.19: Parallel coordinates visualsation of a dataset with 200 records and 5 dimensions in XDAT [Enguerrand de Rochefort, 2015]. Distinctive feature implemented in XDAT, is manual clustering. The user can select a portion of the dataset using the sliders, and mark the set of active records as a cluster, using the view on the right. An arbitrary number of clusters can be defined. The user can select a different colour for each cluster. [Screenshot created by the author.]

The scatter plot, parallel coordinates, and the table view in XDAT are partially connected. When clusters are added or removed from the parallel coordinates plot, these changes are automatically applied to the scatter plot visualisation. Records selected in any of the three views are automatically selected in all other views. However, when records are filtered out in the parallel coordinates, they remain visible in the scatter plot.

As seen from the overview given above, XDAT provides a standard set of interactions, which can be accessed in a relatively intuitive way. Manual clustering is a feature which can further enhance the process of visual exploration. However, the values of the selected records can be only seen in the table representation, which forces user to constantly switch between the table and parallel coordinates visualisation.

4.5.3 OECD Statistics eXplorer

The Organisation for Economic Co-operation and Development (OECD) provides a free, web-based visual data exploration tool [NComVA, 2014], [Jern, 2009]. This tool is implemented in Flash [Adobe, 2015], and is optimised for use with regional statistics data. The visualisation is pre-loaded with the data provided by the OECD, but users can load their own datasets from text files. It provides a comprehensive set of interconnected visualisations: scatter plot, scatter plot matrix, table lens, data table, bar chart, time bar chart, distribution plot, time graph, parallel coordinates and geographic map.

The parallel coordinates visualisation in the OECD eXplorer, shown in Figure 4.20, provides a standard set of interactive features. Sliders are available on the axes to filter the records. Axes can be inverted

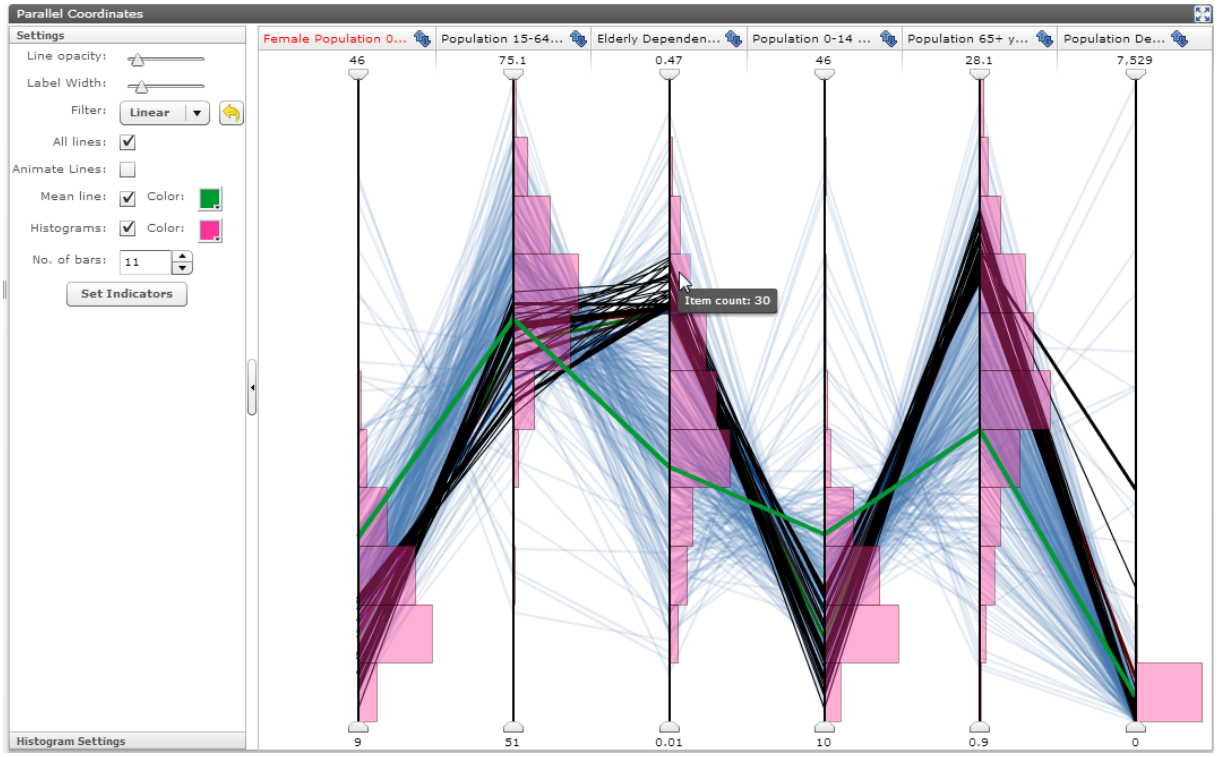


Figure 4.20: The parallel coordinates visualisation in the OECD Regional eXplorer [NComVA, 2014] provides a standard set of interactions. A distinctive feature is the interactive histograms. Clicking on a histogram bin causes all records mapped to the axis range occupied by the selected bin to be selected. [Screenshot created by the author.]

using the button next to the axis label, or scaled by manually defining the value after clicking on any of the labels which indicate the current minimum of maximum values of records. Using the "Settings" view, the user can re-order the axes, change the opacity and other visual properties of the polylines, and add or remove the histograms and the mean line. Multiple records can be selected either by clicking on the plot area on which polylines cross, or by clicking on the histogram bins, which selects all records mapped to the axis interval occupied by the selected bin.

When the sliders are moved across an axis, the polylines are not interactively filtered in and out, but are only removed from the view once the slider is released. Furthermore, the selected records are not labelled. Like in XDAT, users have to refer to the data table in order to view the exact values of selected records.

4.5.4 ParallAX

ParallAX [T. Avidan and S. Avidan, 1999], shown in Figure 4.21, is a commercial data visualisation tool developed by Alfred Inselberg. Datasets can be imported from text files with `.DAT` file extension, in which records are separated by a line break, and elements between the records are separated by spaces. Other data formats are not supported.

ParallAX provides a comprehensive set of record selection mechanisms, for which the term "query" is used. Using the "New query" option which can be accessed from the "Query" menu, a new query set can be created and assigned a colour. All records selected within a query set are then displayed in that colour. Users can manipulate the record display within a query set using the following interactions:

- *Pinch*. When this interaction is activated by clicking on the "P" toolbar button, two arrows appear on the screen, using which users can define a range of records of interest by moving the arrows across the display.

- *Interval*. Similar to the pinch query, activating this interaction using the "I" button in the toolbar causes two arrows to appear on the screen, above (or under) which a label is displayed, showing the current value of the arrow's position on the axis. This allows selection of records within a particular value interval.
- *Angle*. Activating this interaction by clicking on the "A" button in the toolbar and then clicking on an axis, allows users to define the angle range of interest between a selected axis and an axis on its left, by moving the arrows which appear on the screen. This allows selection of only those records, whose angle between the line segment and the first axis is within the specified angle range.
- *Flip*. Axes can be selected using the buttons underneath the axis lines containing the axis label and inverted (flipped) by clicking on the appropriate button in the toolbar.

Users can define an arbitrary number of query sets, and combine them using conjunction, disjunction, and complement operators. Records selected within one or more query sets can be isolated, allowing exploration of records of interest without the displaying all other records. Despite having a different name, the possibility to define query sets corresponds to the manual clustering feature in XDAT.

ParallAX implements the automatic calculation of axis orderings (so-called Hamiltonian paths) which allow comparison between every pair of axes, as described in Section 5.1. In addition, users can define their own axis orderings using the parameter panel in which the calculated orderings are displayed. This panel is shown in Figure 4.22. ParallAX also provides a synchronised scatter plot view, and a clustering mechanism which uses the nested cavity algorithm [Inselberg, 2010, page 406].

Although it provides a comprehensive set of features for visual data exploration, ParallAX is a rather unintuitive tool. All interactions must be explicitly activated by clicking on an appropriate button in the toolbar. To be able to use the provided features, users often need to refer to the manual.

4.5.5 Parallel Coordinates in WPF

Wlodek [2009] provides an example implementation of parallel coordinates using Microsoft's WPF [MSDN, 2015e] technology. The source code of this example is freely available for download [Wlodek, 2010]. Using the source code, a demo application can be compiled an executed.

The number of features in this parallel coordinates implementation is very limited. Datasets cannot be imported from any sources. However, random datasets with 10, 30, 50, 100, or 200 records can be generated. Figure 4.23 shows a visualisation of a generated dataset with 200 records and 6 dimensions. In addition to the axes and polylines, which represent the dimensions and records of a dataset, record values in every dimension are visualised with ellipses, which are drawn at the points where polylines interesect with the axes. Single or multiple records can be selected simply by clicking on the polylines. Selected records are highlighted and labels are drawn to indicate record's value in each dimension. Every axis can be zoomed in or out individually. Axes can be re-ordered using drag-and-drop. When a selected axis is dropped over another axis, the two axes simply swap positions. An interesting feature, whose usefulness is highly questionable, is the possibility to move all points up and down the axis.

As seen from the overview given above, this implementation cannot be used for visual exploration of any datasets other than random ones generated by the application itself. Furthermore, it provides a very limited set of interactions. However, the author's intention was not to produce a usable application, but to demonstrate how parallel coordinates can be implemented in WPF. The provided source code was used as a starting point for implementing aggregated parallel coordinates visualisation described in Chapter 5. The limitations of the Wlodek's approach, and the steps taken in order to improve and extend this implementation are explained in Chapter 6.

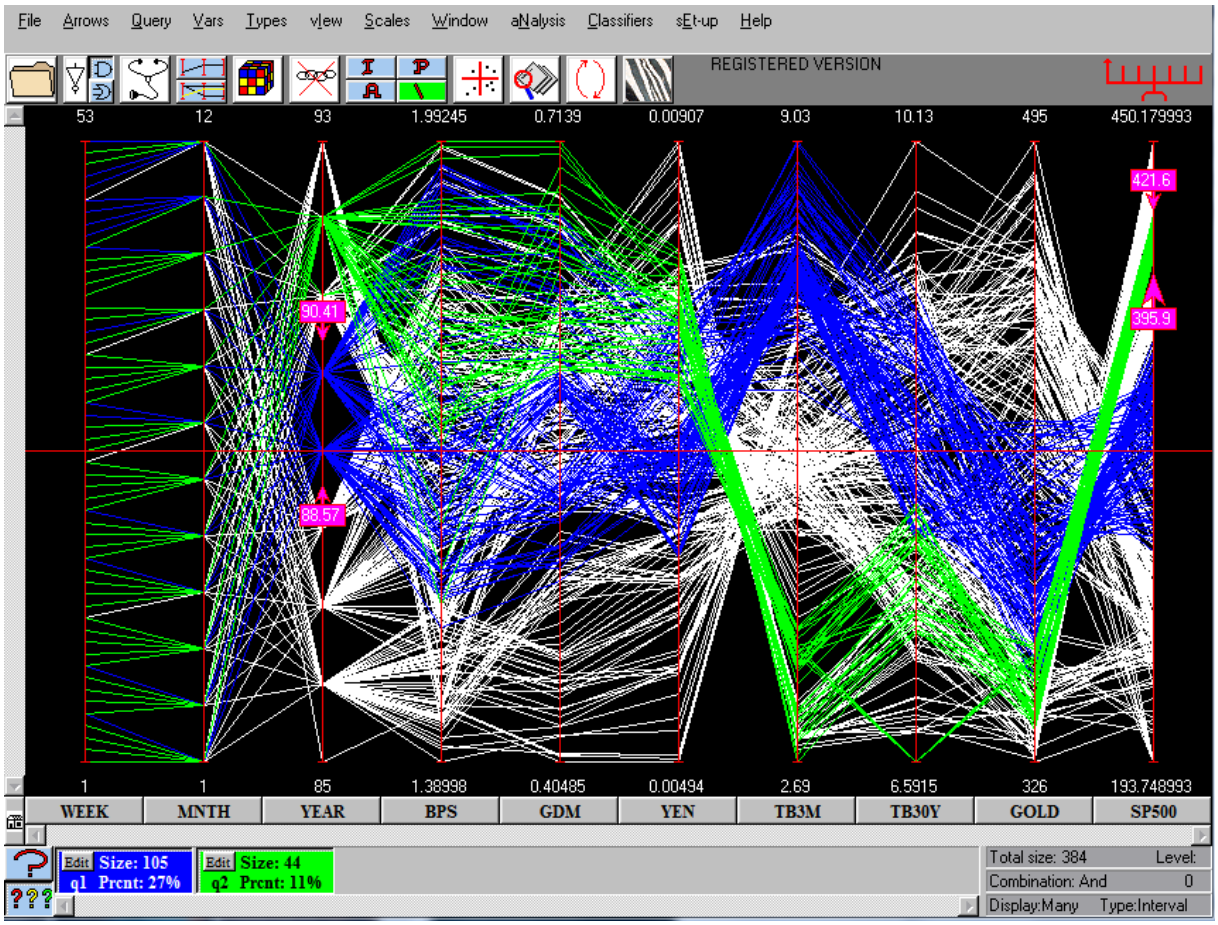


Figure 4.21: Parellel coordinates visualisation in ParallAX [T. Avidan and S. Avidan, 1999]. All interactions have to be explicitly activated using the options in the menubar or the icons in the toolbar. Sets of queries can be defined and assigned a colour, causing all records selected within a set to be displayed in that colour. Here, two sets are defined (blue and green). Query sets can be added or removed from display using the coloured buttons at the bottom, or combined into a new set using boolean operators, which can be accessed from the toolbar. Axis labels are represented as buttons. Clicking on an axis button selects an axis, enabling the axis for manipulation, such as inversion.
[Screenshot created by the author.]

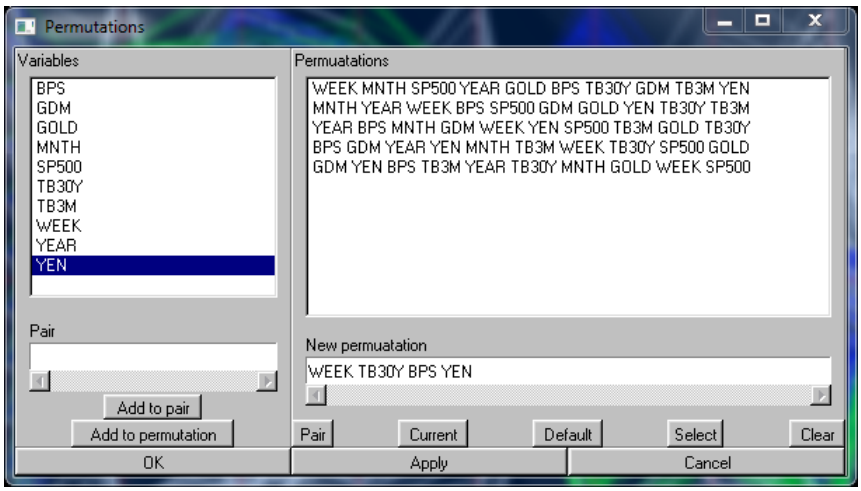


Figure 4.22: The Permutations panel in ParallAX showing a minimal set of axis ordering permutations (so-called Hamiltonian paths) which allow comparison between all pairs of axes. Users can view permutations and apply them to the current parallel coordinates display. Additionally, users can define their own axis orderings by selecting the axis labels from the left in a particular order. [Screenshot created by the author.]

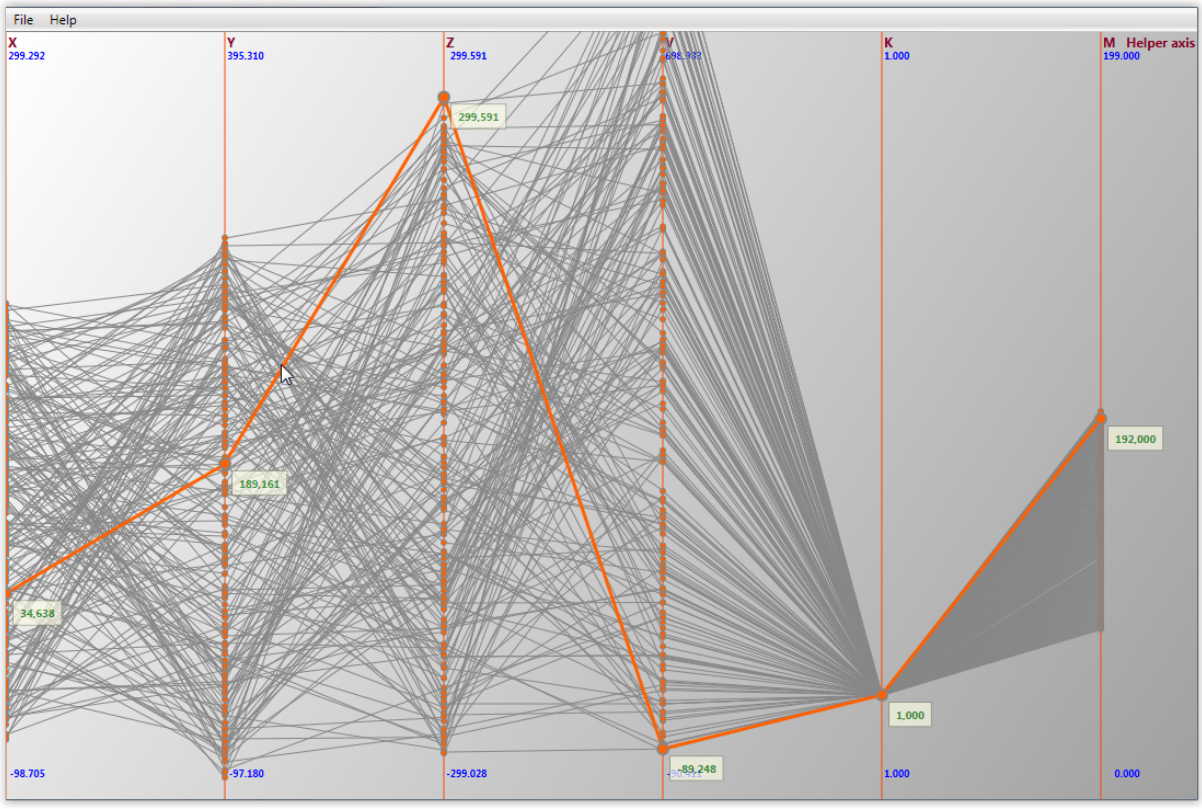


Figure 4.23: The WPF parallel coordinates application implemented by Pitor Wlodek [Wlodek, 2009]. In this view, 200 records are visualised. Points where the polyline segments intersect with the axis are indicated by drawing an ellipse. When a polyline is hovered over, it is highlighted with a different colour and drawn thicker. The corresponding ellipses on all axes are enlarged and show a tooltip with the current value. [Screenshot created by the author.]

Chapter 5

Aggregated Parallel Coordinates

" All animals are equal, but some animals are more equal than others. "

[George Orwell, Animal Farm]

As described in Chapter 2, datasets produced by simulating the influence of different setups on a car's performance on a given racing track are extremely complex. These datasets contain a large number of records and are high-dimensional. Some dimensions have a hierarchical structure, as they contain parameters related to the overall performance, as well as those related to different segments of the track (straights, corners, corner segments) and different parts of the vehicle (front, rear, left, right). The main goal of this thesis was to explore the possibilities for visualisation and exploration of such datasets. The parallel coordinates visualisation technique, described in Chapter 4, is an effective way of visualising multi-dimensional data. However, standard implementations of parallel coordinates cannot deal with dimension hierarchies in the dataset.

Aggregated Parallel Coordinates (APC) is a variation of parallel coordinates, developed at the AVL Racing [AVL, 2015], as a part of this thesis. This technique deals with the hierarchical structure of some dimensions in race car simulation datasets. It also provides filtering features especially designed to better support visual data exploration of race car simulation datasets. This visualisation is implemented in Microsoft .NET framework 4.0, with Windows Presentation Foundation (WPF) [MSDN, 2015e] as the rendering sub-system, and C♯ as the programming language. The use of these technologies was required by AVL. Visual Studio 2013 [Microsoft, 2015c] was used as the Integrated Development Environment (IDE).

The implemented parallel coordinates plot is not intended to be used on its own, but in combination with other visualisations. For this reason, it was developed as a library, which can easily be imported into any WPF application. Aggregated Parallel Coordinates library is already being used as a part of SimBook, a visualisation tool developed by AVL Racing. In addition to the parallel coordinates plot itself, a simple WPF application, named APC Testbed, was also built as part of this thesis to serve as a testbed and demo application. This application enables full interaction with the aggregated parallel coordinates plot, and provides interface to all implemented features. The user guide for this application can be found in the Appendix A.

In the following sections, an overview of the features implemented in APC is presented. Special attention is given to dimension aggregation, the feature which enables visualisation of hierarchies within dimensions of the dataset. This is followed by an overview of the integration of Aggregated Parallel Coordinates into AVL SimBook.

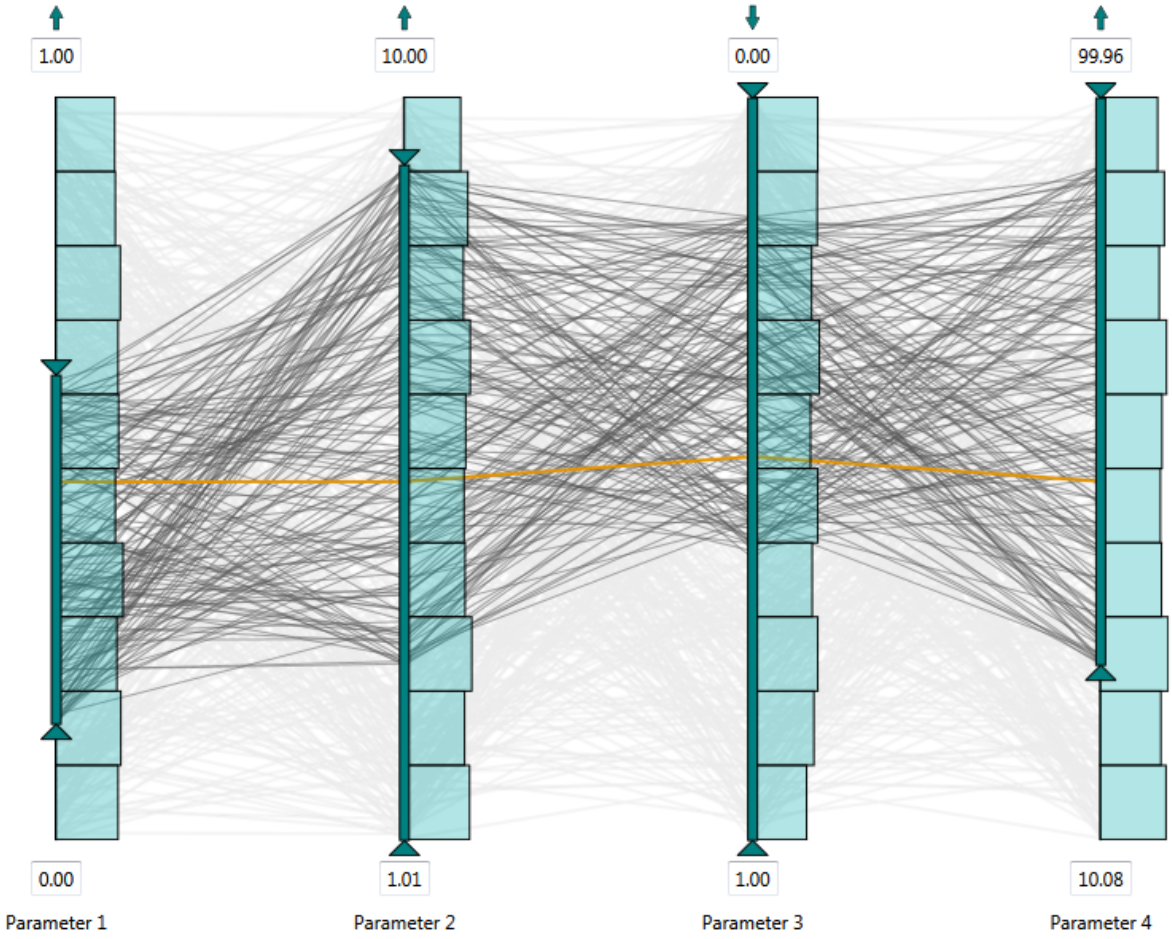


Figure 5.1: Aggregated parallel coordinates (APC). Standard features include axis inversion, filtering records using the sliders, displaying histograms on the axis, and displaying the mean line. [Image created by the author]

5.1 Main Features

The Aggregated Parallel Coordinates, shown in Figure 5.1, provides a series of features and interactions. Some of these features are considered to be standard, since they are implemented in many other parallel coordinates visualisations, while others provide somewhat distinctive functionality. The following interactive features are implemented in the APC:

- **Record Filtering**
  The records in a race car simulation dataset represent values of a particular car setup. Since the main task in visual analysis of race car simulation data is finding the optimal setup, filtering is considered to be the most important interaction. This interaction is implemented by adding sliders to the top and the bottom of each axis. Both sliders can be moved up and down the axis independently. When a slider is moved, a label appears on top of it, showing the current numerical value mapping of the slider's position on the axis. The space between the upper and lower slider works as a middle slider, and can be used to move the selected range along the axis, as shown in Figure 5.2. During the sliding process, the mouse does not have to be directly over the slider, but can be moved freely across the plot. For this reason, the active sliders are highlighted. The sliders can be moved back to their initial position using the axis context menu, shown in Figure 5.5

  The records can be also filtered by changing the axis scaling at a local and global level. When the

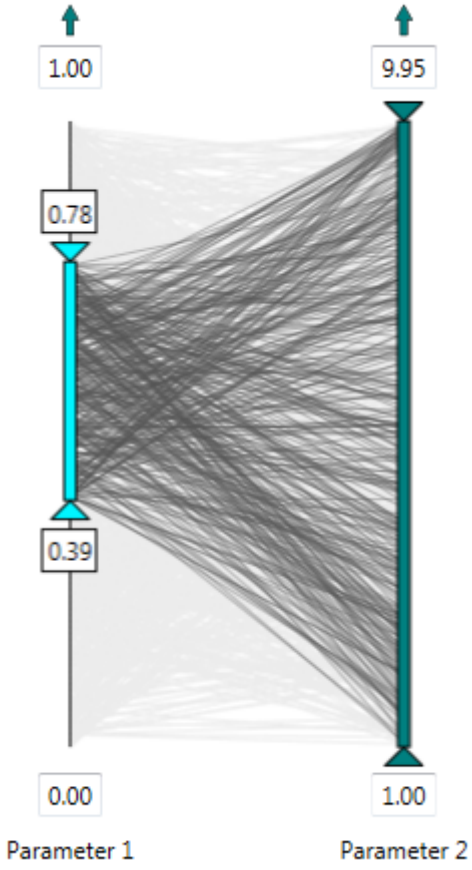


Figure 5.2: The upper and lower sliders on the axes can be moved independently. The space between them acts as a middle slider, which can be used to move the selected value range along the axis. When any of the sliders is activated, a label is added above or below the slider. If the middle slider is being moved, both labels are visible. The label shows the current numerical value of the slider position, and moves with the slider along the axis. [Image created by the author]

axes are scaled at a global level, the same extreme (minimum and maximum) values are applied to all axes. On a local level, an axis can be scaled by inserting the desired record range in one or both extreme labels. Since records which do not belong to the user-selected range are automatically removed from the view, this feature acts as a zoom interaction on a local level. The scaling of axes can also be changed by right-dragging the mouse over the screen. Upon releasing the right mouse button, the axes affected by the drawn rectangle are automatically scaled to the range corresponding to the rectangle's height, as shown in Figure 5.3. This filter interaction supports focusing only on items in the selected range. The scaling can be reset using the axis context menu shown in Figure 5.5.

- **Record Selection**
  Two types of record selection are supported. A single record can be temporarily selected (highlighted) simply by placing a mouse over it. The highlighted polyline is then displayed in a different colour, and the labels with values of the currently selected record are displayed over the record's position on each axis. A record can be permanently selected by control-clicking on the appropriate polyline. Selection of multiple records is possible by left-dragging a rectangle over the polylines which are to be selected. Control-clicking on an already selected record removes it from the set of the currently selected records. Clicking on a blank area deselects all records. Record selection is shown in Figure 5.4.

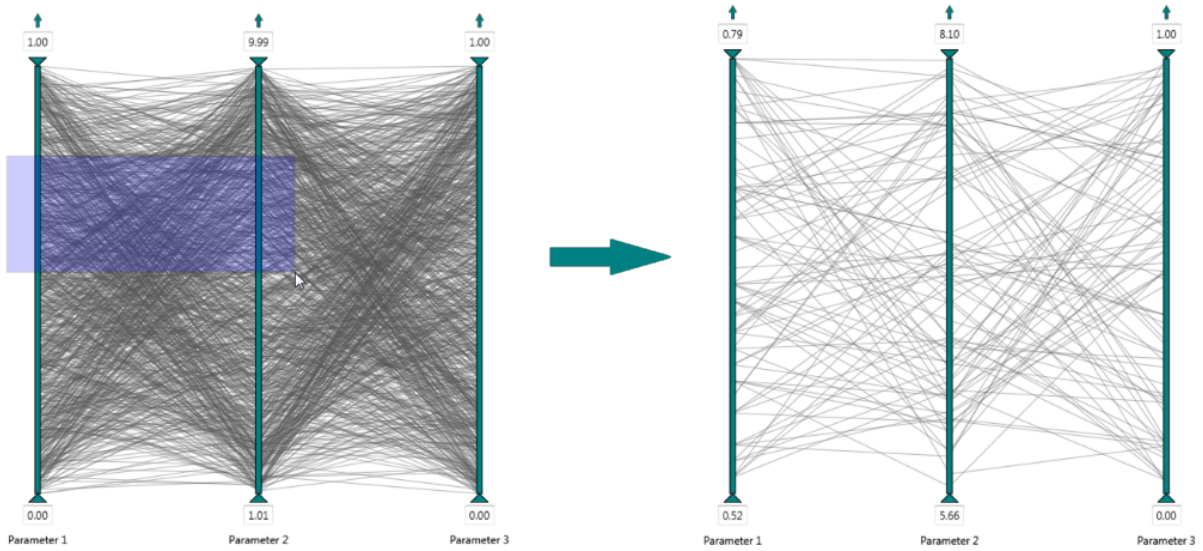


Figure 5.3: By right-dragging the mouse, the affected axes are automatically zoomed to the selected range. The scaling of the other axes is not affected. Records which are out of the selected range are removed from the view. [Image created by the author]

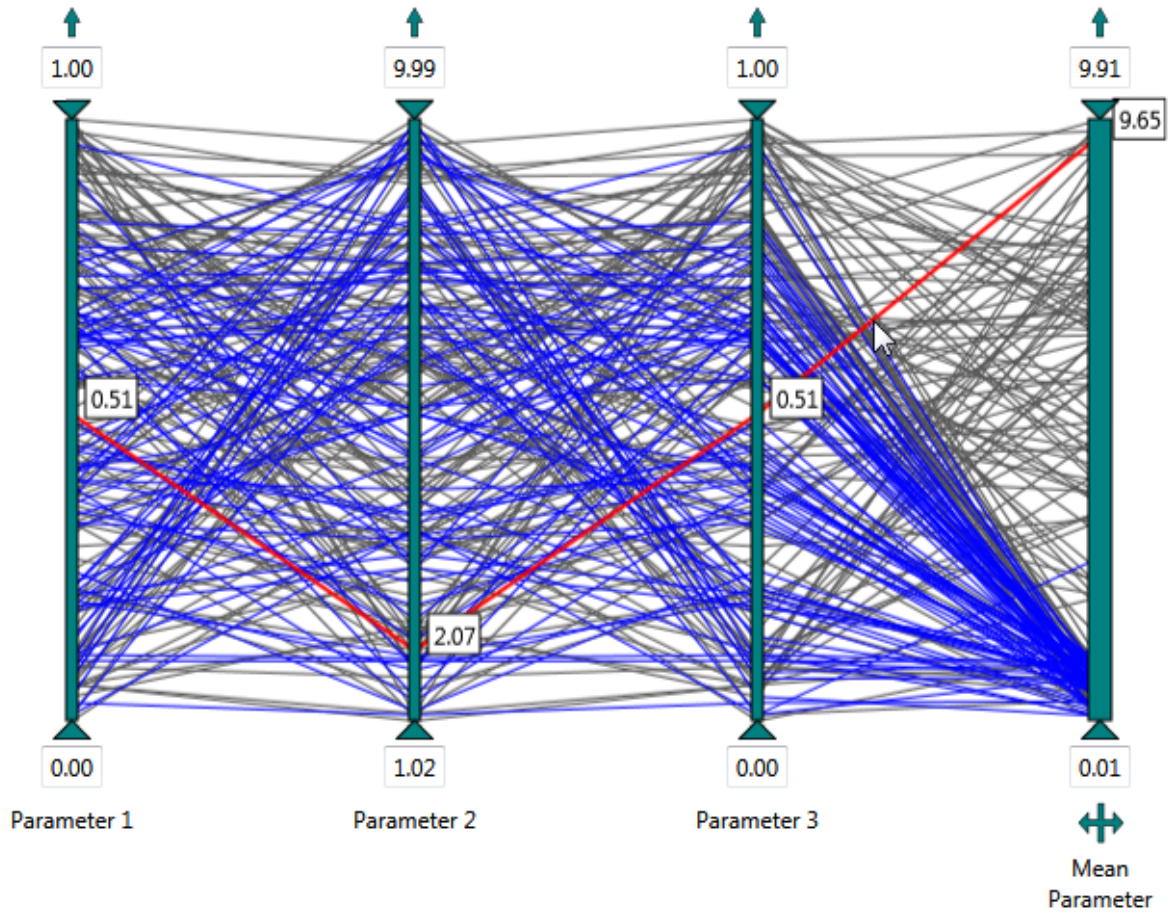


Figure 5.4: APC supports record highlighting and (multiple) record selection. A single record is temporarily selected (highlighted) as the mouse hovers over it, and it is displayed in red. Vales of the highlighted record as displayed in labels on each axis. Other (permanently) selected records are displayed in blue. [Image created by the author]

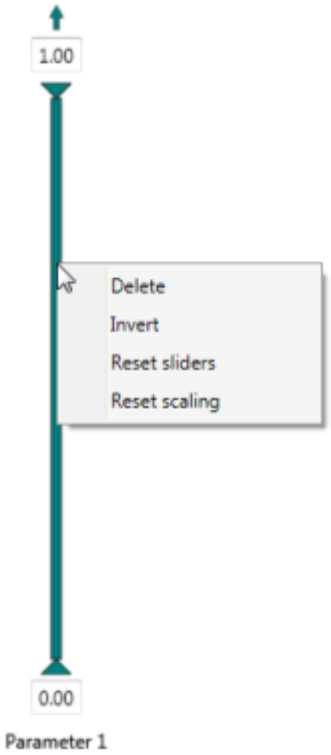


Figure 5.5: The axis context menu supports removing the axis, flipping it, reseting the scaling and moving the sliders to the initial positions. [Image created by the author]

- **Record Context Menu**
  A record context menu, whose contents can be defined externally and passed to the Aggregated Parallel Coordinates library, can be activated by right-clicking a particular record. When an item in the context menu is selected, the particular application using the library is notified. When it receives the notification about the selected item, the application can define what steps should be taken. An example application of this context menu is discussed in Section 5.3.

- **Relative Numerical Representation**
  Apart from the absolute numerical values, records can be represented relative to a specified record. If relative numerical representation is enabled and no reference record has been explicitly defined, the records are represented relative to the mean record. The record context menu can be used to support selection of the reference record. The selected reference record is mapped to the zero position on each axis, and all other records are then mapped relative to this value. This feature is very useful when comparing vehicle setups, because knowing the absolute value is often not as important as knowing by how much the value of a record in a given dimension changes when applying one setup with respect to a selected setup. An example usage of this feature is also described in Section 5.3.

- **Adding/Removing Dimensions**
  Single or multiple dimensions can be dynamically added or removed from the view. Dimensions can be added simply by using the implemented drag-and-drop. When an item is dropped on the view, and the data passed along with the dropped object is a parameter of the dataset, an axis is created and added to the view. Single or multiple axes can be removed from the view using the axis context menu, shown in Figure 5.5.

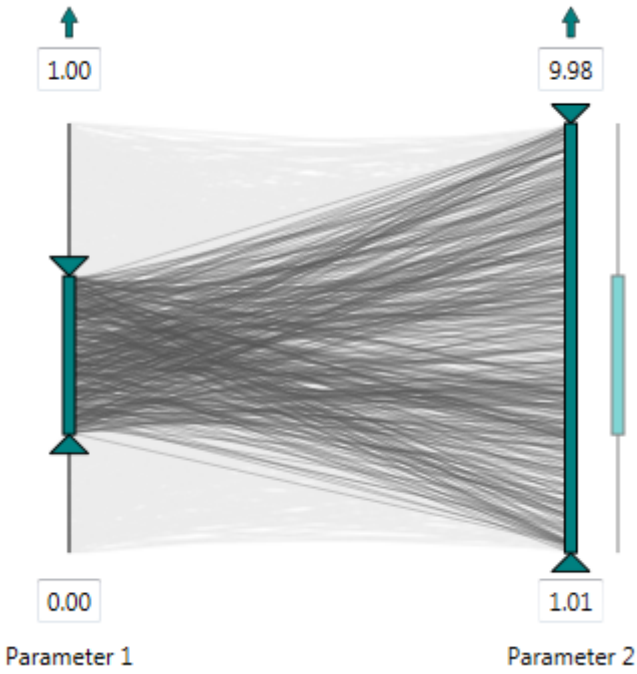


Figure 5.6: The axis for Parameter 1 has been selected for drag-and-drop. Its ghost representation can be moved freely across the plot, while the original representation remains at the same position. When the ghost is dropped, here to the right of the axis for Parameter 2, the original representation is moved to that position. [Image created by the author]

- **Re-ordering Axes**
  Drag-and-drop can be used to move axes across the plot and reorder the axes. When an axis is selected for drag-drop operation, its "ghost" representation is created. The ghost axis can be dragged freely over the view, as shown in Figure 5.6. The original axis representation remains in the same position until the "ghost" is dropped on a different location. The view is then reloaded with the new axis ordering.

  APC can calculate the minimal number of axes orderings which enable comparison of each axis to all the other axes, as described in Chapter 4. The calculated permutations are calculated in the library and can be accessed externally by a particular application using the library, and applied on demand.

- **Axis Inversion**
  By default, minimum record values are mapped to the lower part of an axis, and maximum record values are mapped to the upper part of an axis. This mapping can be inverted by using the arrow button provided on top of each axis. Since these buttons can be removed from the view in order to save space, axis inversion can also be done using the appropriate option in the axis context menu.

- **Adding Histograms to the Axes**
  Histograms can be added to the axes in order to visualise the distribution of the dataset records on the axis. The default number of histogram bins is 10, but this can be changed on demand.

- **Displaying a Mean Line**
  A new (meta) record can be created based on the mean values of the currently displayed records. This record can be visualised in form of a mean line, and dynamically added or removed from the plot.

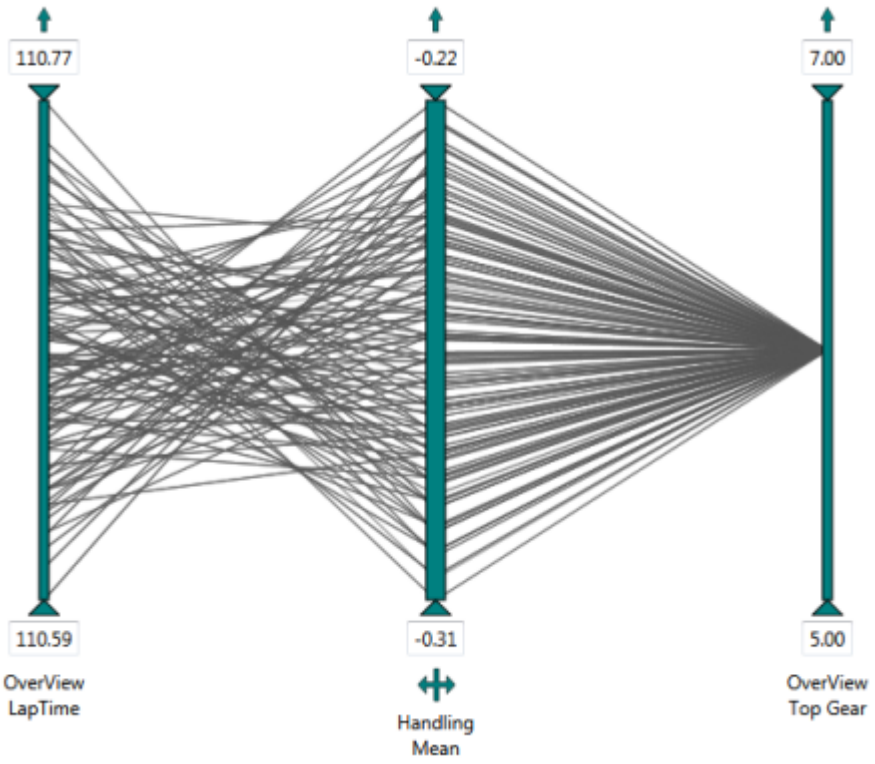


Figure 5.7: Parallel coordinates with an aggregated dimension. The middle axis is thicker than the other ones, indicating that it is aggregated. The button underneath the axis can be used to replace the existing aggregated axis with the axes contained within. [Image created by the author]

- **Changing the Polyline Opacity**
  The opacity of the displayed polylines in the APC can be changed on demand. As described in Chapter 4, reducing the polyline opacity can be used in order to uncover visual clusters when visualising datasets with a large number of records.

5.2 Dimension Aggregation

Hierarchies in parallel coordinates have previously been discussed only in the context of hierarchical clustering of records, as a method for handling large datasets [Ying–Huey, M.O. Ward, and Rundensteiner, 1999]. However, no known research has been conducted on visualising datasets with hierarchies within dimensions.

APC was created to visualise datasets described in Chapter 2 by following the "Overview first, zoom and filter, then detail-on-demand" [Shneiderman, 1996] principle. The "overview" is provided by aggregating elements of a hierarchy into a single dimension. For example, if the dataset contains the same parameter calculated for each corner of a circuit, these dimensions are aggregated and visualised as one dimension. The aggregation is done by calculating the mean values of all records. Aggregated dimensions are treated like any other dimensions. Filtering and other interactions can be applied in the same way as on non-aggregated dimensions. However, aggregated dimensions are visually distinctive, and provide additional interactions. As shown in Figure 5.7, an aggregated dimension has a thicker axis than the other dimensions to indicate that it is a meta-dimension representing or "containing" two or more dimensions.

Aggregated axes are not only marked by their thickness, but also by a button placed underneath them. This "expand" button is used to replace the aggregated axis by its child axes. The child axes can

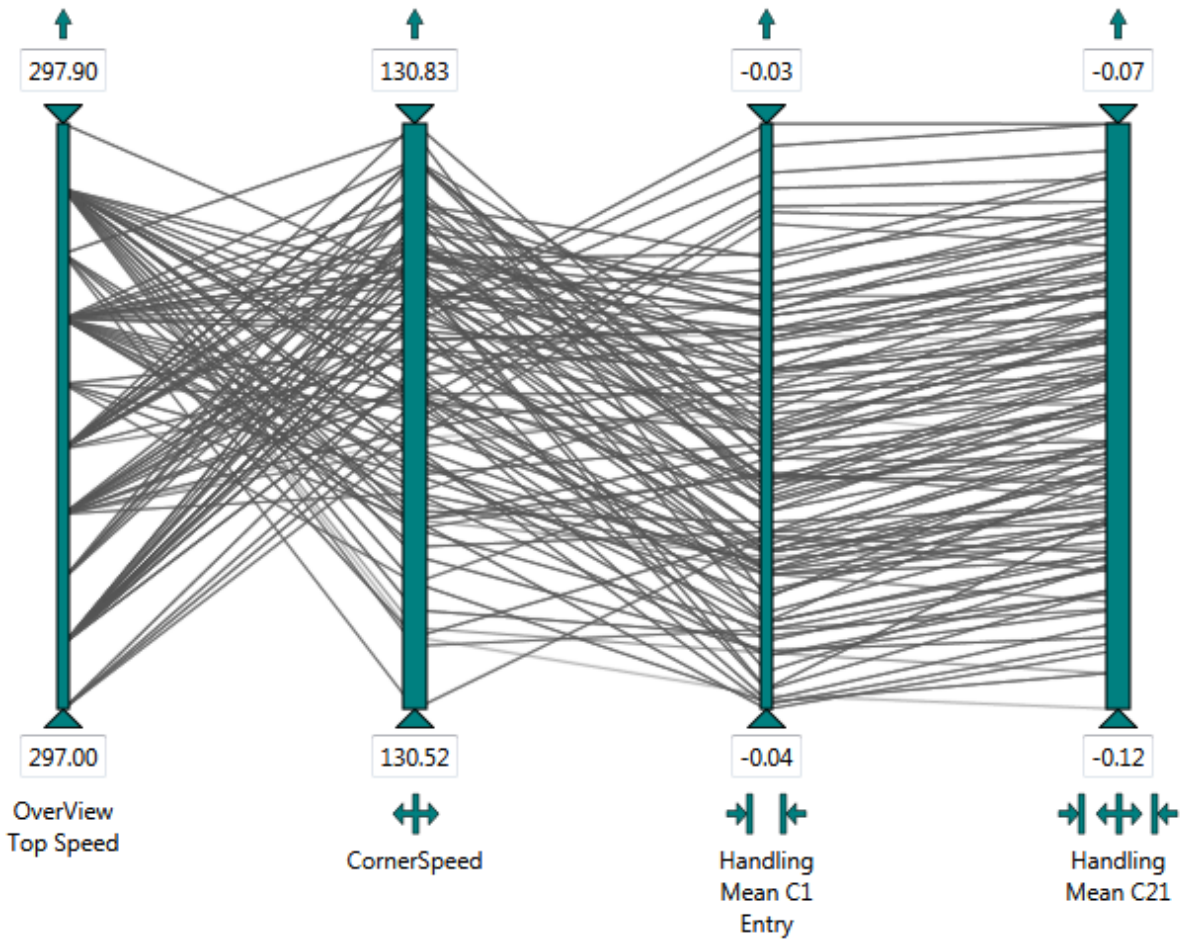


Figure 5.8: Four different types of axes in aggregated parallel coordinates. The first axis does not belong to any hierarchy. The second axis is an aggregated axis. It has a thicker body and a button for expanding. The last two axes are child axes, which have been added to the view by expanding an aggregated axis. The third axis does not contain any children itself, and provides a button for collapsing back to the parent. The last axis is a child axis that has its own children. This axis provides two buttons: one for collapsing back to the parent, and one for expanding (replacing it by its child axes). [Image created by the author]

be easily identified by a button underneath them, which enables aggregating, or "collapsing", the axes back. Any child axis can be an aggregated axis itself. Such child axes have a thick middle slider, and provide buttons both for expanding and collapsing. As shown in Figure 5.8, there are four types of axes in the aggregated parallel coordinates:

- axes that do not have a parent or a child,
- aggregated "root" axes with children but no parent,
- axes with a parent, but no children,
- axes with both parent and children.

Since axes can be freely moved across the plot, expanded axes which have the same parent do not necessarily have to be located next to each other. Identifying which axes have the same parent is possible by hovering over a "collapse" button, which will highlight the sibling axes, as shown in Figure 5.9. In case a sibling axis was expanded even further, that is, replaced by its children, all its child axes are also highlighted, since they would also be removed from the view if the "collapse" button is clicked.

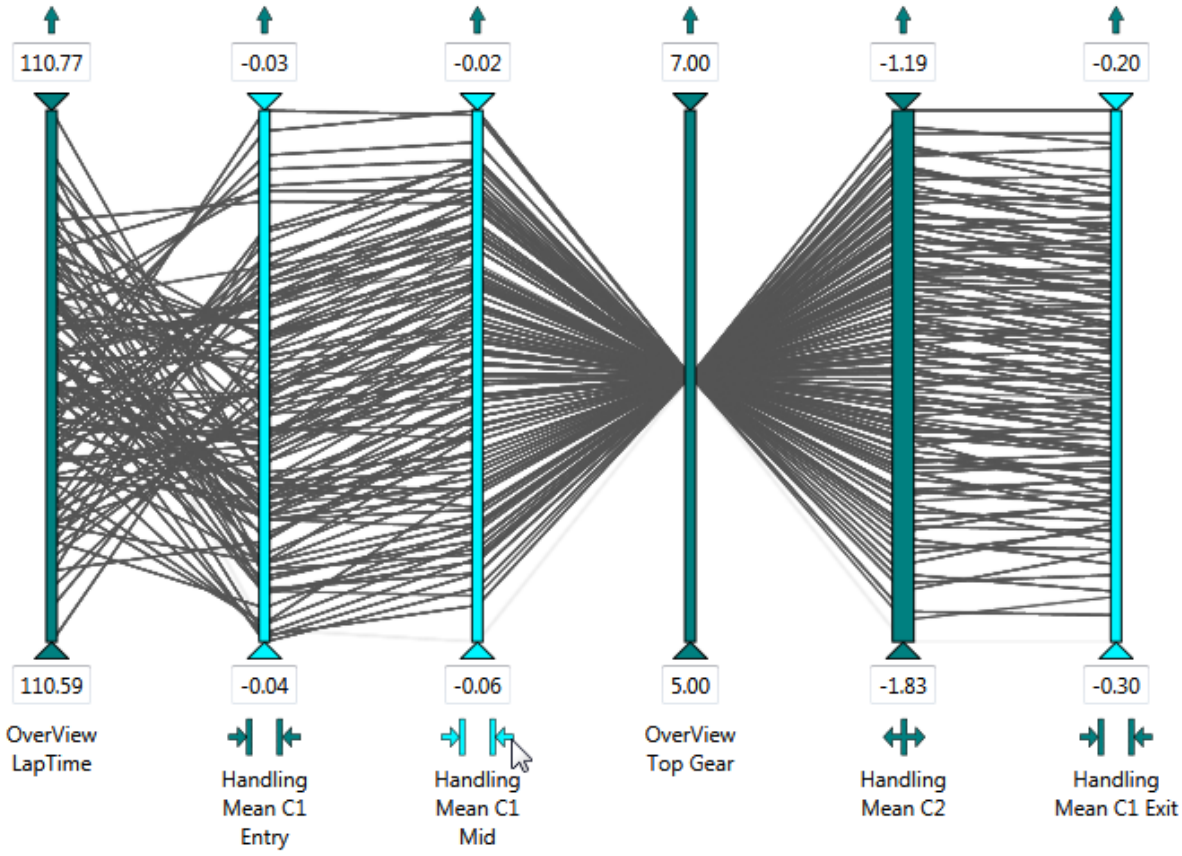


Figure 5.9: Hovering over the "collapse" button highlights the axes that will be collapsed back by clicking on it. This interaction provides easy identification of dimensions at the same level of the hierarchy. [Image created by the author]

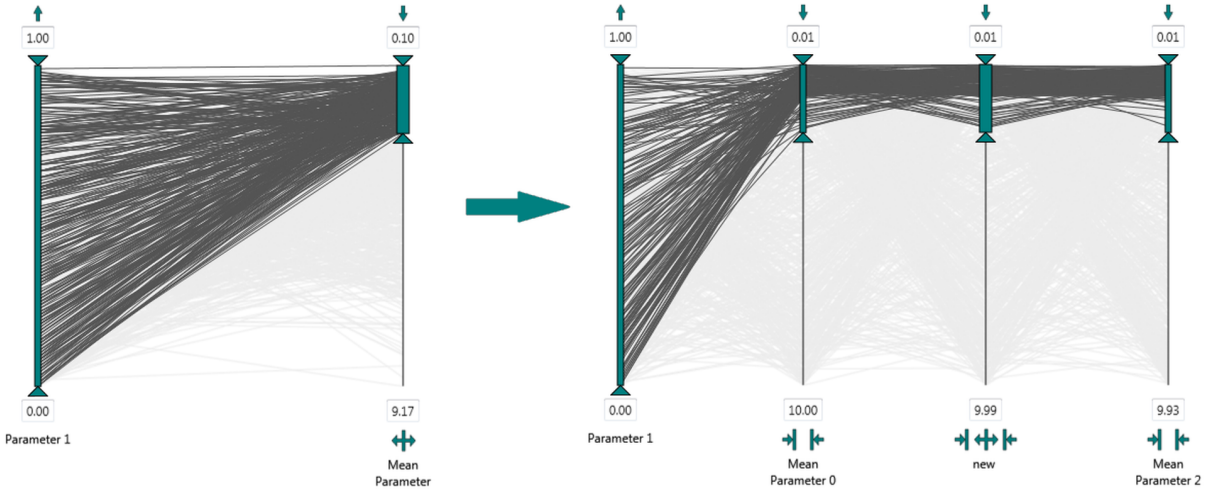


Figure 5.10: The second axis in the left image is an aggregated axis. When the axis is expanded, any scaling, orientation, and current slider values inherited by its child axes. [Image created by the author]

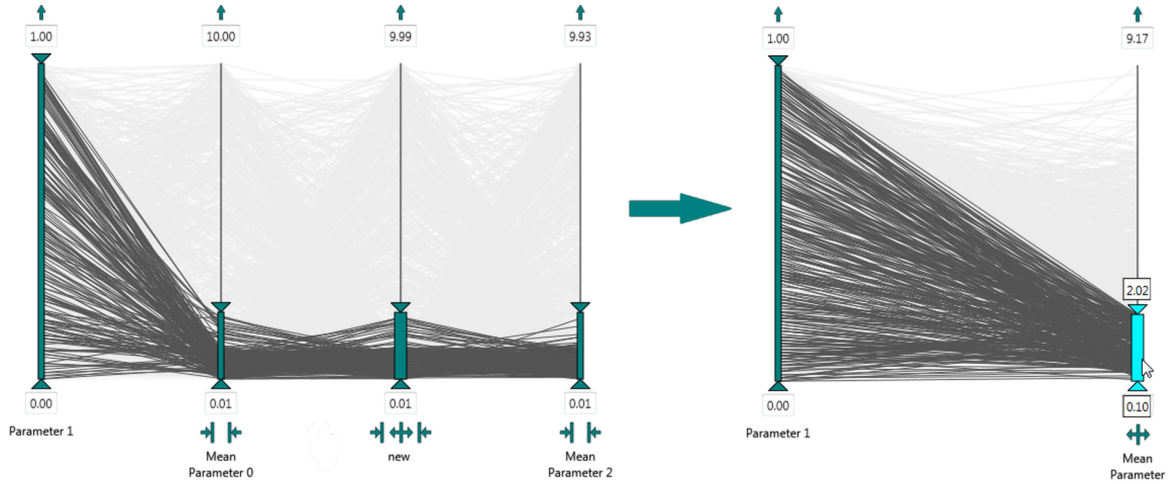


Figure 5.11: When the child axes are collapsed, the minimum and maximum slider values of all children are calculated. The sliders on the parent axis are then set to these values. [Image created by the author]

When an aggregated axis is expanded, its child axes are added at the same position in the plot as the aggregated axis. If the parent axis was inverted at the moment of expanding, all its children will also be inverted. The children inherit any scaling and slider positions from the parent. This behaviour is shown in Figure 5.10. In case the parent axis is collapsed, certain states will be taken derived from its children. If the majority of children are inverted at the moment of collapsing, the aggregated axis will also be inverted. In case the number of inverted children is the same as the number of non-inverted children, the parent axis will have the same state as it had before collapsing. The minimum and maximum values of all sliders on child axes are calculated, and the sliders of the collapsed parent are placed to the corresponding position, as shown in Figure 5.11. The same applies when deriving scaling from child axes.

Based on the axis thickness and buttons provided under the axes, it can be easily determined whether an axis is a child, a parent, or both. However, one cannot easily determine how deep the hierarchy is, or at which level of hierarchy the displayed axes are. In datasets produced by race car simulations, a typical hierarchy is a parameter with values calculated for every segment of every corner in a circuit. In this case, the root axis is an aggregation of every corner in the circuit, and has as many children as the circuit has corners. Since every corner of the circuit has three segments, every child parameter of the

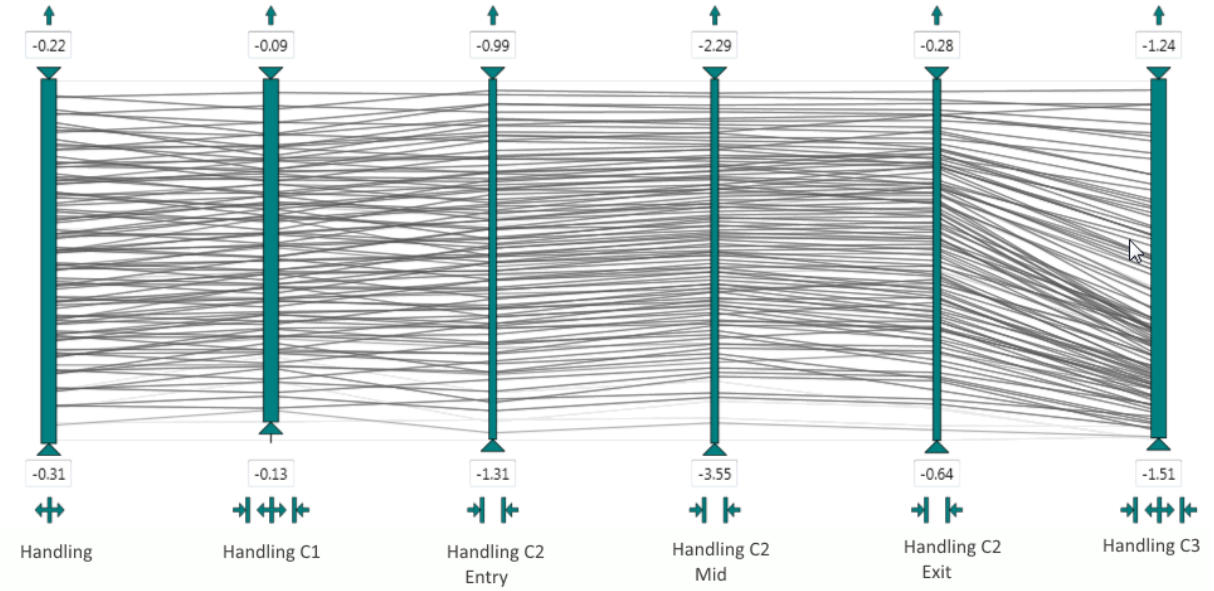


Figure 5.12: A portion of a typical race car simulation dataset visualised with aggregated parallel coordinates. The first dimension (Handling) is an aggregated dimension and represents the root of a parameter hierarchy. Expanding this dimension replaces it with its child nodes: Handling C1, Handling C2, and Handling C3 which, in this case, represent three corners of the track. These three dimensions are themselves aggregated (meta) dimensions, since they are created by calculating mean values of the three segments (entry, mid, and exit) of corners 1, 2, and 3. Handling C2 Entry, Handling C2 Mid, and Handling C2 Exit are the leaf elements of the parameter hierarchy and have been added to the view by expanding dimension Handling C2. Although the hierarchy level is not explicitly visualised, users can easily identify which level of hierarchy is represented by an axis based on the parameter names. [Image created by the author]

root will be an aggregation of three axes: entry, mid and exit. Hence, the level of hierarchy of currently displayed dimensions, as well as the child count, are both evident from the dimension naming, as shown in Figure 5.12.

For those cases in which the level of hierarchy is not inherited from the data itself, an attempt was made to encode the information about the current hierarchy level in the axis height. There were several problems with this approach. Having axes of different heights in the same view affects the possibility to compare the axes directly. If one record has the same value in adjacent dimensions, this would be indicated by the straight line between the axes, provided that the axes have the same scaling and the same height. This, and other similar patterns, cannot be identified as easily if the axes do not have the same height. Furthermore, the axes heights would have to be significantly different in order to enable easy identification of the hierarchy level. This would produce either very tall or very small axes when visualising deep hierarchies.

Another possibility to encode the information about the hierarchy depth would be to use colour coding. This was not implemented, because colour coding is already used in AVL SimBook to encode other kinds of information. Using different colour intensities to encode this information was also considered, but not implemented, since it would make the comparison between the items in deep hierarchies difficult. Conveying information about the current hierarchy level within the parallel coordinates plot does not seem possible without giving up on the advantages of its general properties. Connecting the parallel coordinates plot to a separate hierarchy visualisation would be the only possibility to visualise the level of hierarchy of the displayed axes.

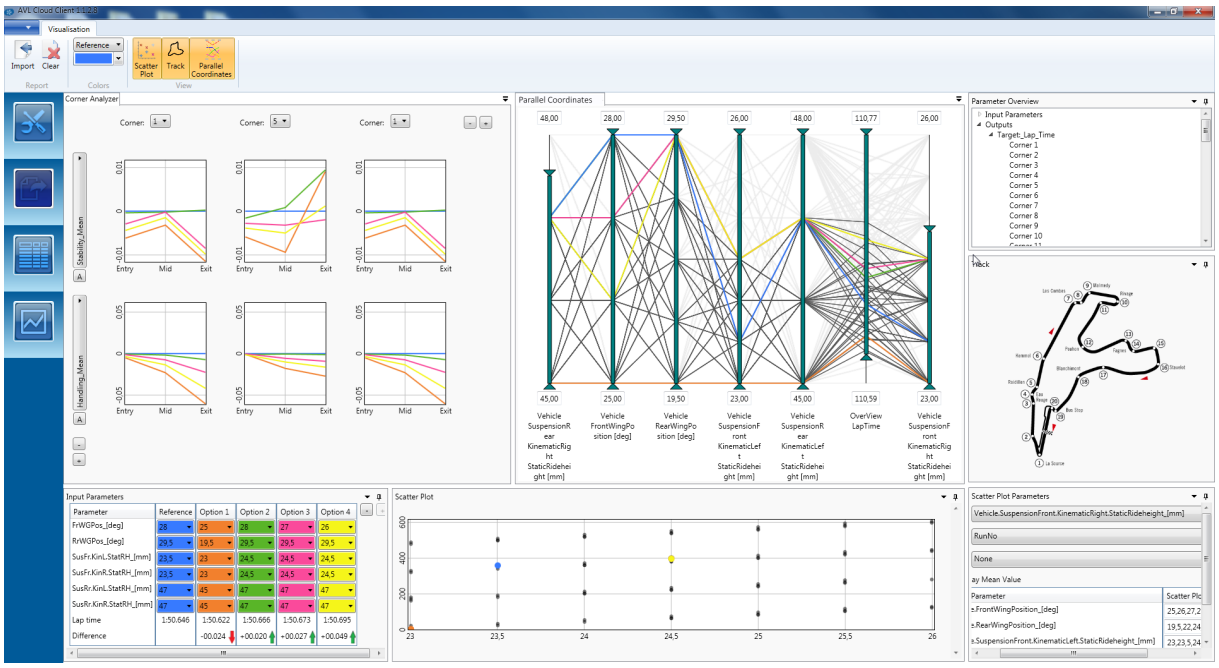


Figure 5.13: SimBook provides several different visualisations to support visual exploration of race car simulation datasets. These visualisations include a small multiples representation of corner segment values (top left), parallel coordinates (top middle), parameter tree view (top right), track (right), setup table (bottom left), and scatter plot (bottom middle). Users can include, remove, resize, and reposition the views according to their needs. [Image created by the author]

5.3 Application of APC in SimBook

SimBook is a tool developed by AVL Racing, which is especially designed for visual exploration of race car simulation data. SimBook was developed independently but in parallel with this thesis, and many details of its design are a result of research conducted as part of this thesis. As shown in Figure 5.13, SimBook consists of the following basic panels:

- corner analyser,
- parameter tree,
- setup table,
- parallel coordinates, and
- scatter plot.

Each of these panels can be moved independently, docked at any position within the window, tabbed next to other views, or displayed as a separate window. This enables customisable layout positioning which can be adapted to screens of all sizes. Single views can be dynamically added, removed, or be hidden from the layout.

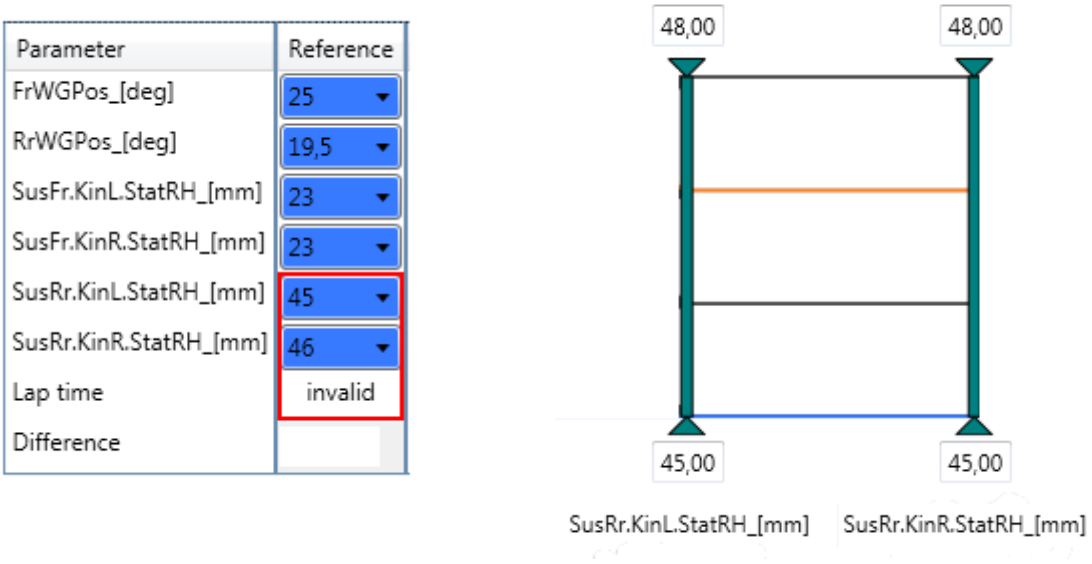
The parameter tree view provides a hierarchical overview of all parameters within a dataset. At the top level, parameters are sorted into two categories: input and output. Input parameters are part of the car setup. The output parameters were calculated by the simulation based on the car setup. Output parameters sometimes have hierarchical structure within the dimensions. This means, for example, that the parameters which are calculated for all straight segments or all corners of the circuit are contained in sub-categories within the tree view. Expanding a corner category, shows the segments of the corner.

In the setup table, input parameters are displayed as rows, while columns represent different combinations of the input parameters (setups). The setup table always contains at least two columns. The first column represents a reference setup, while all other columns are optional setups. The last two rows of the setup table show the time in which the car would finish one lap of the track with that setup, as well as the difference to the lap time of the reference setup. Values within the setup column can be changed by selecting an item from the drop-down menu in a corresponding table cell. Each setup is colour-coded.

If the parallel coordinates view is enabled, it is automatically populated with the input parameters. The polylines corresponding to the currently selected reference and optional setups are displayed in the appropriate colour. The polyline context menu can be used to change the reference, or any of the optional setups. Changing the setup in the parallel coordinates view automatically changes its values in the setup table. Although the setup table and the parallel coordinates both display the same information, there are several advantages to using the parallel coordinates representation to select the reference and the optional setups. One advantage is that all setups (not only the reference and the options) are displayed at the same time. While the setup table visualises only the difference in the lap time between the reference and the optional setups, parallel coordinates enable easy comparison of lap times for all setups, enabling the user to easily identify and filter out the setups which result in too high lap times. Furthermore, the input set can contain parameters which are directly dependant. One example are the setups with parameters that have to have the same value for both the left and right side of the vehicle. Such parameters can be easily identified in the parallel coordinates view, provided that they are placed next to each other. In the setup table, on the other hand, it is possible to select one value for the left and a different one for the right side of the vehicle, but such actions will result in an "invalid lap". The same error will be shown in case a selected combination of parameters is missing from the dataset, for whatever reason. This cannot happen in parallel coordinates view, since only valid records are displayed. This behaviour is shown in Figure 5.14. Using the setup table for setup selection is an advantage in case the user needs to explore a particular setup, for which the input values are known. Furthermore, the setup table provides a good overview of the absolute values of the currently selected setups.

Additional parameters (dimensions) can be added to the parallel coordinates panel, simply by dragging the parameter from the tree view and dropping it on the plot. If the parameter containing subdimensions is dropped on the panel, the parameter is displayed as an aggregated axis. Apart from the complete input and output sets, parameters at any level of hierarchy can be individually added to the view.

The Corner Analyser is a separate panel (see Figure 5.13), which uses a line chart matrix to represent the values of parameters in different segments of the circuit corners. The matrix can contain up to 5 rows and columns of line charts. One parameter can be selected for each row, using the drop-down menu on the left. A corner can be selected for each column. All the line plots within one row have the same scaling. The parameters can be represented as absolute and relative values. In the relative value representation, the reference setup has zero values and is displayed in the middle of the plot, while the values of other setups show the difference with respect to the reference value. This enables direct comparison between the entry, mid, and exit segments of different corners for a given parameter. In the corner analyser, only the reference and optional setups are represented. For any selected corner, a detail view can be displayed. The details are shown as parallel coordinates in a separate panel, by double clicking on a plot. An example of a detail view panel is shown at the right side of the Figure 5.15. This parallel coordinates plot contains three axes: entry, mid, and exit segments of the selected corner. An axis which shows the overall lap time can be added to the plot, by clicking on the check box under the plot. Furthermore, any other parameter can be added to the detail view using the drag-and-drop. If the plot in the corner analyser is displaying the relative values, the detail view will also show the relative values. Changing this setting in the corner analyser will automatically change the value representation in the detail view. If the plot for which details are displayed uses absolute values, the detail view will apply global scaling on the parallel coordinates. The scaling can be changed by clicking on the appropriate check box under the plot. Just like the parallel coordinates panel with the input and other parameters,



(a) In the dataset, the rear left and rear right suspensions have the same value. If the same values are not selected in the setup table view for both parameters, the lap time entry is set to "invalid".

(b) The parallel coordinates visualisation clearly shows that the rear left and rear right suspensions have the same values in all records of the dataset.

Figure 5.14: In the setup table, selecting a value combination which is not available in the dataset causes an error. This cannot happen when selecting setups with parallel coordinates, since only records available in the dataset are visualised.

[Images created by the author]

the detail view panel can also be used to change both the reference and the optional setups. The two parallel coordinate panels are synchronised, which means that when one or more parameters are selected in the detail view, the same parameters are automatically selected in the other parallel coordinates view, as shown in Figure 5.16.

As seen from the overview given above, aggregated parallel coordinates allow easy setup filtering, and provide an effective representation of the dataset hierarchies. This improves the process of visual exploration of race car simulation data significantly. Many features in the APC were developed specifically for use in SimBook. These features include the relative value representation, global axis scaling, and polyline context menus. On the other hand, features like inverting axis or displaying histograms on the axes, are available, but typically not used when exploring these datasets. Since race car simulation datasets tend to contain thousands of records, special attention was given to performance optimisation of APC. This is discussed in the next chapter.

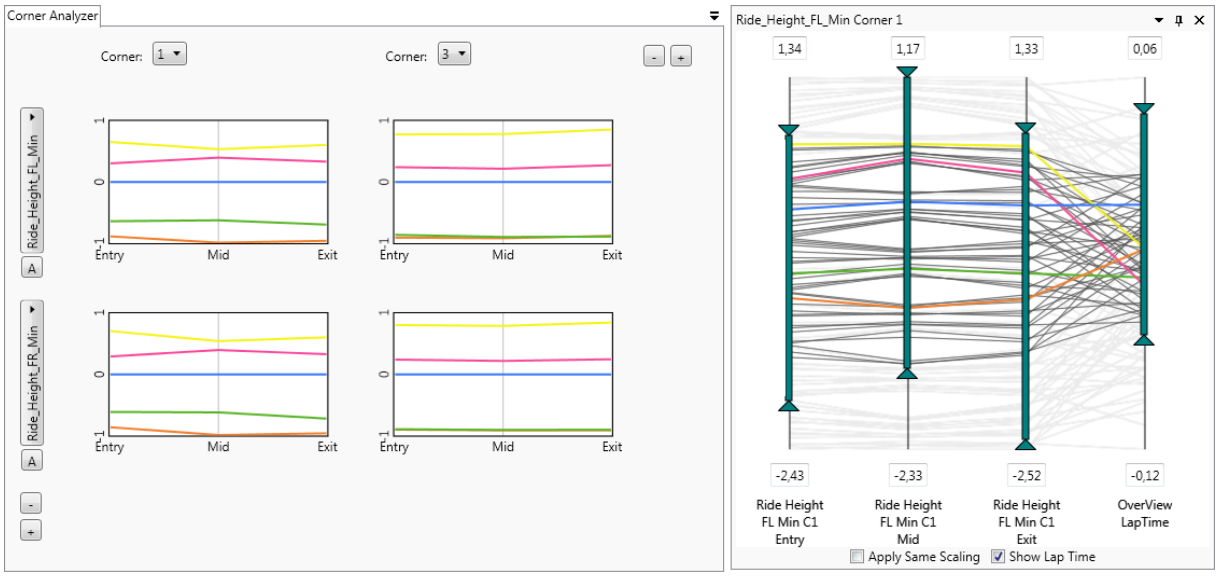


Figure 5.15: The Corner Analyser displays a line chart matrix for selected parameters in selected corners. The "detail view" for every corner segment plot in the Corner Analyser is provided in an additional parallel coordinates panel. [Image created by the author]

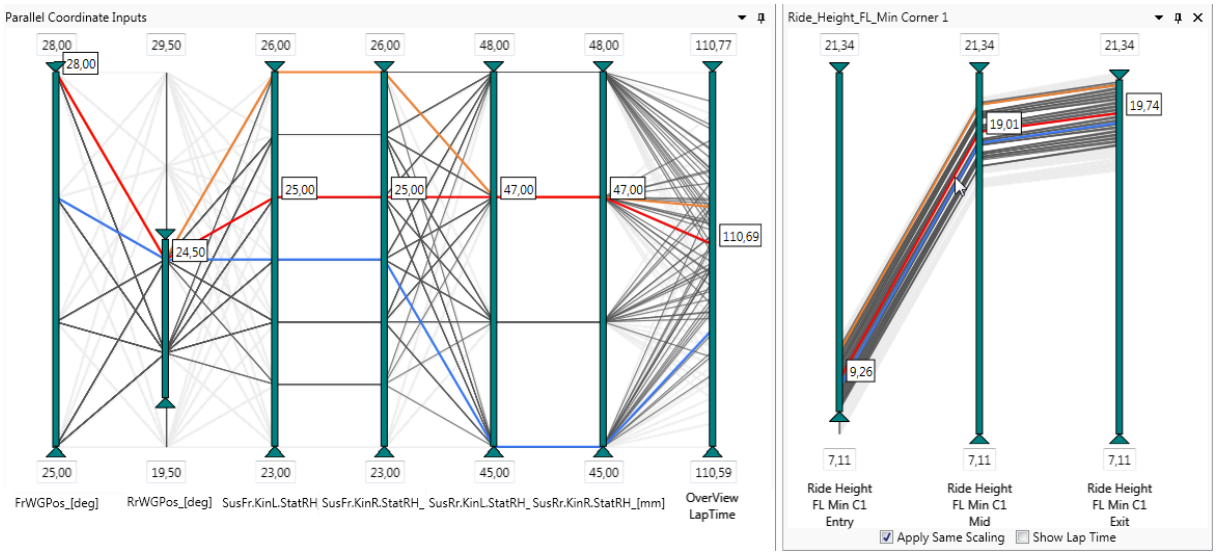


Figure 5.16: The parallel coordinates panels in SimBook are synchronised. Selecting a polyline in one panel highlights it automatically in the other panel. Items which are filtered out in one view are also filtered out in the other view. [Image created by the author]

Chapter 6

Performance Optimisation

"Before software can be reusable it first has to be usable."

[Ralph E. Johnson - Computer scientist]

Windows Presentation Foundation (WPF) was introduced as a part of .NET framework 3.0. WPF is a library which provides a comprehensive set of features for building Windows client applications. In its core, WPF has a resolution-independent, vector-based rendering engine. One of the most important properties of WPF is the separation of application appearance and its behaviour. The appearance is generally defined using Extensible Application Markup Language (XAML), while the behaviour is implemented either in C♯ or Visual Basic. The connection between the appearance and the behaviour logic is established by means of data binding. When the data is updated in the business logic, the changes are automatically reflected to the appearance (provided that the appropriate notification event is triggered), and vice versa.

To access the machine hardware, WPF uses Microsoft DirectX APIs. Depending on the available graphics hardware, a WPF application can be rendered in three tiers [MSDN, 2015d]:

- Rendering tier 0 - The entire application is rendered in software.
- Rendering tier 1 - Hardware acceleration is used for rendering some graphic objects.
- Rendering tier 2 - Hardware acceleration is used for rendering most of the graphic objects.

Tiers 1 and 2 are supported with DirectX version 9.0 or higher. When supported, hardware acceleration is used for most of the 2D rendering, as well as for 3D rasterisation and other 3D features.

WPF provides a large number of user interface elements. These elements can be used as they are provided, combined with other elements, or extended according to the needs of the developer. Templates can be defined for controls (user interface elements such as buttons), in order to override the default appearance, or to define how custom-created controls should be presented. Groups of existing or custom-made controls can be combined into a single control, called a user control. User controls can be built as libraries and imported into other applications. The aggregated parallel coordinates (APC) plot was created using WPF and C♯, and is implemented as a user control library.

The focus of this chapter is the performance of aggregated parallel coordinates. The first implementation of APC yielded very bad performance. Memory consumption was very high, and responsiveness was low when visualising datasets with large number of records. In the following text, the first implementation of the plot is explained, and the causes of the low performance are identified. The steps taken in order to improve the performance are explained in detail. This is followed by a short overview of the achieved results.

6.1 Initial Implementation and Performance Problems

In the early stages of implementation, research on any available parallel coordinates implementations in WPF was conducted. The results were very limited, and the only implementation found was that implemented by Wlodek [2009], which is described in Chapter 4. Since the source code of this implementation is available on GitHub [Wlodek, 2010], it was examined and taken as a starting point for the implementation of the aggregated parallel coordinates.

Wlodek's implementation was done in .NET framework version 3.5. It was implemented by extending the Control class, which is a base class for user interface elements whose appearance is defined by a ControlTemplate [MSDN, 2015c]. By defining custom control templates, both the appearance and the behaviour of each visual element (control) can be manipulated when certain events are triggered. In a control template, the visual appearance (style) of the Chart control was defined as a Canvas (a control which defines an area upon which child elements are placed) consisting of collections of axes, points and lines. Styles are defined for ChartLine, Axis, ChartPoint, and other controls.

The aggregated parallel coordinates were initially implemented by following a similar principle. The plot itself was implemented by deriving from UserControl. This class is used to declare a set of controls. It consists of XAML code and a code-behind file (a file with extension `.xaml.cs`) [Moser, 2015]. The inital XAML code for the aggregated parallel coordinates is shown in Listing 6.1. A top-level Grid element defines a drawing area for the plot. Three collections of items were placed on top of it: a collection of Axis objects, a collection of polylines (AxisPointConnection objects), and a collection of AxisPoint objects to represent the data points. The code-behind contained the logic for resizing the plot, as well as the logic for managing the collections of visual controls. Axis, AxisPoint, AxisPointConnection and other visual elements were implemented by deriving from Control class and defining a Style within a ControlTemplate. The Style for AxisPointConnection, shown in Listing 6.2, defined it as a Poyline shape, whose colour, line thickness and position properties were set by data binding. The same approach was used for every control.

While this approach works well with a small number of records on a relatively static chart, performance issues appeared as soon as the filtering interaction was implemented and tested on a few hundred records. The main reason for these issues was the number of times single visual objects were redrawn. When using sliders to filter records, visual properties such as polyline colour and depth-index inside the canvas were simultaneously changed on a large number of polylines. Slider properties such as position, label margin, and label content were also updated with every slider movement. Furthermore, the colour of the axis points was also changed for those points which were filtered-out. Thus, filtering the records caused a large number of visual elements to be re-rendered several times, resulting in reduced responsiveness to user interactions.

6.2 Optimisation Steps

As soon as the performance problems were detected, the collection of AxisPoint controls was removed from the plot, since showing single points on axes takes many resources and is a redundant feature. This minor optimisation improved memory consumption, as well as the application responsiveness when filtering the records. However, the overall performance was still not satisfactory, even when rendering was done in tier 2. Further analysis was required in order to detect the cause of the performance issues, based on which a series of optimisation steps was implemented. These steps are described in the following sections.

```
<UserControl x:Class="ParallelCoordinates.ParallelCoordinatesPlot"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006
                 "
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             mc:Ignorable="d"
             x:Name="control"
             >
 <ScrollViewer>
  <Grid>
   <Canvas Name="chartCanvas">
    <ItemsControl ItemsSource="{Binding ElementName=control, Path=Axes}">
     <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
       <Canvas />
      </ItemsPanelTemplate>
     </ItemsControl.ItemsPanel>
    </ItemsControl>

    <ItemsControl ItemsSource="{Binding ElementName=control, Path=ChartPoints}">
     <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
       <Canvas/>
      </ItemsPanelTemplate>
     </ItemsControl.ItemsPanel>
    </ItemsControl>

    <ItemsControl ItemsSource="{Binding ElementName=control,
                                        Path=AxisPointConnections}">
     <ItemsControl.ItemsPanel>
      <ItemsPanelTemplate>
       <Canvas/>
      </ItemsPanelTemplate>
     </ItemsControl.ItemsPanel>
    </ItemsControl>
   </Canvas>
  </Grid>
 </ScrollViewer>
</UserControl>
```

Listing 6.1: The inital XAML code for aggreagated parallel coordinates. `Grid` defines a drawing area for the plot. Three collections of items are drawn: `Axes` representing dataset dimensions, `AxisPointConnections` representing dataset records, and `ChartPoints` representing single data items.

```
<Style TargetType="{x:Type pc:AxisPointConnection}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type pc:AxisPointConnection}">
        <Polyline
         Stroke="{Binding
                  RelativeSource={RelativeSource TemplatedParent},
                  Path=Color}"
         StrokeThickness="{Binding
                           RelativeSource={RelativeSource TemplatedParent},
                           Path=LineThickness}"
         Points="{Binding
                  RelativeSource={RelativeSource TemplatedParent},
                  Path=Points}"/>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Listing 6.2: Style definitin for AxisPointConnection in the initial APC implementation. Here, AxisPointConnection is defined as a polyline, whose stroke, stroke thickness and layout is defined by data binding.

6.2.1 Replacing Heavy-Weight Shape By Light-Weight Geometry

The first step taken in order to optimise the performance of the aggregated parallel coordinates was changing how the visual objects are drawn. The initial implementation consisted of controls with templates which were drawing objects such as Polyline, Rectangle, etc.. All these objects derive from the Shape class, which is considered to be heavy-weight, since it provides features such as layout and event handling [MSDN, 2015f]. The light-weight alternative to drawing 2D shapes is using Geometry and classes which derive from it. These objects are considered to be light-weight, because they cannot be rendered automatically when its properties change: they have to be explicitly drawn inside a DrawingContext. However, because they are drawn explicitly, the system does not have to allocate resources for monitoring their state. Since the visual elements in the aggregated parallel coordinates are always drawn in a same way (for example, AxisPointConnections are always drawn as polylines), defining control templates is a redundant feature. So, instead of deriving from the Control class, all visual elements (like AxisPointConnections) derive from FrameworkElement class. This class is one level above Control in the inheritance hierarchy, and does not support templates. Custom FrameworkElement objects are drawn by overriding the OnRender(DrawingVisual) method. This method is automatically called whenever InvalidateVisual() is called. Inside the OnRender method, the elements are not drawn as Shape, but as Geometry objects. Geometry class inherits from the Freeze class, which means that these objects can have a modifiable, and a read-only (frozen) state. Modifiable objects have to be monitored by the system in order to update any unmanaged resources owned by such objects. By calling the Freeze() method, the object is declared immutable. Immutable objects are read-only, and their state cannot be changed, so the system does not have to spend any resources monitoring changes made upon them. In the optimised implementation, AxisPointConnection implements the OnRender method as shown in Listing 6.3.

6.2.2 Reducing The Number of Render Calls

The next step in performance optimisation was reducing the number of times the application is renders visual objects. To better support interaction with other visualisations, the visual properties of each record are defined in a class called RecordProperty, while the polylines themselves are drawn in the AxisPointConne

```
protected override void OnRender(DrawingContext drawingContext)
{
  //... initial checks
  StreamGeometry geo = new StreamGeometry();
  using (StreamGeometryContext context = geo.Open())
  {
  context.BeginFigure(_points[0], true, false);
  context.PolyLineTo(_points.Skip(1).ToArray(), true, true);
  }
  geo.Freeze();
  drawingContext.DrawGeometry(null, new Pen(RecordProperty.Color, RecordProperty.
      LineThickness), geo);

  //...draw labels if the item is highlighted
}
```

Listing 6.3: AxisPointConnection in the optimised APC implementation overrides the OnRender method of its parent FrameworkElement class. Polylines are drawn within StreamGeometryContext. By calling the Freeze() method, the StreamGeometryContext is declared immutable, so system does not have to spend resources by monitoring property changes on this object.

ction class. Whenever a record is hovered over, or filtered in or out, one of the three methods implemented by RecordProperty is called: AppearInBackground(), AppearHighlighted(), or AppearDefault(). These methods set a series of flags which define visual and other properties of the records, as shown in Listing 6.4. When the setter of a property is called, the property value is updated and an OnPropertyChanged event is triggered, as shown in Listing 6.5.

In the initial implementation, the style which was defined the appearance of the AxisPointConnection, used data binding to retrieve the current value of any visual properties. Any changed properties were identified by the property name, which is passed as a parameter, so triggering OnPropertyChanged("Color") event, for example, would notify the system that the colour of the polyline was changed. The system would then re-render the polyline with the new colour. Since automatic re-drawing is not possible in the optimised implementation, the system does not have to be notified every time each property changes. Instead, it is enough to notify the AxisPointConnection that re-drawing is necessary when the state of a record changes. The optimised AppearDefault method is shown in Listing 6.6. Instead of calling the setter of the properties, their fields are set directly, so that the properties do not trigger a property changed event. The OnPropertyChanged("") event is triggered only after all flags and values have been set. An empty string is passed instead of the name of the property. This notifies the listener that some property has changed, but not which one. Inside AxisPointConnection, this event is handled as shown in Listing 6.7. AxisPointConnection does not check the name of the changed property, it only sets internal parameters, such as hit test visibility, according to the current state of the RecordProperty object, and calls InvalidateVisual(), which triggers re-rendering of the affected polyline. So, instead of re-drawing the polyline six times, for six properties set within the AppearDefault() method, this is done only once.

6.2.3 Layers and Bitmap Caching

Another significant reduction in the number of times each AxisPointConnection is redrawn was achieved by placing the polylines into layers. Since displaying the filtered-out records in the background can provide significant help in the process of visual exploration, not displaying the filtered-out records would affect the usefulness of the plot. In order to solve this problem two layers of polylines are drawn: one in the background, and one in the foreground. The polylines in the background are drawn as one FrameworkElement consisting of many polylines. This means that, for example, clicking on the background

```
public void AppearDefault()
{
  LineThickness = DefaultLineThickness;
  Color = DefaultColor;

  if (!_hasCustomColor)
  {
    ZIndex = _defaultZIndex;
  }
  else
  {
    MAX_Z_INDEX++;
    ZIndex = MAX_Z_INDEX;
  }

  IsMultiSelected = false;
  IsInBackground = false;
  IsHighlighted = false;
}
```

Listing 6.4: The initial implementation of AppearDefault method in RecordProperty class. Colour and line thickness are set to default values. Depth index is set to maximum, and flags indicating whether the record is multi-selected, in background or highlighted are set to false.

```
private SolidColorBrush _color;
public SolidColorBrush Color
{
  get { return _color;}
  private set
  {
    _color = value;
    OnPropertyChanged("Color");
  }
}
```

Listing 6.5: The Color property definition within RecordProperty class. When the value of the property is set, the OnPropertyChanged event is triggered, notifying the system that the colour of the record has changed.

```
public void AppearDefault()
{
  _lineThickness = DefaultLineThickness;
  _color = DefaultColor;
  if (!_hasCustomColor)
  {
    _zIndex = _defaultZIndex;
  }
  else
  {
    MAX_Z_INDEX++;
    _zIndex = MAX_Z_INDEX;
  }

  _isMultiSelected = false;
  _isInBackground = false;
  _isHighlighed = false;
  OnPropertyChanged("");
}
```

Listing 6.6: The optimised implementation of AppearDefault method in RecordProperty class. Instead of setting the properites themselves, their fields are set directly, thus avoiding unnecessary triggering of OnPropertyChanged events each time a value of a property is set. Instead, the OnPropertyChanged is triggered only once, after all values have been set, thus notifying the system, that the visual appearance of the record has changed, but without specifing which properties have changed.

```
private void HandleRecordPropertyChanged(object sender, PropertyChangedEventArgs e
    )
{
  IsHitTestVisible = !RecordProperty.IsInBackground || !RecordProperty.
      IsOutOfBounds;
  Visibility = (RecordProperty.IsOutOfBounds || RecordProperty.IsInBackground) ?
          Visibility.Hidden : Visibility.Visible;
  InvalidateVisual();
}
```

Listing 6.7: The event handler for OnPropertyChanged event triggered within RecordProperty object. The internal parameters are set according to the current state of RecordProperty object without checking the name of the changed property.

```
protected override void OnRender(DrawingContext drawingContext)
{
  if (RecordProperty.IsOutOfBounds || Visibility == Visibility.Hidden || _points.
      Count < 2)
    return;
   //otherwise, draw the polyline
}
```

Listing 6.8: The OnRender method of AxisPointConnection first checks if the record has been fildered-out or removed from the view by changing the axis scaling. If so, the method exits without rendering the record.

layer would not provide the information, about which polyline was affected by the click, but only that the object itself was clicked. Since no interactions are performed upon the polylines in the background, the only disadvantage of drawing all polylines in one object is the fact that rendering such a complex object takes some time. However, this object only has to be re-rendered when:

- the size of the plot changes,
- the scaling of one or more axes changes,
- one or more axes are inverted,
- axes are added or removed from the view.

These events do not occur as frequently as slider movements do. The second layer, which is drawn in the foreground, contains the same polylines as the background layer. These are drawn as single objects in order to provide easy support for hit-testing. When an item is filtered out, InvalidateVisual() is called, but the OnRender() method simply returns without drawing it, as shown in Listing 6.8. The item is drawn again if and when it is filtered back in. So, instead of re-rendering AxisPointConnection every time it is filtered in and out, one additional object consisting of all polylines is drawn in the background once, while all AxisPointConnection objects are rendered only when they are filtered in.

This approach did not produce optimal results at first, since whenever single pixels were rendered, several visual layers had to be processed. This problem was evident when moving a semi-transparent "ghost axis" across the plot, or when drawing multi-select rectangles. Drawing several layers requires many pixels to be rendered several times. While such "overlay" objects are drawn, all layers beneath remain unchanged, so re-rendering all these complex layers beneath is not necessary. The problem of redundant pixel rendering was solved by introducing bitmap caching.

The rendering performance of complex UIElement objects can be improved by creating a bitmap, which is cached in video memory [MSDN, 2015a]. Instead of displaying the live control, a cached image of the control is displayed as long as the visual object has not changed. The BitmapCache class has three properties: EnableClearType, RenderAtScale, and SnapsToDevicePixels. The EnableClearType flag defines whether the text within the bitmap is rendered with ClearType [MSDN, 2015b] on an opaque background, or with grayscale antialisaing. The SnapsToDevicePixels flag has to be set whenever pixel-alignment within the bitmap has to be done correctly, which is the case when rendering ClearType. RenderAtScale defines the bitmap scaling. Changing the cached element itself, or any of the three bitmap properties, causes the cache to be re-generated.

In order to make use of bitmap caching, the aggregated parallel coordinates plot is split six several layers, as shown in Figure 6.1. All layers are declared as Canvas elements, and contain collections of FrameworkElement objects. The elements are assigned to the layers depending on how frequently they change. The first layer contains only the background polylines. The second layer contains the basic axis

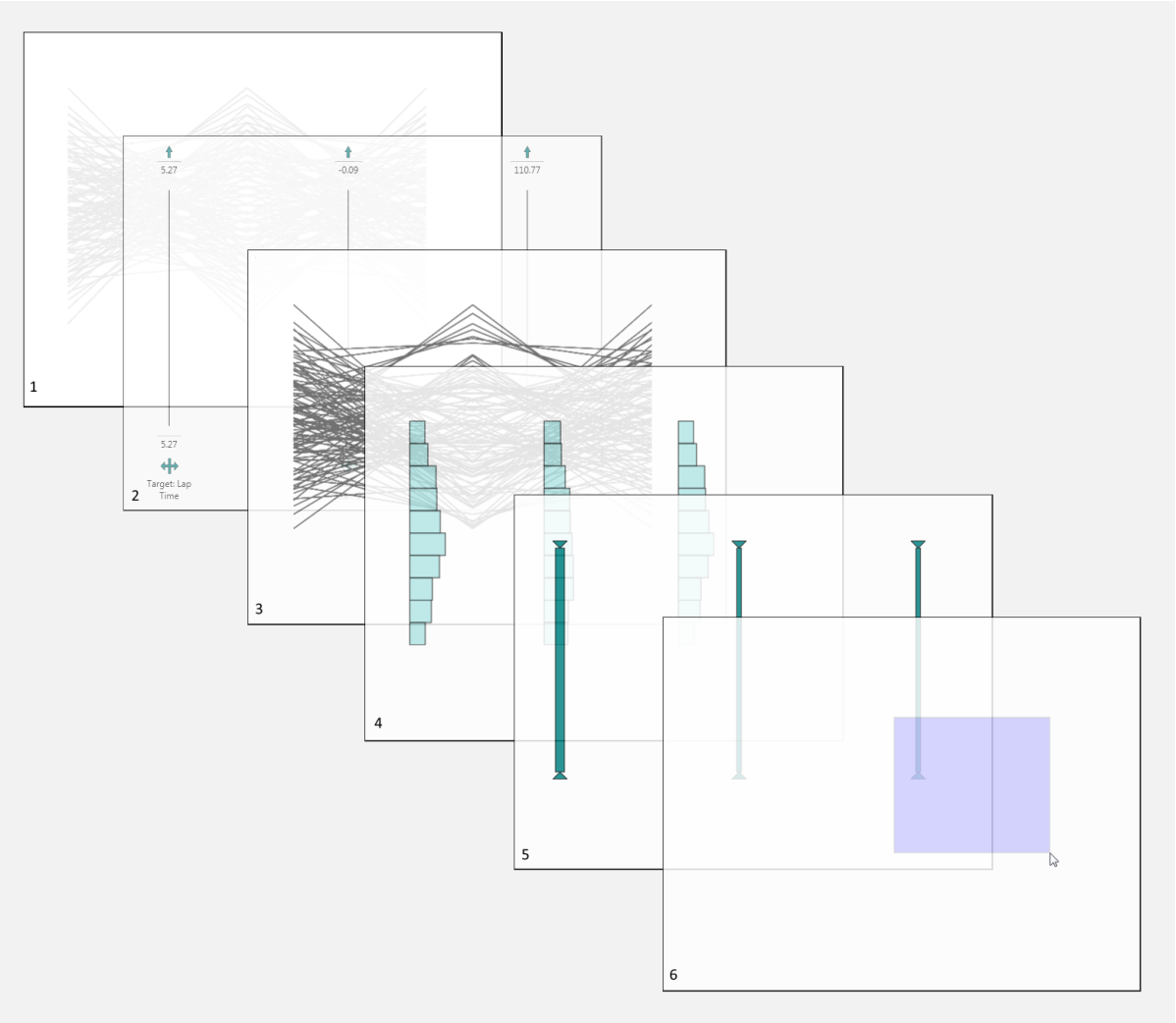


Figure 6.1: The aggregated parallel coordinates are rendered in six layers. Layers in the background are re-drawn less frequently than the layers in the foreground. Apart from the final layer, bitmap caching is used on every layer in order to reduce the number of times the items are rendered. While transient items, such as multi-select rectangles, are being drawn in the final (sixth) layer, all other layers are not re-rendered. Instead, the cached bitmap image is displayed. [Image created by the author.]

elements, such as invert buttons and labels. The third layer contains active polylines (those that have not been filtered out). The fourth layer contains any histograms. The fifth layer contains all axis sliders. These five layers are cached separately. The last layer is reserved for transient elements such as ghost-axes, zoom or multi-select rectangles. These elements are not persistent, so caching the layer in which they are drawn is not necessary. The final XAML code for the aggregated parallel coordinates is shown in Listing 6.9.

Bitmap caching is enabled by default, but can be turned off on demand. For this reason, the BitmapCacheMode is added to each layer in the code-behind, rather than in XAML. If the bitmap caching should be used, it is set for first five rendering layers, otherwise, the cache mode is set to null, as shown in Listing 6.10.

```
<ScrollViewer>
<Canvas Background="White" AllowDrop="True" Name="plotCanvas">

  <Canvas Name="backgroundPolylineCanvas" Canvas.Top="0" Canvas.Left="0"
          IsHitTestVisible="False" AllowDrop="True"/>

  <Canvas Name="axisBaseElementsCanvas" Canvas.Top="0" Canvas.Left="0">
    <ItemsControl ItemsSource="{Binding ElementName=control, Path=AxesElements}">
      <ItemsControl.ItemsPanel>
            <ItemsPanelTemplate>
        <Canvas />
        </ItemsPanelTemplate>
      </ItemsControl.ItemsPanel>
    </ItemsControl>
  </Canvas>

  <Canvas Name="polylineCanvas">
    <Canvas>
        <!--items control declaration... -->
    </Canvas>

    <Canvas Name="histogramCanvas">
      <!-- items control code... ->
    </Canvas>

    <Canvas Name="sliderCanvas">
        <!-- items control declaration ...-->
    </Canvas>
  </Canvas>

  <Canvas Name="overlayCanvas" Canvas.Top="0" Canvas.Left="0"/>
</Canvas>
</ScrollViewer>
```

Listing 6.9: In the optimised APC implementation, the `Canvas` defining the area within which the plot is drawn contains six layers. The first layer contains only the background polylines. The second layer contains the basic axis elements, such as invert buttons and labels. The third layer contains active polylines (those that have not been filtered out). The fourth layer contains any histograms. The fifth layer contains all axis sliders. These five layers are cached separately. The sixth and final layer is reserved for transient elements such as ghost-axes, zoom rectangles, or multi-select rectangles.

```
if (UseBitmapCaching)
{
  backgroundPolylineCanvas.CacheMode = new BitmapCache();

  var axisBaseElementsBitmapCache = new BitmapCache();
  axisBaseElementsBitmapCache.EnableClearType = true;
  axisBaseElementsCanvas.SnapsToDevicePixels = true;
  axisBaseElementsCanvas.CacheMode = axisBaseElementsBitmapCache;

  var activePolylineBitmapCache = new BitmapCache();
  activePolylineBitmapCache.EnableClearType = true;
  activePolylineBitmapCache.SnapsToDevicePixels = true;
  activePolylineCanvas.CacheMode = activePolylineBitmapCache;

  //... the same for all other layers
}
else
{
  backgroundPolylineCanvas.CacheMode = null;
  axisBaseElementsCanvas.CacheMode = null;
  //...
}
```

Listing 6.10: By default, bitmap cahing is enabled and applied to five `Canvas` layers. For demostration purposes, bitmap caching can be disabled by setting the `UseBitmapCaching` flag to false, in which case the `CacheMode` of all five layers is set to `null`.

6.3 The Results

After applying the optimisation steps described above, both memory consumption, and responsiveness of the application was vastly improved. In terms of responsiveness, the initial implementation of the parallel coordinates plot would become almost unusable when visualising a dataset of 5000 records in 5 dimensions. Moving a slider would cause a significant time delay between moving the mouse and displaying the results of applying that filter. While the optimised implementation takes some time to load 5000 records in 5 dimensions, after all elements are loaded, the sliders can be moved freely along the axis, almost without any lag. If a large number of records are filtered in, a certain lag can be experienced while moving the slider back to its initial position on the axis, because at this point records are filtered back in, which means that they have to be rendered again. However, filtering records out (which is the most frequent interaction), does not cause any lag at all, since items which are filtered out are simply not rendered. However, their shadows are visible in the background layer.

The memory consumption of the aggregated parallel coordinates implementation was vastly reduced after applying the optimisation steps. In the first implementation, all visual elements were drawn as custom controls, upon which templates were applied. With the points drawn on axes, a working memory set of the executed code, as shown by the Windows Task Manager, for a plot with 1000 records in 4 dimensions, was 180 KB. Removing the points from the axes reduced the working set for the same plot to 150 KB. This implementation was only drawing axes and polylines; other functionality (sliders, axis labels, polyline labels, dimension aggregation, etc.) was not implemented at that stage. The final, optimised, full-featured demo application (including the plot and a toolbar) has a working memory set of 160 KB.

The major reduction of memory consumption was achieved by drawing the visual elements as `Geometry` objects, instead of using the heavy-weight `Shape`. To demonstrate the difference in memory consumption, memory allocation profiling was done on two versions of the aggregated parallel coordi-

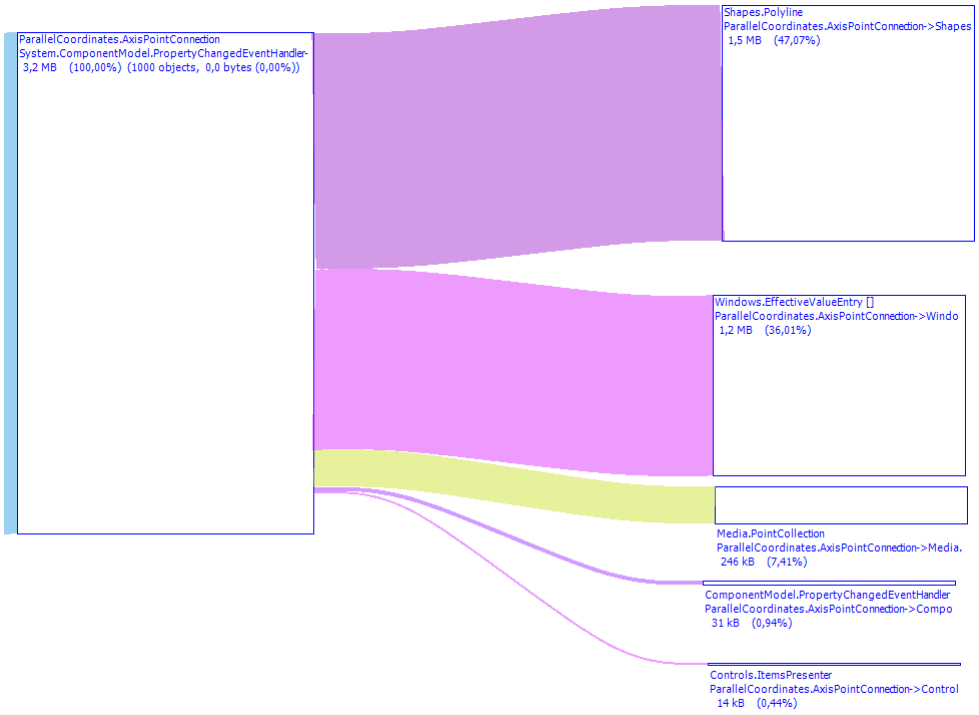


Figure 6.2: The CLR heap graph of managed memory allocated for 1000 AxisPointConnection objects in the initial implementation. [Screenshot created by the author.]

nates. In the first version, the AxisPointCollection objects were drawn as Polyline objects. The second version used Geometry objects. The rest of the code was kept the same. The profiling was done on a dataset consisting of 1000 records in 4 dimension, using CLR Profiler [Microsoft, 2015a]. The overall amount of allocated managed memory for the first implementation was 9.8 MB. As shown in Figure 6.2, 3.2 MB of memory were allocated for AxisPointConnection objects, 47% of which was used by the Polyline objects, while 36% were used to keep track of the values such as the polyline thickness, its colour, and other properties. For the optimised version, only 6.0 MB was allocated in total. As shown in Figure 6.3, in the optimised version only 897KB of managed memory was allocated for the same number of AxisPointConnection objects. This reduction is caused both by the fact that the rendering data requires much less space (430KB), and that the system has to keep track of fewer property values, since the Geometry objects do not redraw themselves automatically when a value of a property changes.

During the optimisation process, drawing an additional layer of background polylines was implemented in order to improve the responsiveness when filtering records. At a first glance, this might seem as a waste of resources, since all the polylines are drawn twice. However, unlike active polylines, which are all drawn as separate objects, background polylines are drawn in a single FrameworkElement. The render data for such an object with 1000 polylines takes less than 250 KB of managed memory, while drawing 1000 polylines in separate objects requires 1.5 MB. Since drawing several polylines inside one object requires less memory than drawing several objects consisting of one polyline, an obvious question to ask is whether it would be more efficient to also draw the layer with the active polylines within one object. By implementing this approach, the entire object would have to be redrawn every time a record is filtered in or out. Since rendering such complex object is time-consuming, the responsiveness of the application would be reduced. Furthermore, drawing polylines within one complex object would make it impossible to use built-in hit-testing, since the polylines would not be a part of visual tree. Hit-testing in that case would have to be performed by adding an additional, invisible layer of polylines. A unique colour would have to be assigned to each polyline, based on which polylines would be uniquely identifiable, which would be necessary in order to implement hit-testing. Applying this approach would, in

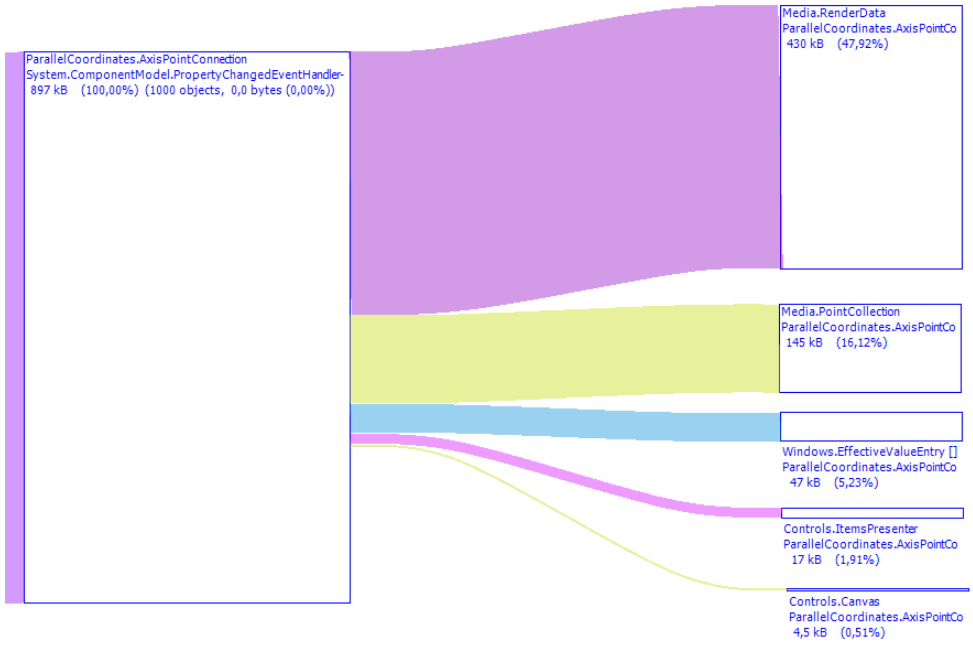


Figure 6.3: The CLR heap graph of managed memory allocated for 1000 AxisPointConnection objects in the optimised implementation. [Screenshot created by the author.]

turn, require using additional memory, so the memory consumption would not be vastly decreased when compared with the memory consumption of the currently applied approach.

Chapter 7

Selected Details of the Implementation

"The details are not the details. They make the design."

[Charles Eames, Designer]

In Chapter 6, a detailed description of how the aggregated parallel coordinates plot is drawn in order to provide the optimal performance was given. In that context, the drawing of the polylines other relevant details of implementation were explained. This chapter focuses on other important aspects of the implementation. Special attention is given to the way dimension aggregation is implemented in the plot, and how an existing data structure was extended to support this feature. The details of the implementation of filtering using sliders are also discussed.

7.1 Data Structure

The data structure used in aggregated parallel coordinates is an extension of the data structure used in AVL SimBook. The decision to use this structure was made in order to provide easier integration of the aggregated parallel coordinates plot into SimBook, and avoid the overhead which would be created by the conversion between different data structures. An overview of the data structure is provided in the class diagram in Figure 7.1. The root of the structure is the Parameter class, which represents a single dimension in the dataset. It holds the information about the name of the dimension, as well as the values of each dataset record in that dimension. Each dimension of the dataset visualised in SimBook is either one of the input parameters for which the influence of different combinations of values on the vehicle performance was simulated, or an output parameter, whose values are a result of the simulation. To differentiate between these dataset dimension types, the Parameter class is extended by InputParameter and OutputParameter. The OutputParameter is extended by the CornerSpecificParameter and the StraightSpecificParameter, in order to provide additional information in case a single dimension has been calculated separately for corners or straight segments of the track. The CornerSegmentSpecificParameter class extends the CornerSpecificParameter, and represents dimensions which are calculated for the segments of the corners.

To support the dimension aggregation, the MeanValueParameter class was added to the above described data structure. This class acts as a container for a collection of Parameter objects, which represent the dimensions to be aggregated within one axis. It extends the Parameter class directly. When the MeanValueParameter object is created, a new virtual dataset dimension is created, whose records are mean values of records contained in the collection of Parameter objects. The collection of objects aggregated in the MeanValueParameter can contain any Parameter type, including the MeanValueParameter itself. Such logical structure enables creating parameters hierarchies of any depth.

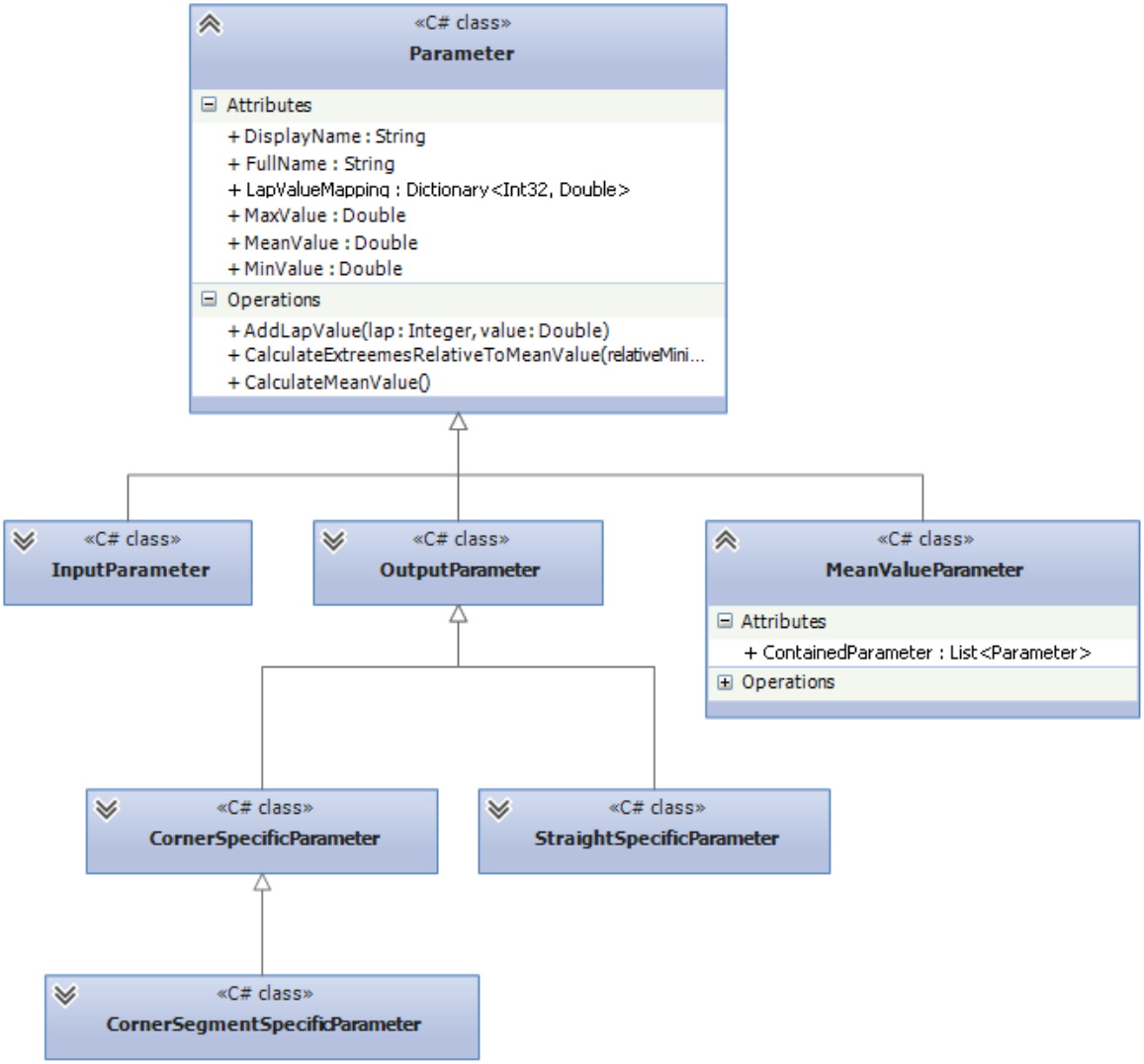


Figure 7.1: The class diagram of the Parameter data structure. A parameter corresponds to a dimension in traditional parallel coordinates terminology. The root element contains general information about a dimension, and the values of the dataset records in that dimension. The extended classes provide additional information about specific dimensions found in the datasets produced by race car simulations. The MeanValueParameter objects represent aggregated dimensions, and are created based on a collection of child Parameter objects. [Diagram created by the author using Visual Studio 2012.]

7.2 Aggregated Axes

Aggregated parallel coordinates library accepts a collection of Parameter objects as a dataset to be visualised. For each Parameter object, the plot draws one axis. More specifically, for each Parameter object, one Axis object is created. The Axis objects are logical, rather than visual, and are responsible for creating all visual elements of an axis, and for calculating their position in the plot. If the dataset passed to the plot contains any MeanValueParameter objects, instead of instantiating the Axis class, its extension, AggregatedAxis is instantiated. This class represents the root of an aggregated axis hierarchy. When a root axis is expanded, the ExpandChildren() method, which is implemented in the AggregatedAxis class, checks the collection contained in its MeanValueParameter object. If the collection contains further MeanValueParameter objects, an instance of AggregatedExpanded class is created, while for every other Parameter, an ExpandedAxis object is created. As shown in Figure 7.2, the AggregatedExpandedAxis extends AggregatedAxis, while ExpandedAxis extends Axis directly. These two axis types both implement the IExpanded interface. The AggregatedExpandedAxis represents dimensions in the dataset hierarchy which have both a parent, and their own children. Since it extends AggregatedAxis, it has all functionality necessary to keep track of and expand its own children when required, and by implementing the IExpanded interface, it also knows about its parent axis, to which it can be collapsed when necessary. The ExpandedAxis objects represent leaf elements of the dimension hierarchy, and can only be collapsed to their parent axis.

Instances of Axis hold a collection of objects created by instantiating classes which implement the IAxisElement interface. An overview of these classes is shown in Figure 7.3. Every class which implements the IAxisElement interface is responsible for drawing a specific part of the visual representation of axes. In the base Axis class, instances of the following classes are created: InvertButton, BodyLine, Slider, ExtremeLabel and Histogram. AggregatedAxis adds an ExpandButton object to the collection, after all other visual elements are created and added to the collection. Furthermore, it overrides the AddMiddleSlider() method in order to make sure that its visual representation contains an instance of the AggregatedAxisMiddleSlider instead of the MiddleSlider. Both the ExpandedAxis and a AggregatedExpandedAxis add an instance of the CollapseButton to the base collection of visual elements. The IAxisElement objects hold a reference to the Axis object to which they logically belong. Based on the information contained in this reference, these objects are able to draw themselves in the correct position in the plot. A reference to a corresponding instance of the Axis class is also used to handle user interactions with the visual representation of an axis. For example, when a user clicks on a collapse button, the CollapseButton object calls the Collapse() method provided by the IExpanded interface implementation whose reference is held by the CollapseButton object. The Collapse() method calls the CollapseChildren() method implemented by its parent axis.

7.3 Sliders

Record filtering in aggregated parallel coordinates is possible by moving either the upper, lower, or the middle sliders. As already mentioned in Chapter 6, the visual properties of every record are defined in a class called RecordProperty. This class contains the data about the current state of the record on the plot, while the drawing of the record is performed elsewhere. Each record in the currently visualised dataset has a unique ID. This ID is used to map the record to its own instance of the RecordProperty class. For example, when the position of the upper slider of an axis moves down, the IDs of all records positioned above the upper slider are retrieved from a list of records sorted by Y screen coordinate for that axis. Using these IDs, the RecordProperty objects of all records to be filtered-out are retrieved, and the AppearInBackgrond() method is called in order to set the visual properties of the records according to their new, “in background” state.

Consider the state of a single record which is filtered-out on two axes, as shown in Figure 7.4. After the record is filtered out on the first axis, its visual properties will be changed by calling the

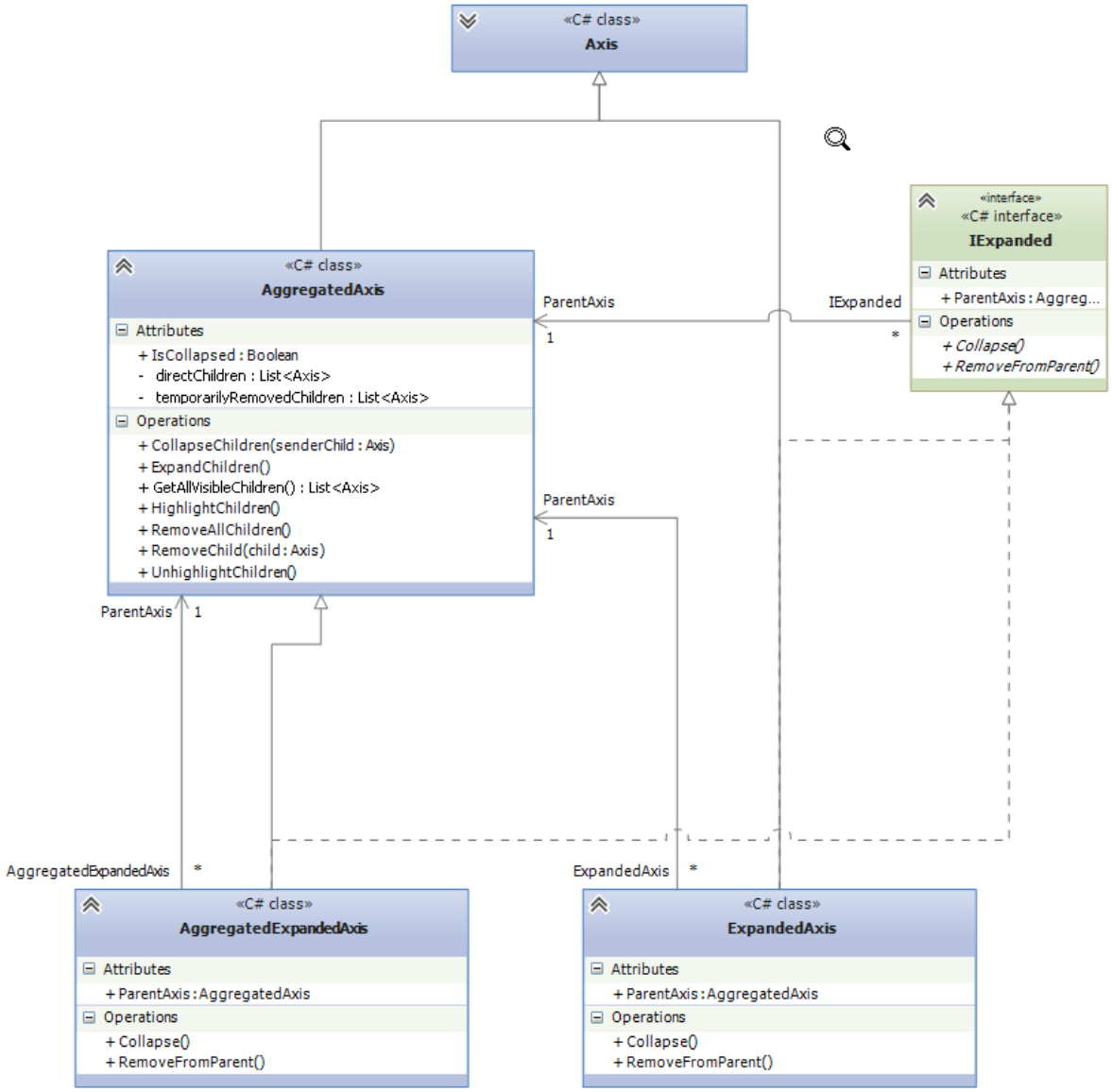


Figure 7.2: The class hierarchy of the logical axis implementation. The root class is extended by AggregatedAxis, which represents the root of a dimension hierarchy. The "leaf" dimensions are represented by the ExpandedAxis. The AggregatedExpandedAxis represents dimensions of the hierarchy which have both a parent and their own children. This class is logically an aggregated axis, since it has its own children, but it also provides the same functionality as ExpandedAxis, by implementing the IExpanded interface. [Diagram created by the author using Visual Studio 2012.]

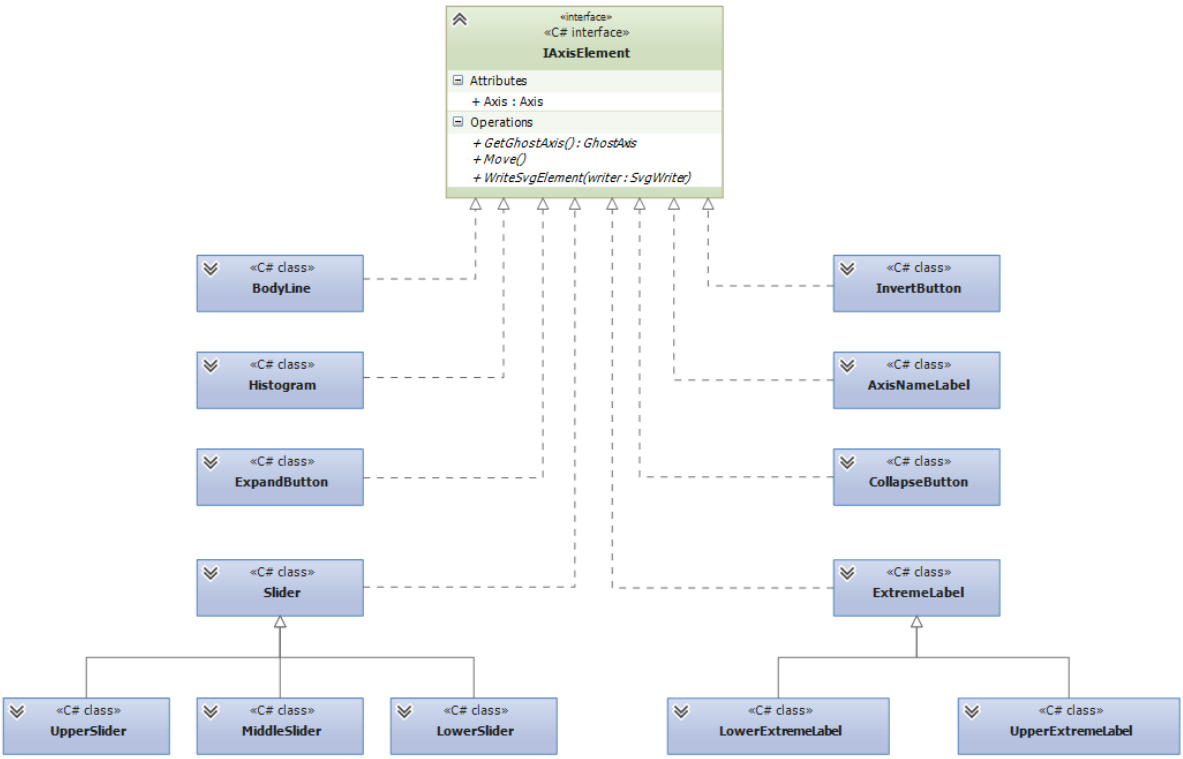


Figure 7.3: The class hierarchy of visual axis elements. Each class is responsible for drawing the visual representation of a specific part of a logical axis. All elements hold a reference to the logical axis, through which visual elements gain access to all information they require to draw themselves, as well as methods for handling user interactions. [Diagram created by the author using Visual Studio 2012.]

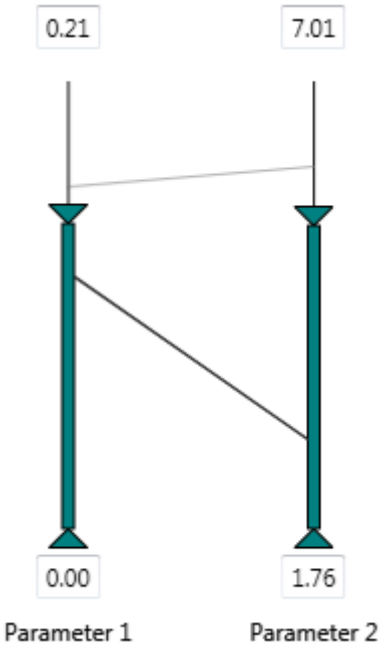


Figure 7.4: The record displayed in the background is filtered out on both axes. Keeping track of all axes on which the record has been filtered out is necessary in order to prevent filtering the record back in when the slider on only one axis moves above the record's position. [Screenshot created by the author.]

AppearInBackground() method. When the slider on the second axis moves, the AppearInBackground() method will be called again, but the representation will not change, since the record is already filtered out. In order to prevent filtering the record back in when the slider on only one axis moves above the record's position, the RecordProperty has to keep track of which axes the AppearInBackground() method has been called for.

One possibility for keeping track of the axes on which a record is filtered-out is simply adding a counter, whose value is incremented whenever the record is filtered out on any axis, and decremented when the record is filtered back in. The record would be filtered back in only in case the counter's value is set to 0. However, if for any reason a method which changes the value of the counter is called too many or too few times, the counter will not show the real number of axes on which it is filtered out. To avoid this problem, each axis is assigned a unique ID, based on which the RecordProperty object can keep track of the axes on which a record is filtered out.

The unique ID is generated in the constructor of Axis as follows:

```
ID = Guid.NewGuid().ToString();
```

In C♯, the Guid structure is used to create a Globally Unique Identifier (GUID), which is a 128-bit integer. Since this number is so large, producing two same GUIDs is very unlikely, which makes its usage for unique identification appropriate.

The portion of code shown in Listing 7.1 is executed in UpperSlider class when the slider moves on the axis. Using its reference to the logical Axis object, the UpperSlider can access the IDs of records stored in a list, which is sorted according to the Y position of the record on the axis. The `_lastVisibleIndex` variable is used to keep track of the index of first record in the sorted list which is not filtered out. So, when the slider moves, starting from the last record that was filtered in, it takes every record from the list, checks if it is positioned above the slider, and if so, it calls the Hide method for that record. If the record is positioned below the slider, the `_lastVisibleIndex` is updated, so that the next time the slider is

```
for (int i = _lastVisibleIndex; i < _axis.SortedAxisPoints.Count; i++)
{
 var id = _axis.SortedAxisPoints[i];
 if (_axis.AxisPointOrderMapping[id].AxisCoordinate.Y < SliderTipPosition.Y )
 {
   GlobalSettings.LAP_PROPERTY_MAPPING[id].Hide(ref _axis.ID);
 }
 else
 {
   _lastVisibleIndex = i;
   break;
 }
}
```

Listing 7.1: Portion of the code executed when the upper slider is moved down. Starting from the index of the last record which was filtered in, the position of every record in a list sorted according to vertical position on the axis is checked. If the record is positioned above the slider, the Hide() method is called for that record. Otherwise, the `_lastVisibleIndex` is updated, so that the next time the slider is moved down, the loop can start from that index in the list, and not from the beginning. After updating `_lastVisibleIndex`, the loop is broken.

```
public void Hide(ref string sender)
{
  if (!senders.Contains(sender))
  {
    senders.Add(sender);
    if (!IsInBackground)
      AppearInBackground();
  }
}
```

Listing 7.2: The implementation of Hide method in RecordProperty. The ID of an axis on which the record is filtered out, is passed as parameter. If the ID is not contained in a local list, it is added to it, and the AppearInBackground method is called. Otherwise, the reocrd has already been filtered out on that axis, in which case the method simply exits without taking any action.

```
for (int i = _lastVisibleIndex - 1; i >= 0; i--)
{
    var round = _axis.SortedAxisPoints[i];
    if (_axis.RealPointToOrderMapping[round].Y >= SliderTipPosition.Y)
    {
      GlobalSettings.LAP_PROPERTY_MAPPING[round].Show(ref _axis.ID);

    }
    else
    {
        _lastVisibleIndex = i;
        break;
    }
}
```

Listing 7.3: Portion of the code executed when the upper slider is moved up. Starting from the index of the last record which was filtered out, a list of records sorted according to vertical position on the axis is traversed back, and the vertical position of each record is checked. If the record is positioned below the slider, the Show() method is called for that record. Otherwise, the `_lastVisibleIndex` is updated, after which the loop is broken.

moved down, the loop can start from that index in the list, and not from the beginning.

The implementation of Hide method is shown in Listing 7.2. It accepts the ID of an axis as a parameter, and checks if the collection of parameters held by the RecordProperty object already contains that ID. If not, it adds it to the collection and, if the record is not already displayed in the background, it calls the AppearInBackground() method. In case this method is called twice for a single record, it will recognise that the record is already filtered out on that axis, and will simply exit.

When an upper slider moves up the axis, the code shown in Listing 7.3 is executed. Starting from the index of the last record which was filtered out on an axis, the list of points sorted by the Y coordinate is traversed backwards, and the Y coordinate of the current slider position is compared to the Y coordinate of currently examined record. If a record is below the slider, the Show method will be called on a corresponding RecordProperty object.

The implementation of the Show method is shown in Listing 7.4. Just like the Hide method, it accepts the ID of an axis as the parameter, and checks if the ID is in the collection of axis IDs for which the record has been filtered out. If the collection contains the axis ID, the ID is removed from the list. Only if the collection is empty after removal, the record will be displayed as filtered-in by calling the AppearDefault() method. If the ID is not present in the collection, the method has been called for a record which is not filtered-out on that axis, in which case the method simply exits without changing the state of the record.

```
public void Show(ref string sender)
{
  if (_senders.Contains(sender))
  {
    _senders.Remove(sender);
    if (_senders.Count == 0)
    {
      AppearDefault();
    }
  }
}
```

Listing 7.4: The implementation of Show method in RecordProperty. The ID of an axis on which the record is filtered in, is passed as parameter. If the ID is contained in a local list, it is removed from it. Otherwise, the reocrd is already been filtered in on that axis, in which case the method simply exits without taking any action. The AppearDefault method is called only if the local list is empty, that is, when the record has been filtered in on all oaxes.

Chapter 8

Outlook

"The future is always beginning now."

[Mark Strand - American poet.]

Aggregated parallel coordinates (APC) provide a set of basic features necessary for visual exploration of multi-dimensional datasets with hierarchical dimensions. They are optimised for easy integration with SimBook, a software application used inside AVL Racing. The set of implemented features support visual exploration of multivariate datasets produced by race car simulations. However, current implementation lacks some common interactions, which were not implemented as a part of this thesis due to time limitations.

The APC implementation does not support zoom interaction. Although axis scaling can be used in order to zoom in or out of a single axis, this feature is used to filter out records which are not of interest, and is not an appropriate replacement for the standard zoom interaction. Furthermore, the scaled axes are not marked in any way, so users cannot know that any records have been removed from the current view. Marking axes on which the scaling has been changed is another possible extension to the current implementation.

Another useful extension to the current implantation would be keeping track of the user's interactions. In addition to adding support for undo/redo interactions, a list of performed interactions could be visualised in a separate view. This would allow adding new filter interactions by enabling users to combine previously applied filters using boolean operators.

When an aggregated axis is expanded, it is removed from the view, and its child axes are placed at the position where the aggregated axis was. Simply reloading the display with a new set of axes might be somewhat confusing to novice users. This problem could be solved by adding an animated transitions, which would visualise the expanding process, thus effectively informing the user about how many new axes are added to the view, and how the other axes are re-positioned in order to make room for the newly added axes.

The records in race car simulation datasets have only numerical values. To add support for categorical data, the data structure used in aggregated parallel coordinates would have to be extended. Furthermore, this data structure currently provides dimension aggregation only by calculating the mean values of the dataset records. Further extensions could add support for other ways of dimension aggregation, for example, by calculating minimum, maximum, or median values of dataset records.

Another possibility to extend aggregated parallel coordinates is to implement some of the clutter reduction methods described in Chapter 4. This would improve the process of visual data exploration with aggregated parallel coordinates even further. The implementation of edge bundling would be particularly interesting.

The overview given above describes some possibilities to improve and extend the currently implemented aggregated parallel coordinates visualisation. Another example of future work would be conducting usability studies. Testing how the implemented expand and collapse interactions are perceived by novice users, as well as the visual distinctiveness of any aggregated axes would be particularly interesting.

Chapter 9

Concluding Remarks

The main focus of this thesis was visualisation of race car simulation datasets. Chapter 2 provided an overview of some of the factors which influence the behaviour of a car on a racing track. This was done in order to illustrate the complexity of datasets produced by race car simulations, which were characterised as both hierarchical and high-dimensional. Information visualisation, a type of visualisation which provides mechanisms for visual representation and exploration of abstract concepts, was described in Chapter 3. Since interaction is one of the two important aspects of information visualisation, an overview was given of interactive features commonly implemented in many visual data exploration tools. The second aspect of the information visualisation, the visual representation, depends on the type of data being represented. Since race car simulation datasets contain both hierarchical and multi-dimensional relationships, an overview of some of the most common techniques for visualising these two types of data was also given in Chapter 3.

Parallel coordinates, one of the most effective technique for visualising multi-dimensional data, were discussed in detail in Chapter 4. Common interactive features and extensions were presented, as well as different methods for effective presentation of datasets with a large number of records. An example was given of how parallel coordinates can be used to explore complex datasets, and of how applying appropriate selection and filtering mechanisms can lead to discovery of new knowledge. Furthermore, several extensions were described, which deal with specific problems inherent in the standard parallel coordinates implementation. The chapter concluded with an overview of five software applications which provide parallel coordinates visualisation.

Of all the described techniques for visualising multi-dimensional data, parallel coordinates seemed to be the most effective way to represent race car simulation datasets. Since standard implementations do not provide a mechanism for dealing with hierarchies within the dataset, aggregated parallel coordinates (APC) were introduced. Chapter 5 described this extension in detail, with the main focus on "dimension aggregation", which is the unique feature introduced with aggregated parallel coordinates. It also showed how this visualisation is used in combination with order methods, such as small multiples, scatter plots, table representation, and tree views, to support the process of visual exploration of race car simulation data.

Chapter 6 guided the reader through the steps taken to optimise the performance of the initial implementation. Among others, this chapter shows how the selection of proper rendering methods can have a huge influence on the performance of the visualisations implemented in WPF. Other selected details of implementation were provided in Chapter 7.

Aggregated parallel coordinates provide sufficient features to support the process of visual exploration of race car datasets, and are already being used for this purpose within AVL Racing as part of AVL SimBook. However, some common interactive features, like zooming, are still missing in the current implementation. During the implementation process, a few new features were considered, but not implemented due to time limitations. The most prominent new feature would be adding animated tran-

sitions to visualise the process of dimension aggregation. Furthermore, minor extensions to the current implementation would be required in order to fully support the representation of datasets which are not generated by race car simulations.

Appendix A

User Guide

This guide is intended for users of the APC Testbed application, which was implemented as a part of this thesis. As shown in Figure A.1, this application consists of the following visual modules:

- a toolbar, which, among others, provides controls to load a dataset,
- a list or tree-view of the dataset dimensions,
- a panel with controls for plot manipulation,
- the aggregated parallel coordinates plot.

The following sections provide information about how the APC Testbed can be used to import or generate and visualise data with aggregated parallel coordinates. The implemented customisation options are explained. Furthermore, this guide explains how the created visualisation can be exported to an SVG file, and how the default application layout can be changed.

A.1 Loading the Data

The dataset can be loaded into the application either by importing it from a CSV file, or by generating a random dataset. Two types of datasets can be imported: general hierarchical datasets, and AVL racing datasets. If an AVL racing dataset is available to the user, it can be imported by clicking the "Import racing dataset" button, after which a dialogue will open and the user can navigate through the file system and select a CSV file to be imported. When importing such a dataset, a special parser is used to detect and load hierarchies within the dataset. For other types of dataset, the "Import dataset" button can be used. This causes the data to be parsed by a general CSV parser. Note that the aggregated parallel coordinates plot can only represent numerical records. Categorical values in the dataset will be set to 0. If any of the two parsers fails to parse the data, an error message is displayed, as shown in Figure A.2.

A random datasets can also be generated by clicking on the "Generate" toolbar button. The dialogue, shown in Figure A.3 opens. In this dialogue, the following properties of the generated dataset can be specified:

- The number of records in the generated dataset,
- The number of single dimensions (dimensions without a hierarchical structure),
- The number of nested dimensions (dimensions with a hierarchical structure).

If the number of nested dimensions is 1 or higher, the user can also select:

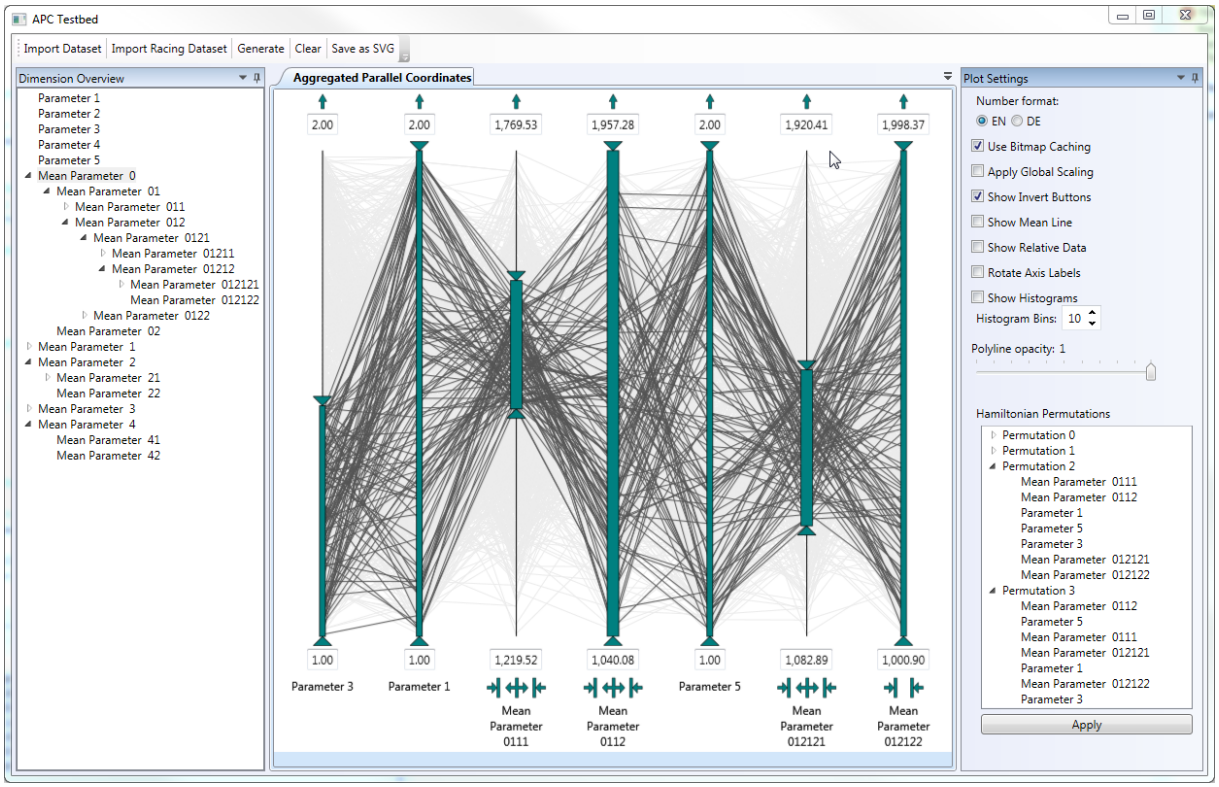


Figure A.1: The APC Testbed, a demo application for aggregated parallel coordinates. The plot is displayed in the middle. The toolbar enables importing or generating a dataset. The panel on the left provides a list or tree view of the loaded dataset. The panel on the right provides controls to manipulate the plot. [Screenshot created by the author.]

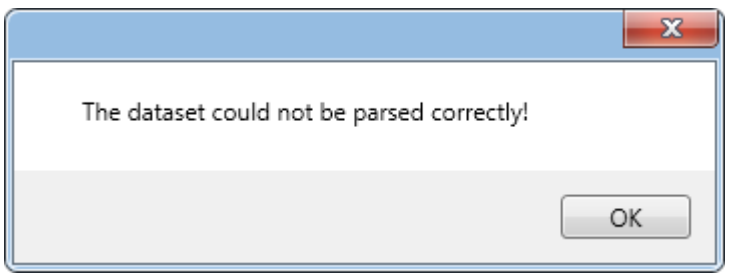


Figure A.2: The error message shown in case the imported data could not be processed by the application. [Screenshot created by the author.]

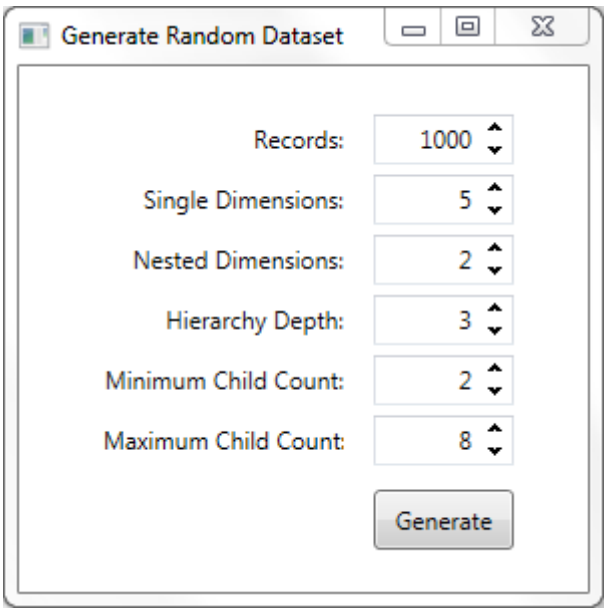


Figure A.3: The dialogue shown after clicking on the Generate toolbar button. The user can specify the record and dimension count in the generated dataset, as well as properties of the nested dimensions. [Screenshot created by the author.]

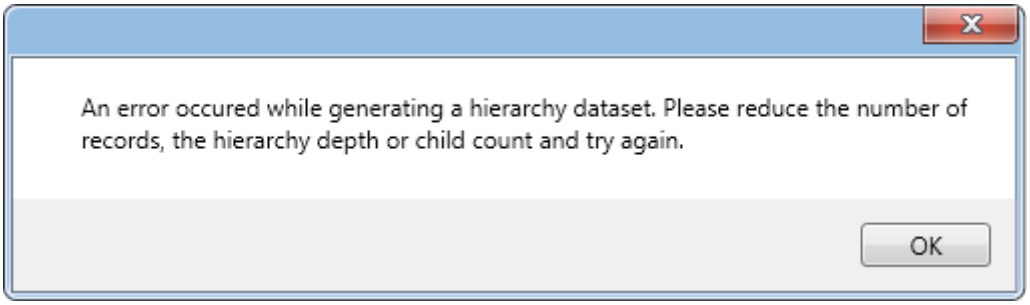


Figure A.4: The error message shown if the application cannot generate the specified dataset, which can occur when generating very large hierarchies. [Screenshot created by the author.]

- The maximum hierarchy depth,
- The minimum number of children at a single hierarchy level,
- The maximum number of children at a single hierarchy level.

If the number of nested dimensions is set to 0, these fields are disabled. Note that generating deep hierarchies with a large number of records can take a long time. If dataset generation fails, the error message shown in Figure A.4 is displayed.

After the dataset has been successfully loaded, either by importing or by generating, the dimensions of the dataset are visualised in the "Dimension Overview" panel. An example of a tree view visualisation of a generated dataset with hierarchical dimensions is shown in Figure A.5.

A.2 Using and Customising the Plot

The aggregated parallel coordinates plot implements a series of interactions. Table A.1, provides an overview of the mouse and keyboard controls which can be used to interact with the plot. For an overview of the functionality implemented in the plot, please refer to Chapter 5.

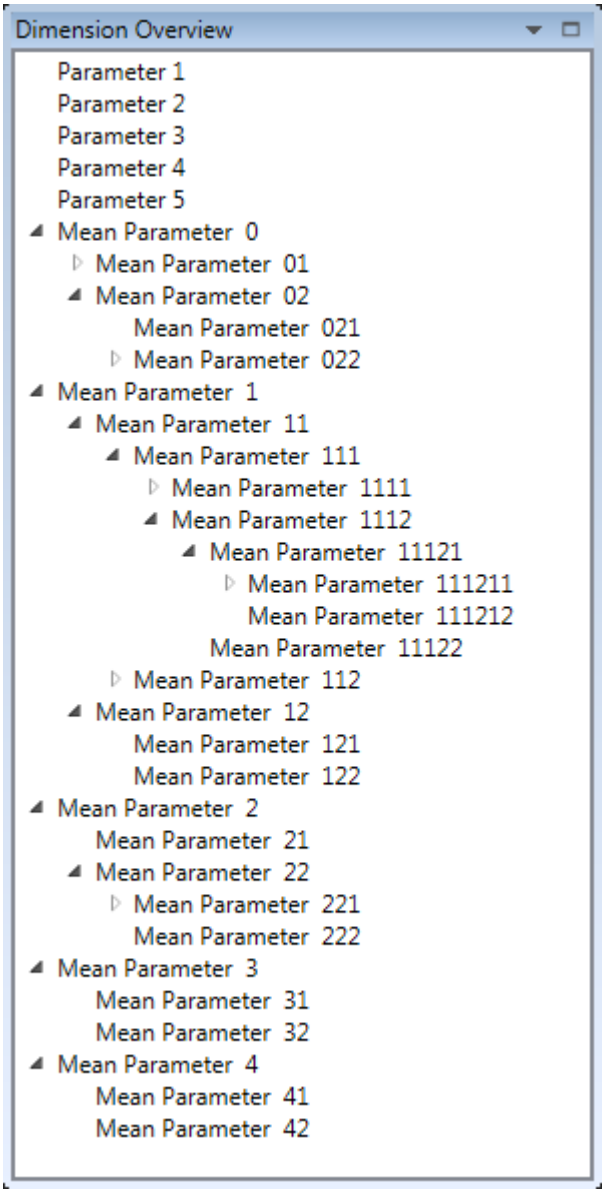


Figure A.5: The dimension overview for a generated dataset. [Screenshot created by the author.]

Goal	Interaction
Highlight single record	Hover the mouse over a record's polyline.
Select multiple records	Draw a multi-select rectangle by keeping the left mouse button pressed while dragging the mouse over the affected range.
Add single record to the set of selected records	Control-left-click on an unselected record.
Remove single record from the set of selected records	Control-left-click on a selected record.
Clear multiple record selection	Left-click on an empty space anywhere on the plot.
Set the reference record	Right-click on the record and select the appropriate context menu option.
Move the axis sliders	Left-click on the slider, and drag the mouse vertically with the left mouse button pressed.
Invert single axis	Left-click on the Invert button above the axis, or use right click on the axis and select the "Invert" option from the context menu.
Change scaling of a single axis	Enter the selected value range in the upper or lower axis extreme value label.
Select scaling range of single or multiple axes	Draw a zoom rectangle by keeping the right mouse button pressed while dragging the mouse over the affected axes. The height and position of the rectangle defines the new scaling.
Remove, reset scaling or slider position of a single axis	Right-click on the axis and select the appropriate option from the context menu.
Remove, invert, reset scaling or reset sliders of multiple axes	While holding the `Ctrl` key pressed, select the axes by left-clicking on them. When all axes are selected, use the right mouse button click to show the context menu and select the appropriate item.
Reset scaling on all axes	Double-right-click on an empty space anywhere on the plot.
Move an axis to a different position	Right-click on the invert button, axis body, any of the sliders or axis label. Drag the mouse, while keeping the right mouse button pressed, to the desired axis position. After releasing the mouse button, the axis will move to the new location.
Expand an aggregated axis	Click on the expand button under the axis.
Collapse the child axes	Click on the collapse button under the axis.

Table A.1: Interactions supported by APC Testbed.

The Settings Panel, shown in Figure A.6, provides an interface for customisation of the aggregated parallel coordinates plot. The controls provided by the Settings Panel include:

- **Changing the Numerical Value Representation**
  The numerical number representation can be changed by using the two radio buttons in the settings panel. Currently, English and German number representations are supported.
- **Use Bitmap Caching**
  Bitmap caching is enabled by default. Disabling the bitmap caching option results in rendering an increased number of pixels, which may cause decreased responsiveness, especially when visualising a large number of records. Keeping this option enabled is recommended.
- **Apply Global Scaling**
  Enabling this option applies global scaling on all axes.
- **Remove Invert Buttons**
  The invert buttons are displayed by default, but can be removed on demand, in case axis inversion (flipping) is not used frequently. Removing the invert buttons allocates more height to the axes, which is a valuable asset when visualising a large number of records. When the buttons are not displayed, axes can be inverted using the axis context menu.
- **Show Mean Line**
  If this option is enabled, the mean values of all records in the dataset are calculated and a mean line is added to the plot. The mean line is displayed somewhat thicker then the other lines, and has a different colour.
- **Show Relative Data**
  Enabling this option causes the plot to display relative instead of absolute numerical values of records, based on a reference value. Enabling this feature, automatically enables displaying the mean line, because the reference value is set to be the mean value by default. The reference record can be selected using the record context menu within the plot.
- **Rotate Axis Labels**
  If this option is enabled, the name labels of the axes are rotated to the right by 30°.
- **Show Histogram**
  Enabling this option adds histograms to each axis. The number of histogram bins can be changed by using the provided text box. The minimum number of histogram bins is 1, and the maximum is 100.
- **Change Line Opacity**
  By default, the polylines in the plot have full opacity. This can be changed using the provided slider. Note that opacity is not updated while the slider is moved, but only upon release. The current opacity value is shown in the label above the slider.
- **Hamiltonian Permutations**
  Axes in the plot can be compared only in case they are displayed next to each other. The minimal number of axis orderings which allow direct comparison of each axis with every other axis is calculated by the application by finding all distinct Hamiltonian permutations, as described in Chapter 4. These orderings are visualised in a tree view within the settings panel. Each root element of the tree view represents one permutation of axis ordering. Expanding a permutation element shows a list of axis names, as they are ordered in that permutation. A selected permutation can be applied to the plot by clicking on the button below the tree view.

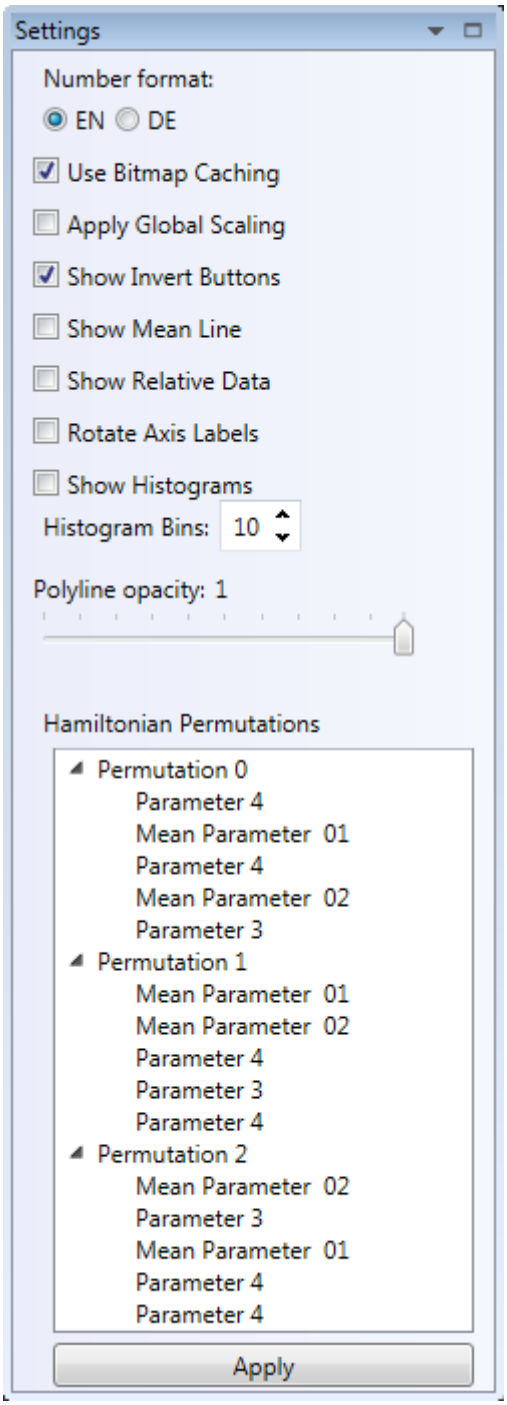


Figure A.6: The Settings Panel provides controls which enable customisation of the aggregated parallel coordinates plot. [Screenshot created by the author.]

A.3 Exporting the Plot

The current plot can be exported as a Scalable Vector Graphics (SVG) file. After clicking on the "Save as SVG" button, a dialogue will open, within which the user can select the location to which the file can be saved. SVG files can be rendered by most web browsers, and support free scaling of the content, which means that its contents can be displayed in any size, without affecting image quality.

A.4 Changing the Application Layout

The default application layout can be changed by manipulating the position and visibility of the Dimension Overview and Plot Settings panels. The panels can be moved simply by dragging them away from their position. As shown in Figure A.7, when a panel is moved from its position, an additional control appears over the application window, showing where the selected panel can be placed. The panels can be placed vertically, on the left or the right side of the plot, or horizontally, above or below the plot. The panels can also be displayed as tabs, within the plot view or as separate windows outside the application's window. Furthermore, one panel can be docked within the other panel, with the same positioning possibilities as when its docked within the application's window. Using the "unpin" icon displayed in the panel's top right corner, a panel can be hidden from the view. When a panel is hidden, it is displayed as a small icon on the left or right side of the application window. The panel can be "pinned" back to the layout at any time.

The two panels occupy a large amount of space within the application, which leaves less space for displaying the plot. Using the above described docking functionality, the layout can be adapted to the screen size available. The panels can be placed horizontally, if more width is required for displaying the plot, or vertically, when the plot needs more height, or completely hidden from the view, when they are not needed.

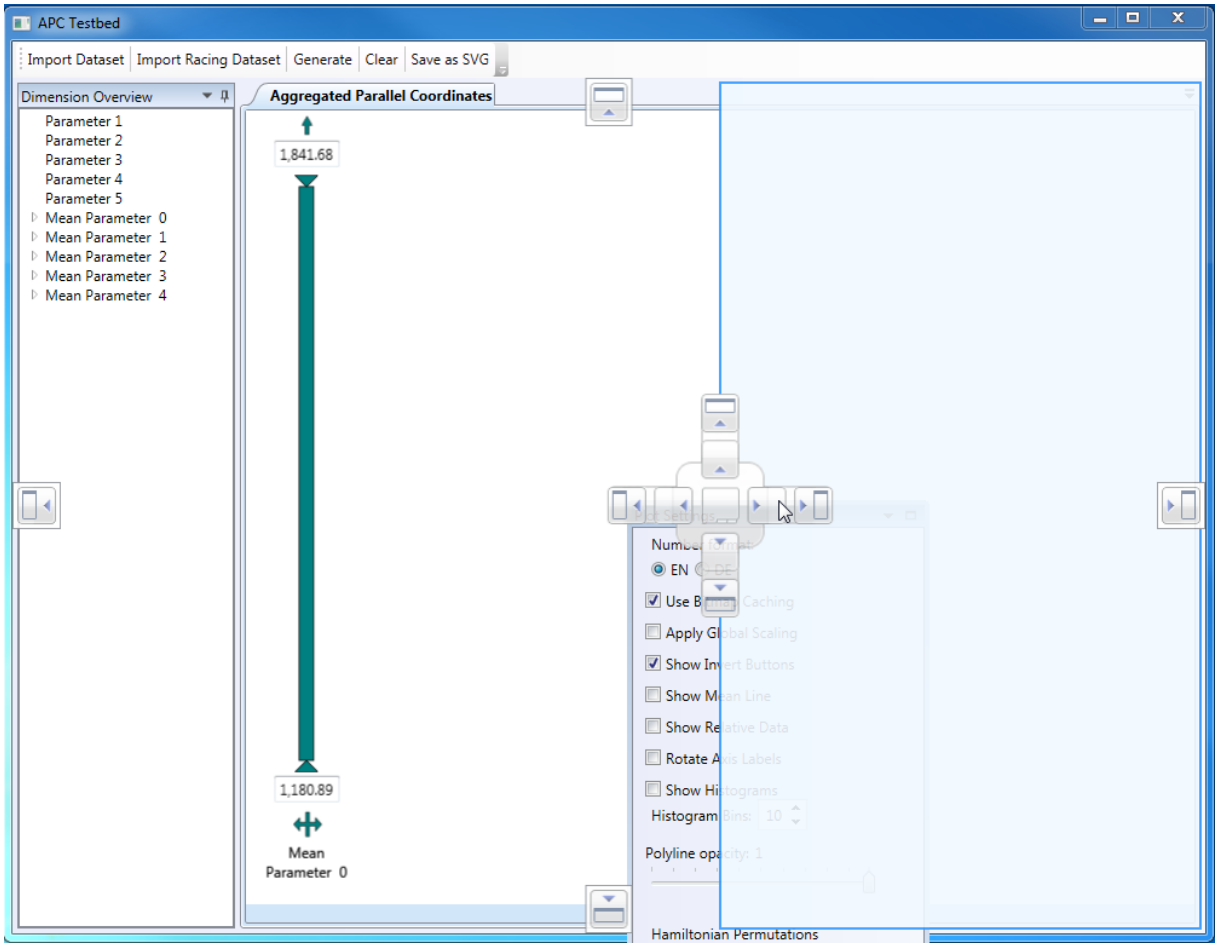


Figure A.7: Changing the application layout. When a panel is moved from its position, an additional control appears over the application window showing at which positions the selected panel can be docked. Placing the mouse over a part of this control draws an outline rectangle showing how the panel will be docked. [Screenshot created by the author.]

Appendix B

Developer Guide

This guide describes how the aggregated parallel coordinates can be used and extended. It is divided into two parts. The first part is intended for developers who want to use the implemented visualisation as a library, within another WPF application. The second part provides a short overview of the internal code structure, and is intended for developers who have access to the source code and intend to change the current implementation, or add a new functionality to it.

B.1 Using Aggregated Parallel Coordinates as a Library

Since the aggregated parallel coordinates plot was developed as a UserLibrary, it can be easily imported into any WPF application. This guide explains how to import and use the aggregated parallel coordinates library in a WPF application.

The following `.dll` files have to be added to the references of the WPF project in which the plot is to be used:

- `AggregatedParallelCoordinates.dll` - contains the APC implementation
- `Avl.ClouClient.Parameters.dll` - contains the data parser and the Parameter hierarchy,
- `Avl.ClouClient.Utilities.dll` - contains utility classes used by the `AggregatedParallelCoordinates.dll`.

After the required libraries are added to the project references, a reference to the aggregated parallel coordinates assembly has to be added to the namespace of the root XAML tag within which the aggregated parallel coordinate plot is embedded as follows:

```
xmlns:apc="clr-namespace:AggregatedParallelCoordinates;assembly=
    AggregatedParallelCoordinates"
```

Once the assembly reference has been added to the namespace, the parallel coordinates plot can be added to the view as follows:

```
<apc:AggregatedParallelCoordinatesPlot/>
```

The code above will display an empty aggregated parallel coordinates plot.

```
public static DragDropEffects DoDragDrop(
  DependencyObject dragSource,
  Object data,
  DragDropEffects allowedEffects
)
```

Listing B.1: Declaration of the DoDragDrop method. The first parameter identifies the object which starts the drag-and-drop operation. The second parameter is the data payload. The third paramereter identifies the allowed visual effects to be shown while dragging the object.

B.1.1 Loading the Data

There are two ways to pass the data to the plot: by binding an Observable<Parameter> collection to the DataItem dependency property, which is exposed by the AggregatedParallelCoordinatesPlot class, or by using a drag-and-drop operation.

The DataItems property is declared as ObservableCollection<Parameter>. After the collection is populated with Parameter objects, the parallel coordinates plot will add one axis for each object, and as many polylines as there are record in the dataset. Any changes to the local reference of this collection will automatically be taken over by the parallel coordinate plot, which means that if additional objects are added or removed from the collection at runtime, these changes will automatically be processed by the plot, and the appropriate axes will be added or removed from the view. Assuming the DataContext object contains an ObservableCollection<Parameter> named PlotData, the following XAML will pass the collection to the plot:

```
<apc:AggregatedParallelCoordinatesPlot DataItems="{Binding PlotData}"/>
```

The collection of Parameter objects can be created either by creating the objects manually, or by parsing a CSV file. For CSV parsing, the ParameterParser implements the following method:

```
public static List<Parameter> Parse(string fullFilePath)
```

This method returns a list of Parameter objects, which will be set to null if the file cannot be parsed. If the Parameter collection contains any MeanValueParameter objects, these objects will automatically be shown as aggregated axes.

The second method of adding data to the plot, is by using drag-and-drop operations, which can be started by calling the DragDrop.DoDragDrop method. The declaration of this method is shown in Listing B.1. The second parameter to the method contains the data that is passed. The method is called as shown in Listing B.2. The important part is setting the data of the DataObject with the name "Parameter" and the Parameter object. When the ParallelCoordiante user control detects that an object is dropped, it will check if a data object with the name "Parameter" is passed. If so, the parameter will be added to the DataItems collection, and the plot will be reloaded with the new dataset. Note that both methods for data passing can be combined. An initial dataset can be added to the view, by passing a collection of Parameter objects. This collection can be updated at runtime either externally, by adding or removing items from the collection, or within the plot, by dropping Parameter objects on it.

B.1.2 Customising The Plot

A series of dependency properties are exposed in order to provide access to the implemented functionality. An overview of the exposed properties is provided in Table B.1. If any of the property val-

```
public void DoDragDrop(Object sender, Paremeter parameter)
{
  DataObject data = new DataObject();
  data.SetData("Parameter", parameter);
  DragDrop.DoDragDrop(sender, data, DragDropEffects.Move);
}
```

Listing B.2: Implementation of the DoDragDrop method for a selected Parameter. A DataObject is created with the name "Parameter" and a Parameter object to be dropped. This DataObject is then passed as the second parameter to the DoDragDrop method.

```
<apc:AggregatedParallelCoordinatesPlot Name="PCPlot"
ShowHistogram              = "{Binding Settings.ShouldShowHistograms}"
HistogramBinCount          = "{Binding Settings.HistogramBinCount}"
DataItems                  = "{Binding Settings.PlotData, Mode=TwoWay}"
ApplySameScalingOnAllAxes  = "{Binding Settings.ApplySameScaling}"
AllowAxisInversion         = "{Binding Settings.AllowAxisInversion}"
RotateAxisNameLabel        = "{Binding Settings.RotateNameAxes}"
ShouldShowMeanLine         = "{Binding Settings.ShowMeanLine}"
StringFormat               = "{Binding Settings.CurrentCultureSelection}"
UseRelativeValues          = "{Binding Settings.UseRelativeValues}"
ContextMenuItems           = "{Binding Settings.PolylineContextMenuItems}"
ReferenceRecordID          = "{Binding Settings.SelectedRecordId}"
UseBitmapCaching           = "{Binding Settings.UseBitmapCaching}"
/>
```

Listing B.3: AggregatedParallelCoordinatesPlot user control as referenced in the APC Testbed. Data binding is used on all dependency properties, since the user can customise the plot properties in a different view.

ues is changed at runtime, the plot will be reloaded with the new changes. An example of how the AggregatedParallelCoordinatesPlot user control is used within the APC Testbed is shown in Listing B.3.

The default colours of visual elements, and the line thickness of polylines in selected and normal states, are defined statically within the GlobalSettings class, as shown in Listing B.4. They can be accessed and changed externally.

B.1.2.1 Creating and Using a Record Context Menu

As shown in Table B.1, a collection of MenuItems objects can be passed to the plot. These objects will be shown in a context menu when a right-click is performed on a specific record in the plot. A MenuItem is created as follows:

```
MenuItem item = new MenuItem();
item.Header = "Make me green";
item.Tag = 1;
```

The Tag property must be set, so that the menu item can be identified later. When an item in the context menu is selected, a GlobalEventContextMenuItemSelected event will be triggered. This event is defined as follows:

```
public class GlobalEventContextMenuItemSelected :
    CompositePresentationEvent<ContextMenuSelectedItem>
```

Name	Type	Description
DataItems	ObservableCollection<Parameter>	A collection of Parameter objects which are to be visualised by the plot. The default value is null.
ShowHistogram	bool	If set to true, histograms are added to each axis. The default value is false.
HistogramBinCount	int	The number of histogram bins to be shown. The default value is 10.
ApplySameScalingOnAllAxes	bool	If set, the same (global) scaling is applied to all axes. The default value is false.
UseRelativeValues	bool	If set, the relative values are shown instead of absolute. The default value is false.
ReferenceRecordID	int	If the UseRelativeValues flag is set to true, the record with this ID is used as the reference value, and the values of all other records are set relative to the reference. The default value is -1, which is the ID of the mean record calculated by the plot.
AllowAxisInversion	bool	If set, invert button is added at the top of each axis. The default value is true.
ShouldShowMeanLine	bool	If set, a polyline representing the mean values of the current dataset is added to the plot. The default value is false.
UseBitmapCaching	bool	If set, the plot will use bitmap caching for performance optimization. The default value is true.
RotateAxisNameLabel	bool	If set, the axis name labels are rotated by 30° to the right. The default value is false.
StringFormat	IFormatProvider	Sets the language format for number representation. The default value is CultureInfo.CurrentCulture.
ContextMenuItems	ObservableCollection<MenuItem>	The collection of MenuItem objects which will be shown in a record context menu.

Table B.1: Dependency properties exposed by the ParallelCoordinates user control.

```
public static SolidColorBrush HIGHLIGHTED_COLOR;
public static SolidColorBrush DEFAULT_COLOR;
public static SolidColorBrush MULTISELECTED_COLOR;
public static SolidColorBrush IN_BACKGROUND_COLOR;
public static SolidColorBrush MEAN_LINE_COLOR;
public static SolidColorBrush SLIDER_DEFAULT_COLOR;
public static SolidColorBrush SLIDER_SELECTED_COLOR;

public static SolidColorBrush HISTOGRAM_FILL_COLOR;
public static SolidColorBrush LAYOUT_COLOR;
public static SolidColorBrush AXIS_HIGHLIGHT_COLOR;

public static double HIGHLIGHTED_LINE_THICKNESS;
public static double DEFAULT_LINE_THICKNESS;
```

Listing B.4: Static definitinon of default values of different visal properties within the GlobalSett ings class.

```
private void OnItemSelected(ContextMenuSelectedItem selectedItem)
{
 int recordId = selectedItem.RecordId;
 int tag = selctedItem.Tag;
 RecordProperty lp = GlobalSettings.RECORD_PROPERTY_MAPPING[recordId];
 if(tag == 1)
 {
   lp.SetDefaultColor(Burshes.Green);
 }
}
```

Listing B.5: Event handler implemementation for a selected record context menu item. The ID of the record, and the tag of the context menu item are retrieved from the event payload. Using these two parameters, the appropriate action can be performed upon the record.

The payload for this event is a ContextMenuSelectedItem object. This object contains a tag, which uniquely identifies which item in the context menu was selected, as well as the record for which the context menu was shown. In order to handle the selection of an item in the context menu externally, one has to subscribe to the GlobalEventContextMenuItemSelected event as follows:

```
GlobalEventContextMenuItemSelected.Instance.Subscribe(OnItemSelected);
```

OnItemSelected is an example of a delegate method which is executed when the event occurs. The ContextMenuSelectedItem payload is passed to the delegate automatically when an event occurs. The menu item tag and the polyline ID can be extracted from the payload. The polyline ID is the key of the selected record in the Dictionary<int, RecordProperty> mapping. The RecordProperty mapping contains the visual properties of the record polylines. So, assuming a context menu item “Make me green” was selected, the delegate method can be implemented as shown in Listing B.5.

B.1.2.2 Visualising and Applying Hamiltonian Permutations

The aggregated parallel coordinates plot calculates the minimal number of permutations of axis orderings which allows direct comparison between every pair of axes. Whenever these permutations change, the AggregatedParallelCoordinates plot publishes the following event:

```
public class GlobalEventHamiltonianPermutationChanged :
    CompositePresentationEvent<Dictionary<int, List<string>>>
```

The payload of this event is a mapping of the permutation ID to the list of names of the axes ordered as the axes are ordered in a permutation. The name of the axes are passed in order to support visualisation of the permutations. To apply an ordering permutation, simply publish the event GlobalEventApplyHamiltonianPermutationRequested , and pass the ID of the permutation as payload:

```
GlobalEventApplyHamiltonianPermutationRequested.Instance.Publish(id);
```

B.1.3 Exporting the Plot to an SVG File

The AggregatedParallelCoordinates user control can write the SVG file from the current representation. In order to make use of this functionality, the GlobalEventSVGExportRequested event has to be triggered as follows:

```
GlobalEventSVGExportRequested.Instance.Publish(pathToTheFileToWriteTo);
```

The payload of this event is of type `string`, and represents the path to the file in which the SVG representation of the plot will be written if the path exists.

B.2 Extending Aggregated Parallel Coordinates

The APC Testbed demo application, as well as the user control library itself, were implemented using Visual Studio 2013. The code provided as a part of this thesis contains, a `.sln` file. Opening this file in Visual Studio 2013 will load all necessary modules. After the project has been successfully loaded, the Solution Explorer panel will contain two projects:

- Testbed - contains the demo application code,
- AggregatedParallelCoordinates - contains the aggregated parallel coordinates user control code.

In the following sections, a walk-through of the code contained in these projects is provided, which should give developers an idea of how each project is implemented. The knowledge gained by reading the walk-through should provide a good starting point for extending both the plot and the demo application.

B.2.1 Testbed Project

The APC Testbed is implemented using the MVVM pattern [MSDN, 2015g]. The AvalonDock library is used to provide the docking layout [Xceed, 2015]. The structure of this project is shown in Figure B.1. The View consists of the following XAML files:

- `MainWindow.xaml` - defines the layout of the application using elements from the AvalonDock library,
- `MenuView.xaml` - defines the content of the toolbar,
- `GenerateDatasetView.xaml` - defines the content of the Generate Dataset dialogue,
- `DimensionTreeView.xaml` - defines the content of the Dimension Overview panel,

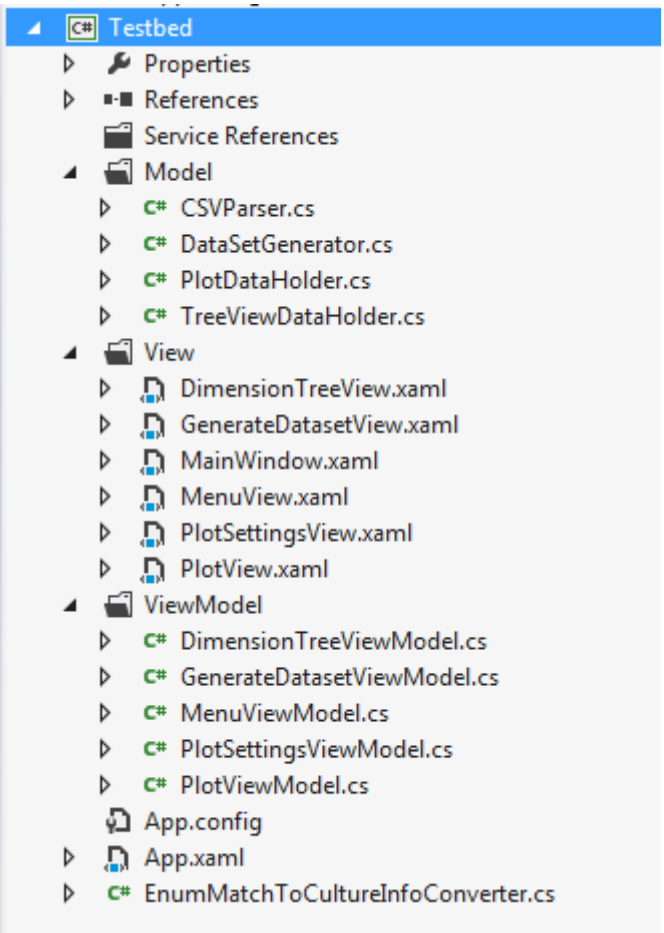


Figure B.1: The structure of the Testbed project as shown by the Visual Studio Solution Explorer. [Screenshot created by the author.]

- `PlotSettingsView.xaml` - defines the content of the Plot Settings panel,
- `PlotView.xaml` - defines how the parallel coordinates plot is used by passing all necessary data.

When the application is started, each of these views instantiates its own ViewModel class in the code-behind (the corresponding `.xaml.cs`file) , and sets its `DataContext` to that instance. The view objects are only responsible for the visual representation of their contents. The ViewModel classes contain the code that handles any changes in the view. All ViewModel classes have access to the classes in the `Testbed.Model` namespace. Apart from the CSVParser and DataSetGenerator, the Model contains two singleton classes: PlotDataHolder and TreeViewDataHolder. These two classes provide access to the resources shared between the ViewModel objects. The following example demonstrates the communication between the View, ViewModel, and Model objects when the "Import Dataset" toolbar button is pressed:

1. The view executes the `ImportDatasetCommand` which is defined within the MenuViewModel,
2. The delegate method which executes the `ImportDatasetCommand` in the MenuViewModel shows the "Open File" dialogue.
3. After the user selects the `.CSV` file to be imported, the MenuViewModel calls the appropriate method within the CSVParser, which belongs to the Model.
4. Assuming that the dataset is correctly parsed, the CSVParser returns a list of Parameter objects.

5. The MenuViewModel passes the parsing results to the TreeViewDataHolder, which creates the tree view hierarchy.

6. The DimensionTreeView observes the changes in the TreeViewDataHolder using the binding to the DimensionTreeViewModel and loads the tree view item as soon as they are available in the TreeViewDataHolder.

In order to add new controls to the application, the View has to be modified either by extending one of the already available visual modules, or by adding new ones. In case new modules are added, the `MainWindow.xaml` should be modified in order to define the position of the visual module in the application layout. A new ViewModel object has to be created, within which the changes in the displayed data and user interactions are handled using data binding and the Command pattern.

B.2.2 AggregatedParallelCoordinates Project

The "AggregatedParallelCoordinates" project contains the full implementation of the aggregated parallel coordinates plot. The structure of this project is shown in Figure B.2. The plot layout is defined in the `AggregatedParallelCoordinates.xaml` file, while the code-behind is responsible for adding the visual elements to the plot, resizing the plot to fit its contents, and handling the user interactions. When data is added to the plot either using the drag-and-drop, or by passing a collection of objects through the `DataItems` dependency property, the ReloadPlot method is executed. This method creates new visual components, and resizes the plot to fit them.

For each Parameter object within the dataset, an Axis object is created. The Axis object is responsible for creating and calculating the position of all necessary axis visual elements. The visual elements implement the IVisualAxisElement interface, and extend the FrameworkElement object either directly or indirectly. While the logical Axis is responsible for creating and positioning the visual elements, rendering is done by the IAxisElement elements. The following classes implement this interface:

- InvertButton,
- ExtremeLabel,
- Body,
- Slider,
- CollapseButton,
- ExpandButton,
- Histogram

For each AggregatedParallelCoordinatesPlot object, one AxisContainer and one DataSetManager object is created. The DataSetManager maintains information about the dataset as it is passed to the plot. AxisContainer, on the other hand, holds the collection of Axis objects, whose visual elements are currently displayed in the plot. This class contains a custom implementation of the OnCollectionChanged event. By listening to this event, the AggregatedParallelCoordinatesPlot monitors the state of the collection, in order to add or remove axis visual elements from the view when necessary. Within the AggregatedParallelCoordinatePlot, references to visual axis elements are distributed across several ObservableCollection instances, depending on the layer (the Canvas object) in which they are drawn.

Apart from creating the visual axis elements, the logical Axis is also responsible for calculating the position on the axis body of each dataset record within the dimension. An AxisPoint object is created for

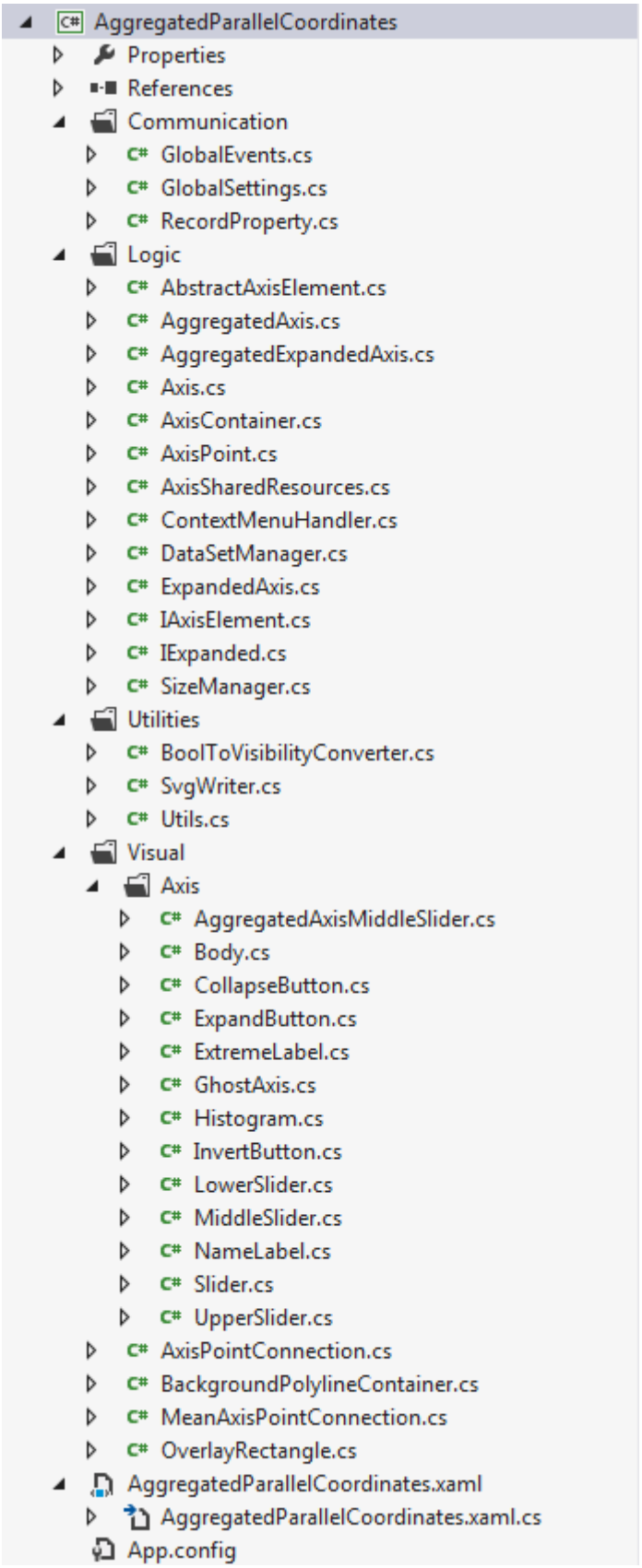


Figure B.2: The structure of the AggregatedParallelCoordinates project, as shown by the Visual Studio Solution Explorer. [Screenshot created by the author.]

each record. These objects contain information about a single record, such as its value and its position on the screen, and are mapped to the record ids.

For each record of the dataset, AxisPointConnection and RecordProperty objects are created. The RecordProperty object holds the information about the visual properties of each record. A mapping of RecordProperty objects to record IDs is created in the GlobalsSettings. This mapping is static, and is created only once, no matter how many times AggregatedParallelCoordinatesPlot is instantiated. This is done in order to share the information about the visual properties of records across several plots. Each AxisPointConnection object contains a reference to a corresponding RecordProperty object. Apart from that, each AxisPointConnection holds a reference to the AxisContainer. AxisPointConnection is derived from FrameworkElement, and overrides its OnRender method in order to draw a polyline based on the information about the position of the record points on the displayed axes, and the information contained in the RecordProperty object. The polylines are re-renderd every time the axis count or the scaling of the axes change, as well as when a visual property of a record changes (when a record is highlighted, for example).

A collection of AxisPointConnection objects is stored in the AggregatedParallelCoordinatesPlot, and drawn inside the corresponding `activePolylineCanvas`. The BackgroundPolyline is a visual object which is drawn in the `backgroundPolylineCanvas`. This object is rendered as a set of polylines, one for each item in the collection of AxisPointConnection objects. Re-rendering of these object is required only when the scaling of the axes changes, or when new items are added to the plot. Since it is drawn in the background layer, this object is covered by the items in the `activePolylineCanvas`, until the user starts filtering out records. For this reason, the rendering priority of this object is low, so the AggregatedParallelCoordinatesPlot renders it only when the application is in an idle state.

As previously mentioned, the AggregatedParallelCoordinatesPlot is also responsible for resizing the plot. Whenever new visual elements are added or removed from the plot, the it calls the AdaptCanvasSize() method implemented in the SizeManager class. This method calculates the distance and height of the axis body based on the current plot size. If any of these two parameters is smaller than the defined minimum, the plot is resized to fit all visual elements.

In addition to adding visual elements to the view and setting the plot size, the AggregatedParallelCoor dinatesPlot is also responsible for handling user interactions, global events, and changes in dependency properties. It overrides the following events:

- OnPreviewMouseLeftButtonDown,
- OnPreviewMouseLeftButtonUp,
- OnPreviewMouseRightButtonDown,
- OnPreviewMouseRightButtonUp,
- OnMouseMove,
- OnDragOver,
- OnDrop.

The following global events are defined in the `GlobalEvents` file:

- GlobalEventContextMenuItemSelected,
- GlobalEventSVGExportRequested,
- GlobalEventHamiltonianPermutationChanged,

- GlobalEventApplyHamiltonianPermutationRequested.

The GlobalEventHamiltonianPermutationChanged is triggered when the number of displayed axes changes, in order to notify subscribers about the minimal number of axes permutations which allow direct comparison between all axes. These permutations are calculated by the FindAdjacencyPermutations, which is defined in the Utils class. AggregatedParallelCoordinatesPlot subscribes to the GlobalEventApplyHamiltonianPermutationRequested event in order to apply a permutation when requested externally. The AggregatedParallelCoordinatesPlot also subscribes to the GlobalEventSVGExportRequired event, and writes an SVG representation of the current plot state to a file whose path is passed as a payload, when the event is triggered. The writing of the SVG file is done using the SvgWriter class. A GlobalEventContextMenuItemSelected event is published when an item in the polyline context menu is selected.

Bibliography

Adobe [2015]. *Flash*. 2015. http://adobe.com/de/products/flashplayer.html (cited on page 51).

Alcula [2009]. *Online Scatter Plot Generator*. 2009. http://alcula.com/calculators/statistics/scatter-plot/ (cited on page 22).

Andrews, Keith [1996]. "Browsing, Building, and Beholding Cyberspace: New Approaches to the Navigation, Construction, and Visualisation of Hypermedia on the Internet". PhD thesis. Graz University of Technology, Austria, Sept. 1996. http://ftp.iicm.tugraz.at/pub/keith/phd/andrews-1996-phd.pdf (cited on page 13).

Andrews, Keith [2014]. *Fluid Diagrams*. 2014. http://projects.iicm.tugraz.at/fluiddiagrams/ (cited on page 16).

Andrews, Keith [2015]. *Information Visualisation: Lecture Notes*. 2015. http://courses.iicm.tugraz.at/ivis/ivis.pdf (cited on pages 9, 13).

Andrews, Keith and Helmut Heidegger [1998]. "Information Slices: Visualising and Exploring Large Hierarchies Using Cascading, Semi–Circular Discs". In: *Late Breaking Hot Topic Paper, IEEE Symposium on Information Visualization (InfoVis'98)*. (Reseach Triangle Park, North Carolina, USA). Oct. 1998, pages 9–11. http://ftp.iicm.tugraz.at/pub/papers/ivis98.pdf (cited on pages 17, 19).

Andrews, Keith, Wolfgang Kienreich, et al. [2002]. "The InfoSky Visual Explorer: Exploiting Hierarchical Structure and Document Similarities". *Information Visualization* 1.3/4 (Dec. 2002), pages 166–181. ISSN 1473-8716. doi:10.1057/palgrave.ivs.9500023 (cited on pages 17, 19).

Andrews, Keith, Werner Putz, and Alexander Nussbaumer [2007]. "The Hierarchical Visualisation System (HVS)". In: *Proc. 11th International Conference on Information Visualization (IV '07)*. (Zurich, Switzerland). IEEE Computer Society, July 2007, pages 257–262. doi:10.1109/IV.2007.112 (cited on page 14).

Andrews, Keith, Josef Wolte, and Michael Pichler [1997]. "Information Pyramids: A New Approach to Visualising Large Hierarchies". In: *Late Breaking Hot Topics, Proc. IEEE Visualisation '97 (Vis '97)*. (Phoenix, Arizona, USA). Oct. 19, 1997, pages 49–52. http://ftp.iicm.tugraz.at/pub/papers/vis97.pdf (cited on page 17).

Andrienko, Gennady and Natalia Andrienko [2004]. "Parallel Coordinates for Exploring Properties of Subsets". In: *Proc. 2nd International Conference on Coordinated & Multiple Views in Exploratory Visualization (CMV '04)*. IEEE Computer Society, July 13, 2004, pages 93–104. ISBN 0769521797. doi:10.1109/CMV.2004.13. http://carpex.usal.es:8080/anai/upload/andrienko2004.pdf (cited on pages 39, 40).

Artero, Almir Olivette, Maria Cristina Ferreira de Oliveira, and Haim Levkowitz [2004]. "Uncovering Clusters in Crowded Parallel Coordinates Visualizations". In: *Proc. IEEE Symposium on Information Visualization (InfoVis 2004)*. (Austin, Texas, USA). IEEE Computer Society, Oct. 10, 2004,

pages 81–88. ISBN 0780387791. doi:10.1109/INFOVIS.2004.68. http://vis.computer.org/vis2004/dvd/infovis/papers/artero.pdf (cited on page 39).

Avidan, Tova and Shlomo Avidan [1999]. "ParallAX — A Data Mining Tool Based on Parallel Coordinates". *Computational Statistics* 14.1 (1999), pages 79–89 (cited on pages 41–45, 52, 54).

AVL [2015]. *AVL Racing*. 2015. https://www.avl.com/racing (cited on page 57).

Balzer, Michael, Oliver Deussen, and Claus Lewerentz [2005]. "Voronoi Treemaps for the Visualization of Software Metrics". In: *Proc. ACM Symposium on Software Visualization (SoftVis 2005)*. (St. Louis, Missouri, USA). ACM, May 14, 2005, pages 165–172. ISBN 1595930736. doi:10.1145/1056018.1056041. http://kops.uni-konstanz.de/handle/123456789/6011;jsessionid=DE2110778233D6A5656B3984D504BA00 (cited on page 17).

Beaudoin, Luc, Marc-Antoine Parent, and Louis C. Vroomen [1996]. "Cheops: a Compact Explorer for Complex Hierarchies". In: *Proc. 7th IEEE Visualisation Conference (Vis '96)*. (San Francisco, California, USA). IEEE Computer Society, Oct. 1996, pages 87–92. doi:10.1109/VISUAL.1996.567745. http://pages.infinit.net/lbeaudoi/cheops.html (cited on page 17).

Buchheim, Christoph, Michael Jünger, and Sebastian Leipert [2002]. "Improving Walker's Algorithm to Run in Linear Time". In: *Proc. 10th International Symposium on Graph Drawing (GD 2002)*. (Irvine, California, USA). Lecture Notes in Computer Science. Springer, Aug. 26, 2002, pages 347–364. ISBN 3540361510. doi:10.1007/3-540-36151-0. http://dirk.jivas.de/papers/buchheim02improving.pdf (cited on page 13).

Buering, Thorsten, Jens Gerken, and Harald Reiterer [2006]. "User Interaction with Scatterplots on Small Screens – A Comparative Evaluation of Geometric–Semantic Zoom and Fisheye Distortion". *Transactions on Visualization and Computer Graphics* 12 (2006), pages 558–568. doi:10.1109/TVCG.2006.187 (cited on page 21).

Card, Stuart K., Jock D. Mackinlay, and Ben Shneiderman [1999]. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1999. ISBN 1558605339 (cited on page 9).

Chernoff, Hermann [1973]. "The Use of Faces To Represent Points in K-Dimensional Space Graphically". *Journal of the American Statistical Association* 68.342 (June 1973), pages 361–368. doi:10.2307/2284077. http://lya.fciencias.unam.mx/rfuentes/faces-chernoff.pdf (cited on page 24).

Cook, Dianne and Deborah F. Swayne [2007]. *Interactive and Dynamic Graphics for Data Analysis With R and GGobi*. Use R! Springer, 2007. ISBN 0387717617. http://ggobi.org/ (cited on pages 49, 50).

d'Ocagne, Maurice [1885]. *Coordonnées parallèles et axiales : Méthode de transformation géométrique et procédé nouveau de calcul graphique déduits de la considération des coordonnées parallèles*. Gauthier–Villars, 1885. https://archive.org/details/coordonnesparal00ocaggoog (cited on page 33).

DotPDN LLC [2015]. *paint.net*. 2015. http://getpaint.net/index.html (cited on pages 26, 39, 47, 49).

Ellis, G. and A. Dix [2006]. "Enabling Automatic Clutter Reduction in Parallel Coordinate Plots". *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pages 717–724. doi:10.1109/TVCG.2006.138 (cited on page 37).

Enguerrand de Rochefort [2015]. *XDAT*. 2015. http://xdat.org/index.php (cited on pages 50, 51).

Fédération Internationale de l'Automobile [2015]. *F1 Technical Regulations*. 2015. http://formula1.com/inside_f1/rules_and_regulations/technical_regulations/ (cited on page 3).

Gillespie, Thomas D. [1992]. *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, 1992. ISBN 1560911999 (cited on page 4).

Graham, Martin and Jessie Kennedy [2003]. "Using Curves to Enhance Parallel Coordinate Visualisations". In: *Proc. 7th International Conference on Information Visualisation (IV 2003)*. (London, UK). IEEE Computer Society, July 16, 2003, pages 10–16. ISBN 0769519881. doi:10.1109/IV.2003.1217950. http://www.iidi.napier.ac.uk/c/publications/publicationid/2760350 (cited on page 48).

Hackl, Christian [2011]. "Parallel Coordinates – Exploratory Data Analysis with Parallel Coordinates and the Multi-Dimensional Explorer". Master's Thesis. Institute for Information Systems and Computer Media (IICM): Graz University of Technology, Mar. 15, 2011. http://ftp.iicm.tugraz.at/pub/theses/chackl.pdf (cited on page 46).

Hauser, Helwig, Florian Ledermann, and Helmut Doleisch [2002]. "Angular Brushing of Extended Parallel Coordinates". In: *Proc. IEEE Symposium on Information Visualization (InfoVis 2002)*. (Boston, Massachussets, USA). IEEE Computer Society, Oct. 28, 2002, pages 127–130. ISBN 076951751X. doi:10.1109/INFVIS.2002.1173157. http://mediavirus.org/parvis/parvis_full.pdf (cited on pages 33, 34).

Hughes, Timothy, Young Hyung, and David A. Liberles [2004]. "Visualising Very Large Phylogenetic Trees in Three Dimensional Hyperbolic Space". *BMC Bioinformatics* 5 (Apr. 2004), page 48. doi:10.1186/1471-2105-5-48. http://caida.org/publications/papers/2004/bioinformatics/ (cited on page 14).

Inselberg, Alfred [2010]. *Parallel Coordinates, Visual Multidimensional Geometry and its Applications*. Springer Science + Business Media, 2010. ISBN 0387215077 (cited on pages 33, 35, 36, 41, 45, 46, 53).

Investopedia [2015]. *Standard & Poor's 500 Index - S&P 500*. 2015. http://investopedia.com/terms/s/sp500.asp (cited on page 41).

iRacing [2015]. *Vehicle Setup Components*. 2015. http://iracing.wikidot.com/components (cited on page 6).

Jern, Mikael [2009]. "Collaborative Web-Enabled GeoAnalytics Applied to OECD Regional Data". In: *Cooperative Design, Visualization, and Engineering*. Edited by Yuhua Luo. Volume 5738. Lecture Notes in Computer Science. Springer, 2009, pages 32–43. ISBN 3642042643. doi:10.1007/978-3-642-04265-2_5 (cited on page 51).

Johansson, Jimmy, Matthew Cooper, and Mikael Jern [2005]. "3-Dimensional Display for Clustered Multi-Relational Parallel Coordinates". In: *Proc. 9th International Conference on Information Visualisation (IV 2005)*. (London, UK). IEEE Computer Society, July 6, 2005, pages 188–193. ISBN 078039464x. doi:10.1109/IV.2005.1. http://webstaff.itn.liu.se/~jimjo/papers/IV05/paperIV05.pdf (cited on pages 46, 47).

Johansson, Jimmy, Patric Ljung, et al. [2005]. "Revealing Structure Within Clustered Parallel Coordinates Displays". In: *Proc. IEEE Symposium on Information Visualization (InfoVis 2005)*. (Minneapolis, Minnesota, USA). Oct. 23, 2005, pages 125–132. ISBN 078039464X. doi:10.1109/INFVIS.2005.1532138. http://ifs.tuwien.ac.at/~mlanzenberger/teaching/ps/ws07/stuff/Johansso-IV2005.pdf (cited on page 39).

Johnson, Brian and Ben Shneiderman [1991]. "Tree–Maps: A Space–Filling Approach to the Visualization of Hierarchical Information Structures". In: *Proc. IEEE Visualization '91*. San Diego, California, USA: IEEE Computer Society, Oct. 1991, pages 284–291. doi:10.1109/VISUAL.1991.175815. https://cs.umd.edu/~ben/papers/Johnson1991Tree.pdf (cited on page 14).

Katz, Joseph [1995]. *Race Car Aerodynamics - Designing for Speed*. Bentley Publishers, 1995. ISBN 0837601428 (cited on page 6).

Keim, Daniel A., Ming C. Hao, et al. [2010]. "Generalized Scatter Plots". *Information Visualisation* 9.4 (Dec. 2010), pages 301–311. doi:10.1057/ivs.2009.34. http://kops.uni-konstanz.de/bitstream/handle/123456789/17475/Keim.pdf?sequence=1 (cited on page 21).

Keim, Daniel A., Florian Mansmann, et al. [2006]. "Challenges in Visual Data Analysis". In: *Proc. 10th International Conference on Information Visualization (IV 2006)*. (London, UK). IEEE. July 5, 2006, pages 9–16. ISBN 0769526020. doi:10.1109/IV.2006.31. http://bib.dbvis.de/uploadedFiles/87.pdf (cited on page 10).

Khan, Suniya Sadullah [2007]. "Analysis of Simulation Techniques and Taguchi Methods as Applied to Optimise the Setup of a Formula 3 Race Car". Master's thesis. Cranfield University, 2007 (cited on pages 3, 4, 6).

Kohlhardt, Chris and Clint Dickson [2015]. *Gliffy*. Mar. 12, 2015. http://gliffy.com/ (cited on page 24).

Kosara, Robert, Fabian Bendix, and Helwig Hauser [2006]. "Parallel Sets: Interactive Exploration and Visual Analysis of Categorical Data". *Transactions on Visualization and Computer Graphics* 12.4 (2006), pages 558–568. doi:10.1109/TVCG.2006.76. http://kosara.net/papers/2006/Kosara_TVCG_2006.pdf (cited on pages 48, 49).

Kreuseler, Matthias and Heidrun Schumann [1999]. "Information Visualization Using a New Focus+Context Technique in Combination with Dynamic Clustering of Information Space". In: *Proc. 1999 Workshop on New Paradigms in Information Visualization and Manipulation (NPIVM '99)*. (Kansas City, Missouri, USA). ACM, Nov. 6, 1999, pages 1–5. ISBN 1581132549. doi:10.1145/331770.331772 (cited on page 14).

Lamping, John O. and Ramana B. Rao [1997]. "Displaying Node-Link Structure With Region of Greater Spacings and Peripheral Branches". 5619632. Apr. 1997. http://freepatentsonline.com/5619632.html (cited on page 14).

Lamping, John and Ramana Rao [1994]. "Laying out and Visualizing Large Trees Using a Hyperbolic Space". In: *Proc. 7th Annual ACM Symposium on User Interface Software and Technology (UIST '94)*. (Marina del Rey, California, USA). ACM, Nov. 2, 1994, pages 13–14. ISBN 0897916573. doi:10.1145/192426.192430. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.6827&rep=rep1&type=pdf (cited on pages 13, 16).

Lichman, M. [2013]. *UCI Machine Learning Repository*. 2013. https://archive.ics.uci.edu/ml/datasets/Auto+MPG (cited on page 28).

MarketWatch [2015]. *Map of the Market*. 2015. http://marketwatch.com/tools/stockresearch/marketmap (cited on page 18).

Math Images [2015]. *Hamiltonian Path*. 2015. http://mathforum.org/mathimages/index.php/Hamiltonian_Path (cited on page 35).

McDonnell, K. T. and K. Mueller [2008]. "Illustrative Parallel Coordinates". In: *Proc. 10th Joint Eurographics / IEEE – VGTC Conference on Visualization (EuroVis 2008)*. (Eindhoven, The Netherlands). Eurographics, May 26, 2008, pages 1031–1038. doi:10.1111/j.1467-8659.2008.01239.x. http://www3.cs.stonybrook.edu/~mueller/papers/ktm-eurovis2008.pdf (cited on page 37).

Microsoft [2015a]. *CLR Profiler for .NET Framework 4*. 2015. http://microsoft.com/en-us/download/details.aspx?id=16273 (cited on page 84).

Microsoft [2015b]. *Excel*. 2015. https://products.office.com/en-us/Excel (cited on pages 28, 29).

Microsoft [2015c]. *Visual Studio 2013 Update 4*. 2015. http://microsoft.com/en-us/download/details.aspx?id=44921 (cited on page 57).

Milliken, William F. and Douglas L. Milliken [1995]. *Race Car Vehicle Dynamics*. Society of Automotive Engineers, 1995. ISBN 1560915269 (cited on pages 3, 4, 6).

Ming, Hao C. et al. [2010]. "Visual Analytics of Large Multidimensional Data Using Variable Binned Scatter Plots". In: *Proc. Visualization and Data Analysis (VDA 2010)*. (San Jose, CA, USA). Volume 7530. SPIE Proceedings. SPIE. Macmillan, Jan. 18, 2010. doi:10.1117/12.840142. http://www.inf.uni-konstanz.de/gk/pubsys/publishedFiles/MiDaSh10.pdf (cited on page 21).

Morris, Christopher J., David S. Ebert, and Penny L. Rheingans [2000]. "Experimental Analysis of the Effectiveness of Features in Chernoff Faces". In: *Proc. 28th AIPR Workshop: 3D Visualization for Data Exploration and Decision Making*. Volume 3905. Society of Photo Optical, June 2000, pages 12–17. ISBN 0819435171. doi:10.1117/12.384865. https://engineering.purdue.edu/~ebertd/papers/Chernoff_990402.PDF (cited on page 26).

Moser, Christian [2015]. *The Differences Between CustomControls and UserControls*. 2015. http://wpftutorial.net/customvsusercontrol.html (cited on page 74).

MSDN [2015a]. *BitmapCache Class*. 2015. https://msdn.microsoft.com/en-us/library/system.windows.media.bitmapcache(v=vs.110).aspx (cited on page 80).

MSDN [2015b]. *ClearType Overview*. 2015. https://msdn.microsoft.com/en-us/library/ms749295(v=vs.110).aspx (cited on page 80).

MSDN [2015c]. *Control Class*. 2015. https://msdn.microsoft.com/en-us/library/system.windows.controls.control(v=vs.110).aspx (cited on page 74).

MSDN [2015d]. *Graphics Rendering Tiers*. 2015. https://msdn.microsoft.com/en-us/library/ms742196.aspx (cited on page 73).

MSDN [2015e]. *Introduction to WPF*. 2015. https://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx (cited on pages 53, 57).

MSDN [2015f]. *Optimizing Performance: 2D Graphics and Imaging*. 2015. https://msdn.microsoft.com/en-us/library/bb613591(v=vs.110).aspx (cited on page 76).

MSDN [2015g]. *WPF Apps With The Model-View-ViewModel Design Pattern*. 2015. https://msdn.microsoft.com/en-us/magazine/dd419663.aspx (cited on page 116).

Munzner, Tamara [2014]. *Visualization Analysis and Design*. A K Peters/CRC Press, 2014. ISBN 1466508914 (cited on page 9).

Munzner, Tamara and Paul Burchard [1995]. "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space". In: *Proc. 1st Symposium on Virtual Reality Modeling Language (VRML 95)*. (San Diego, California, USA). New York, NY, USA: ACM, Dec. 13, 1995, pages 33–38. ISBN 0897918185. doi:10.1145/217306.217311. http://graphics.stanford.edu/papers/webviz/webviz.72dpi.pdf (cited on page 14).

NComVA [2014]. *OECD Regional eXplorer*. Oct. 17, 2014. http://stats.oecd.org/OECDregionalstatistics (cited on pages 23, 25, 31, 51, 52).

Oracle [2015]. *Java*. 2015. http://java.com/en/ (cited on page 50).

Pirolli, Peter and Ramana Rao [1996]. "Table Lens As a Tool for Making Sense of Data". In: *Proc. Workshop on Advanced Visual Interfaces (AVI'96)*. (Gubbio, Umbria, Italy). Gubbio, Italy: ACM, May 27, 1996, pages 67–80. ISBN 0897918347. doi:10.1145/948449.948460. http://www2.parc.com/istl/projects/uir/publications/items/UIR-1996-06-Pirolli-AVI96-TableLens.pdf (cited on page 24).

Rao, Ramana B. and Stuart K. Card [1997]. “Method and System for Producing a Table Image Showing Indirect Data Representations”. 5632009. May 1997. http://freepatentsonline.com/5632009.html (cited on page 21).

Rao, Ramana and Stuart K. Card [1995]. “Exploring Large Tables with the Table Lens”. In: *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI’95)*. (Denver, Colorado, USA). ACM, May 7, 1995, pages 403–404. ISBN 0897917553. doi:10.1145/223355.223745. http://sigchi.org/chi95/proceedings/videos/rr_bdy.htm (cited on pages 21, 24).

Reingold, Edward M. and John S. Tilford [1981]. “Tidier Drawings of Trees”. *IEEE Transactions on Software Engineering* 7.2 (Mar. 1981), pages 223–228. ISSN 0098-5589. doi:10.1109/TSE.1981.234519. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.150.3559&rep=rep1&type=pdf (cited on page 14).

Robertson, George G., Jock D. Mackinlay, and Stuart K. Card [1991]. “Cone Trees: Animated 3D Visualizations of Hierarchical Information”. In: *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI’91)*. (New Orleans, Louisiana, USA). ACM, May 1991, pages 189–194. ISBN 0897913833. doi:10.1145/108844.108883. http://www2.parc.com/istl/groups/uir/publications/items/UIR-1991-06-Robertson-CHI91-Cone.pdf (cited on pages 13, 16).

Robertson, George G., Jock Mackinlay, and Stuart K. Card [1994]. “Display of Hierarchical Three-Dimensional Structures With Rotating Substructures”. 5295243. Mar. 1994. http://freepatentsonline.com/5295243.html (cited on page 13).

Schulz, Hans Jörg [2011]. “Treevis.net: A Tree Visualization Reference”. *Computer Graphics and Applications, IEEE* 31.6 (Nov. 2011), pages 11–15. doi:10.1109/MCG.2011.103 (cited on page 13).

Schulz, Hans Jörg [2015]. *A Visual Bibliography of Tree Visualization*. 2015. http://vcg.informatik.uni-rostock.de/~hs162/treeposter/poster.html (cited on page 13).

Schulz, Hans Jörg, Steffen Hadlak, and Heidruno Schumann [2011]. “The Design Space of Implicit Hierarchy Visualization: A Survey”. *IEEE Transactions on Visualization and Computer Graphics* 17.4 (Apr. 2011), pages 393–411. doi:10.1109/TVCG.2007.7051. http://www.informatik.uni-rostock.de/~hs162/pdf/tvsurvey.pdf (cited on page 12).

Shneiderman, Ben [1996]. “The Eyes Have It: A Task by Data Type Taxonomy For Information Visualizations”. In: *Proc. IEEE Symposium on Visual Languages (VL’96)*. (Boulder, Colorado, USA). IEEE Computer Society Press, Sept. 3, 1996, pages 336–343. doi:10.1109/VL.1996.545307. https://cs.umd.edu/~ben/papers/Shneiderman1996eyes.pdf (cited on pages 11, 63).

Shneiderman, Ben and Martin Wattenberg [2001]. “Ordered Treemap Layouts”. In: *Proc. IEEE Symposium on Information Visualization (InfoVis 2001)*. (San Diego, California, USA). IEEE Computer Society, Oct. 22, 2001, pages 73–78. doi:10.1109/INFVIS.2001.963283. https://cs.umd.edu/~ben/papers/Shneiderman2001Ordered.pdf (cited on pages 14, 18).

Smith, Carroll [1978]. *Tune To Win - The Art and Science of Race Car Development and Tuning*. Aero Publishers, 1978. ISBN 0879380713 (cited on page 4).

Stasko, John T. and Eugene Zhang [2000]. “Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualizations”. In: *Proc. IEEE Symposium on Information Visualization (InfoVis 2000)*. (Salt Lake City, Utah, USA). IEEE Computer Society, Oct. 2000, pages 57–65. doi:10.1109/INFVIS.2000.885091. http://cc.gatech.edu/gvu/ii/sunburst/ (cited on page 17).

Strasnick, Steven L. and Joel D. Tesler [1996]. “Method and Apparatus for Displaying Data Within a Three-Dimensional Information Landscape”. 5528735. June 1996. http://freepatentsonline.com/5528735.html (cited on pages 13, 15).

Swayne, Deborah F. et al. [2006]. *GGobi Manual*. Sept. 2006. http://ggobi.org/docs/manual.pdf (cited on page 49).

The GTK+ Team [2014]. *The GTK+ Project*. 2014. http://gtk.org/ (cited on page 49).

Tufte, Edward R. [2007]. *The Visual Display of Quantitative Information*. 2nd edition. Graphics Press, 2007. ISBN 0961392142 (cited on page 27).

Van Wijk, Jarke J. and Huum van de Wetering [1999]. "Cushion Treemaps: Visualization of Hierarchical Information". In: *Proc. IEEE Symposium on Information Visualization (InfoVis '99)*. (San Francisco, California, USA). IEEE Computer Society, Oct. 24, 1999, pages 73–78, 147. doi:10.1109/INFVIS.1999.801860. http://www.win.tue.nl/~vanwijk/ctm.pdf (cited on page 17).

Walker, John Q. II [1990]. "A Node-Positioning Algorithm for General Trees". *Software – Practice and Experience* 20.7 (July 1990), pages 685–705. doi:10.1002/spe.4380200705. http://cs.unc.edu/techreports/89-034.pdf (cited on pages 13, 14).

Ward, Matthew, Georges Grinstein, and Daniel Keim [2010]. *Interactive Data Visualisation – Foundations, Techniques and Applications*. A.K. Peters, 2010. ISBN 1568814739 (cited on pages 9, 20, 21).

Wikipedia [2014]. *Radar Chart*. Oct. 17, 2014. http://en.wikipedia.org/wiki/Radar_chart (cited on page 27).

Wlodek, Piotr [2009]. *Parallel Coordinates in WPF*. Apr. 6, 2009. http://pwlodek.blogspot.co.at/2009/04/parallel-coordinates-in-wpf-part-1.html (cited on pages 53, 55, 74).

Wlodek, Piotr [2010]. *WPF Parallel Coordinates Source Code on GitHub*. Oct. 10, 2010. https://github.com/pwlodek/CodeGallery/tree/master/src/ParallelCoordinatesDemo (cited on pages 53, 74).

Xceed [2015]. *AvalonDock*. 2015. https://avalondock.codeplex.com/ (cited on page 116).

Yale [1997]. *Scatterplot*. Yale University. 1997. http://www.stat.yale.edu/Courses/1997-98/101/scatter.htm (cited on page 20).

Yi, Ji Soo et al. [2007]. "Toward a Deeper Understanding of the Role of Interaction in Information Visualization". *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pages 1224–1231. doi:10.1109/TVCG.2007.7051. http:/cc.gatech.edu/~stasko/papers/infovis07-interaction.pdf (cited on page 11).

Ying–Huey, Fua, M.O. Ward, and E.A. Rundensteiner [1999]. "Hierarchical Parallel Coordinates for Exploration of Large Datasets". In: *Proc. 10th IEEE Visualization Conference (Vis'99)*. (San Francisco, CA, USA). IEEE Computer Society, Oct. 24, 1999, pages 43–508. ISBN 078035897X. doi:10.1109/VISUAL.1999.809866. http://www-devel.cs.ubc.ca/~tmm/courses/533/readings/vis99_HPC.pdf (cited on pages 37, 39, 63).

Zhou, Hong, Weiwei Cui, et al. [2009]. "Splatting the Lines in Parallel Coordinates". In: *Proc. 11th Eurographics / IEEE – VGTC Conference on Visualization (EuroVis 2009)*. (Berlin, Germany). Eurographics, June 10, 2009, pages 759–766. doi:10.1111/j.1467-8659.2009.01476.x. http://cse.ust.hk/~huamin/euvis09_hong.pdf (cited on page 39).

Zhou, Hong, Xu Panpan, et al. [2013]. "Edge Bundling in Information Visualization". *Tsinghua Science and Technology* 18.2 (Apr. 2013), pages 145–156. doi:10.1109/TST.2013.6509098. http://vis.pku.edu.cn/research/publication/tsinghuaSci&Tech13_edgeBundling.pdf (cited on page 37).

Zhou, Hong, Xiaoru Yuan, et al. [2008]. "Visual Clustering in Parallel Coordinates". *Computer Graphics Forum* 27.3 (Sept. 29, 2008), pages 1047–1054. doi:10.1111/j.1467-8659.2008.01241.x. http://www.cse.ust.hk/~huamin/eurovis08_zhou.pdf (cited on pages 37, 40).