Exploration of Gene Expression Data in a Visually Linked Environment

Master’s Thesis

Alexander Lex, Bakk.rer.soc.oec.
alex@icg.tugraz.at

May 2008

Supervision:
Univ. Prof. DI Dr. techn. Dieter Schmalstieg
Abstract

The development of high throughput genetic analysis techniques, such as microarray DNA chips, opened the gates for the simultaneous analysis of gene expression of thousands of genes. This data can be used for diagnostic and research purposes alike. With ever more data being generated, the problem of deriving knowledge from it increases, not at last due to its multidimensionality. In this thesis the integrated application of two established multidimensional information visualization methods is proposed: parallel coordinates and heat maps.

The true value of knowledge about gene expression regulation can only be exploited when information about the biological context is available. That knowledge is modeled in special graphs, called pathways. Previous work in the Caleydo framework established a 2.5D pathway visualization method. The goal of this thesis is to bring the pathways into context with high level gene expression visualization methods, by employing visual links among pathway elements and expression data. The visual links are drawn among up to five independent views, arranged in 3D, simultaneously, thereby showing relations which were previously difficult to identify.

Keywords: Parallel coordinates, heat map, visual links, gene expression, pathways, Caleydo
Zusammenfassung


Pledge

I hereby certify that the work presented in this master’s thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.
Acknowledgements

I would like to express my gratitude to my supervisor, Univ. Prof. DI Dr. techn. Dieter Schmalstieg for his continuing support and guidance and for the provision of facilities at the institute. I am especially thankful to my friend DI Marc Streit, with whom I closely collaborated during this thesis, and since the very first day of my studies. He was always there to discuss all kinds of problems, even long after working hours over a beer or a game of squash. Thanks also to DI Michael Kalkusch for his initial ideas and pointers.

I will be forever grateful for the continuing support of my amazing girlfriend MMag Heidemarie Purgstaller, who was always understanding of my physical and social absence in the last months. Thank you for all our past and future years.

My parents, Edith and Georg Lex are the ones who enabled me to persue my studies in the way I did. Thank you for your continuing help, guidance and understanding through all my life. Thank you, Richard, for being my brother, all our adventures and for helping me get my mind off work from time to time.

Thanks to Berni, for riding the mountains with me for 15 years now and Robert for rocking with me and being the friend that you are.

Christian, Martin, Tobias and Richard, thank you for your friendship and collaboration on all the assignments over the years.

Finally, I would like to thank all members of the institute for helping out whenever needed and welcoming me as you have.
# Contents

1 Introduction
   1.1 Problem Statement and Contribution ........................................... 8
   1.2 Structure of this Document ....................................................... 9
   1.3 Biological Background
      1.3.1 Gene Expression Regulation ............................................... 11
      1.3.2 Pathways ................................................................. 12

2 Related Work
   2.1 Information Visualization ......................................................... 14
      2.1.1 Overview First, Focus+Context ......................................... 15
      2.1.2 Zoom and Filter ......................................................... 17
      2.1.3 Details on Demand ....................................................... 17
      2.1.4 Relate: Multiple Views, Linking & Brushing ................................ 17
      2.1.5 History ................................................................. 19
      2.1.6 Extract ................................................................. 20
   2.2 Parallel Coordinates
      2.2.1 Crossing Problem .......................................................... 21
      2.2.2 Viewing Multiple Relations .............................................. 23
      2.2.3 Viewing many Dimensions ................................................. 25
      2.2.4 Cluttering
         2.2.4.1 Using Transparency ................................................... 28
         2.2.4.2 Random Sampling ...................................................... 29
         2.2.4.3 Clustering ............................................................. 29
         2.2.4.4 Density based highlighting .......................................... 31
         2.2.4.5 Polyline Averaging .................................................... 31
      2.2.5 Brushing and Interacting in Parallel Coordinates ......................... 32
      2.2.6 Multiple View Integration ................................................. 33
      2.2.7 Extensions ............................................................... 34
      2.2.8 Application to Gene Expression visualization .................................. 34
      2.2.9 Discussion ............................................................... 35
   2.3 Heat Maps ................................................................. 37
      2.3.1 Discussion ............................................................... 39
   2.4 Pathway Visualization ............................................................ 39
      2.4.1 Discussion ............................................................... 41
   2.5 Visual Links ............................................................... 41
      2.5.1 Discussion ............................................................... 43

3 Concept
   3.1 Parallel Coordinates ........................................................... 46
Chapter 1

Introduction

Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

Winston Churchill

The sequencing of the human genome finished by the year 2003 brought the ongoing research effort to understand the foundations of life to the conscious of the public. This process began in 1952 by the work of Sanger et al., who realized that proteins are perfectly ordered and concluded that there must be a mechanism encoding those proteins [Eisenberg2006]. However, the sequencing of the genome does not mark an end of those efforts. On the contrary, only few genes are considered to be fully understood, the function of a majority is still unknown.

All that new knowledge of genomic sequences of many different species, gene functions, etc. made new ways of conducting research in the life sciences necessary. As a consequence, the field of bioinformatics emerged. With bioinformatics methods, one can, for example, compare a given sequence of amino acids to the genome of thousands of species within seconds or browse huge databases maintained by the different biotechnology institutions.

Advances in technology have made it possible to analyze the expression regulation of all genes simultaneously using microarrays, thus allowing to build profiles, which are used for example for cancer classification and personalized medicine. Such data can contain tens of thousands of data entries per experiment, with an ever growing number of experiments being conducted. This makes it necessary to analyze the data with computer aided methods. Statistical analysis is commonly used, but due to its nature, has its limitations.

Thus the importance of the field of information visualization in the life sciences has grown. Information visualization utilizes the human capability of understanding pictures better than just lists of numbers. It allows the discovery of knowledge in data that would not be possible with other means.

1.1 Problem Statement and Contribution

Three years ago the Graz University of Technology was approached by the Institute for Pathology of the Medical University of Graz with the request to form a partnership in developing a visualization framework for genetic data. A visualization framework - Caleydo - has been developed at the Institute for Computer Graphics and Vision. The first 2 years
where dedicated to the development of the backbone of the system, followed by a focus on
the visualization of pathways and their interdependencies. Those pathways were enriched
with information on gene expression regulation, however, the analysis of gene expression
regulation by itself was not yet implemented. This work focuses on this topic: the analysis
of gene expression data with the means of information visualization, in order to put them
into context of pathways (that model biological processes). The following questions can be
answered with such an analysis:

- What genes have a high or low expression?
- What genes are expressed differently in different experiments (for example, healthy
tissue vs. tumor tissue, or time series experiments, or experiments with tissue from
different organs)?
- Are genes with similar expression related in some way?
- In which biological processes is a gene with a suspicious or interesting expression
pattern involved?

The analysis is therefore a process that can be approached from two sides: proving a
preexisting hypothesis or creating a new hypothesis by finding interesting relations and
patterns.

The visualization of large and multidimensional data is a complex and tedious process.
Different visualization and especially interaction methods can help to discover new know-
ledge. We chose two methods which most closely match our requirements: Heat Maps
and Parallel Coordinates.

This paper describes how we apply those methods and enrich them with novel features
that simplify the process of data exploration. In addition we present a new method to
bring the data of the two visualization methods in context with respect to one another,
and, most interestingly, with the preexisting pathway visualization by utilizing the concept
of Visual Links.

1.2 Structure of this Document

This section is followed by a brief introduction to the biological backgrounds of this work
(section 1.3), as readers from the field of computer science might not be familiar with
them. Chapter 2 discusses the related work on information visualization in general (sec-
tion 2.1), parallel coordinates (section 2.2), heat maps (section 2.3), pathway visualization
(section 2.4) and the concept of visual links (section 2.5). The proposed methods to achieve
the goals are described in chapter 3. Software design and implementation are the topics of
chapter 4. The results are discussed in chapter 5, followed by a conclusion and analysis of
future work in chapter 6.
1.3 Biological Background

Life science researchers estimate that between 10 and 100 million living species exist today on earth. As diverse as they are, they all share a common denominator. All organisms are made of cells [Alberts2002, cp. 1.1]. What distinguishes living organisms from other things is their way of reproducing, their heredity. This creates a link between the parent and the offspring. The cause for this link is the shared information, encoded in genes [Alberts2002, cp. 1.1]. While most living organisms on earth have only one single cell, the human body consists of more than $10^{13}$ individual cells. In every cell the whole hereditary information is encoded. The medium for the storage of this information is deoxyribonucleic acid, DNA. DNA is a long polymer made of four basic monomers: A (Adenine), T (Thymine), C (Cytosine), G (Guanine) [Alberts2002, cp. 1.1]. DNA is encoded by a long chain of pairs of nucleotides (the aforementioned monomers plus a sugar) embedded in a double-helix structure (see figure 1.1). Genes are continuous segments on the DNA which have a specific function and constitute a unit of inheritance.

![DNA and its building blocks](image)

**Figure 1.1: DNA and its building blocks.** (A) shows the basic building blocks of DNA, which are a sugar phosphate and a base. (B) depicts a single DNA strand. In (C) the construction of the double strand is explained. (D) shows the straightened double strand, and (E) the double helix which is the result of the two strands twisting around each other [Alberts2002].

As evident in figure 1.1 the different monomers pair up in two combinations - T with A and G with C. Because of this the double helix actually holds the encoding information twice, once on each strand. DNA is always created by copying a template [Alberts2002, cp. 1.1]. Besides replicating itself when a cell divides, DNA is also the model for other products: **RNAs** (ribonucleic acids) and, as a consequence, **proteins**. The process of the production of RNA is called **transcription** [Alberts2002, cp. 1.1]. During transcription the double helix is split up temporarily (by the RNA polymerase) and the RNA is encoded on
the unbound monomers.

From RNA, in a process called translation proteins are produced. This is achieved using a “large multimolecular machine, the ribosome” [Alberts2002, cp. 1.1], which visits each monomer on the mRNA (messenger RNA) and encodes amino acids sequences complementary to the information contained in the mRNA. There are 20 different types of amino acids. **Enzymes** are special proteins which catalyze chemical reactions in the cell [Alberts2002, cp. 1.1]. Proteins have many other functions as well (eg. structural and mechanical functions). Each protein performs a specific task, according to the gene that encoded it. Each protein corresponds to one or more gene(s) [Alberts2002, cp. 1.1]. Because of that our genes determine all functions and processes in our body.

### 1.3.1 Gene Expression Regulation

An organism’s DNA encodes all of the RNA and protein molecules required to construct its cells. Yet a complete description of the DNA sequence of an organism be it the few million nucleotides of a bacterium or the few billion nucleotides of a human no more enables us to reconstruct the organism than a list of English words enables us to reconstruct a play by Shakespeare. [Alberts2002, cp. 2.7].

How much of a protein is produced within a cell is defined by the **expression regulation** of individual genes. The information on how much of a protein should be produced is encoded in regulatory DNA which is interspersed in the DNA encoding the protein, in so-called non-coding regions. These regions bind to protein molecules that are present in the local cell and through that mechanism regulate how much of a specific protein is actually produced and therefore determine the function of a cell [Alberts2002, cp. 1.1].

The study of gene expression regulation is important as it can give clues to the function of a gene as well as information on which gene is involved in which type of process [Alberts2002, cp. 2.8]. Furthermore it can be a diagnostic tool. One can, for example, find out of which type of cancer a tumor sample is. This is due to the fact that the different cancer types have unique expression patterns [Alberts2002, cp. 2.7]. As a consequence targeted therapy is possible. Other applications include the study of the effects of drugs, personalized medicine and many more.

Several ways to study gene expression regulation exist. One method is to use **reporter genes** - genes that give clues to their expression regulation. This involves replacing the coding portion of a gene with a reporter gene. Other approaches are hybridization techniques. [Alberts2002, cp. 2.8]. These techniques share one property: only the expression regulation of one single gene can be observed at a time.

**DNA microarrays** - developed in the 1990s - revolutionized the analysis of gene expression regulation [Ball2007, p. 372], [Alberts2002, cp. 2.8]. Microarrays are small plates, made of glass, plastic or silicone. Two different techniques were developed: one commercial, by Fodor et al. at a private company (Affymetrix[^1^]), based on photolithography and

[^1^]: [http://www.affymetrix.com/]
one at Stanford University School of Medicine. The former is more efficient, to produce in large quantities. For the latter method small spots of DNA are printed on a slide by a robot [Ball2007, p. 372]. That has the advantage of being easily producible and is therefore common in academic settings. The spots are nucleotide sequences of specific genes.

The probe consist of cDNA (complementary DNA, which is more stable [Ball2007, p374] than mRNA) which is produced from mRNA and labeled with a fluorescent substance. Once this is applied to the chip, the nucleotide fragments hybridize to the matching spot. Typically the probe consists of two tissue samples - the sample studied, for example marked with red, and a reference sample, marked with e.g. green. Thus genes that are overly expressed in the red probe relative to the green probe shine red and vice versa [Alberts2002, cp. 2.8].

After hybridization the array is washed and then optically scanned, resulting in an image similar to figure 1.2 which is then processed [Ball2007, p374]. However, the data is not 100% correct, as wrong hybridization and noise can occur.

1.3.2 Pathways

There are two types of pathways: signaling and metabolic. Signaling pathways describe signal transduction in the cell. Metabolic pathways are graphs that model chemical reactions within cells [Alberts2002, cp. 1.2]. The chemical reactions are catalyzed by enzymes,
which are direct products of the transcription and translation of the DNA. However, while the DNA of human beings and many other organisms were sequenced, the function of those genes is, to a large extend, still unclear [Kanehisa2000]. Pathways haven been traditionally printed as huge graphs on paper, but work with those graphs is tedious. Therefore several groups developed online databases containing pathways, genes and chemical compounds, trying to overcome the shortcomings of the paper-based approach by providing search features and hyperlinking. Many such projects exists, two prominent and widely used are KEGG (Kyoto Encyclopedia of Genes and Genomes) [Kanehisa2000] and BioCarta. Examples of both are shown in figure 1.3.

Figure 1.3: BioCarta and KEGG pathways. (a) shows the BioCarta pathway “Cortico-steroids and cardioprotection”. The cell border, the nucleus, the DNA etc. are easily distinguishable. (b) is the “Cysteine metabolism” in a more formal presentation.

While those pathways model the principal way chemical reactions are processed, they do not take into account deviated expression levels. If one compound is not produced due to a severe underexpression, a whole process might break down. Traditionally it was the role of physicians or biologists to manually match expression levels to reactions in pathways. This tedious and error-prone tasks can nowadays be achieved with software, which is explained in detail in section 2.4.
Chapter 2

Related Work

This work builds upon two main pillars: Information Visualization and knowledge from the Biomedical Domain, in particular knowledge of the function of genes and gene expression, which was addressed in section 1.3.

General concepts of Information Visualization will be presented in section 2.1, followed by a detailed section on Parallel Coordinates (section 2.2) which are the core of this work. Heat Maps, the common way to visualize gene-expression regulation are the topic of section 2.3. Computer aided visualization of pathways is discussed in section 2.4. Visual Links, discussed in section 2.5 which are basically connection lines between separate views, are an important aspect to bring the different visualizations into relation.

2.1 Information Visualization

Information Visualization (InfoVis) is the field in computer science that conveys abstract information using images. It therefore distinguishes itself from Scientific Visualization where visualization methods are natural. Examples for scientific visualization are to show the heat spreading on an exhaust pipe, or to render a car in a computer aided design program. On the contrary, InfoVis deals with topics such as the visualization of file sizes on a hard disk or social networks.

Information visualization has been widely used for a long time. One of the most prominent early examples is the Tableaux Grahiques et Cartes Figuratives de M. Minard drawn in 1869, which shows the devastating losses of human life during Napoleons Russian campaign (see figure 2.1).

Information visualization makes use of the human ability to perceive pictures which is far superior to the ability of analyzing long textual lists [Shneiderman1996].

According to Shneiderman information visualization can be categorized into seven “type by task taxonomies (TTT)” [Shneiderman1996]:

- 1-dimensional: Linear types like lists etc.
- 2-dimensional: Maps.
- 3-dimensional: Real world objects such as mechanical parts, chemical structures, the human body etc.
- Temporal: Time lines for medical records, video cutting etc.
- Multi-Dimensional: Statistical data.
Figure 2.1: Visualization of the losses in Napoleon’s Russian campaign 1812. Drawn by Charles Minard in 1869 [Tufte1983, page 41].

- **Tree**: Structures such as file systems, ancestry etc.
- **Network**: Graphs, an extension of the tree structure, where each node can be linked to any other, without a strict hierarchy. Examples are the internet, metabolic pathways, etc.

Within the Caleydo framework the network and the multi-dimensional TTTs are relevant. Pathway graphs are in the network taxonomy, while gene expression regulations are clearly within the domain of the multi-dimensional taxonomy.

This work focuses on the visualization of gene expression regulation and therefore will mainly consider examples and methods of the multi-dimensional domain.

The Information-Seeking Mantra by Shneiderman [Shneiderman1996] is a guideline for designing information visualization systems. In short it states:

Overview first, zoom and filter, then details-on-demand

The extended version adds *relate, history* and *extract*. While being criticized for not being formally evaluated and especially for not being suitable for evaluation of InfoVis systems by [Craft2005], it is nevertheless important in understanding the basic principles behind an information visualization system. This opinion is also shared by [Craft2005], and as the vast amount of citations of Shneiderman’s original paper shows, by many researchers within the InfoVis community.

### 2.1.1 Overview First, Focus+Context

The natural way to visualize a large set of data is to show the whole data set first. This helps to understand the complexity and the size of the data. It may also allow to see structures, which are not recognizable from a closer viewpoint. These are the
reasons why Shneiderman included **Overview First** in the visual information-seeking mantra.

It also is the natural way to look at something. Imagine that a GIS (geographic information system) like Google Maps would start at a random point, zoomed-in. However, if no structure, like in a map of the world, is inherent in the data, one will have to question whether it is good to start with an overview first. If an analysis is focused on a particular subgroup it might not make sense. Imagine a system visualizing relations of web-sites. It would not make much sense to show relations of all web-sites on the internet first and then zoom to the website of interest. One can argue that the same is true for pathway visualization as discussed in [Streit2007](#) [Streit2008](#). Initially showing all pathways does not make sense, instead it is useful to provide some kind of initial input information to select a certain subset. In the case of websites this could be the output of a search query, in the latter case it could be the name of a pathway or a gene.

**Focus+Context** describes a concept that puts one subset of the data in focus while still showing an overview of the other data, therefore a user can always see the context of an entity in focus. One example for an application of focus+context are **Fisheye Views**, originally developed by Furnas et al. [Furnas1986](#). They magnify the object in focus while showing the rest distorted and smaller. An example of a fisheye view applied to the Paris metro map can be seen in figure 2.2.

![Fisheye View of Paris Metro Map](Sarkar1994)

Figure 2.2: A graph representing the Paris metro system with a fisheye applied [Sarkar1994](#).

Other early focus+context implementations include **The Document Lens** [Robertson1993](#), **The Perspective Wall** [Mackinlay1991](#) and **The Hyperbolic Tree** [Lamping1995](#), all developed at Xerox PARC.

The aforementioned methods are so-called **distortion-oriented** methods, as categorized by Kosara et al. [Kosara2003](#). Other techniques are **overview methods**, where one window provides the focus and a separate window the context, **Filtering** where additional
information is shown on a particular subpart, for example with a magic lens \cite{Bier1993} and In-Place F+C, where information is pointed out to the user, for example by highlighting. Filtering in this context should not to be confused with filtering in the upcoming section.

2.1.2 Zoom and Filter

**Zooming** is a modification of the viewpoint. One has to distinguish between **zooming-in** and **zooming-out** \cite{Craft2005}. The former increases the size of an object in a view, by reducing the distance between a viewpoint and an element. Thereby other information, that is not within the currently viewed frame, is clipped. To navigate in such a clipped view it is necessary to **pan**, which means moving the view within the plane perpendicular to the object-viewpoint axis.

The latter, zooming-out, reduces the size of an object in a view by increasing the distance between a viewpoint and an object. Previously clipped information becomes visible. While these two operations are formally inverse, they are very different as far as cognition is concerned \cite{Craft2005}. Zooming can also be seen as “filtering through navigation” \cite{Craft2005}. A formal explanation on zooming and panning is given in \cite{Furnas1995}.

**Filtering** is the process of removing unwanted or uninteresting information from a current view. From an implementation point of view filters are similar to **brushes** (see section 2.1.4), however they remove information instead of highlighting them. Filtering is usually achieved by employing user interface widgets. For interactivity it is important that filters are dynamic (see figure 2.3), which is also called **dynamic querying** \cite{Ahlberg1995}. That means that changes are immediately reflected in the visualization. Composite filters (negation, and, or etc.) are applied similar to composite brushes and are of great value for a fine-grained selection.

2.1.3 Details on Demand

In a scatterplot with 1000 data points it is infeasible to show detailed information on all entries, as the screen-space is limited. But even with unlimited space it would not make sense to show all available information at once. In large data spaces, elements are selected based on specific criteria. Only if an object matches those criteria, and therefore is likely of interest to the user, detailed information should be displayed. This avoids visual clutter \cite{Ahlberg1995}.

Common implementations of the details on demand approach are pop-up windows as shown in figure 2.3 \cite{Ahlberg1995}, linked browser views \cite{Streit2008} and tool-tip information \cite{Craft2005}.

2.1.4 Relate: Multiple Views, Linking & Brushing

**Multiple View Interfaces** are a significant development in information visualization. With the increase in available screen real estate multiple view interfaces became possible.
Multiple view systems are defined by using two or more views of either different data or different aspects of the same data for one single visualization task. They have been widely used in all kinds of systems, such as CAD (computer aided design), GIS and InfoVis systems. They are often used for focus+context, for example to show the current focus in a large window, while providing the context in a smaller, orientation window. Plumlee et al. [Plumlee2006] have evaluated multiple views interfaces and proven their value for systems that show one window for navigation (overview) and one for focus. They showed that subjects make much more use of a multi-window interface than of the ability to dynamically zoom-in and out.

When using different visualizations, the multi-view approach allows using the strengths of each one, while compensating for shortcomings of single methods. An example is shown in figure 2.4. There scatterplots, where brushing is easy, are accompanied by a treemap, a visualization form that makes it trivial to understand the relative “size” of an element and a geographical map, which is important for this type of data.

A guide for the development of multiple view interfaces was contributed by Baldonado et al. [Baldonado2000]. They formalized a set of rules for when to use multiple-view interfaces.

**Linking & Brushing** is a concept for connecting two or more views. Brushing is the process of interactively selecting a set of elements in one view. As a result they are highlighted, for example by changing their color. There are many different types of brushes,
such as a simple selection on click, rubber-band brushes and others. Often those brushes depend on the type of visualizations. Groundbreaking work in the definition of brushes, such as composite brushes, data driven brushing, simultaneous display of multiple brushes etc. were carried out by Martin et al. [Martin1995].

Through linking multiple views are connected within the data space. The elements brushed in one view are also highlighted in the same view, where the effects of the brushing operation can be studied [Hauser2004]. However cognitive load in such representations is high because additionally to the cognitive effort of comparison, the user also has to switch contexts [Baldonado2000] and rediscover the brushed entity or entities in the other view.

Most recently novel techniques have been developed trying to reduce the associated cognitive load by actually drawing connection lines between the brushed entities. As this is one of the core subjects of this work, it is explained more detailed in section 2.5.

### 2.1.5 History

Shneiderman [Shneiderman1996] recommends to keep a **history** of user actions to “support undo, replay, and progressive refinement”. While this is nowadays a standard feature for all kinds of GU-based software it is especially valuable for InfoVis systems. Undo in other software is usually employed to correct mistakes, however in InfoVis it is of special relevance.
since progressive refinement of analysis is a common task \cite{Craft2005}.

### 2.1.6 Extract

According to Shneiderman \cite{Shneiderman1996}, **extract** means to preserve discovered information. One can extract query parameters or results (sub-sets). Extracting is usually equivalent to saving to a file. This allows exchanging data with other persons or to import data into other programs, for example for statistical analysis. By saving query parameters it is also possible to save a program’s state and therefore allow continuous exploration, even if a computer is switched off. Again this property is very common in all kinds of software, but not often found in InfoVis applications.

### 2.2 Parallel Coordinates

**Parallel Coordinates** are a visualization method devised for showing multidimensional data on a standard 2D display. They were originally developed by Alfred Inselberg in the early 1980s \cite{Inselberg1985} to show properties of multi-dimensional spaces. A good state-of-the-art report is given in \cite{Siirtola2006}.

Each of the \( n \) dimensions is mapped onto one of \( n \) parallel axis. A single \( n \)-dimensional data point is represented as a polyline that intersects each axis at the value the data point has in the corresponding dimension. \[\text{Figure 2.5}\] shows a five-dimensional point \( C = (c_1, c_2, c_3, c_4, c_5) \) in a parallel coordinate plot with five axis.

![Parallel Coordinate Plot](http://www.math.tau.ac.il/~aiisreal/)

Parallel coordinates can be used to illustrate multidimensional Euclidean geometric objects, such as five-dimensional spheres. However, in information visualization their property to show relations of abstract, even non-numerical \( n \)-dimensional data is of primary interest \cite{Siirtola2006}.
To understand relations of different data points in an n-dimensional space one can draw many polylines in one plot. While this is an important method in InfoVis, there are some conceptual problems. In the upcoming sections we will discuss these and review solutions proposed in the literature.

In figure 2.5 all axes are of the same scale, ranging from negative to positive and share a common origin. While this simplifies some issues with axis annotation and might seem intuitive at first, this approach is not suitable for the visualization of abstract data. With parallel coordinates, one can show relations of attributes which are in no obvious mathematical relation to each other, as it is the case when mapping nominal data to numerical values [Rosario2004].

A good example for this is the car dataset collected by Ramos et al. [Ramos1983], which is often used for testing parallel coordinate systems. The dataset contains 406 different cars with nine variables each. Among those nine variables are ratio scale attributes, such as miles per gallon (MPG), discrete values, such as number of cylinders (CYL), as well as nominal data, such as country of origin. By assigning the nominal data such as country of origin a numerical value the dataset can be viewed in a parallel coordinate browser. This dataset is shown in figure 2.6.

Figure 2.6: A parallel coordinate browser showing the car dataset. Notice that ratio values (e.g. MPG), discrete values (e.g. CYL) and nominal values (e.g. ORIGIN) can be shown next to each other [Siirtola2006].

2.2.1 Crossing Problem

Figure 2.7 illustrates a possible ambiguity, the crossing problem. It is impossible to determine which line segment right of the second axis belongs to which polyline. The crossing problem occurs when two or more polylines coincide in one point on an axis. This is more significant when visualizing discrete or nominal data, as coinciding points are much more likely to occur.

There are several solutions for this problem. The easiest one is to highlight one of the lines that have coinciding points by brushing. While this is effective in most situations, user interaction is required. Another solution is to reorder the axes. However, this is only
effective in special cases with only very few polylines. Furthermore the ordering of the axis is usually not arbitrary, so reordering might not be an option.

Another approach was suggested by Fua et al. [Fua1999]. The idea is to use proximity-based coloring, a method that colors polylines based on a similarity measure, thus making it easy to distinguish between two lines that may coincide in certain regions but are not similar in the overall representation.

Graham et al. [Graham2003] address this problem by drawing curves instead of straight. Figure 2.8 illustrates their approach. They argue that they can avoid the crossing problem and allow the user to follow the lines without the need for brushing.

Graham et al. also suggest to use spreading points for nominal values. They noticed that on axes with nominal values the lines tended to bundle in one point. The situation was improved with the use of curves, however there were still many lines that met in just several points. The properties of nominal values cause the axes to contain many empty spaces. Therefore Graham et al. proposed to use these empty spaces to spread the intersection points between axes and polylines. To visualize that the spread lines belong to one single data point they drew rectangles around the spreading area.
2.2.2 Viewing Multiple Relations

Reordering, duplicating and removing axes are important operations in parallel coordinates. The reason for this is that significant relations between axes not next to each other may remain hidden \cite{Siirtola2006}. This is due to the fact that one axis can only be in relation to a maximum of two other axes, as illustrated in figure 2.9. The axes arrangement in the upper plot does not reveal a relation between miles per gallon and horse power. However, after reordering the axes, so that those two axes are adjacent, the relation, in fact a inverse relation, is obvious.

Figure 2.9: The effects of reordering axes in parallel coordinates. In the upper plot the inverse relation between miles per gallon (MPG) and horse-power is hidden. By reordering the axes in the lower plot the relations are obvious \cite{Siirtola2006}.

Reordering axes is an essential feature of a parallel coordinate browser, but what if a user is interested in relations between one dimension and more than two others? A good example would be the relation of miles per gallon to horsepower, displacement (DISPL) and the number of cylinders. One way to achieve this is to allow duplication of axes. We then can compare as many dimensions as we want, the only limitation is the available screen real estate. However, having the same axis multiple times in a representation is not intuitive, it increases the complexity of the visualization. Furthermore the arrangement and duplication of axes can be tedious, especially when the number of dimensions is large (bigger than 20).

The cause for not being able to compare one axis with more than two others is the nature of 2D. This was recognized by Johannson et al. \cite{Johansson2005}, who developed a 3D extension for parallel coordinates. They define one axis as the axis in focus and arrange a certain number of other axis circularly around the central axis. Figure 2.10
illustrates their approach.

Figure 2.10: *3D multi-relational parallel coordinates showing relations between miles per gallon and five other dimensions of the car dataset. The axis in the center represents miles per gallon. Inverse relations between miles per gallon and horsepower, weight, and number of cylinders are visible [Johansson2005].*

To increase the visibility of trends Johansson et al. also applied clustering, which is the topic of [section 2.2.4.3](#). Obviously the number of comparable dimensions is limited, due to the occurring occlusion and cluttering when too many dimensions are compared. Johansson et al. [Johansson2005](#) stated the limit to be at around 20, but also mentioned that this number highly depends on the structure of the data and the number of clusters desired. They also proposed to view a limited number of relations in succession. This is achieved by showing only half the axes at a time, fading the others out. Furthermore they developed a method that calculates the amount of space (angle) for one dimension based on their relation with the dimension in focus.

A totally different approach was taken by Theisel [Theisel2000](#). He introduced *higher order parallel coordinates*. The idea is based on the realization that the space between the axes is not used efficiently. He argues that a certain area between two axes is not required to understand the relations between the different dimensions. So he proposed to use this space to encode information of the relation of one dimension A, adjacent to B with a dimension D that is not adjacent to A. This is achieved by using free-form curves that are influenced by the additional dimension D between A and B. [Figure 2.11](#) illustrates this approach for the car dataset. In between two main axes an additional axis is added, that has an influence on the free-form lines connecting the main axis. Theisel argues that this method allows a comparison of more than two relations at once. This technique is not limited to one additional dimension in between, [Theisel2000](#) contains figures showing up to six additional axes between two main axes.
2.2.3 Viewing many Dimensions

Parallel coordinates, without any further extensions, can deal well with up to $10^3$ polylines, but the number of easily manageable dimensions is a lot smaller. On conventional screens it is safe to say that a number of less than 20 dimensions is the limit. Other multi-dimensional visualization methods, such as scatterplots and star glyphs have similar or even stronger constraints.

For obvious reasons this is insufficient. Imagine a patient database, with clinical data on 2000 cancer patients. For every patient a large number of entries exist, not even mentioning data from tests like microarray analysis. Other examples would be digital libraries, simulations or surveys [Yang2003a]. An example of a dataset with 215 dimensions and 298 data points is shown in figure 2.12.

Figure 2.11: Higher order parallel coordinates showing the car dataset. The polylines are free-form curves influenced by a weight which is based on a dimension that is not adjacent to the reference dimension. The additional dimensions are shown as gray axes [Theisel2000].

Figure 2.12: Parallel coordinates showing a dataset with 215 dimensions [Yang2003a].
To increase the number of viewable dimensions, dimension ordering \cite{Yang2003a, Peng2004}, based on a similarity measure, and automatic dimension spacing \cite{Yang2003a}, were proposed. One approach for **automatic dimension ordering** places dimensions with a high similarity next to each other \cite{Yang2003a}. **Automatic dimension spacing** takes advantage of the fact that highly similar dimensions do not need a lot of space between the axes, as most lines are just straight. In contrast to that a lot of space is needed between very different dimensions. Obviously such an approach works best when combined with dimension ordering.

However, those approaches do not solve the problem of a large number of dimension. Yang et al. \cite{Yang2003a} propose three ways to handle such cases:

- **Zooming and panning**: This means that all axes are shown at the same time, the user can zoom in/out and pan. The obvious drawback of this approach is the loss of context and orientation when the user zoomed into one region.

- **Manual distortion**: Again, all data is shown initially. The user can select a certain region, the spacing between dimensions in this region is increased. The spacing can be reduced again, when the area is no longer of interest.

- **Structure based spacing distortion**: This method is based on a hierarchy that was constructed in a pre-processing step. Yang et al. \cite{Yang2002, Yang2003, Yang2003a} use clustering to construct such a hierarchy.

The implementation of the structure based spacing distortion uses a **SunBurst** \cite{Andrews1998, Statsko2000} visualization to navigate and manipulate the hierarchy \cite{Yang2002}, illustrated in figure 2.13 (a). It implements a distortion technique that blows up a selected segment to take up large amounts of the available angle, as shown in (b). The resulting selection, which is a subset of the dataset shown in figure 2.12, is shown in (c).

It is important to note that the automatically constructed hierarchy can be modified manually, thus giving the user full control of the arrangements.

Another approach described in \cite{Yang2003a} is **dimension filtering**. It introduces an importance threshold and a similarity threshold. Dimensions are only inducted into the final visualization if they are not similar to another one (determined by the similarity threshold) or found to be relevant enough (determined by the importance threshold). Again they allow user modifications, as this automatic approach could remove relevant dimensions.

**Automatic dimension reduction** is the final method for handling many dimensions we will discuss. Yang et al. \cite{Yang2003} based their approach again on the hierarchy introduced before. They implemented a process called **visual hierarchical dimension reduction**. In this process the user is shown all dimensions within one dimension cluster. The user has to select a representative dimension from either one of the original dimensions or an automatically generated one, which is an average of all others. Then only this dimension represents the whole cluster. This is repeated until all clusters are represented.

The same techniques are also applicable for other visualization methods, such as **Star Glyphs** or **Scatterplots**.
2.2.4 Cluttering

Cluttering is a problem which results from many objects overlaying each other. While all visualization methods presenting each data point as a separate entity are affected as soon as the dataset reaches a certain size \cite{Fua1999}, parallel coordinates tend to clutter more easily. A special case of cluttering is the crossing problem discussed in section 2.2.1.

Basically there are three ways to address large amounts of data \cite{Novotny2006}:

- **Selection**: Visualizing selected subsets.
- **Aggregation**: Merge representations of data to new subsets.
- **Segmentation**: Show a subdivision of a dataset.

The problem is illustrated in figure 2.14 (a). The figure shows a parallel coordinate plot with 2000 data points and three dimensions. Notice that practically no trends and no
details are recognizable. In the following we will discuss a selection of methods that take care of this issue and allow the visualization of datasets with $10^9$ and more data points per dimension.

Figure 2.14: A cluttered parallel coordinate plot with 2000 data points in three dimensions. (a) showing the plot without the use of transparency, (b) with the use of transparency and (c) shows the effect of 4 black areas with 80% transparency stacked on top of each other.

2.2.4.1 Using Transparency

The first method developed to avoid cluttering in parallel coordinates was to use transparency \cite{Wegman1996}. By using transparent polylines one utilizes the effect that the alpha channel of all overlaying elements is summarized, as illustrated in figure 2.14 (c). It shows four boxes stacked on top of each other. Each box is black, but with only 20% of the maximum alpha channel. By rendering the boxes on top of each other the overall transparency is reduced, resulting in the impression that the top-most box is dark-gray.

The results of applying transparency to parallel coordinates are striking, as shown in figure 2.14 (b). It is easy to see the overall trends in the dataset. Up to a certain number of elements, it is even possible to see outliers, which are lost when the alpha value for an individual polyline becomes too small.

Wegman et al. also used this effect in combination with hue and brushing. By, for example, brushing certain elements in red and others in green one can identify overlapping regions easily. The red and green color add up to yellow in areas where both coincide.

One advantage of using transparency over other techniques is that it does not require any preprocessing. The structure of the data practically reveals itself \cite{Wegman1996}. However, when handling very large datasets, this is a disadvantage, as all elements in the dataset are actually rendered, thus reducing the interactivity of the visualization, even on modern computer systems. Another drawback is, that this approach only works for
Actually overlapping or crossing elements and might fail to correctly emphasize parallel elements.

2.2.4.2 Random Sampling

Dix and Ellis [Dix2002, Ellis2006] showed that random sampling can be of high value to visualization. In fact random algorithms have been used with great success in other areas. The basic idea is that a randomly taken subset statistically represents the whole dataset. We know from opinion polls that a survey with about 1000 participants has a very high confidence for predictions representing the population of a whole country. The same is true for visualization. A parallel coordinate system using transparency can easily show several thousands of datasets, which in turn are representative (with a certain confidence) for truly large datasets. The benefits are obvious: random sampling is easy to integrate into existing systems, the handling of a large dataset is not different from the handling of a medium dataset for the user (in contrast to for example aggregation techniques) and performs well.

However, Ellis and Dix also explained that for some use cases, random sampling is not well suited. Generally it works better for exploratory tasks, than for goal driven, specific tasks. This is due to the fact that some outliers and some high-frequency alternations can be lost in the process of random sampling.

Ellis and Dix [Ellis2006] also introduce a method to bring the power of random sampling to a not-omitting visualization with parallel coordinates, the Sampling Lens, a Magic Lens. This technique allows to distinguish trends in very dense data by providing the user with a lens that she can move over the dataset. The data viewed through the lens is sampled, thus reducing the clutter (see figure 2.15).

![Figure 2.15: The sampling lens showing a 10% sample. On the left it can be seen that the lines below the lens are mainly crossing each other. In the center they tend to end in one point and on the right the lines are parallel [Ellis2006].](image)

2.2.4.3 Clustering

Clustering is the process of grouping similar items. This is achieved by applying a similarity measure to all data elements. While there are two basic types of clustering, namely partitional (splitting the input into n clusters on one level) and hierarchical, we are concerned with hierarchical clustering. Hierarchical clustering finds a pair of similar elements and puts them into a cluster. The resulting cluster is compared to other elements or clusters and then
Data visualized with parallel coordinates usually lend themselves to clustering, due to their high dimensionality. Therefore a lot of research was conducted in this area, for example [Fua1999, Johansson2004, Johansson2005b, Andrienko2004, Novotny2006, Kosara2004a].

The foundations were laid by Fua et al. [Fua1999]. Their method uses hierarchical clustering. For each cluster to be visualized a central line is drawn. Then two polylines with no alpha are drawn at the boundaries of the cluster. This is followed by an interpolation of the alpha values from the boundaries to the central line. The result for one cluster is shown in figure 2.16 (a). By combining the central line with the interpolation a user can immediately figure out the main direction, but also the spreading of the cluster. Figure 2.16 (b) shows the application of this method for a dataset containing 230,000 data points.

In some cases the visualization of the spreading of the clusters might lead to undesired occlusions. Therefore Fua et al. created the method of extend scaling. This allows the user to reduce the spreading around the central line, resulting in a much “cleaner” view (compare figure 2.16 (b) to (c)), at the cost of less intuitive understanding of the cluster.

Another important property is to define a level of aggregation [Fua1999]. Depending on the user’s requirements it can make sense to show only two or three main (naturally with a broader band) or many clusters. They satisfy this by introducing multiresolutional cluster displays. It is based on a parameter S, which is a measure for the level of detail. A cluster is selected that has less than or exactly the number of S child clusters. Thus the system can smoothly scale among very few to many clusters.

Johansson et al. [Johansson2004] extended the clustering system as proposed by Fua et al. [Fua1999] by implementing a drill-down feature for the clustered parallel coordinates. They achieved this by visualizing the actual content of a selected cluster in a linked view. Thus they allow exploring the overview in the clustered visualization and the individual
properties of a selected cluster in a separate view.

Another problem with clustered views is that outliers typically get lost. This is true for local (i.e. local to a cluster) and also for global outliers. In [Johansson2005b], Johansson et al. proposed a method to prevent the loss of local outliers and to better visualize the structure of the cluster. To improve the method of interpolating between the boundaries and the center of the cluster they introduced high-precision textures, which are mapped onto the cluster. This allows seeing the structure of the cluster, which conveys more information than interpolating between boundary and center. In addition they computed a texture that emphasizes outliers which can be shown when an analysis is focused on outlier detection. Furthermore they introduced transfer functions for the intensity values of the textures. The problem they tried to address here is that typically on the outer rims of the clusters intensity values can be too low to recognize the data. This might be desirable in some situations, but a problem in others. A square transfer function for example emphasizes regions of high density, while a logarithmic shows the whole cluster where even outliers are visible.

Another approach to handle outliers was taken by Novotny and Hauser [Novotny2006]. They separate the outliers before the clustering, thus avoiding non-representative stretching of the clusters. Figure 2.17 illustrates the results of their approach.

![Figure 2.17: Outlier preserving clustered parallel coordinates.](image)

**Figure 2.17:** Outlier preserving clustered parallel coordinates. The left image shows the dataset with no precautions, the center shows a naive clustering approach. On the right one can clearly see the main clusters and the preserved outliers [Novotny2006].

### 2.2.4.4 Density based highlighting

An approach not based on clustering is density based highlighting [Miller1991, Artero2004]. Artero et al. determine the intensity of a polyline by calculating bi-dimensional frequency histograms for each pair of adjacent axes. The number of histograms is determined by the desired or existing resolution. Polylines are drawn based on the histogram, for every non-empty histogram a polyline is drawn (at least without thresholding applied). The intensity of the drawn line is determined by the value of the histogram.

The advantage of their system lies in the simplicity compared to clustering. It is also a suitable method for data that can not be clustered. The underlying structure of the visualization allows for thresholding based on a gray value, similar to classic image processing. A core property is that it emphasizes local structures (local densities) in contrast to clustering where an overall comparison of polylines is of relevance.
2.2.4.5 Polyline Averaging

Polyline averaging [Siirtola2000] is the process of aggregating polylines that share a common attribute, for example a value on a nominal axis. All the lines sharing this attribute are aggregated in one single polyline. To improve the usefulness, the averaged polylines can be augmented with boxes showing the standard deviation. Polyline averaging fails to convey meaningful results when nominal values are averaged, since no meaningful ordering can be defined.

2.2.5 Brushing and Interacting in Parallel Coordinates

Brushing in general was described in section 2.1.4. In this section we will discuss interaction and brushing techniques specifically for parallel coordinates.

When visualizing high dimensional data sets, real-time interaction is a very valuable method [Siirtola2000]. It can provide insights in dependencies that otherwise can be tedious to understand. Furthermore real-time interaction reduces the time a user needs to achieve a certain task.

Dynamic masking [Fua1999] is the procedure of reducing the intensity or even hiding those polylines that are currently not in a selection. This allows emphasizing brushed entities while still showing trends in the background. Examples are shown in figure 2.18 and figure 2.19.

Visualizing Brushes with color bands were proposed by Martin and Ward [Martin1995]. This helps to show the effects of a brush even when no data is in the brushed region.

Figure 2.18: A range brush on the car dataset. The range is set in a way that shows all cars with four cylinders [Hauser2002].

According to Siirtola [Siirtola2000] selecting and modifying a range from an axis (a 1D brush) is the most important property of a parallel coordinate browser. This operation can be seen as equal to a SQL query in a database. An example would be to show all cars with exactly four cylinders in the car dataset (illustrated in figure 2.18). In such a case
real-time parallel coordinates allow the user to see the change in the dataset when the range of the brush is altered immediately.

2D brushes, such as drawing a rectangle or a rubber-band and thereby selecting all polylines within this rectangle are especially suitable for outlier selection [Siirtola2006]. Just clicking the polylines has a similar effect.

Another 2D brush that is very useful in some situations is the angular brush as introduced by Hauser et al. [Hauser2002]. By defining a certain angle between two axes it is possible to select only those polylines that’s slope is within the defined angle. This allows to filter for data that has a similar behavior as far as inclination between two axes is concerned. An example is shown in Figure 2.19 which reveals polylines that do not follow the general trend of going up between axes two and three.

Figure 2.19: Angular brushing. While most polylines go up between the second and third axis by angular brushing one can detect those that do not follow this trend easily [Hauser2002].

Smooth brushes are brushes that allow non-binary selection [Hauser2002]. They allow to visualize differences between areas of special interest and contextually relevant selections. Also they make selections that are not strictly above or below a certain value possible.

An important aspect concerning brushes is that the usefulness and usability of parallel coordinate systems increase dramatically when the brushes are composable and modifiable [Siirtola2006]. This allows the user to compose complex queries and correct mistakes or refine selections.

For more complex parallel coordinates visualizations, especially those that aggregate values, special brushes have to be designed. One such powerful yet complex method is structure based brushing [Fua1999]. It is based on selecting subspaces in the hierarchy of the hierarchical cluster. The brushing has to be done in a separate window showing the hierarchy.
2.2.6 Multiple View Integration

The integration of parallel coordinates into multi-view systems (see also section 2.1.4) is important because of their excellent properties to support linking & brushing.

Hauser et al. [Hauser2002] implemented a multi-view environment to demonstrate the effect of high level brushes in scatterplots and parallel coordinates. Bertini et al. [Bertini2005] integrated parallel coordinates and radviz, a method conceptually similar to scatterplots. Radviz arranges several dimensions in a circle which are connected with springs that draw them to the dimension origin. The system allows to choose whether the brushed entities are highlighted in a visualization of the whole dataset or only those selected are drawn.

2.2.7 Extensions

Several extensions to parallel coordinates have been proposed. Typical examples are those who present statistical data, such as box plots on the axes [Siirtola2000] or overlaid histograms [Hauser2002].

Another example are parallel sets [Bendix2006, Kosara2006] a concept very similar to parallel coordinates. Parallel sets are well suited for the visualization of categorical data. The basic concept is to draw the categories on an axis as boxes and parallelograms among the different categories. The width of the parallelograms symbolizes the number of relations between the categories. An example is shown in Figure 2.20.

An integration of parallel coordinates and star glyphs was implemented by Fanea et al. [Fanea2005]. The idea is to unfold the polylines into 3D. Thereby each polyline is rotated by a specific angle and the axis up to the intersection with the line is drawn as well. Consequently they also connect side by side arranged star glyphs with polylines in 3D.

Figure 2.20: Parallel Sets. The boxes along the axes represent different categories. The parallelograms connecting the boxes are equivalent to the polylines in parallel coordinates. [Bendix2006].
2.2.8 Application to Gene Expression visualization

The properties of gene expression data make them suitable for an analysis with parallel coordinates. Therefore parallel coordinates were used to visualize gene expression regulation, for example by Zhang2003, Craig2003, Ruebel2006. The commercial tool GeneSpring\(^1\) also uses parallel coordinates to visualize statistical data. Zhang et al. [Zhang2003] use parallel coordinates only in a static manner, without even drawing the axes. The parallel coordinates view of GeneSpring can only be influenced by tuning statistical parameters.

![Gene expression visualization](image)

Figure 2.21: Gene-expression regulation analysis in 2D and 3D. The left image shows a classic parallel coordinate plot with two brushes applied. The brushed lines are colored red. The right image shows the 3D representation of the same data. The third dimension corresponds to the spatial arrangement on the embryo in the upper right corner [Ruebel2006].

The system devised by Ruebel et al. [Ruebel2006] is a fully featured parallel coordinate browser. They also developed a 3D extension to parallel coordinates, shown in [figure 2.21]. Ruebel et al. use the third dimension to visualize the location of the expression values on an embryo. Location of the gene-expression in an embryo is important to determine the shape of the developing animal. The red lines in [figure 2.21] represent the lines brushed in the left image. This view allows a user to see, that the genes \textit{eve} and \textit{ftz} are expressed with less than 20\% reside only on the tips of the embryo.

Craig et al. [Craig2003] present a system to analyze microarray time-series data by using a combination of scatterplots (their main visualization) and parallel coordinates (which they call graph-view). They use parallel coordinates only for the specification of a timeframe to be explored with the scatterplot view.

\(^1\)http://genespring.com - visited 17.04.2008
2.2.9 Discussion

The techniques to avoid the Crossing Problem (curved polylines and spreading on nominal axes) proposed by Grahame et al. [Graham2003] and explained in section 2.2.1 make it much easier to follow the course of a limited number of lines. This comes at the cost of computing performance and added complexity. Because of the added complexity, the system has higher cognitive load for learning it. When this is mastered, the proposed ideas might improve usability in certain use cases. However, it is doubtful that the proposed techniques scale well with a higher number of polylines. Once clutter (see section 2.2.4) becomes a serious issue, those techniques could actually worsen the problem.

The problem of viewing multiple relations at the same time, as discussed in section 2.2.2 has not been solved satisfactorily. The methods of showing one dimension in the center of a 3D ring of axes as proposed by Johansson et al. [Johansson2005] certainly works well for a limited number of dimensions and a limited number of clusters. However, once the complexity of the dataset increases this method will fail. Furthermore the 3D arrangement complicates interaction. Brushing or other operations might not be possible, due to the occlusion of parts of the visualization. The approach taken by Theisel [Theisel2000], using free-form lines that are weighted according to an additional dimension between two main dimensions, is interesting. However, the interpretations is complex and cognitive load is high. He tries to encode more information between two axis, thereby reducing the visibility of the relation between the originally adjacent relations. When using more than one such weight the complexity is further increased.

Viewing many dimensions is a non-trivial task. Automatic dimension ordering and dimension spacing [Yang2003a] are certainly concepts that are useful for increasing the number of visible dimensions. Once the number of dimensions exceeds a certain limit, one needs to employ further techniques, as described in section 2.2.3. Zooming and panning can increase the number of manageable dimensions. The concerns Yang et al. [Yang2003a] express, in particular the loss of context, can be addressed with several methods (see section 2.1.1), for example by providing an overview window. Another possibility would be to generate knowledge about contextual information from other views and pan the parallel coordinates automatically to focus on the relevant part. Zooming based on a hierarchy derived from clustering as proposed in [Yang2003a] is a promising approach, especially for a domain like gene-expression regulation analysis, where the similarity of the dimensions is highly significant.

Dimension reduction methods, as mentioned in section 2.2.3 [Yang2003, Yang2003a] have to be taken with a grain of salt. While they are the only way to visualize large datasets on one screen, one has to keep in mind that because of the extensive simplification and abstraction, detail is lost. However, we believe, that when handled with caution, they can increase the understanding of large datasets.

Cluttering (section 2.2.4) is a serious problem when it comes to the visualization of large datasets. However, all proposed solutions are valuable within their specific domain. Transparency is easy to implement and increases the usefulness of parallel coordinates even for small datasets. Therefore no parallel coordinate visualization should be without it. Ran-
**dom sampling** is a valuable approach which is also easy to integrate into existing systems. One has to be cautious not to use it for cases where the existence of all data elements is relevant. A problem with random sampling is the likely loss of outliers. This issue could be addressed by a preprocessing step similar to [Novotny2006](#) where outliers are identified first. The Sampling Lens might be useful when no transparency or other measure to avoid cluttering is taken. However we doubt its value in combination with more advanced tools. **Clustering** is certainly the way to go when handling datasets with more than $10^4$ polylines. Combined with the outlier detection, drill-down on subsets and high-precision textures we believe that it is the best way to visualize such datasets. An alternative to clustering is a **Density / Frequency** based aggregation approach. Its emphasis for local structures might be superior to clustering in some applications. **Polyline Averaging** is a very fast data aggregation approach. It is probably best suited for statistical analysis. For more thorough analysis it is possibly not sophisticated enough.

**Brushes and interaction**, as discussed in section 2.2.5, are essential features no parallel coordinate system should lack. User-friendliness may be an area where there is still room for improvement. Complex brushes such as **structure based brushes** are difficult to understand and therefore might be applied only by expert users. The **angular brush** is a very useful tool to express similarity (or dissimilarity) independent of an actual value. However, a solution to express similarity that extends beyond the two dimensional approach might be useful. The use of a color band to visualize a brush is certainly good, as long as brushes are limited to operations that can be represented in a continuous area. This method breaks down when applying brushes such as the angular brush, that brushes on other properties than spatial arrangement.

**Multiple view integration** and **Focus + Context**, discussed in section 2.2.6, are essential features for any kind of more complex visualization (see also section 2.1.4) and this is also true for parallel coordinates.

The **Extensions** proposed for parallel coordinates, discussed in section 2.2.7 can actually add a lot of value to a normal parallel coordinate visualization. Especially the statistical augmentations are well suited, since it is likely that users of parallel coordinates have some experience with classic statistical analysis. **Parallel sets** are a simplification that satisfy specific use cases.

**Applications of parallel coordinates to gene-expression regulation analysis** is certainly promising, but surprisingly has not been used extensively. The commercial tool mentioned uses parallel coordinates only as a visualization tool for statistical analysis and thereby loses all the benefits and insights one can gain from actually interacting with parallel coordinates. The approach taken by Ruebel et al. [Ruebel2006](#) is interesting, but they apply it only to a limited use case. Their datasets of genes are small (around five) and they visualize the experiments as polylines, contrary to the more common approach of showing the experiments as axes. Furthermore the number of experiments they visualize is small enough for parallel coordinates not to run into cluttering problems. We believe that the analysis of both genes and experiments as polylines and of the whole genome are interesting challenges.
2.3 Heat Maps

A Heat Map is a visualization method specifically developed by Eisen et al. [Eisen1998] to visualize gene expression regulation derived from microarray analysis. A single data point is colored according to its fluorescence ratio (see section 1.3.1). Data points with a neutral fluorescence ratio are shown in black, with increasing ratios (corresponding to over-expression when compared to the reference) the points are shown increasingly red and with decreasing negative ratios (corresponding to under-expression when compared to the reference) increasingly green. While this color coding is not the best in terms of information visualization, as the three colors can produce rainbow color map artifacts [Borland2007], it is preferred by domain experts for two reasons: First it is equivalent to the effects visible directly on the chip, achieved through the fluorescence and second, it highlights significant over- and under expression well.

The spatial arrangement places all data points of one gene type next to each other in one dimension (row or column) and experiments in the other.

While this presentation by itself is already valuable, its real strength becomes evident, when meaning is assigned to the order of the arrangement. Eisen et al. [Eisen1998] propose to use clustering combined with ordering based on other factors, such as mean intensity, for the arrangement of the genes. This makes use of the observation, that genes, which have similar expression regulation often also have a similar function. Thus clustering automatically groups the genes according to their function. This makes it possible to find similar functions which are not indicated by a similar genetic sequence. To visualize the hierarchical cluster, a dendrogram (a tree-structure) is drawn on top of the color-mapping (see figure 2.22).

The key to clustering is the definition of a similarity measure. Eisen et al. [Eisen1998] found the standard correlation coefficient (i.e. the dot product of two normalized vectors of expression levels) to be a good measure.

The main limitation when using heat maps is the available screen space (the number of genes to be visualized can be in the tens of thousands), which was recognized by Seo and
Shneiderman [Seo2002]. They propose the Hierarchical Clustering Explorer which provides an overview by averaging adjacent leaves. The system follows information visualization guidelines and implements multiple, linked views (e.g. scatterplots and histograms), focus + context and linking & brushing. To emphasize the clusters, it also visualizes borders between clusters directly in the heat map. To modify the level of grouping, one can adjust a similarity grouping, resulting in a finer or broader grouping.

Another method to overcome limited screen space is the usage of bigger screens [Hibbs2005]. On state of the art giga-pixel display walls one can fit much more information. However, even with the best displays it is impossible to fit all elements on the screen.

2.3.1 Discussion

Heat maps are the standard way to visualize gene-expression regulation. They combine several benefits: hot-spots are easily identifiable, they provide a quick overview and they are a natural representation of the microarray. However, there are some drawbacks one has to address when using them in real-life systems: The number of genes that can be displayed is limited by the screen resolution, and focus+context problems emerge when zooming the heat map. Those problems can be solved by applying state-of-the-art information visualization methods, as demonstrated in [Seo2002].

We believe that heat maps have huge potential for visualizing contextual information for other views. The interaction of heat maps and a pathway explorer might be extremely valuable, as a heat map can provide immediate information on the “neighbors” of a gene selected in a pathway. Therefore perhaps similar genes in other pathways, or genes without a known function can be discovered.

2.4 Pathway Visualization

The chemical processes in organisms are an extraordinary complex network of cycles with influences among each other. One gene can be responsible for several independent reactions [Kanehisa2000]. The complexity of metabolic networks is obvious when looking at figure 2.23 which shows the metabolic network as modeled by KEGG.

Metabolic networks can be understood as large graphs. Each of the nodes and edges in figure 2.23 has a meaning, which can be explored by navigating through the network. However, while the digital representation of pathways in browser based systems such as KEGG and BioCarta brings features like searching, linking and navigating, the cognitive load for the user is still high and navigation can be tedious. Especially the problem of focus + context is not solved within those databases.

A concept trying to overcome these problems is by visualizing the whole metabolic network as one big 3D graph [Rojdestvenski2003]. Due to the size and the complexity, navigation in such a graph is a challenging task. Yang et al [Yang2006] tried to simplify this by using a virtual reality environment, a CAVE. Due to the increased available screen space in such an environment, the visualization of a large 3D graph is simpler.
Figure 2.23: The metabolic network as modeled by KEGG. The nodes are collapsed sub-graphs and lead to one of the 362 reference pathways. (available at http://www.genome.jp/kegg/).

However, the layout of those graphs is inferior to the hand-crafted counterparts provided by BioCarta, KEGG and others. They do not contain rich meta-information and the layout is arbitrary from a biomedical point of view. We therefore encounter a dilemma: both approaches suffer severe drawbacks. Streit et al. [Streit2008] recognized this and took a different approach. They used the graphs as provided by the databases and visually linked them with each other, thus providing the well-known, hand-crafted layout while preserving contextual information. We will discuss the underlying concept in more detail in section 2.5.

In section 1.3.2 we discussed that gene expression regulation can have tremendous influence on the metabolism of an organism. Therefore the logical next step is to augment the pathways with gene expression information of concrete experiments.

One approach to visualize gene expression regulation is to draw a color coded heat map element directly on top of the nodes, as proposed by [Wolf2000, Lindroos2002, Mlecnik2005], which is also implemented in the commercial software GeneSpring. However, there are several limitations of this approach. Quite often, for example, one node in a pathway is encoded by several genes, or pathways are simply not suited for an augmentation due to the shape of the nodes, as is the case with BioCarta [Streit2008]. One way to solve these problems is by drawing several overlaid boxes with the gene expression and other meta-information, as shown in figure 2.24 [Streit2008].
Figure 2.24: Augmenting a KEGG pathway. In the background one can see the different nodes in the pathway, that were augmented with gene expression data. The bright blue color means that more than one gene is responsible for the encoding of the enzyme. On mouse-over, a number of boxes containing information on the genes and their expression pops up, as can be seen in the foreground [Streit2008].

2.4.1 Discussion

The importance of pathways in understanding bio-chemical processes in an organism is evident. However, the complexity of interdependencies is high, even when using linked pathways, as provided by KEGG. Approaches that visualize the whole network in one single 3D graph are likely to fail as far as user acceptance is concerned, due to the complexity of navigation. Even the visualization in virtual environments can not countervail this. We therefore believe that a visualization, based on the well-established pathway drawings, is the way to go. However, the handling of those still leaves room for improvement.

The augmentation of pathways with expression data is essential for the understanding of the concrete condition of a patient or an experiment. The limitations of mapping the data directly onto the pathway are obvious. The visualization in pop-up boxes is an improvement but has its disadvantages as well. First, it occludes the rest of the pathway. Second, the method fails when many genes encode one enzyme and third, the number of comparable experiments is not too big.

2.5 Visual Links

The immense benefits of a multiple view environment were discussed in section 2.1.4. Ever since the wide-spread use of brushes in data space [Martin1995] multiple views became
more common. Typically selections in one view were highlighted in the others, filters in one were propagated to the others, etc. It was not until recently that the obvious limitations of this approach have been overcome by drawing connecting lines directly between the affected views.

Figure 2.25: Network visualization by semantic substrates. The nodes within the green respectively the red rectangle represent semantic substrates of one larger network. On the right, the user can choose which edges to show [Shneiderman2006].

Early work in the field includes [Fekete2003], where graph links are overlaid on treemaps, or [Neuman2005] where relations in hierarchical data are visualized by drawing edges. Shneiderman et al. [Shneiderman2006, Aris2007] (see figure 2.25) applied visual links for two visualizations, even when they were of the same type with different data and in 2D. They used the visual links to connect different entities in semantic substrates of one large network. Semantic substrates are groups of items which do not overlap with others. The reason for the use of substrates is the realization that small networks can be visualized more efficiently than large ones. They then allow the user to select which connections are of interest. For example one could connect all elements within a substrate with each other, providing information on the internal relations of the substrate. Most interestingly the system also allows to draw connections to other substrates, therefore visually linking the two substrates. The resulting visualization can be seen in figure 2.25. However, one could argue that the system does not link two separate views but just bridges an artificial subdivision of one large network.

The work of Streit et al. [Streit2007, Streit2008] extended this approach to 2D planes in 3D. They link pathway graphs, which are sub-graphs of the whole metabolic network (see section 1.3.2 and section 2.4). By tilting the 2D graphs into 3D they emphasize the interconnectivity while reducing clutter. The result can be seen in figure 2.26. With more than 600 pathways in the system they continue to rebuild the linked pathway stack (cp. (2) in figure 2.26) and the other representations based on user interaction. To avoid visual clutter they draw only connection lines for one entity at once.

A generalization of these approaches was proposed by Collins and Carpendale [Collins2007]. By augmenting prefuse, a visualization toolkit [Heer2005], to render its views in a 3D
OpenGL environment, they were able to draw connection lines between independent views. Collins and Carpendale draw edges between all related items visible in both views. They argue that they only draw edges between two adjacent views to achieve visual clarity. To see links between non-adjacent views the user has to rearrange the planes.

Besides providing several pre-defined positions for the arrangement of the planes Collins and Carpendale allow the user to manually arrange all planes at will.

Collins and Carpendale bundle the connection lines when there is a 1:n relation, a feature that is supposed to reduce the problem of too many lines crossing each other. An interesting and sophisticated approach for edge bundling in hierarchical systems for non-hierarchical edges has been presented by Holten [Holten2006]. By bundling adjacent edges and color coding the density he can significantly reduce the visual clutter, as illustrated in figure 2.28.

### 2.5.1 Discussion

Visually linking different views is an important concept to improve the usability of multi-window information visualization interfaces. Overcoming the separation of windows is nowadays equal to not using the existing windowing management systems. All discussed systems only link within their own window, some of them actually render the views usually contained in separate windows in a 3D OpenGL environment. This is associated with problems, especially as far as the navigation and the arrangement in a 3D environment.
Figure 2.27: VisLink: interconnections between independent visualizations. The picture shows a tree map, a scatter plot and a map connected with visual links [Collins2007].

is concerned, as our classical input devices are not well-suited for a 6 degrees-of-freedom navigation.

Another problem one encounters is the docking point of an element and the connection line. Figure 2.27 shows a tree map connected to a scatterplot. The corner where the tree-map element is connected, is also the corner of another element, the actually selected element is only distinguishable by the highlighting. Imagine a parallel coordinate visualization - where should the connection line dock? At the beginning of the line? In the center? It is obvious, that, to make this new concept work well for users, additional methods have to be developed.

A serious problem for systems working with connection lines is visual clutter. The connection lines easily obscure the views. For example the text in the box in the tree map in figure 2.27 is not readable. Shneiderman et al. [Shneiderman2006] try to address this problem by only selectively showing connection lines, which is certainly a valid approach. In the example with the tree map a more intelligent placement of the connection lines could also solve the problem. However, when dealing with many connections, intelligent edge bundling, as proposed by Holten [Holten2006], might be the only feasible way.

All discussed 3D systems can not link non-adjacent views, thus hiding connections when
Figure 2.28: Hierarchical edge bundling. The pictures show a software system and its call graph [Holten2006].

an element does not exist in a view between two others. This problem is equivalent to the problem of viewing multiple relations, discussed in the context of parallel coordinates in section 2.2.2, and can possibly be solved in similar ways.
Chapter 3

Concept

Visualizing gene expression data on a large scale, allowing interactive selection, filtering and bringing the information into context with other views, are the main objectives of this work. We chose to implement a parallel coordinate browser, due to its superior features when it comes to the visualization of and interaction with multi-dimensional data. Furthermore we implemented a heat map, due to its well-established position as the state-of-the-art visualization for gene expression data. The integration with the 2.5D pathway explorer which is part of the Caleydo framework [Streit2008] creates new possibilities of understanding gene expression data in the context of pathways (see section 2.4). As discussed in the chapter on related work [chapter 2], no suitable tools for such a task have yet been developed. Commercial gene expression analysis tools, as e.g. GeneSpring, focus on the statistical aspect of the data analysis, providing only limited visualizations. Academic visualization frameworks, like prefuse [Heer2005], are not well suited for the task of pathway analysis, because the interconnections among the pathways can not be shown satisfactorily.

3.1 Parallel Coordinates

When used for gene expression analysis, a parallel coordinates browser has two main use cases: to discover new knowledge from a large set of data and to visualize contextual information when combined with other visualizations, in our case pathways. The former usually deals with larger amounts of data, exceeding the visualization capabilities of parallel coordinates, when no precautions are taken. We therefore implemented two strategies to cope with this issue: using transparency (see section 2.2.4.1) and sampling (see section 2.2.4.2). The analysis of multiple relations is possible by allowing duplicating, moving and removing axes (see section 2.2.2). When many dimensions are viewed simultaneously, the system allows to pan the view. This panning is done automatically when contextual information on a region of interest is available.

In terms of brushing we identified the following types to be relevant for gene expression analysis:

- Individual selection:
  Each polyline can be picked by moving the mouse over or clicking it. This is important for outlier identification and to provide contextual information to the rest of the system

1D selection:
1D selection is the selection of values within a certain range on an axis. This is probably the most important brush, as it allows to filter the data in a straightforward way. However, we found that current parallel coordinates browsers lack usability as far as those brushes are concerned. We therefore believe that an approach, where the range of not desired values is specified, improves usability. To avoid confusion we will refer to this type of brushes as gates from now on. These gates specify regions where no polylines are allowed to pass, much like a dam with variable height does not allow to pass water.

Angular brushing:
There is no other tool that allows brushing for discontinuities as angular brushes. In the case of gene expression analysis the discontinuities are especially significant. One example would be to monitor changes between one experiment and another. The absolute values are not as interesting as the relative changes. With an angular brush the user can select only those polylines with a slope within a certain range. However, current implementations, as e.g. shown in [Hauser2002], suffer from complex handling. Hauser et al. allow the angular brush to be defined freely between two axes. To create a brush one has to click an icon, select the origin of an angle, select the relative position of the lower leg and the relative position of the upper leg. This is potentially confusing for the user. We therefore propose to define angular brushes relative to a polyline. With this approach, the user selects the brush and simply clicks a polyline (line 1 in figure 3.1). A default angle is drawn around the selected line immediately. The shown angle can now be modified by dragging the legs in the directions illustrated by the arrows in figure 3.1. Notice that line 2 in figure 3.1 is deselected, while line 3 is within the defined angle.

Another important property for a parallel coordinate browser is the ability to exchange axes and polylines. With parallel coordinates the choice of what is represented as polylines and what as axes is open to the user. Different analysis tasks require different representations.

Figure 3.1: Angular brushing in Caleydo. The brush is defined on line 1. Notice the legs of the angle that can be dragged in the directions indicated by the arrows. Line 2 is not in the selection, line 3 is.
When a life scientist, for example, wants to analyze three experiments based on samples taken from a patient during a drug trial, it is obvious, that she will be interested to see the behavior of many genes in relation to few experiments. This is emphasized by choosing the experiments as axes. However, when the behavior of only a handful of genes in many experiments is of interest, interpreting the genes as axes are beneficial. Current parallel coordinate browsers do not support such an exchange feature. When an exchange is required one has to alter the data. We believe that it is better to support run-time switching of axes vs. polylines. This is especially significant, when the number of data items is sufficiently small, which is usually the case when parallel coordinates are used for contextual analysis.

3.2 Heat Map

Heat maps are a valuable tool for visualizing gene expression data, especially to identify functional similarities in genes without sequence similarities (see section 2.3). In the context of this work we are primarily interested in their ability as a contextual visualization for pathways and parallel coordinates. This is why we decided to use unclustered heat maps. Clustering is a pre-processing step, which is not within the scope of this work. The implementation of the visualization is designed in a way that clustering can be done according to the filter paradigm, using a command interface (see section 4.2.2).

As already mentioned our objective with the heat map is to allow the visualization of gene expression data in context of pathways. As discussed in section 2.4 current approaches, namely mapping expression data onto the nodes and showing them in pop-up windows, suffer certain drawbacks. We believe that those drawbacks can be overcome by connecting a heat map and the pathways with visual links, an aspect that will be discussed in detail in section 3.4.

When a lot of data is visualized with heat maps the fields are soon becoming fairly small. This is a severe problem for the contextual visualization, as we are interested in the properties of one particular gene. We propose to apply a fish-eye distortion to the heat map in the area of current interest, thereby allowing to view the selected gene in great detail, while still providing contextual information within the heat map (which is only relevant when the heat map is clustered).

To meet the problem of heat maps exceeding the available screen space, we propose to implement a panning mechanism, similar to the parallel coordinates. Again the panning is applied automatically when contextual information is available.

For optimal usage of screen space it should be possible to change the orientation of the heat map.
3.3 View Arrangement

For the implementation of visual links the arrangement of 2D views in 3D space is essential. In *VisLink* [Collins2007], the arrangement of views is completely free, with a certain number of predefined positions activated by keyboard shortcuts. However, Collins et al. note that their arrangement is not satisfactory. The more restricted approach proposed by Streit et al. [Streit2008] of placing views on top of each other also suffers from certain problems, most notably that only adjacent entities can be connected. It is out of this realization that Streit et al. have developed a new, still unpublished system - *The Link Bucket*. As the work presented in this thesis was implemented in the Link Bucket we briefly introduce it here.

Figure 3.2: View arrangement in the Link Bucket. Views are placed on the bottom, the sides and the rim.

[Figure 3.2 illustrates the basic setup of views in the Link Bucket. The Link Bucket is a metaphor for a real-life, square bucket with a bottom, walls and a rim. Views are placed on all these areas, according to their priority. The view in focus is placed on the bottom of the bucket, thereby assigned most of the screen space. Up to four contextual views can be placed on the walls, where the perspective distortion reduces the required screen space while still allowing all parts to be visible. Views of less importance in the current context are placed on one side of the rim, with a miniature representation and a textual annotation. To explore the miniature views in more detail, they are enlarged on mouseover. The other side of the rim holds a memo-pad, where interesting views can be stored for later retrieval.

The Link Bucket itself can not be moved, only restricted navigation is allowed. By turning the mouse wheel the view at the bottom is zoomed into full-screen, using a camera flight to emphasize the real-world bucket metaphor. Views can be exchanged by drag and drop, or by using the navigational panel shown in figure 3.3.

Pathways can be placed in the Link Bucket in two ways: By searching and adding them manually, or by clicking an entity, upon which Caleydo loads related pathways based on a ranking. The views with the highest ranks are placed on the walls, those relevant, but not ranked highly enough on the rim.

Views of no current relevance are swapped-out automatically. To avoid this they can be locked at their position by clicking the lock symbol visible in figure 3.3. Another way to preserve views is to drag them onto the memo-pad.
3.4 Visual Linking

Visual linking is the process of connecting two or more related or identical entities in different views with visible links, for example lines or curves (see section 2.5). Generally all kinds of relations can be visualized. However, within the scope of this work we will only be concerned with occurrences of the same entity in different representations. It is in the nature of pathway graphs that entities can occur several times within one graph, in fact they often do. When visual linking is applied to the aforementioned setup of views in the Link Bucket, one entity can occur many times in a visualization. A simple example: the Link Bucket is filled with one parallel coordinate view, one heat map, each with one occurrence of a gene. Furthermore three pathways are currently in the bucket, each containing 3 instances of the gene. To connect all instances with each other the naive approach would be to simply connect all element with each other, which would result in $n(n-1)/2$ edges to be drawn. For the aforementioned example this would result in 55 connection lines.

It is obvious that this would lead to unnecessary visual clutter. To avoid this, we propose to implement the strategy illustrated in figure 3.4:

All edges associated with one view are joined into one single view-bundling-point, illustrated in red in figure 3.4. These bundling points in turn are connected to a central bundling
point (yellow in Figure 3.4). This results in a reduction to only $n+v$ required connection lines, where $v$ is the number of views. For the aforementioned example this would mean that only 16 edges have to be drawn. Additionally this approach clears up the central and most important area of the Link Bucket, as only one line per view is drawn, thereby further reducing visual clutter.

Figure 3.4: Visual linking edge bundling strategy. Edges are first bundled on a per-view basis (red) and then in a central bundling point (yellow).

Finally, we need to address the problem of the docking points for visual links, as discussed in Section 2.5. It is not obvious where to connect a line when an object has a certain size. This is especially evident when connecting parallel coordinates. Where should the line be connected to – to an arbitrary point on the polyline? We propose another approach which we believe to be useful: a triangle fan. Connecting each line segment of the polyline with a triangle to the view bundling point emphasizes the structure of the polyline. Drawing additional edges at the intersection of the polylines and the axes improves the understanding of the 3D structure. We can thereby produce a well-suited representation of the connections between polylines and points. The per-view bundling approach and the use of transparency on the triangles reduces visual clutter to a minimum.
Chapter 4

Design and Implementation

This chapter will provide an overview of the design and implementation of the software developed in the course of this work. To understand the rationale, it is important to be familiar with the design principles within the Caleydo framework, therefore section 4.2 will introduce those briefly. In section 4.3 we will explain the design of the parts which were developed for this thesis. The following section is concerned with the technologies, tools and third party libraries used in the Caleydo framework.

4.1 Used Technologies

The Caleydo framework is written in the Java programming language. Development is being conducted in Eclipse under Linux and Windows. Caleydo uses the Eclipse Rich Client Platform (RCP) which provides basic application functionality, such as window management, toolbars etc. RCP is based on the Standard Widget Toolkit (SWT), a GUI library developed by IBM in the course of the development of Eclipse. SWT has two major benefits over other toolkits like AWT and Swing. First, SWT has a native implementation on all supported platforms and therefore provides native look and feel and second it is faster than the alternatives.

For accelerated graphics Caleydo uses JOGL which is a wrapper library for OpenGL, thus making it possible to use OpenGL in Java. JOGL is a lightweight layer between OpenGL and Java, exposing much of the underlying structure of OpenGL to the Java programmer. While this is effective in terms of library maintenance, code conversion (from other programming languages) and performance, it combines two different programming models - the object-oriented Java approach and the state machine/procedural approach of OpenGL - in one application.

For testing the Caleydo project uses JUnit a popular Java unit testing framework.

1http://java.sun.com
2http://www.eclipse.org/
4http://www.eclipse.org/swt/
5https://jogl.dev.java.net/
6http://www.opengl.org/
7http://junit.org/
4.2 Framework

Caleydo is a framework developed for information visualization. The goal of Caleydo is not to provide different standard visualization methods, but rather to be a platform for rapid development of new, customized visualizations. This is reflected in the architecture throughout the system. Currently it contains around 850 classes. Details on the design are also explained in [Streit2007].

To explain the architecture, we will follow the basic steps which are executed when Caleydo is started, as illustrated in [figure 4.1](image).

![Figure 4.1: Basic steps executed on Caleydo start-up.](image)

4.2.1 Bootstrap

Caleydo is a flexible system allowing to use many of its parts separately. The set-up of a particular visualization is not implemented in Java, instead it is specified in configuration files. Configuration files can be passed as command line arguments, or, when no argument was specified, by choosing a file from the file system at runtime.

4.2.2 Read Configuration

The aforementioned configuration files are written in XML, which makes them easy to process. Caleydo uses a SAX (Simple API for XML) parser, which is a serial parser. Therefore the parser reads XML files token by token, and does not store them in RAM. This makes it suitable for scenarios where information needs to be processed only once, as is the case with reading configurations.

The configuration files hold the following information (list is not exhaustive):

- Which views are to be constructed.
- What data is to be read.
- Which data mappings are to be read.
- Which view needs which data.
- Which views interact with each other.
- Whether preprocessing for the data is necessary.

We will go into detail concerning the different possibilities for each of those items in the upcoming sections. At this point we are interested in how the Caleydo framework allows such a flexible approach. This is achieved by using an implementation based on the Command Pattern, as described in [Gamma1995, p. 233-242]. All commands in Caleydo
implement the ICommand interface and all commands are registered with the SAX parser. Therefore it is possible to execute commands out of XML files, but also directly from the code. All the operations associated with the listed tasks are implemented as commands, which allows the simple composition of different setups.

4.2.3 Read Data

Caleydo currently supports three different types of data: graphs, lists and mappings. An example for graph data currently used is the metabolic network. The internal graph is constructed with information derived from the pathways. This information is provided by KEGG and BioCarta, but only for one pathway at once. To allow for operations such as neighborhood search, Caleydo constructs one overall graph out of all pathways. While the pathways can contain a single node several times, Caleydo stores every element only once. When a duplicate node is processed, its edges are simply added to the already existing one. Because of this, Caleydo knows about relations between the pathways, which it then can visualize.

Caleydo stores list data such as microarray data using a special Storage Concept, developed by Michael Kalkusch [Kalkusch2006]. A Storage is an encapsulation of primitive arrays. A Storage can hold lists of all kinds of primitive data types as well as Strings and Objects. The rationale for using primitive arrays instead of high level containers is performance. For gene expression data one experiment can have up to 30,000 entries, and the number of experiments can be fairly high (<1000) as well. In higher level Java containers every entry is encapsulated in an object, which is obviously not beneficial when dealing with large quantities.

The drawback of using arrays instead of high-level containers is that it is complex and computationally expensive to perform operations such as adding or removing elements. Caleydo avoids this by never modifying the length of the lists. Instead Caleydo uses a special selection mechanism, which will be explained shortly.

![Diagram of Sets and Storages of a view.](image)

Figure 4.2: The Sets and Storages of a view. A view can hold two different types of Sets: data Sets and selection Sets.

Storages are grouped into Sets, which are containers for Storages and Virtual Arrays. A Virtual Array specifies the order of reading a Storage. It is conceptually similar to a
iterator, containing information about which element to read next. Views hold two types of Sets, *data Sets* and *selection Sets*, as illustrated in [figure 4.2](#). A data Set typically holds several Storages, all filled with data. Selection Sets can be used for two purposes: to implement local selections, or to communicate with other entities in the system via updates (this will be explained in more detail in [section 4.2.5](#)). Which Set is used for what depends on convention. The selection Sets hold lists of indices which correspond to the Storage indices. Thereby the ordering is meaningful and indices can exist 0:n times, as illustrated in [figure 4.3](#).

Figure 4.3: *The selection mechanism employed in Caleydo. The indices are shown above, the stored data within the boxes. The arrows illustrate the references.*

The actual number of Sets and Storages, their relations etc. are configurable via commands.

**Mapping data** is essential for the different parts of Caleydo to work together. This is best explained with an example: a pathway contains enzymes, those enzymes correspond to 1:n genes, which can have one of several representations (different IDs, like the accession number or the NCBI Gene ID, short names, long names, etc.). Genes can have 0:n representations on a microarray. To keep Caleydo independent from concrete visualizations, entities also have a system-internal ID. Therefore, to know which gene name is associated with one entry in a Storage, that represents microarray data, the system has to look up the following mappings:

- Storage Index to System ID
- System ID to Accession Number
- Accession Number to Gene Name

This is achieved by using an ID manager, which stores the mappings in hash maps. The mappings are created on system start-up based on text files where equivalencies are listed. At the time of writing, a simplification of this system is being implemented, where all data is mapped to a single central entity. This reduces start-up time and memory usage, simplifies the conversions (the aforementioned example would be reduced by one step) and maintenance.
4.2.4 Create Views

Views are the entities that are responsible for the display of data. Caleydo knows two basic types of views: OpenGL and SWT views. SWT views are used where no drawing is done, as for example in a browser or a table. Those views are native SWT views, containing widgets such as buttons, sliders etc. Caleydo also provides a mechanism to integrate views that use one of the other Java GUI libraries, AWT and Swing through embedded SWT views. One example for an application of an embedded SWT view is a 2D graph viewer, which uses the JGraph library. It is implemented in Swing and therefore needs to be embedded in a SWT view to be usable in Caleydo.

Visualization views in Caleydo are usually implemented in OpenGL. There are two reasons for that: performance and flexibility. OpenGL allows to use hardware rendering, and therefore can use the processing power of the GPU, leaving the CPU resources available for other tasks. This is especially important for a large number of objects.

A valuable mechanism are OpenGL display lists. They allow to avoid the execution of OpenGL code that has not changed in a display cycle on the CPU. This is achieved by keeping the commands in display lists on the graphics card, and replacing them only when necessary.

However, the main reason for the use of OpenGL is the flexibility it provides when composing single views to a larger one. While Caleydo currently mainly uses 2D visualizations, it can tilt them into 3D thereby creating 2.5D composite views. The benefits of doing so have been elaborated in section 2.5 and section 3.3. Concepts like the Link Bucket or the Jukebox require that views can be nested, which is achieved by the software design modeled in figure 4.4.

All OpenGL views in Caleydo extend the AGLCanvasUser class. This class provides functionality that is common to all OpenGL views. Among those are implementations for methods defined by the JOGL interface GLEventListener that AGLCanvasUser implements, for example the reshape() method. Other interface methods that can not be implemented, like init() and display() are defined as abstract, thereby requiring inheriting views to implement them (cp. figure 4.4).

The ability to nest views requires that a view is able to distinguish whether it is used remotely or locally. One reason for that is, for example, that a view needs to determine whether event handling has to be done locally or is done by the remote class. Therefore the methods initLocal(), initRemote(), displayLocal(), displayRemote() were introduced. The remote and local function each have to call the native functions (init() resp. display()). It is therefore encouraged to implement as much functionality as possible in the native functions, and use the remote and local functions only when needed (this avoids code duplication).

This mechanism allows views to be rendered either as part of a larger setup, or as a standalone view with its own SWT window, or both at the same time. Figure 4.4 for example shows the relations of a remote view, GLCanvasRemoteRendering3D and a local view, GLCanvasSpecializedView (concrete examples are the heat map or a pathway view). Remote views can hold 0:n other views, identifying them by their
system-wide unique ID, which is provided through the inheritance of `AUniqueManagedObject`.

The characteristics of a remote view, specifically how its views are arranged, is determined by concrete implementations of `ARemoteViewLayoutRenderStyle`, which for the Link Bucket is `BucketLayoutRenderStyle` and for the Jukebox `JukeboxLayoutRenderStyle`.

### 4.2.5 Set-up Event System

The event system in Caleydo follows the observer and the mediator patterns described in [Gamma1995](#). Caleydo supports three types of updates: data, selection and view updates. Participants can register at an Event Publisher as reader and/or writer of another participant. This is done using commands, therefore can be done in the XML definition. However, when views are dynamically loaded and removed, as is the case with nested views, it is the responsibility of the managing view to connect its children correctly. Every object can be a sender or a receiver, it only has to implement the `IMediatorSender` resp. `IMediatorReceiver` interface and register with the mediator. The mediator also prevents circular updates.

There are two methods to update a receiver: by just informing it that something happened, or by additionally passing data. The listening view is notified of a change by a call to its `updateReceiver()` method, which is part of the `IMediatorReceiver` interface.
Writing an update is achieved by calling the `updateSelectionSet()` method on a Set. The details of this process will be explained in section 4.3.

### 4.2.6 Runtime

After all these steps a Caleydo visualization finally runs in a loop, waiting for user interaction. However, we have omitted one important concept in the Caleydo framework so far: the managers. Managers are classes that fulfill a central job in the framework. Each manager has only one instance, it is therefore implemented as a singleton [Gamma1995, p. 127-134]. They can be accessed via the `GeneralManager` on which every class, that inherits from `AUniqueManagedObject`, has as reference.

Managers provides functions as diverse as ID management, data management and mapping management. Due to their implementation as a singleton they can only be used for system wide and not for local tasks.

### 4.3 Design

In the following the design of the parts contributing to the system developed will be discussed. First some basic enhancements for the framework are described, such as normalization, picking, selection management and update merging, followed by three sections on Storage based views, first in an abstract form, then concrete for parallel coordinates and heat maps. The final topic is the connection line management in the Link Bucket.

#### 4.3.1 Normalization

The implementation of views such as the heat map and the parallel coordinates require the data to be normalized. This was achieved by implementing a designated command. When normalizing values, it is necessary to know the minimum and maximum values in the data which is to be normalized. As a consequence the Storages had to be modified to provide information on their minimum and maximum. This functionality is implemented as a lazy evaluation, as it is a computationally expensive task and not needed in all cases.

By convention all values in the Storage are normalized in a range between 0 and 1, with single precision (float). To fit different needs, normalization is implemented in two forms: as an in-place method which replaces all values in the Storage by their normalized equivalents, or by creating a new array for the normalized values.

The normalization and the evaluation of minima and maxima is tested with unit tests, as it is crucial for the correctness of the statements generated by an analysis.
4.3.2 Picking

The interaction with views in OpenGL requires using the OpenGL picking mechanism. While this was already used in the Caleydo framework, it was only implemented for one view and for a narrow use-case. With the complexity of interaction in parallel coordinates, and particularly with the complexity of picking in nested views it became evident that the previous approach would not suffice. Furthermore picking in OpenGL is rather complex, which reduces the ability to develop new views rapidly. We therefore decided to implement a PickingManager which inherits the AManager and is therefore accessible for every UniqueManagedObject via the GeneralManager.

The idea of the picking manager is, that every view can get a picking ID, based on the unique view ID, a picking type and an ID that the view can choose freely. The picking type is one of the types defined in an enumeration. Examples are a type for buttons, a type for the selection of polylines, etc. The picking manager generates a system-wide unique ID out of this information, which has to be passed to OpenGL by the glPushName() OpenGL call. This identifies the object for OpenGL.

In each display cycle the view has to call the handlePicking() method, which processes the picks OpenGL encountered. After that the picks are available via the getHits() method, which takes the pickingType and the viewID as parameters.

When the processing of the picks is finished, they have to be cleared using the picking manager’s flushHits() method, again supplied with the same parameters.

To further reduce the complexity when creating new views, some of these functions are called in the abstract AGLCanvasUser class. This class in turn provides a handleEvents() method that is called automatically. The system therefore provides a comfortable callback for events, as most modern GUI toolkits do.

While the PickingManager hides most of the OpenGL picking interface from the user, it can only provide information on which OpenGL object has been picked. Other related information, such as the current mouse position in a scene have to be accessed via a separate interface, the PickingTriggerMouseAdapter. It has a method that returns the screen coordinates. To convert screen coordinates to world coordinates the use of the GLCoordinateUtils class is recommended.

4.3.3 Selection Management

Similar to picking, selection management is crucial for interactive visualizations. It is not sufficient to say one item is selected or not, different types of selections exist. Each applied brush is a different selection. Furthermore selections should be composable, negatable etc.

We therefore decided to implement a selection manager, the GenericSelectionManager. Contrary to the picking manager, it is not derived from AManager as it is not a system-wide tool, but specific to the view. As a consequence each view can hold its private instance of the selection manager.

The implementation is based on hash maps. It has to be initialized with all selectable
elements. It provides methods to check for elements of particular selection types, to check the number of elements of a selection type, to add and remove elements to the types, etc. It guarantees that every entry is only contained once in the manager.

### 4.3.4 Update Merging

To allow **Linking & Brushing**, it is necessary to provide a way of communication among the different views. In Caleydo this is achieved by using Sets, containing information on which entities to display. This process is illustrated in [figure 4.5](#).

![Figure 4.5](image)

**Figure 4.5:** The update merging mechanism in Caleydo.

Every view has at least one Set where the current selection state is stored. Notice that these selections are different from the selections in the previous chapter, since they are system-wide selections. These Sets contain three arrays, one with element IDs, one with the group IDs and one with arbitrary content (of the Java type `Object`).

Such a Set is updated by merging it with a modification Set. When the `updateReceiver()` method of a view is called, one can pass the modification Set. By calling `mergeSelection()` on the local Set and passing the modification Set, the modification Set and the local Set are merged, as shown in the result Set (which is the local Set) in [figure 4.5](#).

The group IDs follow a certain convention: 0 means that the element is not selected, bigger than 0 means that the element is selected and -1 means that the element should be removed from the list. This means that local Set can contain values with 0 or more in the group ID field, the modification can additionally contain -1. When merging, all entries with -1 in the modification Set are removed from the local Set, all entries that occur in the modification Set have their group value set to the group value of the modification Set and all elements in the modification Set that do not occur in the local Set are added.
4.3.5 Storage Based Views

As discussed in section 4.2.4 all OpenGL based views are inherited from AGLCanvasUser. For views heavily based on data stored in lists, such as parallel coordinates, heat maps or scatter plots an additional layer of inheritance was introduced, the AGLCanvasStorage-BasedView (see figure 4.6).

![Figure 4.6: The software design of Storage-based OpenGL views in Caleydo. Notice the additional layer, AGLCanvasStorageBasedView, when compared with figure 4.4.](image)

This additional abstraction makes use of the fact that the aforementioned methods are different visualizations of the same data, with similar properties. Parallel coordinates and heat maps, for example, share not only the data structure, but also the way they handle selections. This abstraction therefore further facilitates the rapid development of new visualization methods and avoids code duplication.

4.3.6 Parallel Coordinates

In the following we will present the software design of the parallel coordinates. For brevity, only a few highlights of the extensive implementation are given.

When using composite brushing, it is not trivial to determine which polylines are actually brushed, specifically when one brush is modified and thereby removes a polyline from the set of selected polylines. In such a case it can happen, that polylines are still selected through another brush. Thus one has to be cautious about removing lines from a selected set. We achieve correct behavior by evaluating all brushes at each render step. Notice, that this is no problem as far as performance is concerned, as we rely on OpenGL display lists. As a consequence calculations are only performed when something in the rendering changed. Every brush produces a list of all its selected entities, which are then merged into one hash map (to avoid duplicate entries) and consequently written to the selection manager.
To allow a run-time exchange of axes and polylines, the rendering is implemented independent of the actual data structure. This allows, that, within the rendering code, only two if conditions are necessary to determine whether a Storage represents a polyline or an axis.

To enable moving, removing and duplication of the axes, a mechanism similar to that illustrated in figure 4.3 is employed. The parallel coordinates hold a list of indices with references to the Storage elements, and one with references to the Storages themselves. The rendering always uses those lists, thereby allowing the move, remove and copy operations on the otherwise inflexible Storages.

4.3.7 Heat Maps

The implementation of the heat map is conceptually similar to the parallel coordinates. Of general interest might be the fish-eye implementation, which is completely implemented in the HeatMapRenderStyle and therefore hidden from the actual implementation. Render styles are a way Caleydo views handle their visual properties, such as color and size. Every time the heat map wants to draw an element it asks the render style for the width and height of the element. The render style knows about selected entities, and returns the values according to a set of rules. Therefore the heat map is not aware that it is actually rendering a fish-eye distortion.

4.3.8 Connection Line Management

Visual linking can be understood as a process with two main parts: the arrangement of the views, and the drawing of the connection lines. The concepts behind the implementation of the Link Bucket and the connection lines were explained in section 3.4. The management of the connection lines are the topic of this section.

The implementation of the connection line management is realized by the interaction of a number of classes, illustrated in figure 4.7. Three classes are at its center: SelectedElementRep, ViewSelectionManager and AGLConnectionLineManager.

The SelectedElementRep class acts as the interface to the views. They store the representation of an element that should be connected. Whenever an element is selected, be it by a brush in that particular view or by an external update, the views have to create a SelectedElementRep for that element. Currently the view can choose between two representations: point and line. The former is relevant for views that want the element to be connected by a simple polyline, as for example pathway views usually do. The latter is meant for views that want the element to be connected with a triangle fan, thereby emphasizing the property of the element. This is typically needed by parallel coordinate views.

Once the element representation is created, the view notifies the ViewSelectionManager (see figure 4.7).

The view responsible for the remote rendering of the other views, an example would be GLCanvasRemoteRendering3D, holds an instance of AGLConnectionLineRenderer, for ex-
Figure 4.7: Connection Line Management in Caleydo. The views produce representations (SelectedElementRep) which they pass along to the SelectionManager. The concrete connection line renderers query the selection manager for the representations and construct the composite connection lines.

ample a GLConnectionLineRendererBucket. The difference between the connection line renderers is the policy on how to connect the views. For the Jukebox, it only connects entities of adjacent views, for the Link Bucket it follows the policy described in section 3.4. In the following we are only concerned with the implementation for the Link Bucket.

The connection line renderer queries the selection manager and retrieves the respective element representations. Notice that the view keeps its reference of the representation, which allows it to update the representation when necessary. An example is when the view is being panned - whereupon the location of the selected element changes.

Depending on the type of representation (line or point), the connection line renderer renders a triangle fan or a set of lines. It creates the bundling point on a per view basis and a central bundling point, were all view bundling points are connected to.
Chapter 5

Results

In this chapter we will elaborate the results achieved by implementing the concepts discussed in chapter 3. First the features of the different parts are presented from a technical point of view. Following that is a use case study for two different approaches in the field of gene expression analysis, to elaborate possible work-flows.

5.1 Parallel Coordinates

Figure 5.1 shows the parallel coordinates view as implemented in the Caleydo framework.

Figure 5.1: Parallel coordinates showing more than 2000 genes as polylines and six experiments as axis. Note the info area on top providing details on demand.
The data shown in this and all subsequent screen-shots (unless noted otherwise) is gene expression data from human and mouse samples. The data is preprocessed and contains about 30,000 elements, of which a sample of about 2,000 data items is shown in figure 5.1. Above the parallel coordinates, an info area is rendered, providing details on demand. In this case, it contains information on the current view, for example the number of polylines and the relation of data and its representations (axes - polylines to experiments - genes). The info area provides details on demand in all views in Caleydo. For the sake of brevity we will omit it in all following figures.

On top of the axes the captions are rendered. Notice also the values at the top and the bottom (in this case 1 respectively 0) for all axis, that indicate the range of the display. The red line is a polyline selected by a click. To provide detailed information on the actual numerical values they are shown at the intersection with the axes.

Figure 5.2 shows the effect of using transparency in parallel coordinates. Both views show the same data set. However, the transparent view (b) in figure 5.2 reveals trends and structures (for example the high density at the bottom) which are occluded in the non-transparent view (a). Transparency does not prevent outliers from being visible, as is evident in figure 5.2 (b). Notice the single lines that are visible at the top, which are still recognizable with transparency enabled.

![Figure 5.2: Parallel coordinates with versus without transparency. In (a), transparency is enabled, in (b) it is not. Both views show the same data (Sample ≈ 2,000)](image)

To cope with an arbitrary number of polylines the alpha values for the polylines are determined by a function of the number of polylines. The limit for the number of polylines is effectively determined by the performance of the computer. The alpha value calculation also takes into account dynamic masking as shown in figure 5.4. When many polylines are masked, the rest of the polylines is drawn with a higher intensity thereby increasing the visibility.
The rendering results for different numbers of polylines are shown in Figure 5.3. Figure 5.3(a) depicts the dataset with a sample of about 5,000 elements, (b) contains approximately 15,000 polylines, and (c) visualizes about 30,000 data items (which is a number that is larger than the number of genes in the human genome).

As evident in Figure 5.3, no new trends can be seen in the plots showing the larger samples, which validates the sampling approach. Even with fewer elements, as in Figure 5.1 with only 2,000 elements, the trends are visible. An example of a small trend recognizable in all representations is the strong density of values close to the maximum between experiment 0 and 1 and the subsequent spreading between experiments 1 and 2.

As far as performance is concerned: interactivity is significantly impaired when rendering 30,000 polylines. On a notebook computer with a NVIDIA Quadro NVS 140M with 128 MB VRAM (video RAM) and an Intel Core2 Duo CPU with 2 GHz clock frequency the system needs a couple of seconds to react on input. Interaction with 15,000 polylines is still not satisfying, with 5,000 it is reactive.

The application of composite brushes is illustrated in Figure 5.4. Image (a) shows the familiar dataset with two gates set. One on the second axis at 0.13, thereby removing many of the polylines concentrated in the low regions. This concentration however, is still visible due to the dynamic masking applied for the deselected lines. The second gate blocks all polylines on the third axis between 0.4 and 1.

By applying an angular brush, as shown in Figure 5.4 (b), on a polyline going down between axes one and two all polylines behaving against this pattern are removed. Notice that this is the majority of the polylines, therefore those behaving contrary to a main trend are emphasized.

When a selection is satisfying to the user, and she wants to explore it in more detail, she can click the “save” button (not shown in the pictures). This removes all polylines currently not in the selection (but visible due to dynamic masking). It also propagates the selection to other views, like the heat map.

Moving, copying and removing of axes are illustrated in Figure 5.5. Picture (a) shows the
Figure 5.4: Parallel coordinates with different brushes applied. In (a) two gates are applied. (c) shows an additional angular brush.

Dataset where an angular brush between the first and the second experiment was applied. The brush was subsequently saved, thus removing all the unwanted polylines from the background. Figure 5.5 (b) shows the duplication of the second and the fourth axis, easily recognizable by the horizontal lines connecting the respective axes.

Notice that the spacing between the axes is automatically adjusted to fit the additional axis. Furthermore, one axis was selected (therefore drawn in yellow) and consequently the second instance of the same axis is highlighted as well.

In Figure 5.5 (c) the first and fourth axes were removed, and both axes copied previously were moved right.

Figure 5.6 illustrates the ability of the parallel coordinates to exchange axes and polylines in real-time. In (a) several brushes are applied to the dataset. Experiments are rendered as axes and genes as polylines. The result of saving the brushes, thereby reducing the number of polylines and clicking the icon for axis-polyline exchange (not shown) can be seen in (b). Notice that there are six polylines (one for each experiment), but more axes (one for each gene) than fit on the screen. The axes right of the yellow axis are clipped. By clicking the yellow axis, the system automatically moves the yellow axis to the left, to reveal the previously clipped axes. This is shown in Figure 5.6 (c).
Figure 5.5: *Moving, removing and duplicating of axes.* (a) shows a subset of the data. In (b) two axes are copied. (c) shows two axes removed and two moved.

Figure 5.6: *Axis - polyline exchange.* In (a) axes are experiments and polylines are genes. This is inverted in (b). (c) shows the effect of panning.
5.2 Heat Map

Figure 5.7 shows the heat map as implemented in the Caleydo framework. The leftmost image depicts the heat map as it is rendered upon startup, with no selections and no distortions. Again, the same dataset is shown, but only around 50 genes are visible. To allow the visualization of more genes concurrently, the spacing can be adjusted. As a consequence it is possible to show approximately five times as many genes as with the shown settings. This comes at a cost: the single elements are not as well distinguishable and the caption next to each element can not be shown. We decided to use a large resolution in the Link Bucket setup, as we use the heat map only as a contextual view for pathways (see section 3.2).

![Figure 5.7: Heat map and selections with fisheye distortions. (a) shows the heat map as rendered on system startup. In (b) one selection, and in (c) two selections are shown.](image)

It is evident, that the heat map can accommodate more experiments than the parallel coordinates. The visualization of six experiments only uses up a narrow band in the center of the screen. At the depicted resolution up to 30 experiments are visible concurrently in a square view, as used in the Link Bucket.

To improve the visibility for contextual visualization we implemented a fisheye distortion for selected elements, as shown in the center of figure 5.7. This fisheye also works for multi-selections as can be seen in the rightmost image.

The heat map can be rendered in two directions, with the experiments aligned vertically and the genes horizontally, or the other way around. We found the former to be advantageous for contextual visualization in the Link Bucket, the latter better for a standalone visualization.
5.3 Link Bucket

The Link Bucket and an associated browser view in an RCP application is shown in figure 5.8. The Link Bucket itself is in the center. On its left are contextually relevant views, which can be swapped into the bucket by clicking them. To the right is the memo pad, where views can be stored, and the trash, which is used to remove views. On top of the Link Bucket the info area is drawn.

Above the bucket one can see two drop-down boxes, where genes and pathways can be searched respectively selected. The browser view right of the main window provides contextual information on the selected entities in the pathways by retrieving the corresponding entry for a gene product from a major genetic database website. This website provides detailed information on the product and links to other relevant bioinformatics databases and to publications concerning the product in PubMed (the widely-used publication search engine for medical topics).

The system allows arranging the windows with the views freely. Typically such a setup is run in a multi-monitor environment, where the Link Bucket is on the main screen, and other views, as the browser, are on a separate screen.

New pathways are loaded by clicking an entity, such as a polyline in the parallel coordinates, a field in the heat map, or a gene respectively enzyme in a pathway. This triggers the system to load all pathways that contain the clicked entity. The new pathways are placed inside the bucket, when space is available. Otherwise, they are put in the rim, from where they can be swapped into the bucket at any time.

In figure 5.9 one can see how the connection lines are drawn. The line bundling of two lines is visible just above the lower pathway, where the two lines coming out of the view are joined. The triangle fan for the parallel coordinates is shown in the left picture. The triangle fan emphasizes the structure of the polyline. In this case, three experiments are over-expressed (i.e. up-regulated) significantly. The right picture in figure 5.9 shows the same gene selected, but this time the genes are shown as axes. Note that the three over-expressed experiments for the selected gene are visible in the parallel coordinates, but the trend is not as obvious.

In a case as depicted in the right picture in figure 5.9, where parallel coordinates and heat map are clipped, the panning discussed previously is done automatically as soon as an element in an other view is selected.

---

1http://pubmed.gov/
Figure 5.8: The Link Bucket in a RCP window with an attached browser view. In the center the bucket can be seen, containing a heat map, a parallel coordinates view and pathways. To the left are other related pathways. On the right is the memo pad and the trash bin. Right of the main window a browser is attached. Genes and pathways can be searched and loaded by using the drop down boxes at the top.
5.4 Use Cases

To demonstrate the actual value of the developed software we will present two use cases from the bio-medical domain in this section. First we will consider a case in which a researcher approaches a problem from a known pathway. The pathway is suspected to play a role in a disease. The second case does not involve prior knowledge about a biological process. Instead different experiments in the data come from a different clinical background.

Notice that the following use-cases are not meant to convey biological truth. Especially the assumptions on the type and origin of the data are made up and are to be understood as illustrations for the interaction only.

5.4.1 Prior Knowledge on Processes

The processes depicted in pathway *HIV-I Nef* is known to have an influence in the drastic declines in CD4 T helper cells, due to the apoptosis (cell death) of uninfected cells in HIV positive patients. This is a known fact that can be derived from this Biocarta pathway and its description. However, there are other processes that play a role in the apoptosis. The life science professional therefore starts Caleydo and loads the *HIV-I Nef* pathway using the integrated pathway search feature. Caleydo adds the expression data of all occurring genes automatically. First he zooms-in to explore the pathway in detail (see Figure 5.10 (a)). He sees the gene *ASK1* which is known to play a role in this process. By clicking the gene and subsequently zooming-out again Caleydo loads all pathways (and the associated expression values) that also contains the *ASK1* gene, shown in (b) in Figure 5.10. However,
the gene is not involved in any other interesting pathways, and is not highly expressed in the sample data he loaded from six patients.

Figure 5.10: Prior knowledge use case. By clicking a suspected gene (a) related pathways are loaded and shown in the Link Bucket (b). The gates in (c) filter the data, whereupon other highly expressed genes can be explored (d).

Thus he filters the expression values, using the parallel coordinates, to contain only highly expressed genes, as shown in figure 5.10 (c). After applying the filter he browses the remaining expression values, and finds a gene, that is highly expressed and also contained in most of the pathways that contain the ASK1 gene, NFKBIA (see figure 5.10 (d)). Hence he decides to inspect the properties of this gene and its role in other pathways.
5.4.2 Gene Expression Data Centered Approach

In a data centered approach, another researcher tries to derive new knowledge without employing preexisting knowledge on gene functions. She therefore starts Caleydo and loads data from different experiments, for example three samples from healthy (e.g., Exp. 0, 1 and 5 in figure 5.11) and three from cancerous tissue (Exp. 2, 3 and 4 in figure 5.11). The data shown is not a random sample, but includes expression data on all genes that occur in any pathway, which are around 3000 for this dataset. Expression data for genes that do not occur in the dataset are omitted.

First the user reorders the axes so that similar ones are next to each other. She then reduces the data by applying two filters: for experiment 0, she excludes all highly expressed values, for experiment 4 all genes with low expression. After applying these filters, the expert is confronted with a view where several patterns are visible, as can be seen in figure 5.11 (a). Two main trends can be identified: gene expression increases significantly between experiment 0 and 1 (an example is the highlighted polyline in (a)) and, even stronger, between experiment 5 and 2 (axes 3 and 4), of which an example is highlighted in figure 5.11 (b).

As the expert is aware that the first three axes represent healthy samples, she chooses to focus on the second pattern, by applying an angular brush that highlights elements with a high slope between experiment 5 and 2. This is illustrated in figure 5.11 (c). After saving the brushes, bringing the axis into the original state and zooming out she now can see the pattern in both, the heat map and the parallel coordinates, figure 5.11 (d). She now clicks one gene after the other whereupon the system starts loading pathways containing the clicked genes. By browsing different pathways she soon finds out that all of the genes coincide in the same element in the pathways (evident in (e) and (f) in figure 5.11). She therefore concludes that she found an enzyme which is encoded by those genes and is up-regulated in cancerous cases and down-regulated in healthy tissue. She now can explore the properties and the involvement of this enzyme in detail and find out whether it plays an important role in the development of this type of cancer.

5.5 User Feedback

While we have not conducted formal user studies yet, initial feedback from our partners from the bio-medical domain was enthusiastic. They especially emphasized the value of the connection lines, which bring the different views into context. While they were unfamiliar with the concept of parallel coordinates they soon recognized its possibilities for interaction and embraced it.
Figure 5.11: *Data centered analysis use case.* (a) and (b) show trends. Brushing is applied in (c), the result is shown in (d). (e) and (f) show the exploration of the selected genes.
Chapter 6

Conclusions and Future Work

The chosen visualization methods for gene expression data have the ability to improve the diagnostic and research working process of life scientists. Parallel coordinates are valuable for their ability to convey trends on a large scale, and especially for interaction. They thereby facilitate a visual exploration process, which can lead to findings that remain hidden with classical statistical analysis. Handling large datasets by using random samples works well for the visualization of large datasets. Whenever the data can be reduced by employing contextual information from other views, we do so. This is the case when using parallel coordinates in combination with pathways, as the number of genes occurring in pathways is sufficiently small to be handled without sampling.

The use of the heat map to visualize contextual information on pathways avoids the problems of other methods, while making efficient use of the available screen space.

The real strength and novelty of the system lies in the ability to put views in context with each other, in a way no other tool can. The multi-layer line bundling allows to connect many entities while reducing visual clutter to a minimum. Combined with the arrangement and navigation in the bucket it allows to view relations of up to 5 views concurrently, where all displayed relations are still visible. The approach does not limit the visualization of relations to adjacent views only.

Future Work

The next step in the development for the parallel coordinates will be to improve the handling of many dimensions. The currently implemented approach of zooming and panning is sufficient for our current use cases. However, once the number of dimensions increases, some improvements have to be made. Probably the best way to achieve this is by clustering the data in a preprocessing step, in both dimensions. This allows improvements in the heat map and also in the parallel coordinates. We believe that a multi level approach following the overview first, zoom and filter paradigm by Shneiderman [Shneiderman1996] is the right way to handle large multidimensional datasets. For parallel coordinates and heat maps, this means, that the first step should be to present a clustered view, but also provide access to the concrete data, by separately visualizing the content constituting the cluster. The bucket is well suited to support this paradigm by allowing to visualize the relations among abstract and concrete views.

For the analysis of gene expression data, it is also of the utmost importance to bring clinical data (data about the subject of an experiment), such as age, sex, disease free survival, etc. into the picture. Only such data can avoid wrong conclusions, or allow to find otherwise hidden patterns. Caleydo is designed to integrate such data, but it has not been done
yet. Parallel coordinates are an excellent tool for the analysis of clinical data. Bringing them into visible relations with the gene expression data may improve the understanding of biological processes.

For the visual links several challenging fields for improvement can be identified. Currently we show one relation among different views (although elements can occur multiple times in one view). When trying to show more than one relation the problem of cluttering increases. Thus we have to develop methods to align and bundle connection lines in such a way that they are easy to follow on the one hand and not occluding other information on the other.

Possible solutions include “connection rings”, or the routing of the connection lines along the edges of the view. With a connection rings, several connections (less than 5) can be shown using connected, color coded rings around the borders of a view, from which connection lines to a selected element are drawn.

![Connection Rings Mockup](image)

**Figure 6.1:** Mockup for connection rings. Elements of one type in a view are connected to a colored ring around the central view.

Figure 6.1 shows a mockup for this approach. When drawing rings around the central view, every view has an edge close to the rings. Thus we can connect elements in a contextual view.
to the central ring, without introducing clutter in other views.

When the objective is to convey trends, not exact connections, the bundling of lines with density color coding, as discussed in [Holten2006] could improve the display.

Finally one could generalize the connection lines in Caleydo to map more relations. Currently we map identity relations. This means that we draw connection lines only to representations of the same element. While this is sufficient for many use cases in gene expression and pathway analysis, other visualization tasks demand a richer set of relations. For example similarity, hierarchical relations (is a subclass of), synonymy, or instantiation are relations that could be interesting to depict.
# List of Figures

1.1 DNA and its building blocks [Alberts2002]. ........................................ 10
1.2 DNA microarray [Alberts2002]. ......................................................... 12

2.1 Napoleons Russian campaign by C. Minard [Tufte1983]. ......................... 15
2.2 Paris metro system fisheye view [Sarkar1994]. ....................................... 16
2.3 Dynamic querying interface and details on demand in IVEE showing a movie database [Ahlberg1995]. ......................................................... 18
2.4 Multiple views in the Snap-Together Visualization [North2000]. ............... 19
2.5 A five-dimensional point represented in a parallel coordinate plot (Alfred Inselberg, available at [http://www.math.tau.ac.il/~aiisreal/]). .............. 20
2.6 A parallel coordinate browser showing the car dataset [Siirtola2006]. ........ 21
2.7 The crossing problem [Graham2003]. .................................................... 21
2.8 Using curves to avoid the crossing problem [Graham2003]. ....................... 22
2.9 The effects of reordering axes in parallel coordinates [Siirtola2006]. ........ 23
2.10 3D multi-relational parallel coordinates [Johansson2005]. ....................... 24
2.11 Higher order parallel coordinates showing the car dataset [Theisel2000]. .... 25
2.12 Parallel coordinates showing a dataset with 215 dimensions [Yang2003a]. .... 25
2.13 Dimension hierarchy browsing with InterRing, selection shown in parallel coordinates [Yang2003a]. ......................................................... 27
2.14 A parallel coordinate plot with 2000 data points in three dimensions. ......... 28
2.15 The sampling lens showing a 10% sample [Ellis2006]. ........................... 29
2.16 Clustered parallel coordinates plot with 230.000 datasets [Fua1999]. .......... 30
2.17 Outlier preserving clustered parallel coordinates [Novotny2006]. .............. 31
2.18 A range brush on the car dataset [Hauser2002]. ..................................... 32
2.19 Angular brushing [Hauser2002]. ........................................................ 33
2.20 Parallel Sets [Bendix2006]. ............................................................... 34
2.21 Gene-expression regulation analysis in 2D and 3D [Tuebel2006]. ............... 35
2.22 Clustered heat map with dendrogram [Eisen1998]. ................................ 38
2.23 The metabolic network as modeled by KEGG (available at [http://www.genome.jp/kegg/]). .......................................................... 39
2.24 Augmenting a KEGG pathway [Streit2008]. ......................................... 40
2.25 Network visualization by semantic substrates [Shneiderman2004]. ............. 42
2.26 The jukebox setup [Streit2008]. .......................................................... 43
2.27 VisLink: interconnections between independent visualizations [Collins2007]. 44
2.28 Hierarchical edge bundling [Holten2006]. .............................................. 45

3.1 Angular brushing in Caleydo. .............................................................. 47
3.2 View arrangement in the Link Bucket. ................................................... 49
3.3 Navigation in the Bucket. ..................................................................... 50
3.4 Visual linking edge bundling strategy ........................................ 51
4.1 Basic steps executed on Caleydo start-up .................................. 53
4.2 The Sets and Storages of a view ............................................. 54
4.3 The selection mechanism employed in Caleydo ....................... 55
4.4 The software design of OpenGL views in Caleydo ................... 57
4.5 The update merging mechanism in Caleydo ........................... 60
4.6 The software design of Storage-based OpenGL views in Caleydo .. 61
4.7 Connection Line Management in Caleydo .............................. 63
5.1 Parallel coordinates and info area ........................................... 64
5.2 Parallel coordinates with versus without transparency .......... 65
5.3 Parallel coordinates with different sampling rates .................. 66
5.4 Parallel coordinates with different brushes applied .............. 67
5.5 Parallel coordinates with moving, removing and duplicating of axes 68
5.6 Parallel coordinates showing axis - polyline exchange ........... 68
5.7 Heat map and selections with fisheye distortions ................... 69
5.8 The Link Bucket in a RCP window with an attached browser view 71
5.9 Differences between genes as polylines and genes as axes in the Link Bucket 72
5.10 Prior knowledge use case .................................................. 73
5.11 Data centered analysis use case ......................................... 75
6.1 Mockup for connection rings ............................................... 77
Bibliography


