Liquid Diagrams:

A Suite of Information Visualisation Gadgets

Dieter Ladenhauf

Liquid Diagrams:

A Suite of Information Visualisation Gadgets

Master's Thesis

at

Graz University of Technology

submitted by

Dieter Ladenhauf

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

17th January 2012

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

Liquid Diagrams:

Eine Sammlung von Applikationen zur Informationsvisualisierung

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Dieter Ladenhauf

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

17. Januar 2012

Diese Arbeit ist in englischer Sprache verfasst.

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

Abstract

The field of information visualisation is no longer only a discipline for academics, but also increasingly for a more general audience, who want to create computer-based visualisations of their own data. Liquid Diagrams addresses this wider audience by providing twelve highly interactive visualisations, which can be exported using high-quality raster (PNG) and vector graphics (SVG).

This thesis gives an overview of existing information visualisation solutions. In addition, it describes the modifications and improvements made to the Liquid Diagrams framework and outlines version 2.0, its current state. Enhancements over version 1.0 include extending the user interface, improving performance, introducing 3d effects, and refactoring the architecture of the framework. Two new variants, a standalone version and a gadget version using cookies, were also created.

Liquid Diagrams 2.0 contains the visualisations area chart, bar chart, line chart, pie chart, parallel coordinates, star plot, bat's wing diagram, polar area diagram, heatmap (choropleth map), treemap, Voronoi treemap, and similarity map. The similarity map uses a force-directed placement algorithm to place similar data entities close to one another. A special focus of this thesis is on 3d extensions to the pie chart and heatmap gadgets.

Kurzfassung

Das Fachgebiet der Informationsvisualisierung ist mittlerweile nicht nur allein eine Disziplin von Akademikern, sondern auch von einem breiten Publikum, das daran interessiert ist computerbasierte Visualisierungen aus den eigenen Daten zu erstellen. Liquid Diagrams adressiert dieses breite Publikum mittels zwölf in hohem Maße interaktiven Visualisierungen, die als qualitiv hochwertige Rastergrafiken (PNG) und als Vektorgrafiken (SVG) exportiert werden können.

Die vorliegende Arbeit gibt einen Überblick über bereits existierende Lösungen der Informationsvisualisierung. Zusätzlich werden die Veränderungen und Verbesserungen, die im Liquid Diagrams Framework gemacht wurden beschrieben und Version 2.0, der aktuelle Stand skizziert. Verbesserungen im Vergleich zu Version 1.0 beinhalten die Erweiterung des Userinterfaces, die Optimierung der Effizienz, die Einführung von 3-D Effekten und die Überarbeitung der Architektur des Frameworks. Zwei neue Varianten, eine eigenständige Version und eine Gadget Version, die Cookies verwendet wurden zusätzlich erstellt.

Liquid Diagrams 2.0 enthält die Visualisierungen Flächen Diagramm, Balken Diagramm, Linien Diagramm, Torten Diagramm, Paralelle Koordinaten, Star Plot, Bat's Wing Diagramm, Polar Area Diagramm, Heatmap (Choropleth Map), Treemap, Voronoi Treemap und Similarity Map. Die Similarity Map benutzt einen Force-Directed Placement Algorithmus, um ähnliche Datenelemente dicht beieinander zu platzieren. Ein spezieller Fokus dieser Arbeit liegt auf den 3-D Erweiterungen des Torten Diagramms und der Heatmap.

Statutory Declaration

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

Place	Date	Signature

Eidesstattliche Erklärung

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

Ort	Datum	Unterschrift

Contents

List of Figures

List of Tables

Acknowledgements

First of all, I want to thank my advisor Keith Andrews for giving me the opportunity to work on such an interesting project. In addition, I want to thank him for correcting the draft versions of this thesis.

I wish to thank my parents Alois Ladenhauf and Waltraud Sommer for constantly supporting me during my study. Without their support, this thesis would not exist. This is why I want to dedicate this work to them. Additionally, I want to thank:

- my sister Eva,
- my girlfriend Jenny,
- and my friends Admir, Johannes, Markus, Michi, Michi, Susi, and Volli

for their support. Last but not least, I wish to thank my friends Florian, Christian, Patrick, and Thomas from Graz University of Technology for their help and presence for successfully completing countless courses at the university.

Thank You.

Dieter Ladenhauf
Graz, Austria, January 2012

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews, 2011c].
- The photograph in Figure 2.13a is used with kind permission of Keith Andrews.
- Figure 6.9 is used with kind permission of Martin Lessacher [2010].

The following are used under the terms of the ACM Copyright Notice (see page xi):

- Figures 2.16a and 2.16b are extracted from the Journal ACM Transactions on Graphics, volume 11 [Shneiderman, 1992].
- Figure 8.7 is adapted from Carlbom and Paciorek [1979], published in ACM Computing Surveys, volume 11.

ACM Copyright Notice

Chapter 1

Introduction

“Data expands to fill the space available for storage”. This corollary of Parkinson’s law (“Work expands so as to fill the time available for its completion” [Parkinson, 1955]) is indicative of the massive amount of data produced every day. According to Hilbert and López [2011], the amount of data created in the world in 2000 was 54.5 exabytes (54.5 * 10^{18} bytes). In 2007 on the contrary, it increased to 295 exabytes. This amount could be stored using 404 billion CD-ROMs. Piling up the CD-ROMs would result in a stack with a height of $1\,\frac{1}{4}$ times the distance from the earth to the moon.

This explosion of data makes the derivation of information a highly difficult task. The field of information visualisation attempts to simplify this procedure by taking the human visual perception system into account. Under the right conditions, a visual representation of data can be more meaningful than the data contained within a table. Tufte [2001, pages 13–14] uses Anscombe’s quartet [Anscombe, 1973] (shown in Table 1.1) to emphasise this statement. The data of all four datasets in Anscombe’s quartet have equal statistical properties (mean and standard deviation), but if they are plotted (seen in Figure 1.1), an entirely different perception is established.

The use of the field of information visualisation is no longer limited to academics, due to the fact that the interest of the general public is constantly expanding. Nowadays, people want to visualise their own data and share it with others, in order to discuss and collaboratively explore the data. Therefore, appropriate visualisation tools have to be developed. Liquid Diagrams aims to satisfy some of this need by providing twelve highly interactive visualisations. The underlying Liquid Diagrams framework was implemented by Martin Lessacher [2009] in 2009 and extended in 2010 [Andrews and Lessacher, 2010; Lessacher, 2010] under the supervision of Keith Andrews.

This thesis is about the further development of the Liquid Diagrams framework. Chapter 2 describes the field of information visualisation and gives examples of its history and its corresponding visualisations. Currently existing information visualisation solutions are discussed in Chapter 4. Here, a distinction between online services, standalone software, and libraries and toolkits is made. The focus of Chapter 3 is on technologies such as HTML, JavaScript, Flash, and Java which can be used to build information visualisation systems. Chapter 5 describes version 2.0, the current state of the Liquid Diagrams framework and points out improvements and modifications made during the present work. The visualisations of the framework are introduced in Chapter 6 and specific changes made to the framework are discussed in Chapter 7. The technical aspect of the implementation is explained in Chapter 8, Chapter 9 gives an outlook and lists possible future extensions and finally, Chapter 10 concludes this thesis.

Statistical Properties	**I**		**II**		**III**		**IV**	
	X	Y	X	Y	X	Y	X	Y
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.1	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.1	4.0	5.39	19.0	12.5
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
Mean	9	7.50	9	7.50	9	7.50	9	7.50
Standard Deviation	10	2.03	10	2.03	10	2.03	10	2.03

Table 1.1: Anscombe's quartet [Anscombe, 1973]. The four datasets have identical statistical properties, mean and standard deviation. How they differ is only revealed by plotting the data, as seen in Figure 1.1.

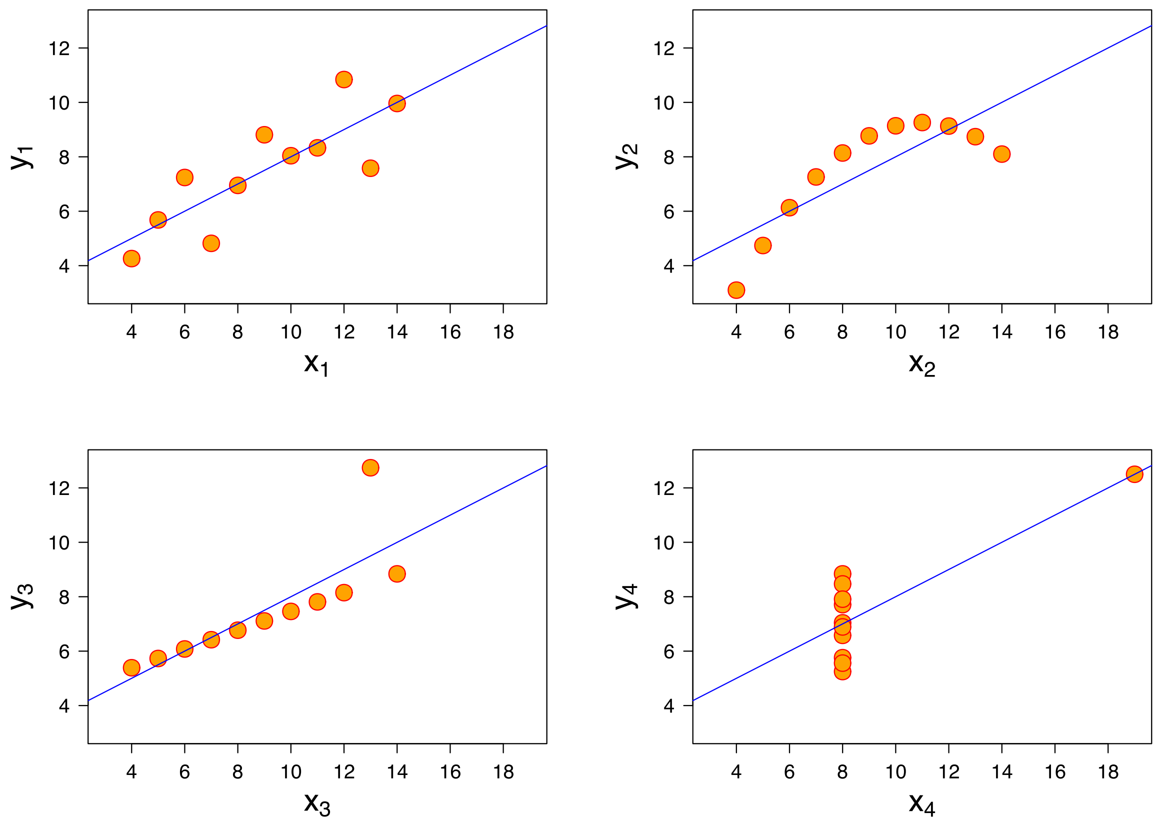


Figure 1.1: A plot of Anscombe's quartet, shown in Table 1.1. Here, the difference between the four datasets can be easily seen. This image is extracted from Wikimedia Commons [Wikipedia, 2011e].

Chapter 2

Information Visualisation

> “ *Graphics* reveal *data. Indeed graphics can be more precise and revealing than conventional statistical computations.* ”
>
> [Edward R. Tufte, The Visual Display of Quantitative Information, 2001.]

2.1 Origins and Historical Examples

According to Tufte [2001, page 32], one of the inventors of modern graphical designs is the political economist William Playfair. Many modern visualisations such as line charts (see Section 2.3.1), bar charts (discussed in Section 2.3.2), and pie charts (see Section 2.3.3) date back to Playfair [1786, 1801]. Florence Nightingale, a British nurse invented the bat's wing diagram (introduced in Section 2.3.8), and polar area diagrams (see Section 2.3.9) in order to visualise how death rates during the Crimean War (1853–1856) were reduced because of improvements in hygiene.

A popular historical example is the cholera map by John Snow, shown in Figure 2.1. According to Tufte [1997, pages 27–37], an outbreak of cholera happened in the Broad Street area of central London on August 31, 1854. Dr. John Snow was able to discover the cause of the outbreak, a water pump in Broad Street and showed that cholera is in fact a disease transmitted by water. Snow created a dot map of central London, displaying each individual death. Tufte [1997] states that Snow was able to identify the cause of the outbreak using this map. However, more recent research has found that Snow's map was drawn *after* the cause of the outbreak was determined [Brody et al., 2000].

Tufte [2001, pages 40–41] states that Charles Joseph Minard showed the first graphic with added spatial dimensions such as time and temperature. In 1896 Minard visualised the march of Napoleon's Russian campaign of 1812, using the graphic shown in Figure 2.2. It displays the fate of Napoleon's army, being decimated from 422,000 to 10,000 solders. According to Tufte [2001, page 40], this graphic may be the best statistical diagram ever drawn.

Jacques Bertin tried to identify all graphical elements which can be used to express data in his book “Semiology of Graphics” Bertin [1967, 1983]. Semiology, also known as semiotics is the study of symbols and their relations. Bertin also built a mechanical matrix permutation device called Domino. He showed that the understanding of a matrix can be improved dramatically by permuting its rows and columns [Henry, 2008, page 78].

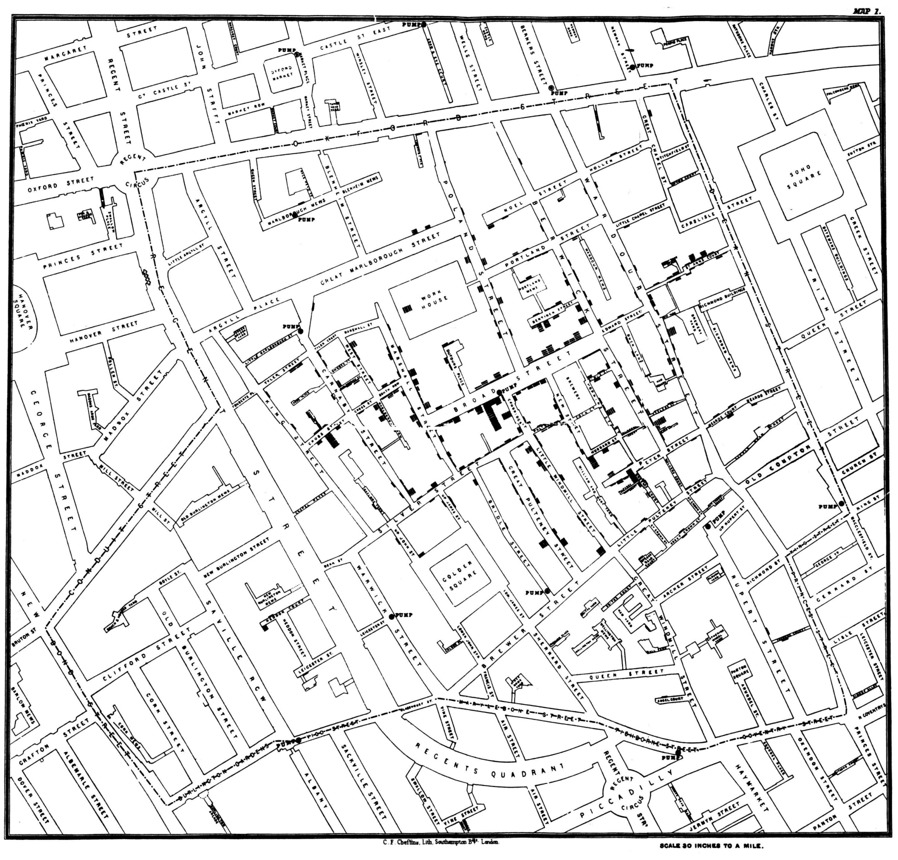


Figure 2.1: John Snow plotted the deaths of the cholera outbreak in London in 1854 on a map. Each death is expressed as a small black bar. This image is extracted from Wikimedia Commons [Wikipedia, 2011b].

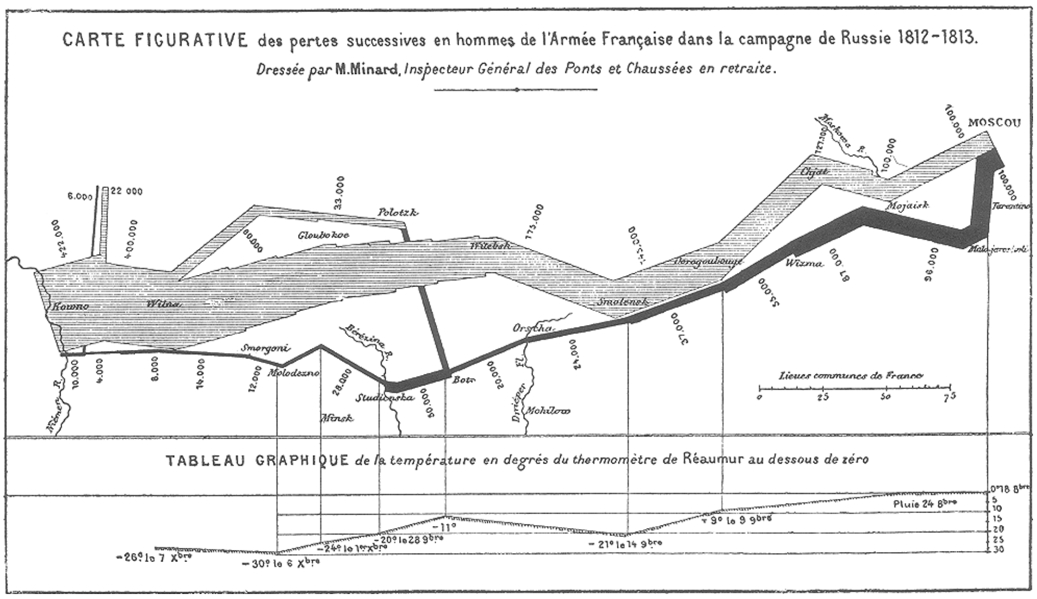


Figure 2.2: Minard's map displaying the decimation of solders in Napoleon's Russian campaign in 1812. The amount of solders is represented by the widths of the coloured areas. The march to Moscow is expressed with a grey shading and the return path of the rest of the army is shaded black. At the bottom, a line indicates the fall of the temperature after the army left Moscow. This image is taken from Wikimedia Commons [Wikipedia, 2011c].

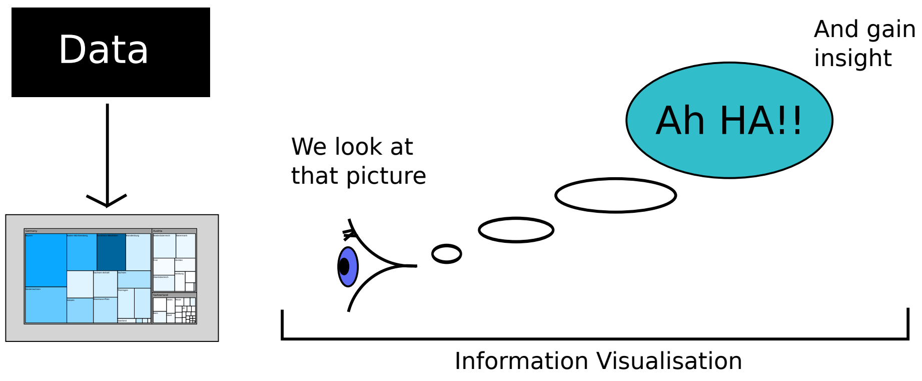


Figure 2.3: Spence's process of information visualisation. This figure is adapted from Spence [2007, page 5].

2.2 Definition

[Spence, 2007, page 5] defines the term "visualisation" by citing a dictionary and concludes that visualisation is a cognitive activity human beings engage in, following the approach of Ware [2004]. Spence [2007, page 5] emphasises that this activity has nothing to do with computers and need not involve a visual experience, because of the presence of other sensory influences such as sound. Nevertheless, Ward et al. [2010, page 3] states that sight is one of the key senses of information understanding. The activity of information visualisation is explained by Spence [2007, page 5] using Figure 2.3. It illustrates the moment when a graphical explanation causes a "Ah HA!" reaction, describing the act of finally understanding the visual representation. Mazza [2009, page 8] states that the term "Information Visualisation" was coined at the end of the 1980s by researchers at Xerox PARC, and was intended to distinguish the new field of creating visual solutions by considering the human cognition.

According to Ware [2004, page 2], the role of visualisations is important in cognitive systems. The human brain has the remarkable ability to provide an adaptive pattern-finding mechanism, coupled with a flexible decision-making apparatus. It is accompanied by almost unlimited information resources such as the World Wide Web, provided by computer systems. Interactive visualisations play an important role in building the interface between those two individual systems. Ware [2004, page 2] states that the performance of the entire system can be enhanced by improving these interfaces. A simplified model of the human visual perception system, using three stages, is described by Ware [2004, pages 20–22]:

Stage 1: Billions of neurons in the eye and the primary visual cortex at the back of the brain simultaneously extract features of the visual field. This are features such as orientation, colour, texture, and movement.

Stage 2: Here, the visual field is divided into regions and simple patterns such as regions sharing the same texture or colour. This process involves both working memory and long-term memory and is performed using slow serial processing.

Stage 3: In this stage, only a few objects at a time are held in the visual working memory. A sequence of visual queries can be triggered in order to be answered using visual search strategies. An example for this procedure would be the task of finding a route on a map. Here, the a search for thick, connected lines, representing streets will be triggered.

According to Ware [2004, pages 149–154], certain simple shapes or colours can be identified in a short amount of time because they "pop out" from their environment. The mechanism behind this effect

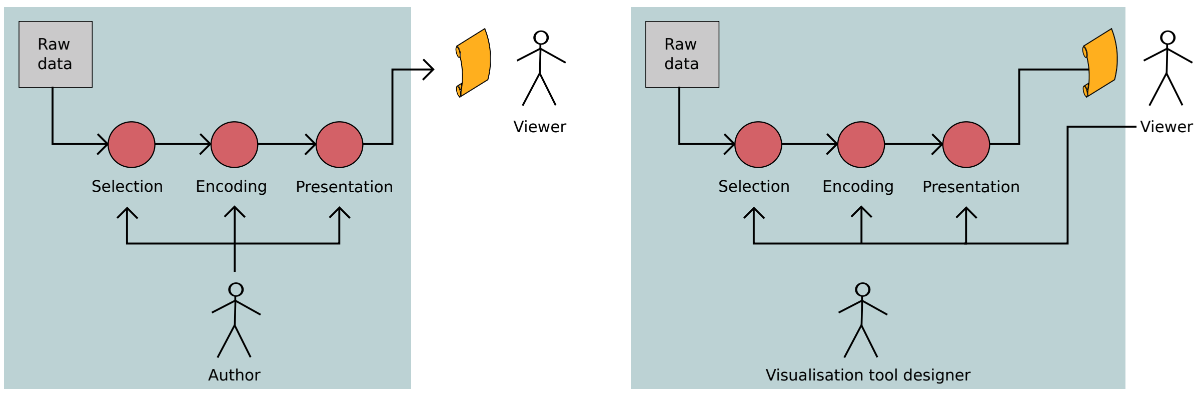
```
890546850695068182901280301067
806876097650978650961200960876
549675968538596854996885976191
899271973054860950586597897970
```

(a) To count the number of occurrences of the digit 3, it is necessary to scan all lines sequentially.

```
890546850695068182901280301067
806876097650978650961200960876
549675968538596854996885976191
899271973054860950586597897970
```

(b) Here, the 3s can be found immediately by noticing the red digits. Colour can be preattentively processed.

Figure 2.4: An example of preattentive processing, adapted from Ware [2004, page 150].



(a) The creation of an image in the pre-computer age. **(b)** The creation and use of an interactive visualisation system.

Figure 2.5: The creation process of a visualisation in the (a) pre-computer and (b) post-computer ages, adapted from Spence [2007].

is called *preattentive processing*. It specifies which visual elements immediately attract one's attention. Figure 2.4 illustrates preattentive processing. In order to find all 3s in Figure 2.4a, it is necessary to sequentially scan all lines, whereas the 3s can be immediately identified in Figure 2.4b, because here, only the red digits have to be scanned. These "pop out" features, which are preattentively processed, fall into the categories colour, form, motion, and spatial position.

Spence [2007, page 12] argues that information visualisation is related to the field of scientific visualisation, which visualises data related to physical entities such as air flowing around the wing of an aeroplane. Information visualisation on the contrary, is based on abstract information structures, and therefore appropriate visualisations have to be invented [Andrews, 2011b]. There is no natural geometry inherent in the data.

According to Spence [2005], the creation process of a visualisation has changed significantly in recent years. This process is visualised in Figure 2.5. In the pre-computer age, an author created an image, performing the tasks selection, representation, and presentation of content. The resulting image was then presented to a viewer, an entirely different person. Now on the contrary, powerful computers enable the creation of interactive visualisation systems, controllable by users. The user is here in some extent the author, using interaction techniques to modify the resulting presentation. Andrews [2011b] states that interaction support is just as important as the underlying visual representation.

Seven basic tasks of user interaction in information visualisation systems are identified by Ben Shneiderman [1996]:

- **Overview:** Gain an overview of the entire collection.
- **Zoom:** Zoom in on items of interest.
- **Filter:** Filter out uninteresting items.
- **Details-on-demand:** Select an item or group and get details when needed.
- **Relate:** View relationships among items.
- **History:** Keep a history of actions to support undo, replay, and progressive refinement.
- **Extract:** Allow extraction of sub-collections and of the query parameters.

The first four of them ("Overview first, zoom and filter, then details-on-demand") are known as Shneiderman's "Visual Information Seeking Mantra".

Tufte [2001, page 51] lists several principles of graphical excellence, a set of guidelines for creating effective graphical representations. According to Tufte, a good graphic is well-designed, represents interesting data, and combines substance, statistics, and design. It is an efficient, precise, and clear representation of a complex idea and tells the truth about the data. "Graphical excellence is that which gives to the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space" [Tufte, 2001, page 51].

2.3 Visualisations

There are many different types of visualisations. For example, Steele and Iliinsky [2011, pages 4–7] make a distinction between the terms *infographics* and *data visualisations* (or *information visualisations*):

- An infographic is a manually drawn, aesthetically rich visual representation of data. Since it is drawn manually, the amount of data contained in the graphic tends to be limited. In addition, an infographic is difficult to change or update and it is hard to recreate with different data.
- Data visualisations and information visualisations on the contrary, are algorithmically drawn graphics, which are easy to recreate with different data. Normally, these visualisations are initially drawn by a human, before their drawing process is automated using an algorithm.

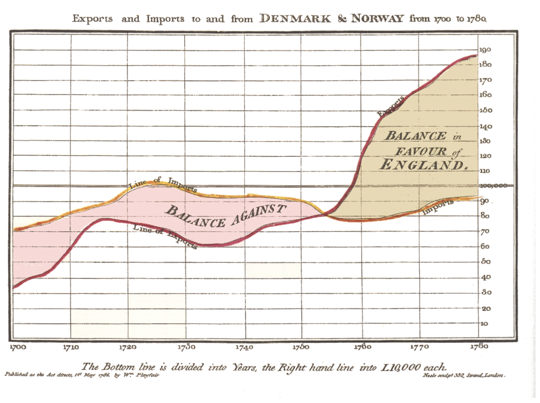
This section describes visualisations contained in the latter category.

2.3.1 Line Chart

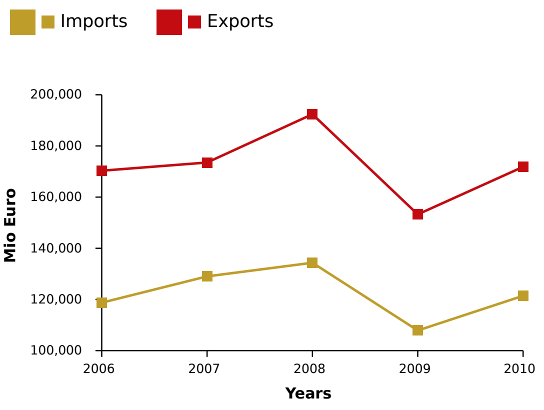
A *line chart* is a visualisation in which data entities are expressed as lines, connecting data points of the corresponding entity. Line charts are used to display trends over time, which is why they are commonly used for financial data. The fist line chart, shown in Figure 2.6a, was created by William Playfair [1786] in 1786. It displays the imports and exports to and from Norway and Denmark from 1700 to 1780. A modern version of a line chart can be seen in Figure 2.6b.

2.3.2 Bar Chart

A data entity in a *bar chart* is expressed by vertical or horizontal bars, in which the height, or in the case of horizontal bars the length, indicates the value of the data entity. Bar charts simplify the comparison of multiple data entities. The first bar chart was also created by William Playfair [1786] in 1786. Figure 2.7a shows Playfair's first bar chart, illustrating the imports and exports of Scotland. Figure 2.7b shows the exports of Scotland in 2009.

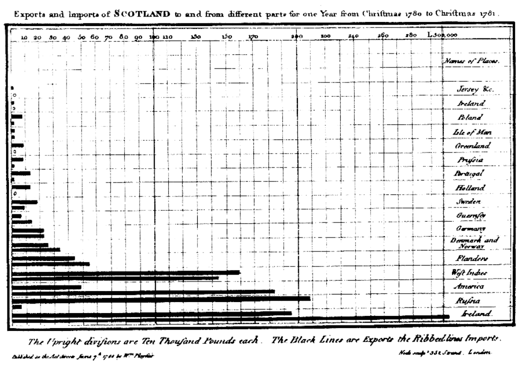


(a) The first line chart by Playfair [1786] showing the exports and imports to and from Denmark and Norway from 1700 to 1780. (This image is extracted from Wikimedia Commons [Wikipedia, 2011f]).

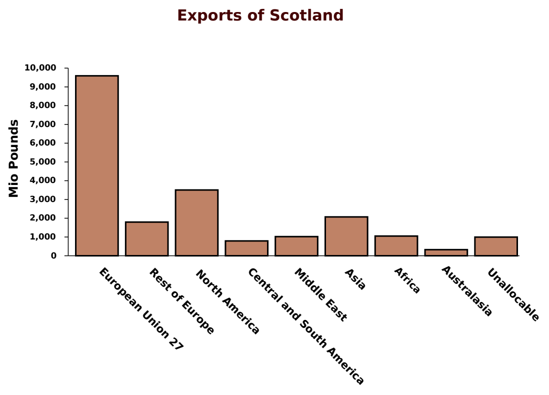


(b) This graphic shows a modern line chart. It displays the exports and imports of Denmark and Norway from 2006 to 2010. It was produced using data from EC [2011] (given in Euro) and DST [2011] (given in Danish crown). The numbers were calculated using the currency rate of one Danish crown equalling 0.134442778 Euros.

Figure 2.6: Two examples of line charts. (a) shows William Playfair's first hand-drawn line chart [Playfair, 1786] and (b) displays a modern computer-generated line chart.

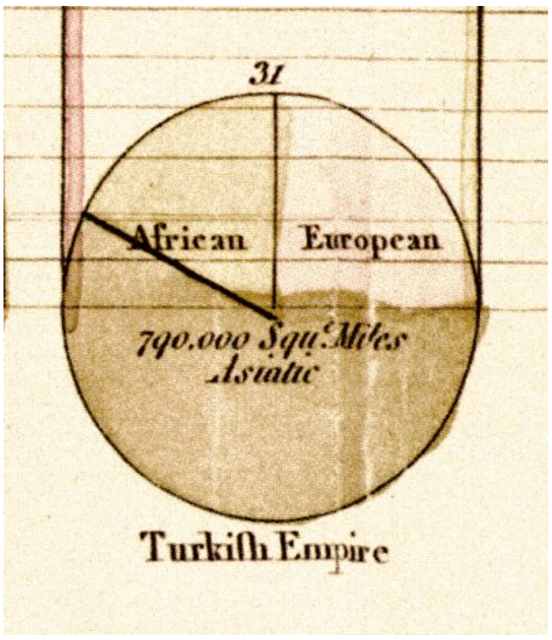


(a) William Playfair's first bar chart displaying the imports and exports of Scotland [Playfair, 1786]. (The image is taken from Wikimedia Commons [Wikipedia, 2011g].)

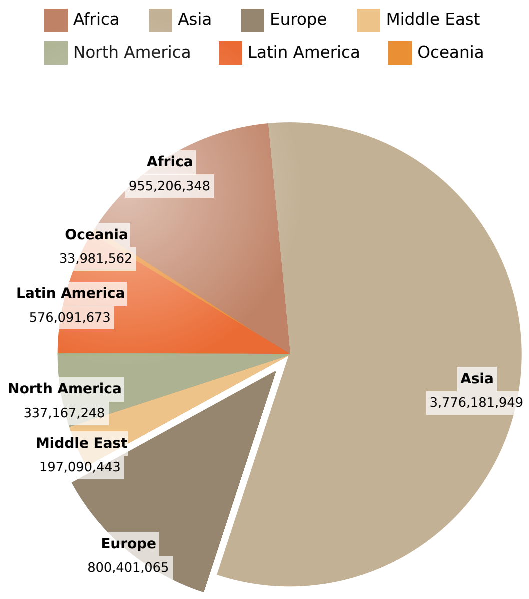


(b) A modern bar chart visualising the exports of Scotland. The data for this example was extracted from the Global Connections Survey [GCS, 2011].

Figure 2.7: Two examples of bar charts: (a) shows the first bar chart by Playfair [1786], (b) shows a modern bar chart.



(a) One of William Playfair's first pie charts [Playfair, 1801], showing the proportion of the Asiatic, European, and African land masses of the Turkish Empire [Spence, 2005]. (This image is extracted from Wikimedia Commons [Wikipedia, 2011d].)



(b) A modern pie chart displaying the world's population by region.

Figure 2.8: Pie charts (a) original by William Playfair [1801] and (b) a modern computer-generated version.

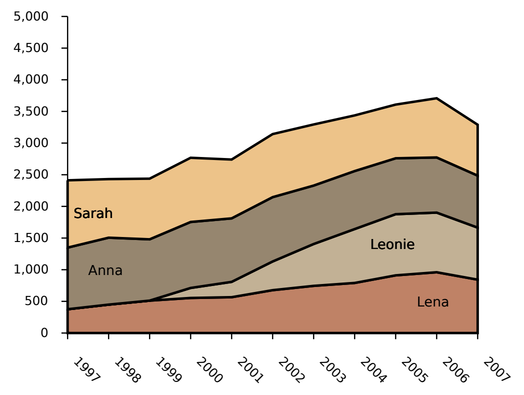
2.3.3 Pie Chart

A *pie chart* uses a circle divided into sectors. The size of each sector expresses the proportional significance of the corresponding data entity. Pie charts are a popular chart type. However, there are disadvantages of pie charts. If the sectors of a pie chart are of roughly equal size, it is difficult to distinguish them. In addition, a comparison of data across different pie charts is highly problematic. There are many variants of pie charts, such as doughnut charts, three-dimensional pie charts, and multi-level pie charts. Figure 2.8 shows two examples of pie charts.

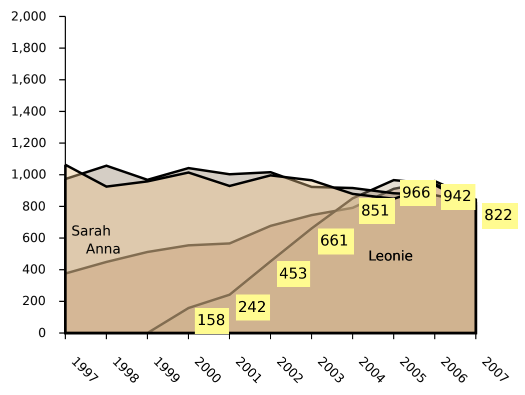
The pie chart also dates back to William Playfair [1801]. Playfair used pie charts as a part of larger figures, illustrating the areas, populations, and revenues of European states [Spence, 2005]. One of the first pie charts, displaying the Asiatic, European, and African land masses of the Turkish Empire is shown in Figure 2.8a.

2.3.4 Area Chart

Area charts are similar to line charts and are also used to display trends over time. Like in line charts, data points of a corresponding data entity are connected with a line. However, the difference between area charts and line charts is that the area beneath a line in the area chart is filled. There are several variants of area charts such as stacked area charts and overlay area charts. They are illustrated in Figure 2.9.



(a) A stacked area chart visualising the number of occurrences of newborn forenames from 1997 to 2007. The areas are drawn on top of each other.



(b) An overlay area chart displaying the same data. Here, the areas overlap each other. For visibility purposes the areas are drawn using transparency.

Figure 2.9: Area charts. (a) a stacked area chart and (b) an overlay area chart.

2.3.5 Star Plot

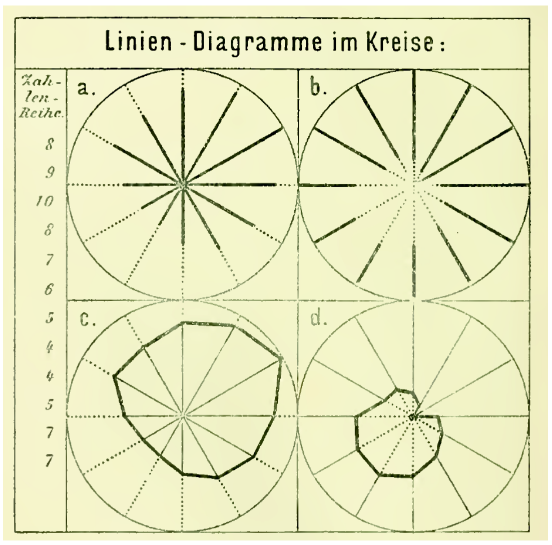
Star plots are used to display multivariate data. A star plot consists of axes which are equally spaced around a circle, originating from the centre. Each of the axes represents a different variable, ranging from the smallest to the largest value. The specific value of a data entity is expressed as the length from the centre of the circle to a point on the axis belonging the corresponding variable. The name star plot comes from the circumstance that the values of neighbouring axes are sometimes connected by a line, resulting in a star-like shape. Star plots date back to Georg von Mayr [1877] and the first ever star plot can be seen in Figure 2.10a. A modern version of a star plot, visualising the same data is shown in Figure 2.10b.

The lowest value of an axis is typically located in the centre. However, star plots are most useful when the axes are created so that good values are on the outside regardless of whether high or low means good. For example, Chambers et al. [1983] used star plots to visualise the well-known cars dataset, in which the axis displaying the price of a car was inverted in order to make sure that lower prices are found on the outside of the plot (are considered good). The cars in this example, adapted in Figure 2.10d, are presented using a star plot matrix, multiple small star plots, one for each car. The corresponding legend, showing the assignment of variables for the displayed axes can be seen in Figure 2.10c.

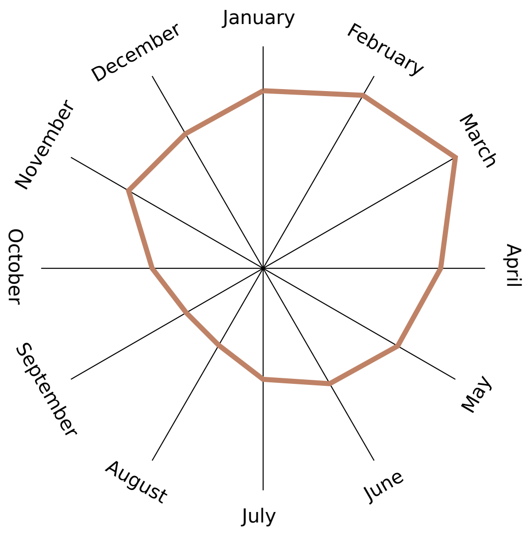
2.3.6 Parallel Coordinates

Parallel coordinates are a visualisation method for displaying multidimensional data. They consist of several vertical axes, each representing one data dimension. Here, the ends of the axes represent the lowest and the highest values of the corresponding dimensions. A data entity consists of several data points, each located on one of the axes. Its position is determined in relation to the lowest and highest value of the corresponding dimension. An entity is visualised by connecting each data point on neighbouring axes. An example of parallel coordinates can be seen in Figure 2.11.

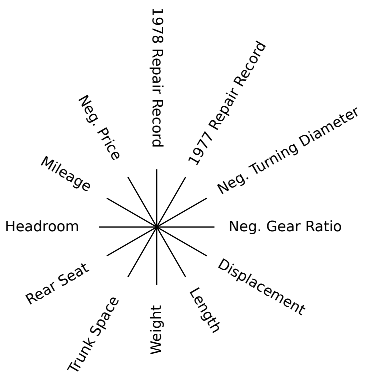
Parallel coordinates date back to Maurice d'Ocagne [1885] in 1885. Later, they were independently re-invented by Alfred Inselberg in 1959 [Inselberg, 1985, 2009]. Parallel coordinates allow the comparison of data entities and dimensions, and are useful for detecting patterns in the data.



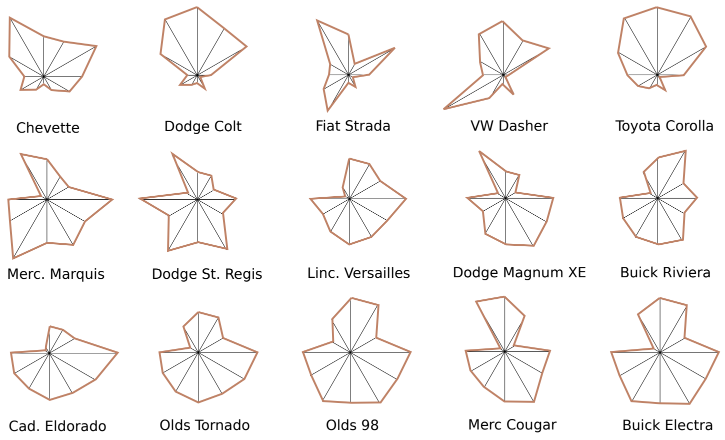
(a) "Linien-Diagramm im Kreise", the first star plot from Mayr [1877] showing four different variants, including an inverted star plot.



(b) A modern star plot visualising the same data used in (a).



(c) Legend: the assignment of the dimensions (such as length, weight, and mileage) of the star plots in (d).



(d) A star plot matrix, adapted from Chambers et al. [1983], visualising the well-known cars dataset. Here, the cars are represented by multiple small star plots, each for one car, arranged next to each other. The assignment of the used dimensions can be seen in (c).

Figure 2.10: Star plots. The first ever star plot by Mayr [1877] is shown in (a) and a modern version of the same star plot in (b). (d) displays star plots visualising high-dimensional data. The corresponding assignment of the used dimensions can be seen in (c). The images shown in (c), and (d) are adapted from Chambers et al. [1983].

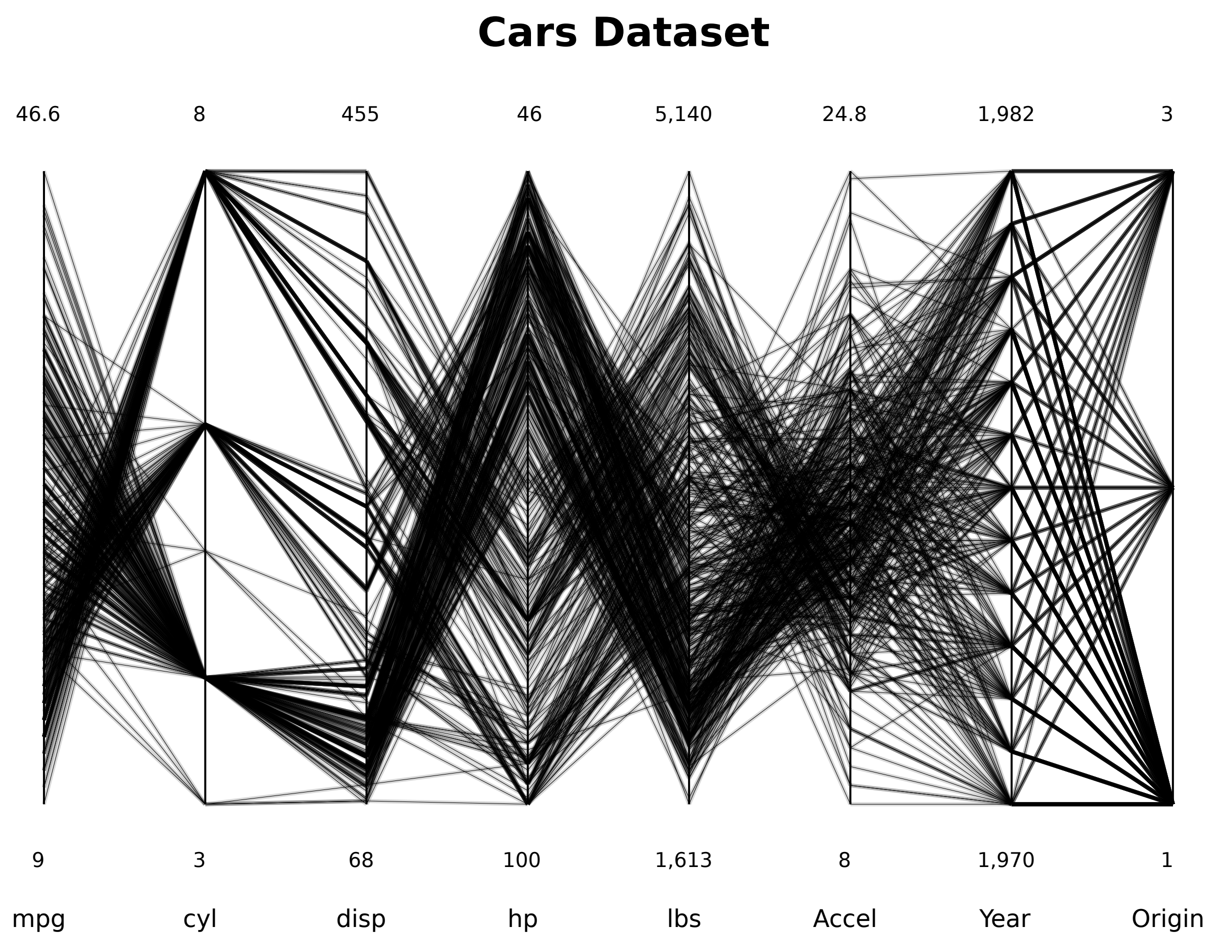


Figure 2.11: The cars dataset visualised with parallel coordinates. Each data dimension is represented by a vertical axis, ranging from the lowest to the highest value of the dimension. Neighbouring data points of a entity (car) are connected. Cars with similar characteristics are represented by polylines with similar shapes. The cars dataset was created by Ramos and Donoho [2011].

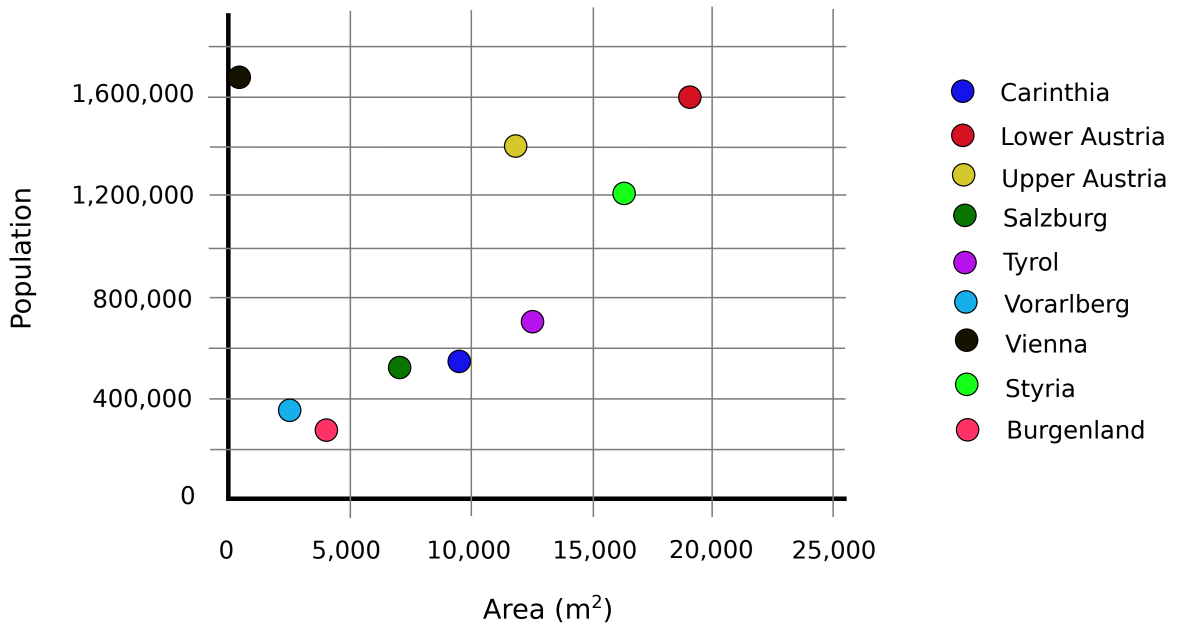


Figure 2.12: The regions of Austria visualised with a scatterplot. The x-axis shows the area and the y-axis the population of the displayed regions. Using a scatterplot, patterns or outliners (such as Vienna in this figure) can be easily spotted.

2.3.7 Scatterplot

A scatterplot is a diagram, displaying two different, independent variables by plotting data entities onto a coordinate grid defined by a vertical and a horizontal axis. According to Friendly and Denis [2005] "the humble scatterplot may be considered the most versatile, polymorphic, and generally useful invention in the entire history of statistical graphics". A scatterplot is a highly intuitive visualisation method, useful to reveal relationships between two variables or to detect patterns in the data. An example of a scatterplot can be seen in Figure 2.12.

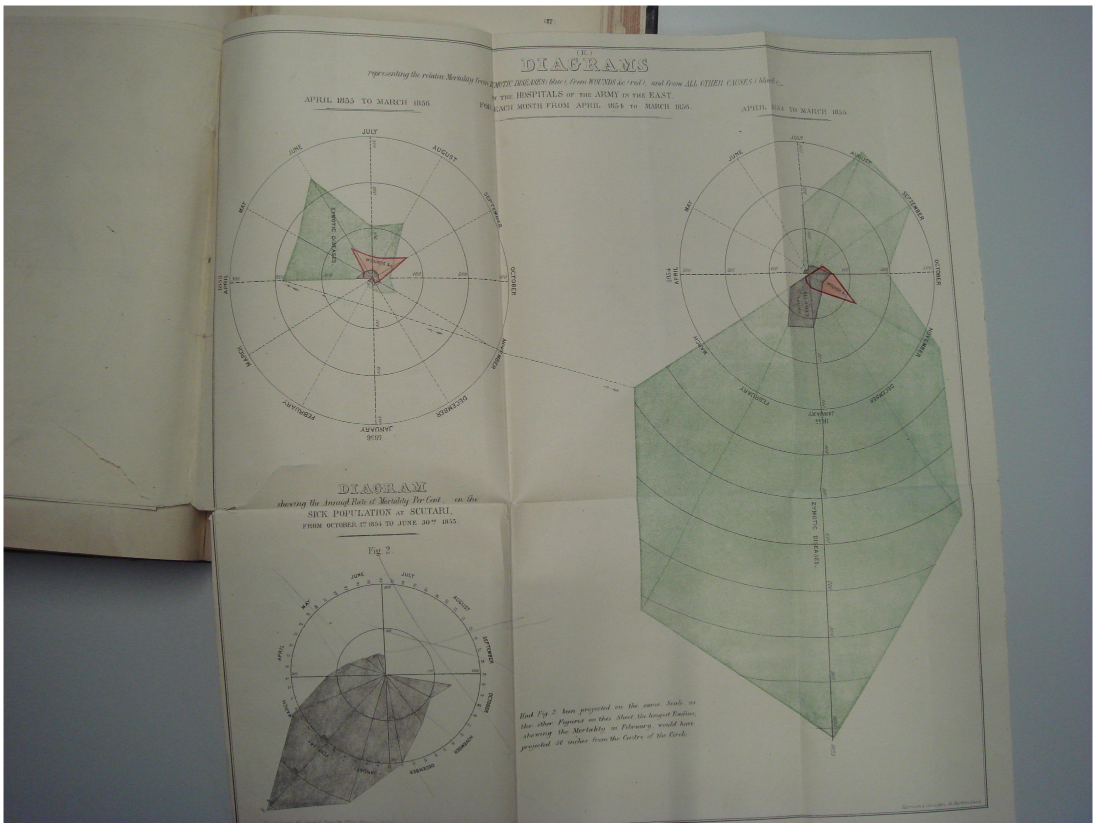
2.3.8 Bat's Wing Diagram

The *bat's wing diagram* dates back to Florence Nightingale, an English nurse, who invented it in 1858. According to Small [1998], Nightingale used the bat's wing diagram to display the death rate from causes such as wounds and disease during the Crimean War (1853–1856). The diagrams visualised how the death rates were reduced because of improvements in hygiene.

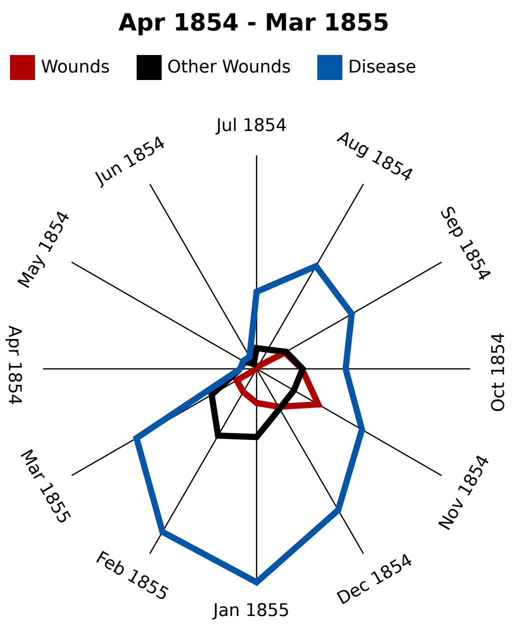
The bat's wing diagram consists of twelve axes, originating from the same center. Each axis clockwise represents one month. A data value is represented by the length of a radial line, starting at the center. The ends of the radial lines on neighbouring axes are connected. This leads to a major disadvantage of the bat's wing diagram: the assumption that the data is expressed using areas [Small, 1998]. This is the reason why Nightingale replaced the bat's wing diagram with the polar area diagram (see Section 2.3.9) in later publications. A modern version of the original diagram can be seen in Figure 2.13.

2.3.9 Polar Area Diagram

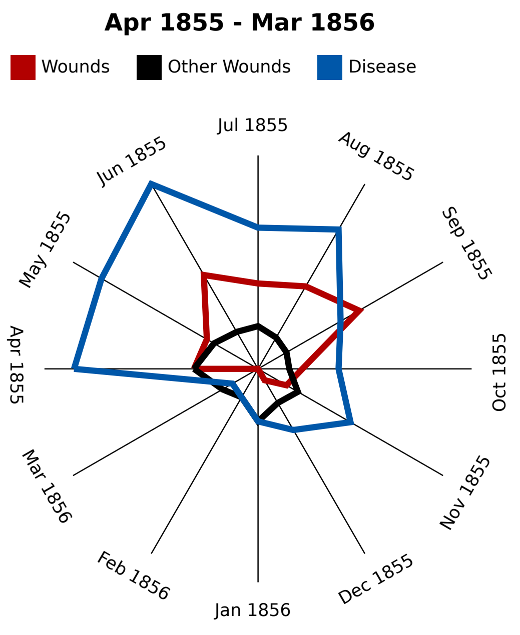
The *polar area diagram* (or wedge diagram) is Florence Nightingale's improvement of the bat's wing diagram, addressing the problem mentioned in Section 2.3.8. The data of a polar area diagram is expressed using its areas. Like the bats's wing diagram it is divided into twelve equal sectors, each representing



(a) Florence Nightingale's bat's wing diagram from 1858. Photograph copyright Keith Andrews, used with kind permission.



(b) The number of deaths from various causes from April 1854 to March 1855.



(c) The death rate from various causes from April 1855 to March 1856. The number of deaths from disease decreased significantly during this timespan.

Figure 2.13: Bat's wing diagrams. (a) displays Florence Nightingale's original bat's wing diagram and (b) and (c) show modern, computer-generated bat's wing diagrams. The diagrams display the death rate from three different causes during the Crimean War (1853–1856).

one month. A data value is drawn as a wedge, reaching from the start to the end of a sector. Here, the area of the wedge is proportional to the data value expressed by it. Polar area diagrams are often incorrectly referred to as "coxcomb" diagrams [Small, 1998]. The original polar area diagram can be seen in Figure 2.14a and modern versions are shown in 2.14b and 2.14c.

2.3.10 Geographic Heatmap

A *heatmap* is a graphical representation of data in which coloured, two-dimensional areas are used to express the value for each data entity. A *geographic heatmap*, also known as *choropleth* map, is a heatmap with cartographic areas. An area of a heatmap is shaded in proportion to the data value assigned to it. Choropleth maps are designed to simplify the analysis of geographic data such as statistical or economic data. The name choropleth map was introduced by the american geographer John Wright [1938] in 1938.

According to Friendly and Denis [2011], the first choropleth map was by Baron Pierre Charles Dupin in 1826. The map, which can be seen in Figure 2.15a, shows the distribution and intensity of illiteracy in France [Dupin, 1826]. Three years later, Balbi and Guerry [1829] created the first comparative choropleth maps, several choropleth maps next to each other, each with a different variable [Friendly, 2007]. This enables the viewer to perform a direct comparison of the displayed variables. The maps illustrated crimes against persons and against property, in relation to the level of education by departments in France. They are shown in Figure 2.15b.

A prism map is a special version of a choropleth map. In addition to the shading, the cartographic areas are extruded, the height of the extrusion indicating a second variable. In this way, two variables can be displayed using the same map.

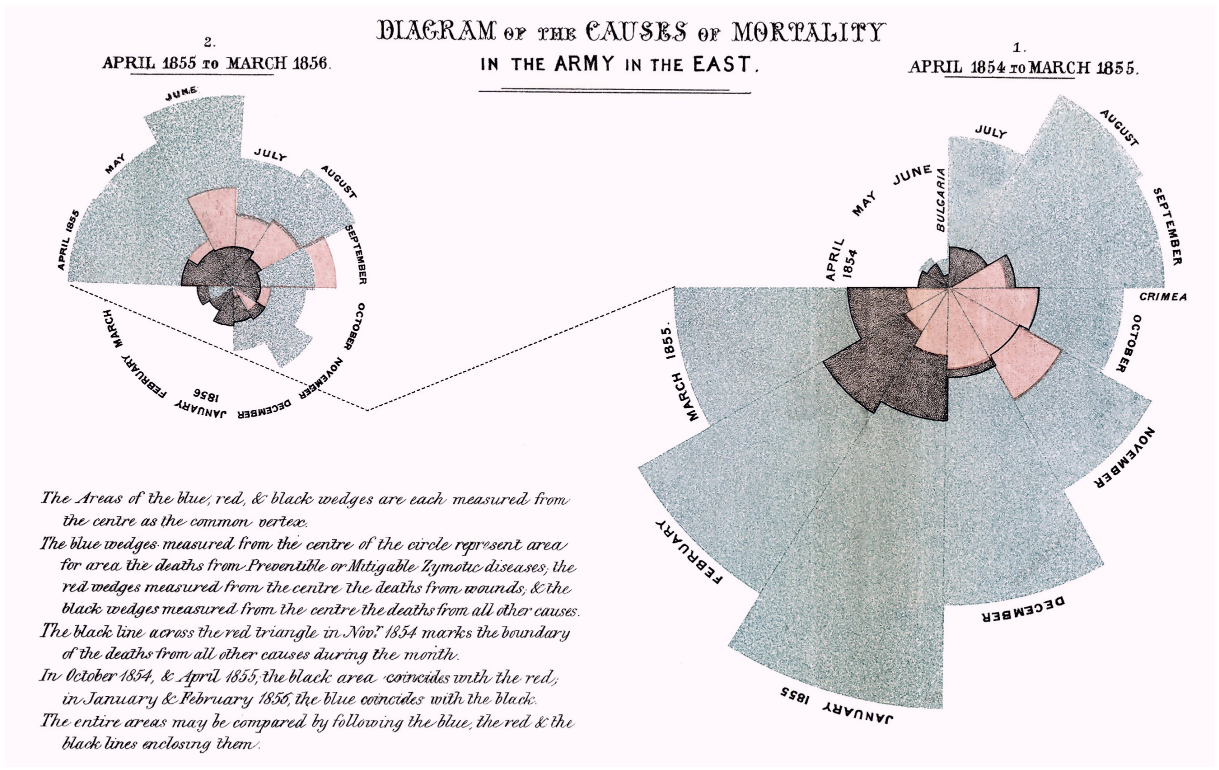
2.3.11 Treemap

Treemaps were invented by Ben Shneiderman in 1991, to visualise the space on hard disks, in order to find large files that could be deleted and to display the amount of shared space used per person [Shneiderman, 2009]. The algorithm in its original version [Shneiderman, 1992] was published in 1992. A treemap uses a two-dimensional space filling approach for visualising hierarchical tree structures [Johnson and Shneiderman, 1991]. It iteratively divides the available space into nested rectangles, in which the elements contained within a hierarchy level share the space of their parent. Figure 2.16 shows two representation methods of the same hierarchy, a traditional node-link drawing and a treemap.

2.3.12 Voronoi Treemap

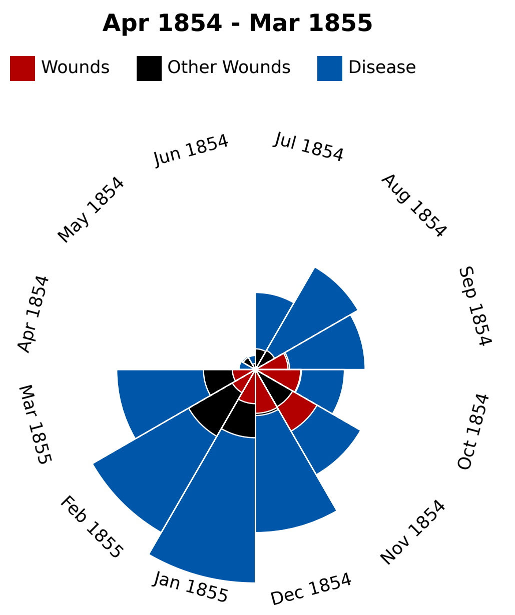
A Voronoi diagram uses a Voronoi tesselation to partition the available space by applying the nearest-neighbour rule [Aichholzer and Aurenhammer, 2002]. The first person to introduce the concept is believed to be George Voronoi, a Russian mathematician. René Descartes [1644] published the first Voronoi-like diagram in 1644, displaying the arrangement of matter in the solar system.

The first implementation of Voronoi diagrams in the context of information visualisation, the InfoSky visual explorer [Andrews et al., 2002; Granitzer et al., 2004] was performed in 2002. InfoSky, shown in Figure 2.17 enables the interactive exploration of large, hierarchically structured document collections. It uses modified, weighted Voronoi diagrams for area partitioning, in which the size of each polygon is related to the number of documents in the corresponding collection. A demo version of InfoSky can be obtained from the Know-Center [2011].

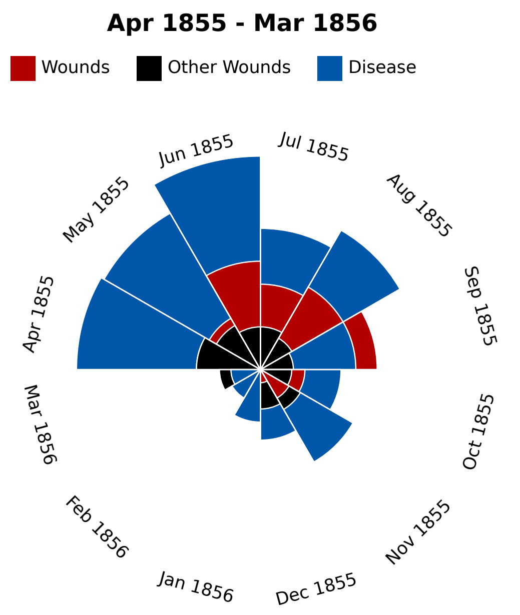


The Areas of the blue, red, & black wedges are each measured from the centre as the common vertex.
The blue wedges measured from the centre of the circle represent area for area the deaths from Preventible or Mitigable Zymotic diseases; the red wedges measured from the centre the deaths from wounds; & the black wedges measured from the centre the deaths from all other causes.
The black line across the red triangle in Nov.r 1854 marks the boundary of the deaths from all other causes during the month.
In October 1854, & April 1855, the black area coincides with the red; in January & February 1856, the blue coincides with the black.
The entire areas may be compared by following the blue, the red & the black lines enclosing them.

(a) Florence Nightingale's polar area diagram from 1858. (Image from Wikimedia Commons [Wikipedia, 2011a])

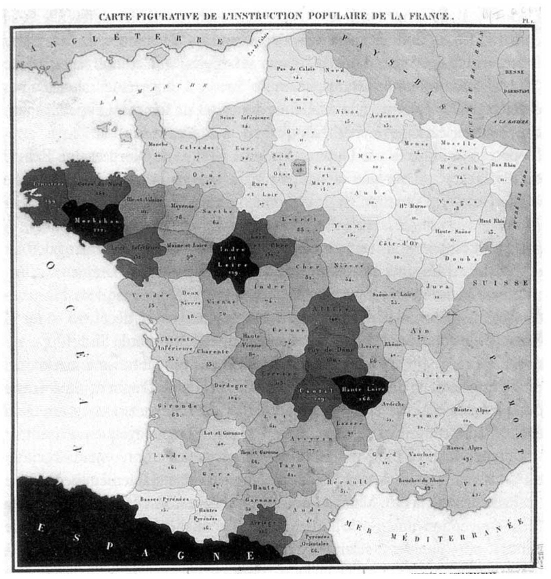


(b) The the number of deaths from various causes during the first year of the Crimean War (1853–1856), visualised with a modern polar area diagram.

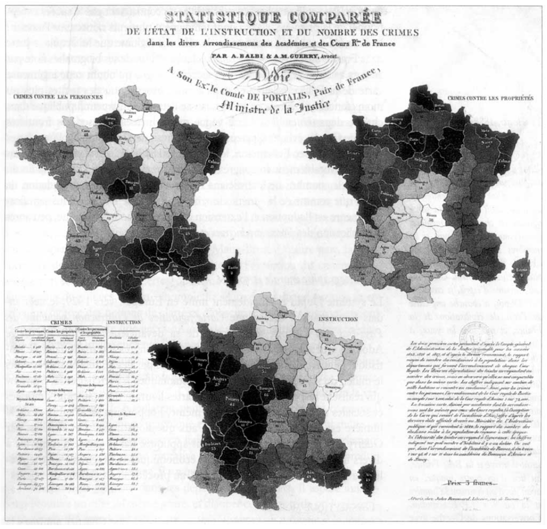


(c) A computer-generated polar area diagram illustrating the death rate from various causes such as wounds and disease during the second year of the war.

Figure 2.14: Polar area diagrams. (a) shows Florence Nightingale's original polar area diagram and (b) and (c) display modern, computer-generated polar area diagrams.

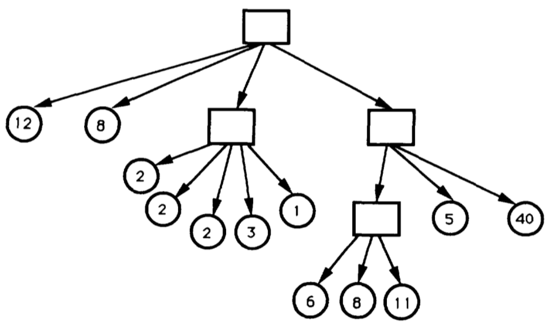


(a) The first choropleth map showing the levels of illiteracy in France [Dupin, 1826].

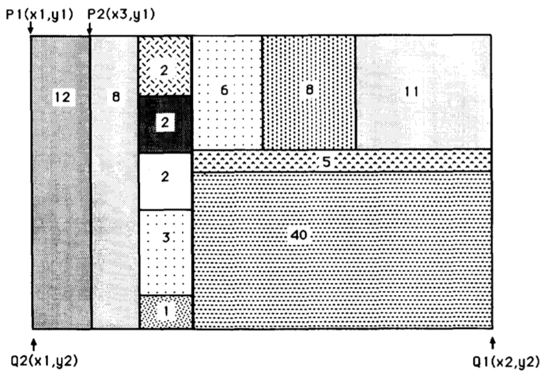

STATISTIQUE COMPARÉE
DE L'ÉTAT DE L'INSTRUCTION ET DU NOMBRE DES CRIMES
dans les divers Arrondissemens des Académies et des Cours Rles de France
PAR A. BALBI & A.M. GUERRY
CRIMES CONTRE LES PERSONNES
CRIMES CONTRE LES PROPRIÉTÉS
INSTRUCTION

(b) The first comparative choropleth maps from Balbi and Guerry [1829]. The map at the top left shows crimes against persons, the map at the top right crimes against property, and the map at the bottom the level of education. A darker shade signifies worse values (more crimes or less education).

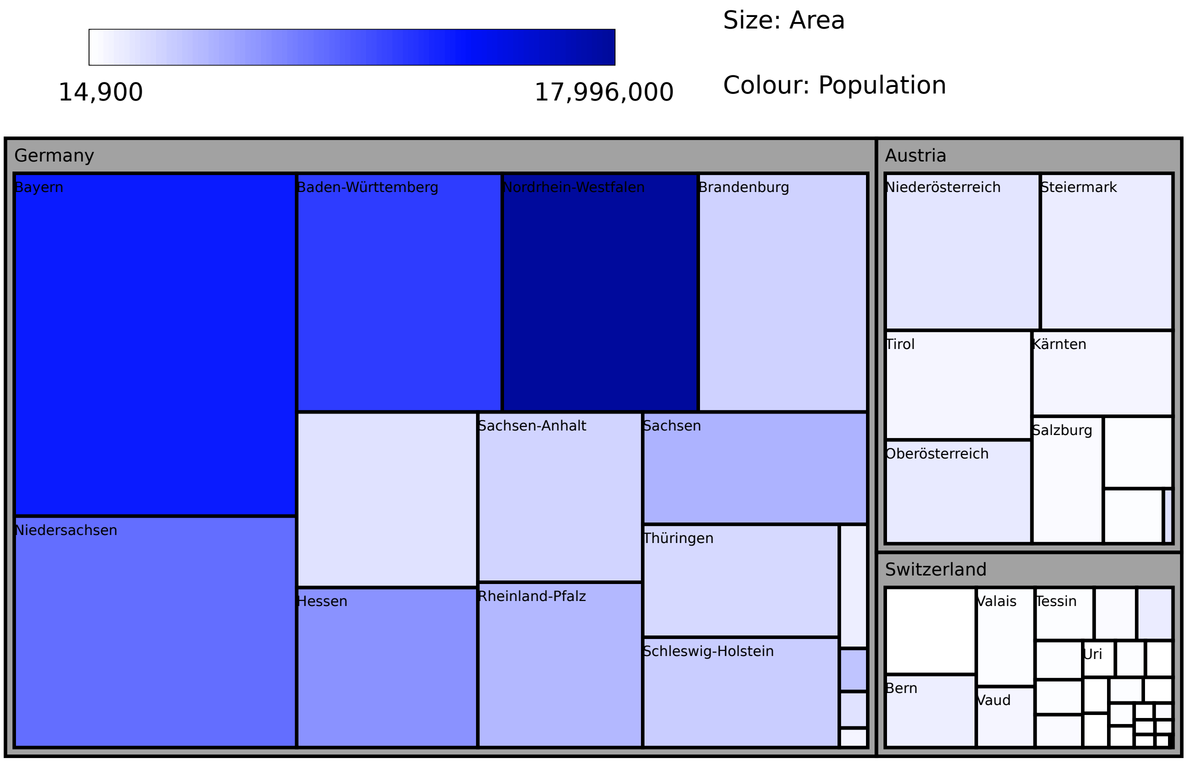
Figure 2.15: Early examples of geographic heatmaps (choropleth maps). (a) shows the first choropleth map and (b) displays the first juxtaposed choropleth maps, allowing direct comparison. The images are extracted from Friendly and Denis [2011].



(a) The traditional node-link drawing for representing tree structures, visualising a file structure consisting of directories (rectangles) and files (circles). The numbers inside the files indicate their sizes. Image from Shneiderman [1992].

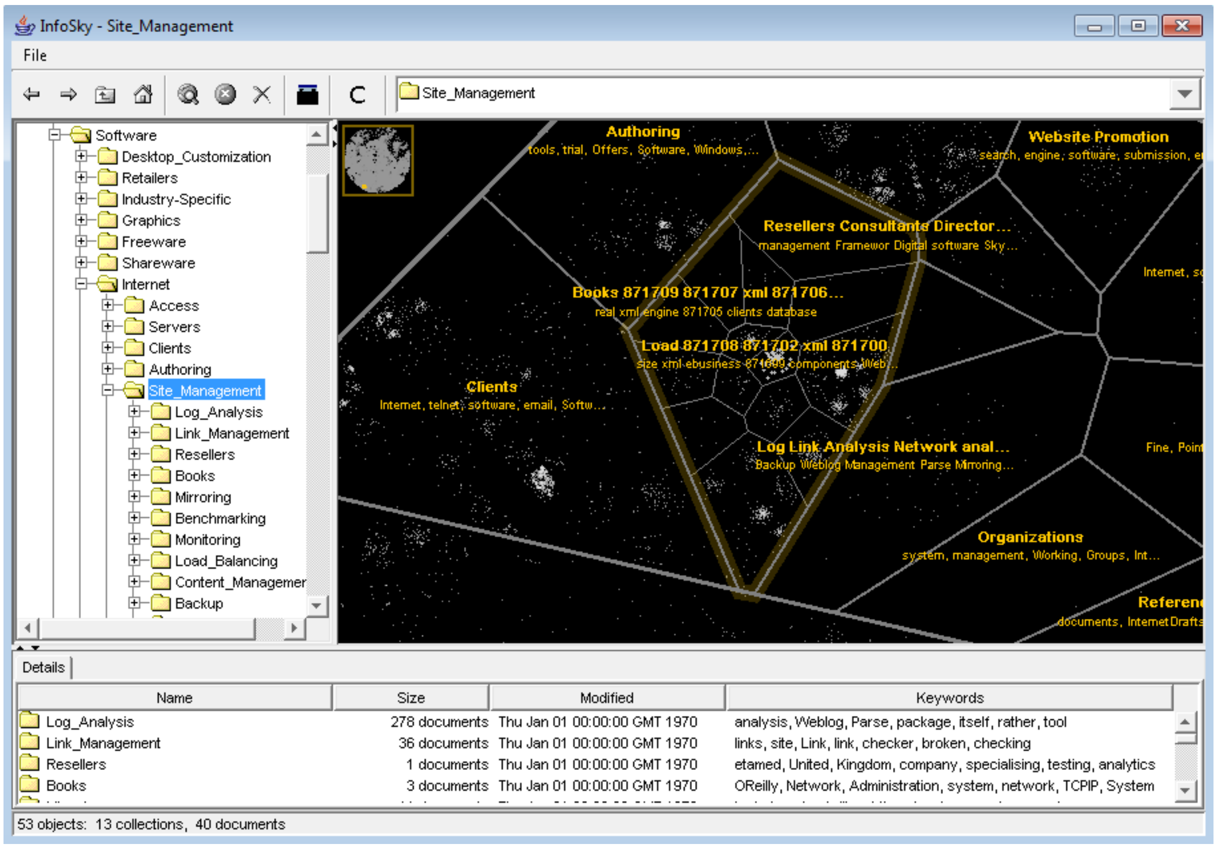


(b) A treemap visualising the same directory structure shown in (a). Image from Shneiderman [1992].



(c) A modern treemap visualising the population and area of regions in Austria, Germany, and Switzerland. The area of a region is expressed as the area of the corresponding rectangle and the absolute population of a region is indicated by its colour.

Figure 2.16: Treemaps are used to visualise hierarchies. (a) shows a 3-level tree structure in the form of a traditional node-link drawing. (b) shows the same tree in a squarified treemap. (c) shows a modern treemap.


InfoSky - Site_Management
File
Site_Management
Software
Desktop_Customization
Retailers
Industry-Specific
Graphics
Freeware
Shareware
Internet
Access
Servers
Clients
Authoring
Site_Management
Log_Analysis
Link_Management
Resellers
Books
Mirroring
Benchmarking
Monitoring
Load_Balancing
Content_Managemer
Backup
Authoring
tools, trial, Offers, Software, Windows,...
Website Promotion
search, engine, software, submission, e
Resellers Consultants Director...
management Framewor Digital software Sky...
Internet, s
Books 871709 871707 xml 871706...
real xml engine 871705 clients database
Load 871708 871702 xml 871700
size xml ebusiness 871699 components Web...
Clients
Internet, telnet, software, email, Softw...
Log Link Analysis Network anal...
Backup Weblog Management Parse Mirroring...
Fine, Point
Organizations
system, management, Working, Groups, Int...
Referen
documents, Internet Drafts
Details

Name	Size	Modified	Keywords
Log_Analysis	278 documents	Thu Jan 01 00:00:00 GMT 1970	analysis, Weblog, Parse, package, itself, rather, tool
Link_Management	36 documents	Thu Jan 01 00:00:00 GMT 1970	links, site, Link, link, checker, broken, checking
Resellers	1 documents	Thu Jan 01 00:00:00 GMT 1970	etamed, United, Kingdom, company, specialising, testing, analytics
Books	3 documents	Thu Jan 01 00:00:00 GMT 1970	OReilly, Network, Administration, system, network, TCPIP, System

53 objects: 13 collections, 40 documents

Figure 2.17: The InfoSky visual explorer, a system for interactively exploring hierarchically structured document collections, uses Voronoi treemaps for area partitioning [Andrews et al., 2002; Granitzer et al., 2004]. It is the first implementation of Voronoi treemaps in the context of information visualisation.

In 2005, the name *Voronoi treemap* was used by Balzer and Deussen [2005] to describe the same idea. Voronoi treemaps use subdivisions into arbitrary polygons instead of the traditional approach of treemaps of splitting the available space into rectangles. This procedure has the advantage of a better visibility of the hierarchical structure in the resulting layout. Additionally, it is possible to create treemap layouts within circles, triangles (see Figure 2.18), and other polygonal outer structures [Balzer et al., 2005].

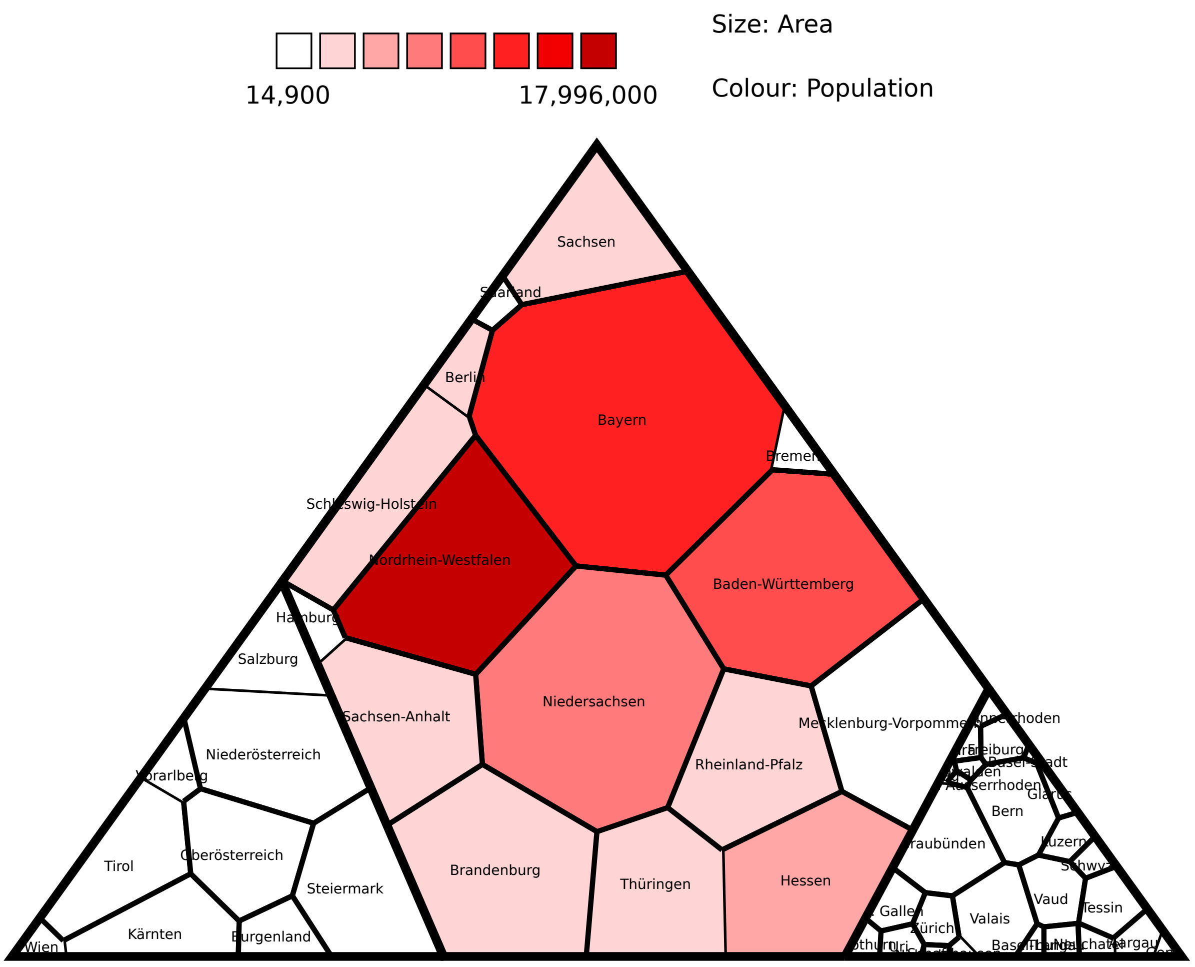


Figure 2.18: A Voronoi treemap displaying the population and area of regions in Austria, Germany, and Switzerland. The area of a region is expressed as the area of the corresponding polygon and the shading of a polygon indicates the population.

Chapter 3

Technologies

“ All you need is trust and a little bit of pixie dust. ”

[James Matthew Barrie, Peter Pan, or The Boy Who Wouldn’t Grow Up, 1904.]

This chapter gives an overview of technologies which can be used to create visualisations. In addition, the prevalence of certain technologies is compared. This is a challenging task, because distribution statistics of internet technologies depend highly on the collecting criteria, varying from studies consisting of a number of surveys to automatic data collection via a script embedded on a web site. In order to provide a more objective view, Figure 3.1 shows the distribution statistics from three different sources: RIAStats [2011], STATOWL [2011], and Millward Brown [2011].

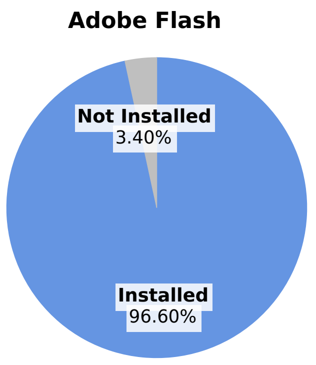
3.1 HTML and AJAX

The main publishing language of the World Wide Web is the HyperText Markup Language (HTML) [W3C, 2011d], invented by Tim Berners-Lee in 1991. HTML in its version 4.01 is the current recommendation of the World Wide Web Consortium (W3C). HTML describes the structure of web sites using elements such as tables, lists, paragraphs, and so forth. Cascading Style Sheets (CSS) [W3C, 2011b], in contrast, are used to define the style of web pages using fonts, layouts, and colours. The structure (HTML) and the style (CSS) of web pages are separated in order to provide better maintainability. This mechanism is often referred to as the separation of content from presentation. Since HTML is designed for static content, it is not suited to provide complex user interactions.

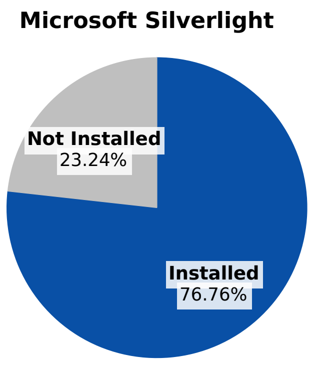
The eXtensible HyperText Markup Language (XHTML), a variant of HTML, is an application of the Extensible Markup Language (XML). It includes the same elements as HTML, but the syntax is slightly different. Due to the fact that XHTML is an application of XML, it is possible to use XML tools such as Extensible Stylesheet Language Transformations (XSLT), a declarative language for XML transformation.

JavaScript, a scripting language allows the extension of HTML with complex user interaction support, such as user-triggered events [W3C, 2011f]. JavaScript code is executed on the client-side by the web browser. It allows dynamic access to and manipulation of the Document Object Model (DOM) [W3C, 2011c] of web pages. The DOM describes all elements of a web page. The term dynamic HTML (DHTML) is used for the combination of HTML, CSS, and JavaScript.

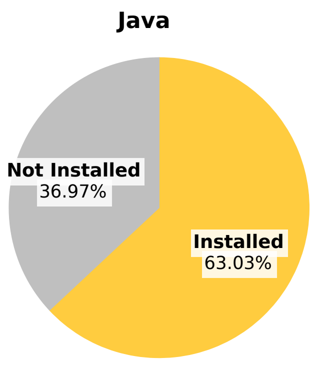
Using the `XMLHttpRequest` object, JavaScript provides the possibility to asynchronously send and retrieve data from a web server. This is called Asynchronous JavaScript and XML (AJAX). Using AJAX,



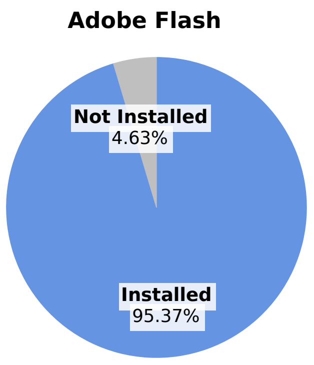
(a) Statistics from RIAStats [2011]. The proportion of users with Adobe Flash Player installed.



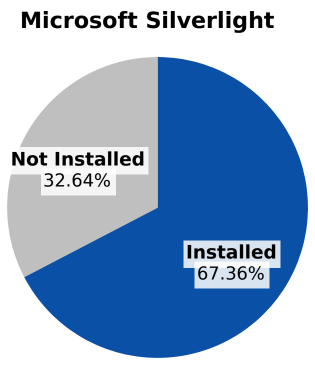
(b) Statistics from [RIAStats, 2011]. The proportion of users with Microsoft Silverlight installed.



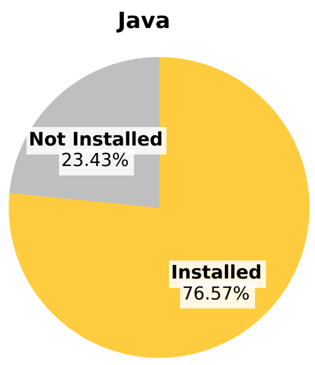
(c) Statistics from [RIAStats, 2011]. The proportion of users with the Java Runtime Environment (JRE) installed.



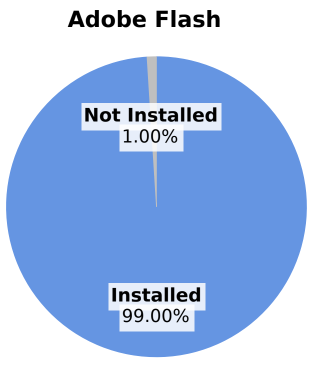
(d) Statistics from STATOWL [2011]. The proportion of users with Adobe Flash Player installed.



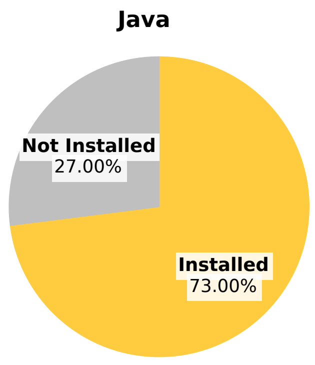
(e) Statistics from [STATOWL, 2011]. The proportion of users with Microsoft Silverlight installed.



(f) Statistics from [STATOWL, 2011]. The proportion of users with the Java Runtime Environment (JRE) installed.



(g) Statistics from Millward Brown [2011]. The proportion of users with Adobe Flash Player installed.



(h) Statistics from [Millward Brown, 2011]. The proportion of users with the Java Runtime Environment (JRE) installed.

Figure 3.1: The distribution of the Adobe Flash plug-in, the Microsoft Silverlight runtime environment, and the Java Runtime Environment (JRE) visualised by pie charts. (a), (b), and (c) display statistics from RIAStats [2011]. The statistics for (d), (e), and (f) are taken from STATOWL [2011]. The statistics in (g), and (h) are from Millward Brown [2011].

it is possible to request only specific data and avoid the web page being fully reloaded. The advantage of this approach is that no additional software, such as a runtime environment, or a plug-in has to be installed. Only a web browser with JavaScript enabled has to be available in order to use AJAX.

3.2 Adobe Flash

Adobe Flash is a tool for creating Rich Internet Applications (RIAs) [Adobe, 2011c]. It adds animation and interactive functionality to web sites and is frequently used for games and advertising purposes. Raster and vector graphics are manipulated by Flash and its attached scripting language ActionScript. ActionScript is an object-oriented programming language with similar semantics and syntax to JavaScript. It is currently available in version 3.0. Flash uses the file format SWF (Shockwave Flash) with its corresponding file extension `.swf` to display its content. Typically SWF files are embedded in a HTML file. Flash needs the Adobe Flash Player, a browser plug-in, in order to display its content. The distribution of the Adobe Flash Player can be seen in Figure 3.1. The statistics are taken from three different sources (RIAStats [2011], STATOWL [2011], and Millward Brown [2011]), in order to provide a broad range of statistics.

Adobe Flash originated from SmartSketch, drawing software developed by Jonathan Gay. In 1996 animation functionality was added and the software and it was released by FutureWave Software with the name FutureSplash Animator. The company FutureWave Software was bought by Macromedia in December 1996, and the FutureSplash Animator became Macromedia Flash 1.0. Flash is currently owned by Adobe, after the acquisition of Macromedia in 2005. [Gay, 2011]

3.3 Adobe Flex

Adobe Flex is an open source framework for developing rich internet applications [Adobe, 2011d]. It was first released by Macromedia in 2004 and the product was integrated in Adobe's product portfolio soon after Adobe acquired Macromedia in 2005. Adobe Flex is partly based on Flash and like in Flash, applications are compiled into SWF files. For this reason Flex needs the Adobe Flash Player to display its content. The difference between Adobe's Flash and Flex is that Flash is intended for designers, using animations together with a timeline concept. Flex targets developers, using a programming model to create applications.

According to Noble et al. [2010, page 1], a Flex application consists of two main parts: ActionScript and MXML. ActionScript is the object-oriented programming language also used in Flash and MXML is a markup language based on the Extensible Markup Language (XML). Graphical components can be created using both ActionScript and MXML. Listing 3.1 shows the ActionScript code necessary to create a button. The same task is performed using MXML in Listing 3.2. The major difference between these two approaches is that the ActionScript code only produces the button object, the MXML version in contrast creates the object and adds it automatically to its parent container.

In Flex, it is possible to save separate MXML and ActionScript files. MXML files, which have the extension `.mxml` can include ActionScript code within a `<fx:Script>` tag. Listing 3.3 shows a simple example of a Flex application. The resulting application can be seen in Figure 3.2.

```
var myButton:Button = new Button();

myButton.label = "My Button";
myButton.width = 200;
myButton.height = 100;
```

Listing 3.1: The ActionScript code necessary to create a button in Adobe Flex. This example produces the same output as the MXML code shown in Listing 3.2.

```
<mx:Button id="myButton" label="My Button" width="200" height="100" />
```

Listing 3.2: MXML code producing a button component. This code is equivalent to the ActionScript code in Listing 3.1.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               minWidth="400" minHeight="300" applicationComplete="init()">
    <fx:Script>
        <![CDATA[

            public function init() : void {
                myButton.addEventListener(MouseEvent.CLICK, setHelloWorld);
            }

            private function setHelloWorld(event:MouseEvent) : void {
                var text:String = myButton.selected ? "Hello World!" : "";
                myText.text = text;
            }

        ]]>
    </fx:Script>

    <mx:VBox verticalAlign="middle" horizontalAlign="center">
        <mx:Button id="myButton" label="Hello World" toggle="true"
                   width="200" height="100" />
        <mx:Label id="myText" fontWeight="bold" fontSize="20"/>
    </mx:VBox>
</s:Application>
```

Listing 3.3: A simple Flex application demonstrating the interaction of MXML components with ActionScript code. The resulting application can be seen in Figure 3.2.

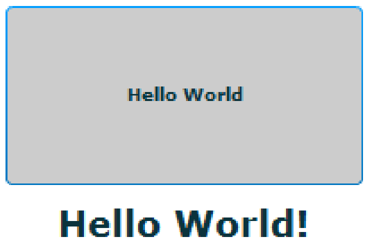


Figure 3.2: The simple application generated by the code in Listing 3.3. A click on the button fires an event causing the textual content of the label to be set.

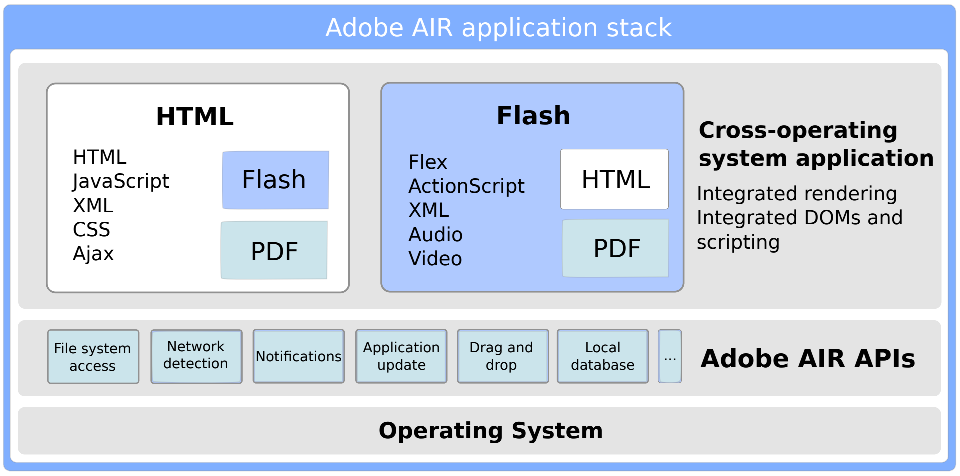


Figure 3.3: The architecture of Adobe AIR. This image is adapted from Adobe [2011b].

3.4 Adobe Integrated Runtime (AIR)

Adobe Integrated Runtime (Adobe AIR) is a cross-operating system runtime [Adobe, 2011a]. It enables developers to deploy rich internet applications (RIAs) on various operating systems using HTML, JavaScript, Adobe Flash, and Adobe Flex. Adobe AIR supports the desktop operating systems Microsoft Windows, and Apple Macintosh. The support for Linux was discontinued after version 2.6. On the mobile device sector Adobe AIR supports operating systems such as Google's Android and Apple's iOS. A set of Adobe AIR APIs, which can be seen in Figure 3.3, enable the use of functionality, such as local file access and storage, network detection, system notifications, drag-and-drop support, database access, and more.

Applications for Adobe AIR have to be installed in order to be executed. For this, the Adobe AIR runtime is needed. An Adobe AIR application can be developed with the use of a simple text editor together with the AIR software development kit (SDK). However, it is recommended to use the Adobe Flash Builder, an integrated development environment (IDE) for AIR development.

3.5 Microsoft Silverlight

Microsoft Silverlight is a tool for developing rich internet applications (RIAs) [Microsoft, 2011c]. It is Microsoft's answer to Adobe's Flash and Flex and its features are very similar to those of Adobe's products. The first release of Silverlight was in April 2007 and its current version is Silverlight 4. The release of version 5 is scheduled for the second half of 2011. Silverlight is available for Microsoft Windows and Apple Macintosh. Unix systems are supported by Moonlight, a free software implementation of Silverlight, developed by Novell [2011]. The distribution of Microsoft Silverlight can be seen in Figure 3.1.

Silverlight applications need the Silverlight runtime environment, a browser plug-in, in order to be executed. These applications typically run within the web browser. However, it is possible to install them on the desktop as well. This functionality, called out-of-browser (OOB), is Microsoft's answer to Adobe AIR. The main benefit of OOB applications is the ability to be launched from the desktop, without the need of an active internet connection.

Microsoft Silverlight uses the declarative language Extensible Application Markup Language (XAML) to create user interface (UI) elements, such as text or buttons. This language is based on XML and can

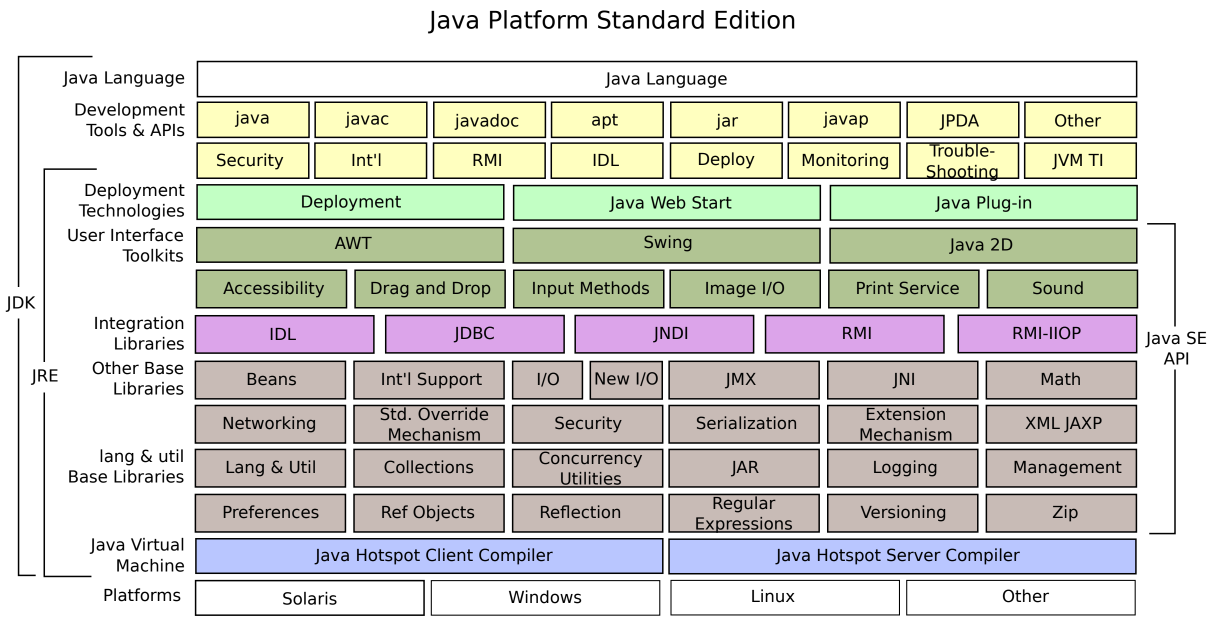


Figure 3.4: The architecture of the Java Standard Edition (SE) platform, showing that the Java Runtime Environment (JRE) sits on top of the underlying operating system. The image is adapted from Oracle [2011c].

be compared to Adobe's MXML. It allows a clean separation of the user interface elements and the code of an application. The major benefit of Silverlight applications is the ability to be fully indexed by search engines. This is possible because the textual content of an application is not compiled and thus is indexable.

3.6 Java

Java is an object-oriented programming language, which originated from a small group of Sun Microsystems engineers led by James Gosling in 1991 [Oracle, 2011a,b]. The Java technology was incorporated into the web browser Netscape Navigator in 1995. The credo of Java is "write once, run everywhere" (WORA), meaning that the execution of Java programs is independent from any operating system. This is achieved by compiling the Java code into Java byte code which is then interpreted by the Java virtual machine (JVM), a software loaded on top of the underlying operating system. Java is a licensed product, having the advantage that its specification looks the same on every execution environment.

A disadvantage of Java is that the execution of a Java program is considered slower, with more memory consumption, than the execution of a program written in native code. Another downside concerns the Java Runtime Environment (JRE). In order to display Java content, the JRE has to be installed previously. The distribution of the JRE can be seen in Figure 3.1. The architecture of the Java platform is shown in Figure 3.4.

3.7 JavaFX

JavaFX is a platform which evolved from the Java platform and enables developers to create rich internet applications (RIAs) [Oracle, 2011d]. It is Sun Microsystems answer to Adobe Flex and Microsoft Silverlight. JavaFX was first released in 2007 by Sun Microsystems. After Oracle acquired Sun Microsystems, it was announced that the support for the JavaFX Script language will be discontinued and the corresponding Script APIs will be ported to Java. JavaFX is currently available in version 2.0.

JavaFX allows the creation of applications, using two programming languages: Java and FXML. FXML is similar to Adobe's MXML and Microsoft's XAML. It is an, XML-based declarative language for creating user interface elements.

3.8 Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) is a language which enables the description of two-dimensional graphics in XML [W3C, 2011a,e]. It is an open standard developed by the SVG working group at the World Wide Web Consortium (W3C). The working group was chartered after the submission of six competing proposals for vector graphic specifications in 1998 [W3C, 2011g]. The first specification of SVG was released in September 2001. In contrast to raster graphics, Scalable Vector Graphics are completely scalable without loss of quality.

The SVG specification in its version 1.1 is a W3C recommendation. It is the most recent version of the full specification. SVG tiny 1.2, another W3C recommendation targets mobile devices. The SVG working group is currently working on the specification for SVG 2.0 which will add new features to SVG.

Elements in SVG can be easily grouped, transformed, and styled. In addition, it is possible to create interactive, and dynamic SVG drawings by using animations. Through a scripting language, the SVG Document Object Model (DOM) can be accessed and its elements, attributes, and properties can be modified. Any graphical object can be extended with event handlers like `onmouseover` and `onclick`.

There are various types of elements in SVG: [W3C, 2011e]

- **Paths:** An outline of a shape is represented by a path in SVG. It can be filled, stroked, and used as a clipping path. The drawing procedure makes use of the concept of the current point in addition to several drawing commands such as *move to*, *line to*, or *curve to*.
- **Basic Shapes:** These are basic elements like lines, rectangles, circles, ellipses, and polygons.
- **Text:** The `text` element is rendered as part of a SVG document fragment. It causes a single textual line to be rendered. Automatic line breaks are not supported.
- **Filling, Stroking and Marker Symbols:** Filling and stroking can be applied to the `text` and `path` elements and the basic shapes. Additionally, some elements such as `path`, `polygon`, and `line` can have marker symbols drawn on top of their vertices.
- **Colour:** The colour property has to be specified using the sRGB colour space and it is used to provide a value for properties like `fill` or `stroke`.
- **Gradients and Patterns:** The filling or stroking of elements can also be achieved using gradients and patterns. Gradients can be linear or radial and patterns can be provided from a vector or image source.
- **Filter Effects:** Scalable Vector graphics support filter effects. These are a series of operations which can be applied to any graphical element in order to modify the original element.
- **Interactivity:** The execution of animations and scripts can be triggered in SVG in response to user-performed actions.

- **Linking:** In Scalable Vector Graphics it is possible to include links to any Internationalized Resource Identifiers (IRIs). These are more generalised counterparts to Uniform Resource Identifiers (URIs).
- **Scripting:** Using the `script` element, scripts can be inserted into a Scalable Vector Graphic. It is equivalent to the `script` element in HTML.
- **Animation:** A Scalable Vector Graphic can change its content over time using animations. Effects like fade-in, fade-out, growing or shrinking objects, or colour change can be applied in this way.
- **Fonts:** In order to provide designers the ability to customise text in Scalable Vector Graphics, fonts can be inserted using the `fonts` element. This mechanism ensures that the image is displayed equally on all platforms, even if a certain font is not installed on the user's computer.
- **Metadata:** Data about an SVG file can be inserted using the `metadata` element.

Scalable Vector Graphics can be created using a simple text editor. Here, the SVG tags have to be typed into the editor and saved as an SVG file. However, a more convenient approach to create SVG files is using a graphical editor such as Adobe Illustrator [Adobe, 2011e], Corel Draw [Corel, 2011], Microsoft Expression Design [Microsoft, 2011a], the freely available Inkscape [Inkscape, 2011], or online solutions like Google Docs [Google, 2011c]. An example of a Scalable Vector Graphic file can be seen in Listing 3.4. Figure 3.5 shows the corresponding image.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="4in" height="2in"
     viewBox="0 0 4000 2000" version="1.1"
     xmlns="http://www.w3.org/2000/svg">

  <!-- Define a triangle as an arrowhead -->
  <defs>
    <marker id="Triangle"
      viewBox="0 0 10 10" refX="0" refY="5"
      markerUnits="strokeWidth"
      markerWidth="4" markerHeight="3"
      orient="auto">
      <path d="M 0 0 L 10 5 L 0 10 z" />
    </marker>
  </defs>

  <!-- outer frame -->
  <rect x="10" y="10" width="3980" height="1980"
       fill="none" stroke="blue" stroke-width="10" />

  <path d="M 1000 750 L 2000 750 L 2500 1250"
        fill="none" stroke="black" stroke-width="100"
        marker-end="url(#Triangle)"  />

  <circle cx="2900" cy="1700" r="100"
        fill="red" stroke="blue" stroke-width="10"  />

</svg>
```

Listing 3.4: An example of a Scalable Vector Graphics file. It specifies a marker in the shape of a triangle and contains a rectangle, a path and a circle. The resulting image can be seen in Figure 3.5.

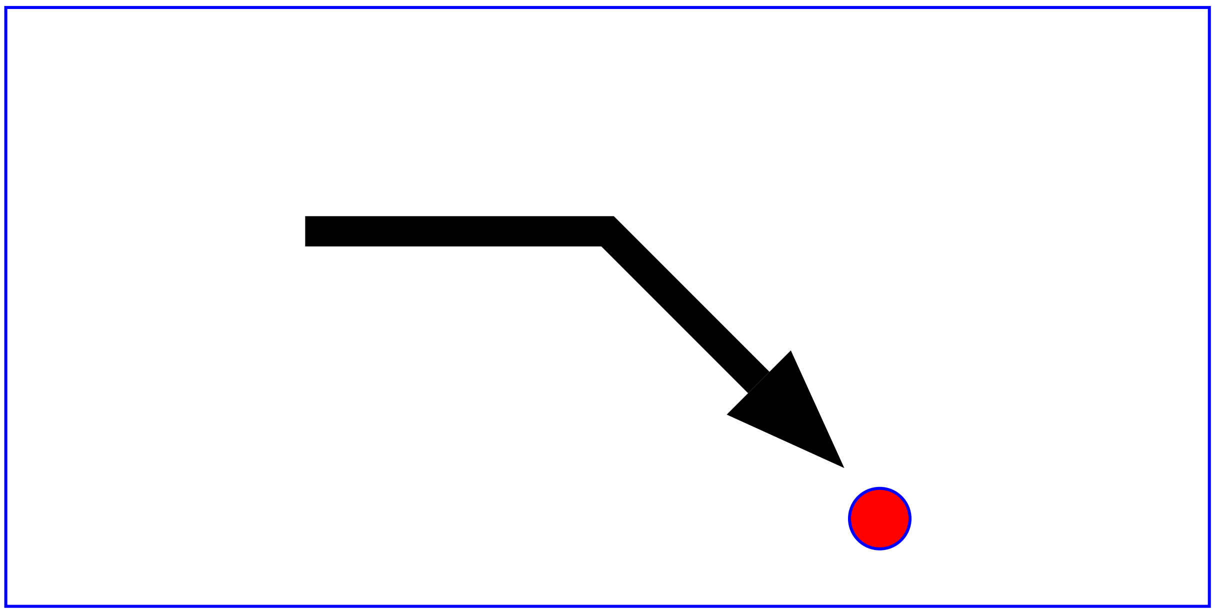


Figure 3.5: This figure shows the resulting SVG image from Listing 3.4.

Chapter 4

Information Visualisation Software and Tools

4.1 Online Services

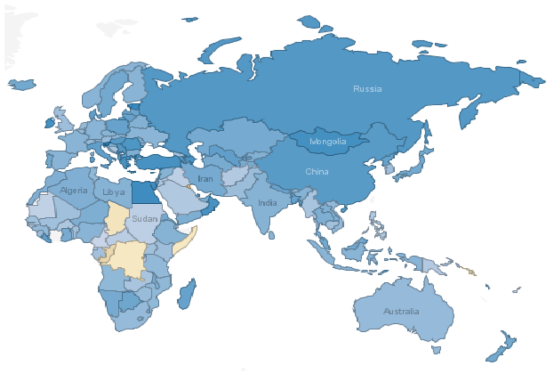
This chapter focuses on services, which are available on the internet and can be accessed using a web browser. Online services have the major benefit of user collaboration. In most cases, users have the possibility to share data or visualisations with each other. They can collaborate during the creation process and can comment on the resulting visualisations. In this way, new insights into the visualised data can be gained.

There are several drawbacks with online services: depending on the speed of the internet connection, online services can be quite slow. If the server crashes, the service can no longer be reached. In most cases, online services are programmed by using a technology such as Adobe Flex or Java, which requires a browser plug-in or in the case of Java, the Java Runtime Environment (JRE), to display its content. If the necessary component is not installed, the service cannot be used. Another disadvantage of online services is that there may be no possibility to keep the uploaded data and the created visualisations private. Several online visualisation services require that any uploaded data and resultant visualisations are visible to all.

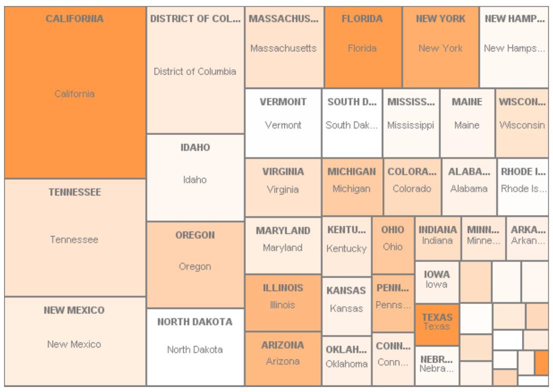
4.1.1 Many Eyes

Martin Wattenberg, the founder of IBM's Visual Communication Lab and Fernanda Viégas created Many Eyes in 2007 [Viégas et al., 2007; IBM, 2011a,b]. The service combines information visualisation with social software. It gives users the possibility to collaboratively explore and discuss visualisations. According to Viégas et al. [2007], the goal of Many Eyes is to "[...] not only serve as a discovery tool for individuals but also as a medium to spur discussion among users." Many Eyes visualisations are implemented using Java. There are many visualisations such as scatter plots, matrix charts, network diagrams, bar charts, bubble charts, line charts, pie charts, treemaps, choropleth maps, and many more. Figure 4.1 shows examples of visualisations in Many Eyes.

In Many Eyes, users can upload their own data and create visualisations. The user has to perform three steps to create a visualisation. The first step is to choose a dataset. This can be done by using an existing dataset, or by uploading a dataset prepared as a tab separated table. The second step is to choose a visualisation type, used to visualise the previously chosen dataset. In the last step, the user has the possibility to customise the look of the visualisation.



(a) Part of the World Map visualisation in Many Eyes, displaying the percentage change of infant mortality of each country in the world. The colour blue expresses falling infant mortality, whereas its rising is indicated using a yellow shading.



(b) A treemap containing each state of the United States. Its shading expresses the number of alien sightings per state. The number of inhabitants is encoded using its areas.

Figure 4.1: Examples of visualisations in Many Eyes.

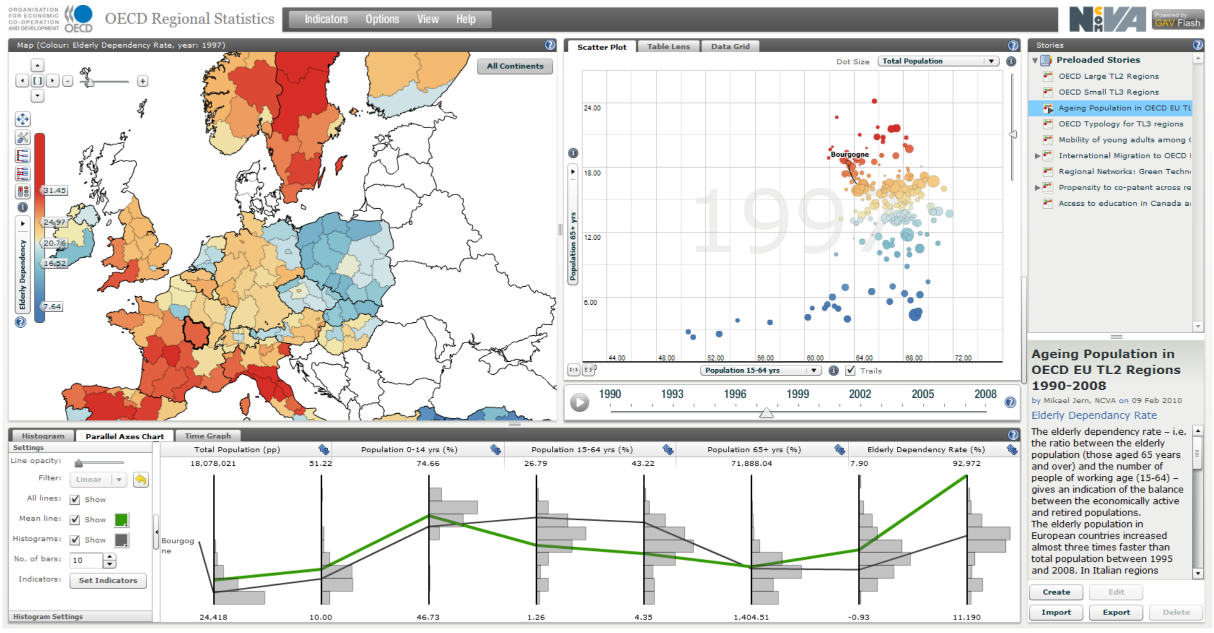
The major benefit of Many Eyes is the social aspect. Users can take snapshots of visualisations and insert comments. This enables users to point out interesting findings and collaboratively explore the visualised data. On the other hand, it is not possible to upload a dataset and to create a visualisation without making it public. Another drawback of the service is the limited export functionality: only the possibility to export the visualisation as a pixel graphic.

4.1.2 Statistics eXplorer

Statistics eXplorer is a tool for interactively analysing statistical data [NCVA, 2011d]. It is customised for use by statistics bureaus and gives users the possibility to explore trends over time while using multiple, synchronised, highly interactive visualisations.

The tool was first released in November 2008 on the OECD [2011] web site and is developed by the The National Center for Visual Analytics (NCVA), a Swedish national research lab, located at the Department of Science and Technology of Linköping University [NCVA, 2011b]. The OECD eXplorer can be seen in Figure 4.2. According to Jern [2009], the OECD eXplorer was developed using GAV Flash (see Section 4.3.3). In 2009, the Open Statistics eXplorer with a research and educational license for non-commercial use was created. This generic platform enables the customisation of the Statistics eXplorer application using the "eXplorer Wizard" [NCVA, 2011c]. Norrköping Communicative Visual Analytics (NComVA), a spin-off company of the NCVA, was founded in June 2010 for commercial deployment of the Statistics eXplorer [NCVA, 2011d; NComVA, 2011a].

Statistics eXplorer starts with various preloaded datasets. To load new data, there are two possibilities. The first possibility is to load data from the attached statistics database. The available data can be browsed and selected, and a particular timespan (if available) can be loaded. The second possibility is to load one's own data from a local source. For that, the data has to match the expected format.



Ageing Population in OECD EU TL2 Regions 1990-2008

by Mikael Jern, NCVA on 09 Feb 2010

Elderly Dependancy Rate

The elderly dependency rate – i.e. the ratio between the elderly population (those aged 65 years and over) and the number of people of working age (15-64) – gives an indication of the balance between the economically active and retired populations.
The elderly population in European countries increased almost three times faster than total population between 1995 and 2008. In Italian regions

Figure 4.2: Aging statistics in Europe, visualised by OECD eXplorer. At the left side a choropleth map allows comparison of the districts of each country and at the right side the corresponding data is visualised using a scatter plot. A parallel coordinates visualisation can be seen at the bottom.

A very useful feature is the possibility to create stories in Statistics eXplorer. Upon pressing the Create button on the right side, the Story Editor appears. With this editor it is possible to report findings together with metadata. The resulting story can be exported as an XML file or published using a "Vislet". A Vislet is an embeddable interactive visual representation of the created story. Saved stories can be imported using the Import button.

Statistics eXplorer is written using Flex and offers many different synchronised visualisations including scatter plots, table lenses, histograms, parallel coordinates, time graphs, and choropleth maps. Each of them offers highly interactive, intuitive handling.

4.1.3 Gapminder

Gapminder [2011] World is a service which allows the exploration of statistical data using animated time series. It evolved in 2006 from the Trendalyzer software and it is developed by the Gapminder Foundataion, which was founded by Ola Rosling, Anna Rosling Rönnlund, and Hans Rosling in 2005. Unfortunately, Gapminder does not allow the uploading of one's own data. However, there are many predefined datasets available which can be visualised.

Gapminder World can be seen in Figure 4.3. It consists of a scatterplot, displaying different data dimensions on each axis. The colours of the circles, representing the countries, indicate the country's geographic region. Additionally, the circle size can be used to express another data dimension. Using the slider component at the bottom enables users to follow changes in the data over time.

In 2007, the Trendalyzer software and its development team was acquired by Google. Google now offers a gadget called Motion Chart in Google Docs. It is very similar to Gapminder and has the advantage that one's own data can be visualised. It is integrated within the Google Visualization API and can be included into a web site using JavaScript [Google, 2011e]. See Section 4.1.5.

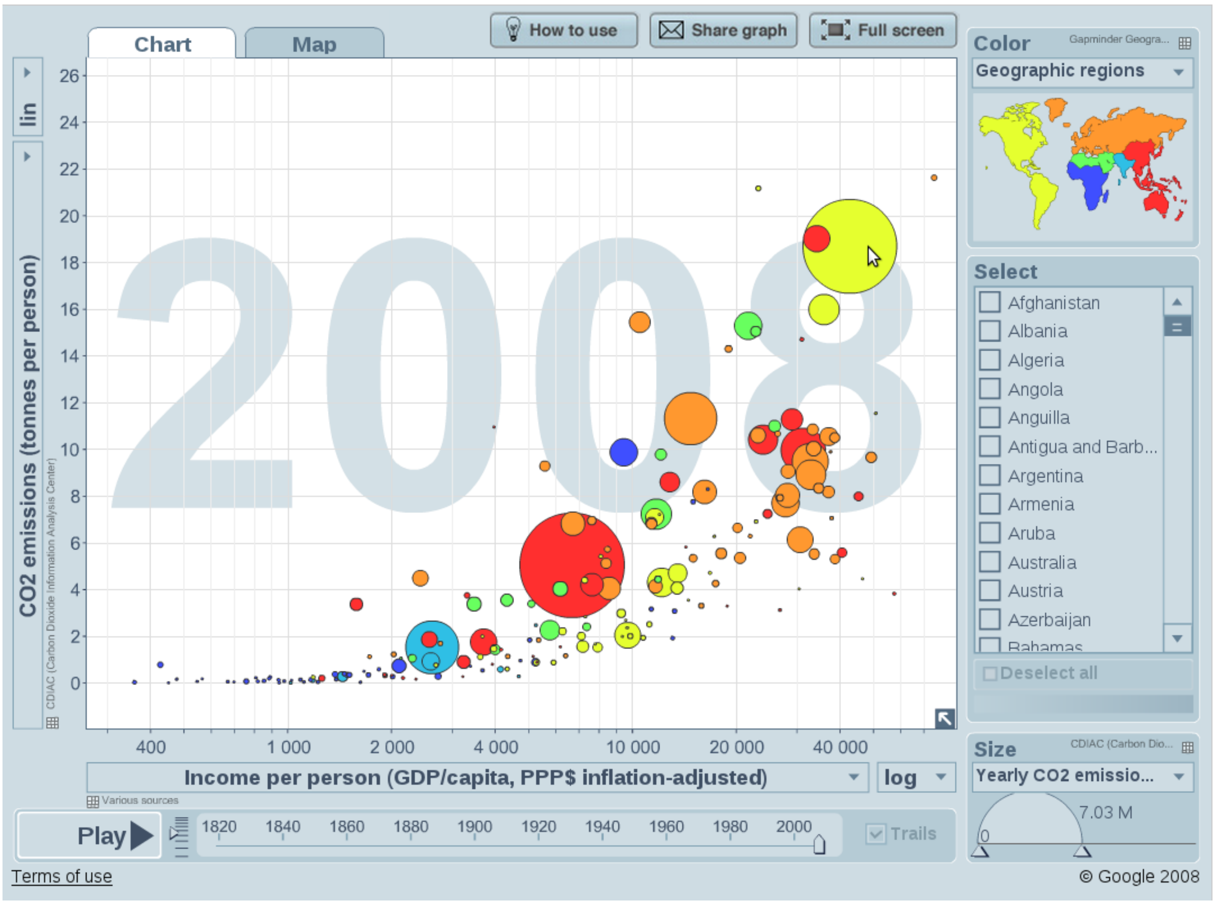


Figure 4.3: Gapminder World visualising CO2 emission statistics. The scatterplot currently displays the CO2 emissions for each country on its y axis and the income per person on its x axis. The colours of the circles indicate the country's geographic region and the size of each circle indicates the country's total CO2 emissions for the displayed year. The slider at the bottom allows the exploration of changes in the data over time.

4.1.4 Google Image Charts

The service Google Image Charts [Google, 2011d] uses the Goolge Chart API to dynamically generate different kinds of charts from URL strings. Both the data and the formatting arguments are included within a long URL. The tool sends a PNG image back, which can be embedded into a web site. The arguments have to fulfill certain requirements, in order to obtain the expected output from the service. Before the tool was opened up for web developers, it was originally built as an internal service for rapid chart embedding. It includes several chart and visualisation types, like bar charts, box charts, candlestick charts, compound charts, dynamic icons, formulas, Google-O-Meter charts, GraphViz charts, line charts, pie charts, QR codes, radar charts, scatterplots, venn charts, and map charts. Figure 4.4 shows an example of a visualisation created with Google Image Charts. The corresponding URL can be seen in Listing 4.1.

```
https://chart.googleapis.com/chart?
cht=map:fixed=-60,0,80,-35&
chs=600x400&
chld=CA-BC|CN|IT|GR|US-UT&
chdl=Vancouver|Beijing|Torino|Athens|Salt+Lake+City&
chco=B3BCC0|5781AE|FF0000|FFC726|885E80|518274&
chtt=Last+Five+Olympic+Host&
schm=f2010+Winter,000000,0,0,10
f2008+Summer,000000,0,1,10
f2008+Winter,000000,0,2,10,1,:-5:10
f2004+Summer,000000,0,3,10
f2004+Summer,000000,0,4,10&
chma=0,110,0,0
```

Listing 4.1: The URL used to create Figure 4.4. It is broken into separate lines for display purposes.

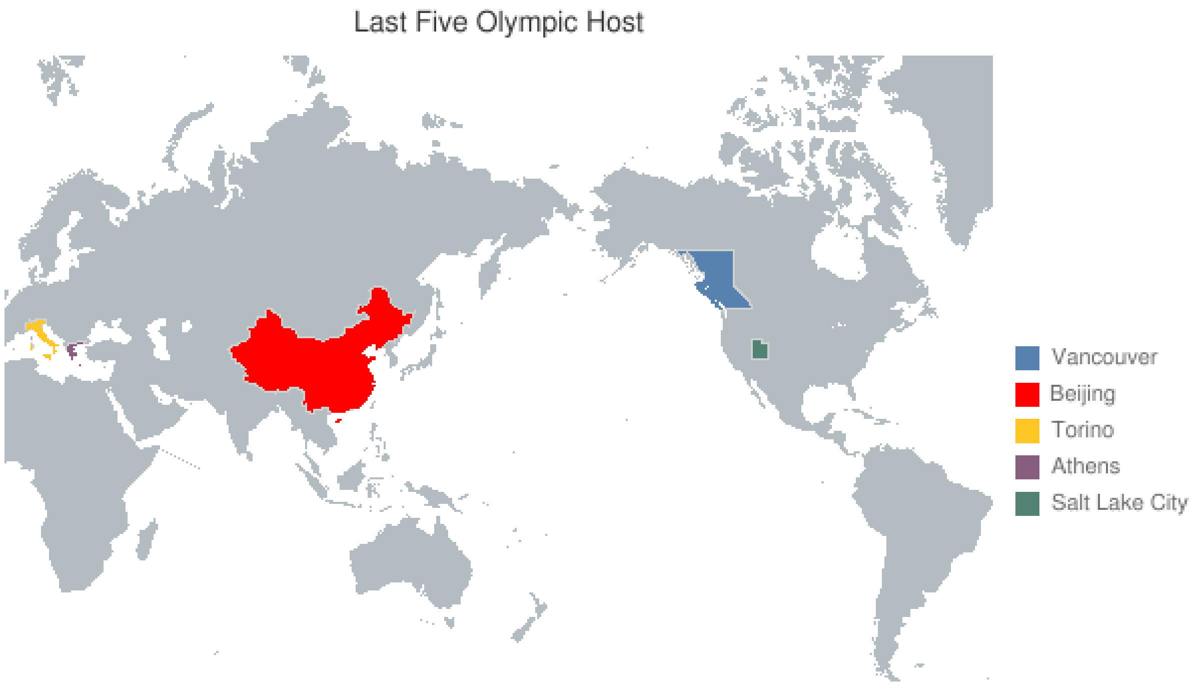


Figure 4.4: The last five olympic host countries visualised with Google Image Charts. This figure was created using the URL in Listing 4.1.

4.1.5 Google Docs

Google Docs is a free web-based service that offers the possibility to create, upload, edit, and share documents [Google, 2011a]. It includes word processor, presentation, spreadsheet, form, and drawing applications. In addition, users can upload all kinds of documents and share them with other users. The service allows collaboration among users in real-time, paired with a revision history system.

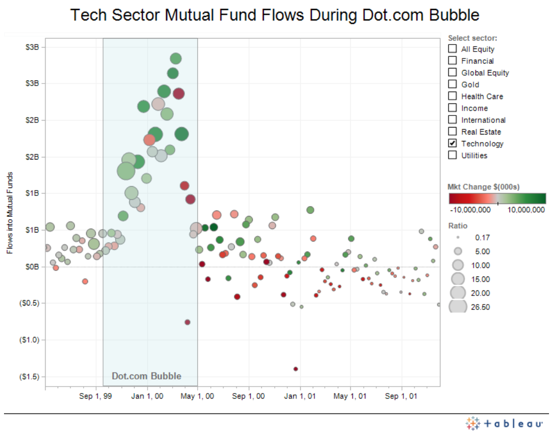
When using the spreadsheet service, users have the possibility to insert a gadget to visualise their data. A gadget is a simple JavaScript and HTML application that can be embedded into the spreadsheet. Using the Google Gadgets API, users can also create and submit their own gadgets [Google, 2011b]. Up to now, the following gadget types are submitted by companies and individuals:

- **Pie Chart and Doughnut Chart:** InfoSoft Global (P) Ltd. contributed this Flash-based visualisation which can be displayed in both 2D and 3D. The appearance of the visualisations can be changed by using the settings panel of the gadgets. Multiple options such as labelling, font, and number options are available. Sadly, there is no export functionality included. The only possibility to save the resulting visualisation is to make a screenshot.
- **Funnel Gadget:** This gadget was also submitted by InfoSoft Global (P) Ltd. It has the same characteristics as the pie and doughnut chart.
- **Pyramid Gadget:** This is the third gadget contributed by InfoSoft Global (P) Ltd. Unfortunately, there is also no export functionality included.
- **Star Plot** A star plot gadget was submitted by Greg Marra. Unfortunately, it has little functionality. The only available option included within the gadget settings is the ability to change the appearance of the legend.
- **Treemap** A treemap is contributed by Yaar Schnittman. It is a squarified treemap, offering two different layout and labelling options. In addition, it is possible to define two colours for the colour range.
- **Event Timeline:** David Huynh and Timeline Fans submitted a gadget displaying a timeline of events.
- **Gantt Chart:** Viewpath [2011] offers the possibility to create a Gantt chart, displaying task schedules.

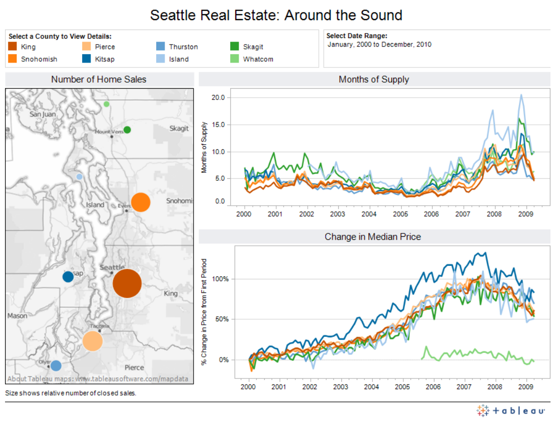
Besides the possibility for companies and individuals to submit their own gadgets, most of the available gadgets are developed by Google itself. This includes pie charts, bar charts, area charts, scatter charts, motion charts, line charts, and many more.

4.2 Standalone Software

This section describes standalone software for creating visualisations. These solutions have to be downloaded and installed locally on a computer in order to use them. The major benefit of this procedure is that the user does not need to subscribe to an online account or to upload or publish their data. Standalone software typically runs faster than an online service and does not require an active connection to the internet. However, a negative aspect of the use of standalone software is the lack of possibilities to share data and visualisations and to collaborate with other users.



(a) A bubble chart visualising fund flows in the technological sector before, during, and after the dot.com bubble.



(b) A visualisation displaying the real estate prices in counties around Seattle.

Figure 4.5: Two examples of visualisations created with Tableau. The images are extracted from Tableau [2011b].

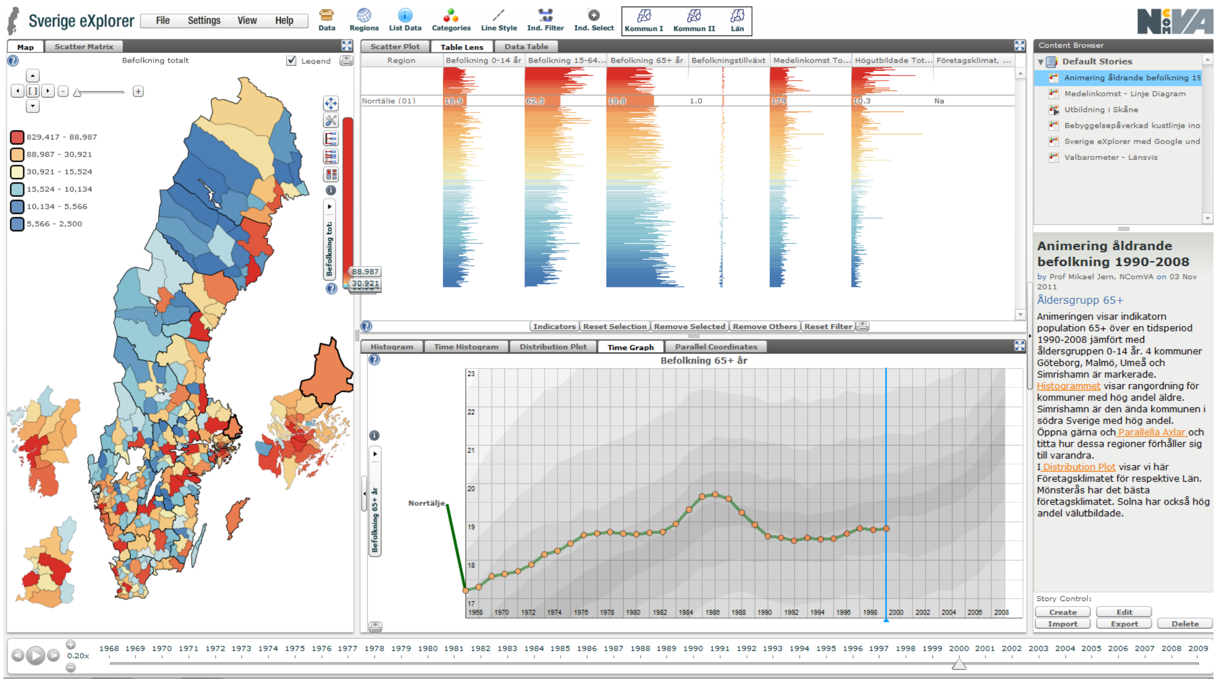
4.2.1 Tableau

Tableau is a family of data visualisation software originating from academic research at Stanford University. Pat Hanrahan, Chris Stolte, and Christian Chabot founded the company Tableau Software in 2003 [Tableau, 2011b; Bloomberg, 2011]. The software family consists of the products Tableau Desktop, Tableau Server, Tableau Digital, and Tableau Public. Apart from Tableau Public, which has limited storage space and a limit of data rows that can be visualised, the other products are not free [Tableau, 2011a]. However, Tableau offers a free 30-day trial version for these software products. Tableau Public is a service allowing users to create and share visualisations on the web. A disadvantage of this product is that these visualisations and the corresponding data are always public. Tableau Digital is the premium version of Tableau Public and allows visualisations and data to be kept private.

In order to create a visualisation, Tableau first has to be connected to a data source. Data sources such as Microsoft Excel, Microsoft Access, MYSQL, PostgreSQL, and many more are available. In the second step, a view containing a visualisation can be created. This is done by dragging and dropping fields onto shelves. Figure 4.5 shows two visualisations created with Tableau.

4.2.2 Statistics eXplorer Desktop Version

Besides the online version of Statistics eXplorer (see Section 4.1.2), NcomVA offers a standalone version as well [NComVA, 2011c]. It can be installed locally on the user's computer and it is available in the versions Sweden eXplorer, Europe eXplorer, and World eXplorer. In addition to the functionality offered by the online version, the desktop version contains an improved GUI layout. It moves commonly used functionality to a new menu bar at the top of the application. Additionally, this version includes new functionality such as a region filter, region highlighting, different line styles, categories and an indicator filter. Figure 4.6 shows a screenshot of the online demo version of the Sweden eXplorer. Sadly, no free version of the desktop version of Statistics eXplorer is available. Only a yearly licence can be purchased.



Animering åldrande befolkning 1990-2008

by Prof Mikael Jern, NComVA on 03 Nov 2011

Åldersgrupp 65+

Animeringen visar indikatorn population 65+ över en tidsperiod 1990-2008 jämfört med åldersgruppen 0-14 år. 4 kommuner Göteborg, Malmö, Umeå och Simrishamn är markerade. Histogrammet visar rangordning för kommuner med hög andel äldre. Simrishamn är den ända kommunen i södra Sverige med hög andel. Öppna gärna och Parallella Axlar och titta hur dessa regioner förhåller sig till varandra. I Distribution Plot visar vi här Företagsklimatet för respektive Län. Mönsterås har det bästa företagsklimatet. Solna har också hög andel välutbildade.

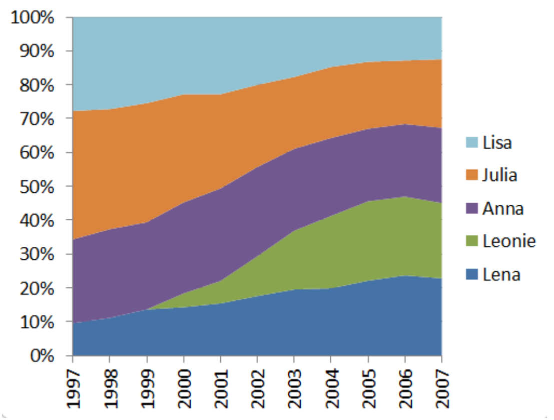
Figure 4.6: The Sweden eXplorer displaying the total population of Sweden. On the left side, a map displays the provinces of Sweden. A table lens can be seen on the top right and a time graph is shown on the bottom right.

4.2.3 Microsoft Excel and OpenOffice Calc

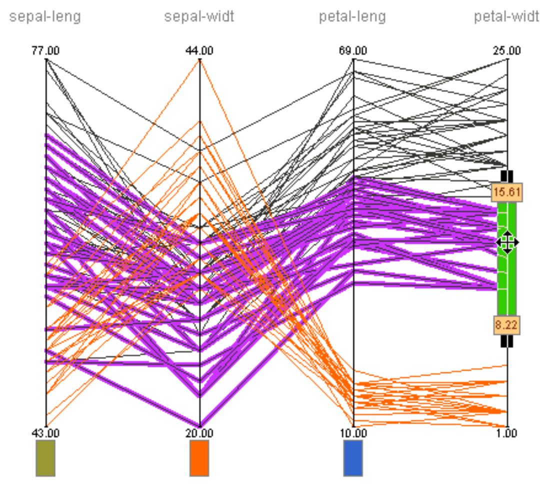
Microsoft Excel is a standalone spreadsheet application included within Microsoft Office [Microsoft, 2011b]. It is available for Windows and Mac and its latest version for Windows is Microsoft Excel 2010. Excel offers advanced functionality with regard to the import, manipulation, and export of data. In addition, various different charts can be produced and inserted into a spreadsheet. Microsoft Excel offers ten different chart types, which can be inserted by selecting a data range on the spreadsheet together with choosing the desired chart type. Unfortunately, the export functionality of charts is limited: there exists only the possibility to export charts using a PDF printer. Figure 4.7a shows an example of a chart created with Microsoft Excel.

The charting functionality of Microsoft Excel can be extended by installing add-ins or templates. An example of an Excel add-in is VisuLab [2011], a project of Hans Hinterberger at the Institute of Computational Science at Swiss Federal Institute of Technology Zurich. With VisuLab it is possible to visualise multidimensional data using parallel coordinates (see Figure 4.7b), Andrew's curves, scatterplot matrices, and permutation matrices. Another extension which can be used to enhance the charting functionality of Microsoft Excel is the open-source template NodeXL [2011], a project of the Social Media Research Foundation. Using NodeXL, it is possible to create network graphs such as the graph in Figure 4.7c.

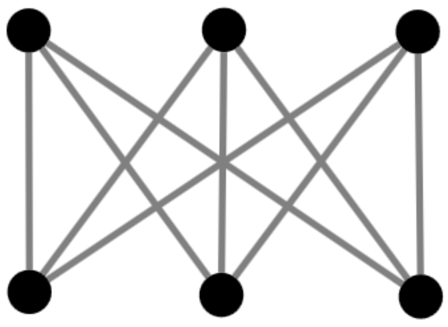
A free alternative to Microsoft Excel is OpenOffice Calc, which offers similar functionality for importing, creating, and modifying data [Apache, 2011]. Like in Microsoft Excel, users can create charts such as bar charts, pie charts, area charts, line charts, scatterplots, bubble charts, and star plots. Unfortunately, the offered charts are not interactive and there is no export functionality. An example of a chart created with OpenOffice Calc is shown in Figure 4.7d.



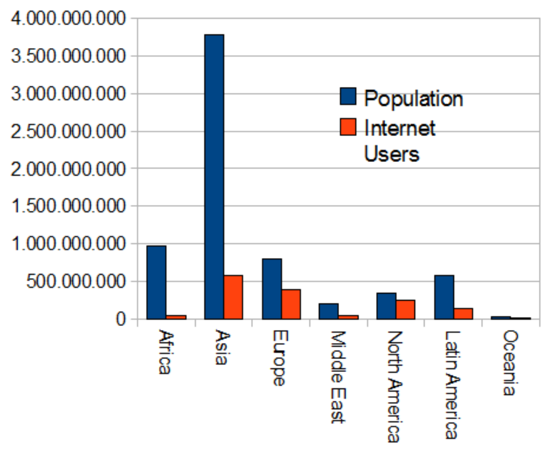
(a) An area chart created with Microsoft Excel displaying the percentage of female forenames from 1997 to 2007.



(b) Parallel coordinates visualisation created with the Excel add-in VisuLab. This image is extracted from VisuLab [2011].



(c) The graph $K_{3,3}$ created with the Excel template NodeXL.



(d) A bar chart created with OpenOffice Calc. It displays the population of the World's regions in relation to the number of internet users.

Figure 4.7: Examples of visualisations created with Microsoft Excel and OpenOffice Calc.

```
var vis = new pv.Panel()
     .width(150)
     .height(150);

vis.add(pv.Rule)
     .data(pv.range(0, 2, .5))
     .bottom(function(d) d * 80 + .5)
   .add(pv.Label);

vis.add(pv.Bar)
     .data([1, 1.2, 1.7, 1.5, .7])
     .width(20)
     .height(function(d) d * 80)
     .bottom(0)
     .left(function() this.index * 25 + 25)
   .anchor("bottom").add(pv.Label);

vis.render();
```

Listing 4.2: Protivis code used to produce the bar chart shown in Figure 4.8a. This example is taken from Protovis [2011].

4.3 Libraries and Toolkits

Libraries and packages give users the possibility to embed a visualisation into a web site or application. The solutions are typically developed by a third party and require developer knowledge in order to set them up.

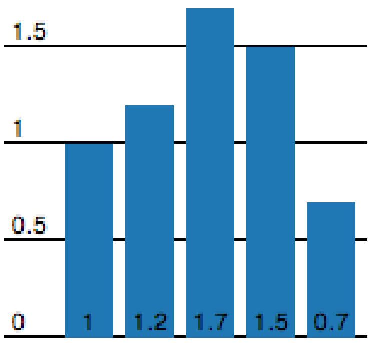
4.3.1 Protovis

According to Bostock and Heer [2009], Protovis [2011] is an embedded domain-specific language, allowing the creation of visualisations by hierarchically composing graphical primitives (called marks). It is a project of the Stanford Visualization Group led by Mike Bostock and Jeff Heer and it is implemented in JavaScript. Protovis was created to close the gap between charting software which allows the creation of predefined charts and vector-based drawing programs. The development team already released a final version of Protovis (v3.3.1) and is now developing the visualisation library D3 (see Section 4.3.2).

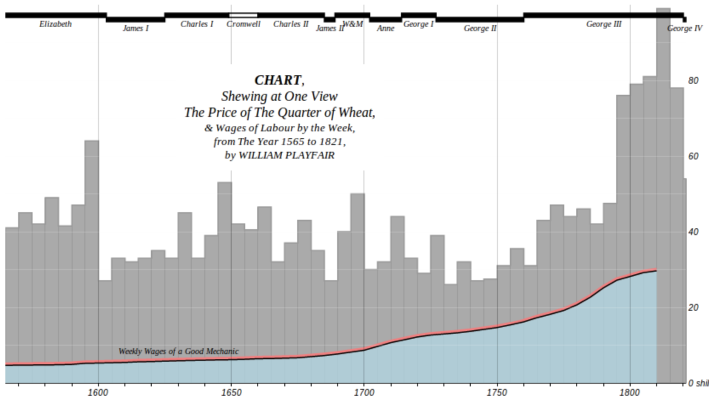
In Protovis, visualisations can be created by composing marks. Marks are graphical primitives such as bars, lines, and labels. They are associated with data and visual properties such as colour and position. Data can be added using the JavaScript Object Notation and transformations can be applied to it. Event handlers can be registered in order to add interactivity to marks. In addition, Protovis offers HTML5, SVG, and Flash rendering support. Listing 4.2 shows an example of Protovis code used to create the bar chart in Figure 4.8a. A redrawn version of Playfair's wheat chart [Playfair, 1801] is shown in Figure 4.8b.

4.3.2 D3

D3 (or d3.js) is a JavaScript library developed by the Stanford Visualization Group [Bostock et al., 2011; Bostock, 2011]. It binds data to the Document Object Model (DOM) and provides various data-driven transformations. Unlike traditional graphics libraries, D3 does not encapsulate the Document Object Model (DOM) of web sites in order to provide the possibility to directly manipulate it. Developing DHTML typically includes debugging of the code, using tools to inspect the scene graph of a document.



(a) A bar chart created with the Protovis code shown in Listing 4.2.



(b) A redrawn version of Playfairs wheat chart, produced with Protovis. It displays the price of wheat, weekly wages, and the reigning monarch from 1565 to 1821 [Playfair, 1801].

Figure 4.8: Two visualisations created with Protovis. The examples are extracted from Protovis [2011].

Using D3, this procedure is still possible, because data is bound to arbitrary document elements. According to Bostock et al. [2011], in D3 “the document is the scene graph”. Two examples created with D3 can be seen in Figure 4.9.

D3 can be compared to other document transformers such as jQuery, CSS, and XSLT. It is more a visualisation “kernel” than a framework. D3 is closely related to Protovis (see Section 4.3.1), a previous project of the Stanford Visualization Group. However, the difference is that the focus of Protovis is on a declaration of static scenes, whereas D3 creates dynamic content using transformations. [Bostock et al., 2011]

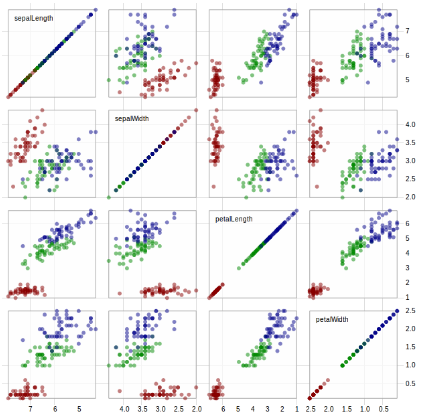
4.3.3 GAV Flash Toolkit

The GAV Flash Toolkit is an interactive visualisation framework, allowing developers to integrate a broad range of visualisations into their applications [NComVA, 2011b; NCVA, 2011a]. GAV Flash is programmed using Adobe Flex and includes the same components used in the Statistics eXplorer (see Section 4.1.2). These components are: choropleth maps, colour legends, data grids, distribution plots, histograms, parallel coordinates, scatter plots, scatter matrices, table lenses, and time graphs. The general architecture of all components used by a GAV Flash application can be seen in Figure 4.10. The toolkit supports various data sources, such as unicode text files (`.txt`), Microsoft Excel files, the World Bank Database API, and partly PC-Axis, and SDMX files. One of statistics eXplorer’s main features, the story-telling functionality, is also included within GAV Flash. Using this functionality enables users to report interesting findings to other users.

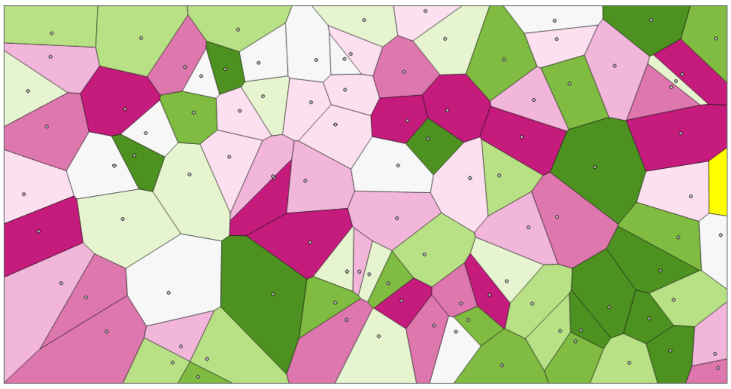
Unfortunately, the GAV Flash Toolkit cannot be downloaded from the NComVA’s web site and therefore cannot be tested.

4.3.4 JavaScript InfoVis Toolkit (JIT)

The JavaScript InfoVis Toolkit [SenchaLabs, 2011] is a open source project by Nicolas Garcia Belmonte. Using the toolkit enables developers to create highly interactive data visualisations. It implements multiple visualisations used in the field of information visualisation such as area charts, pie charts, treemaps,



(a) A scatterplot matrix displaying the well-known Iris flower dataset collected by the botanist Edgar Anderson in 1935 and introduced by Fisher [1936].



(b) A Voronoi diagram created with D3.

Figure 4.9: Visualisations created with the JavaScript library D3. The examples and the corresponding images are taken from Bostock [2011].

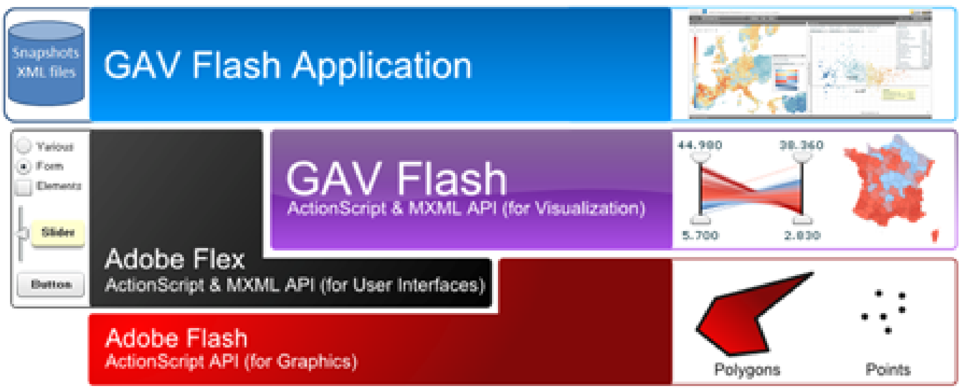
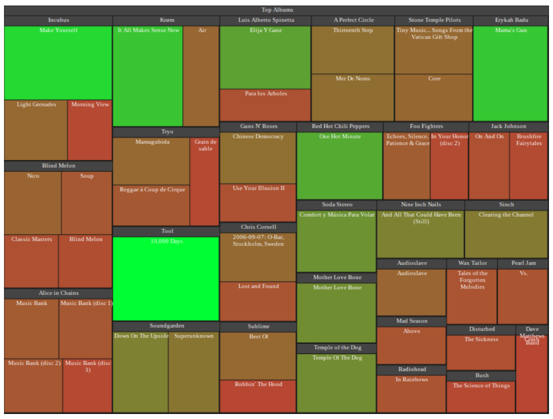
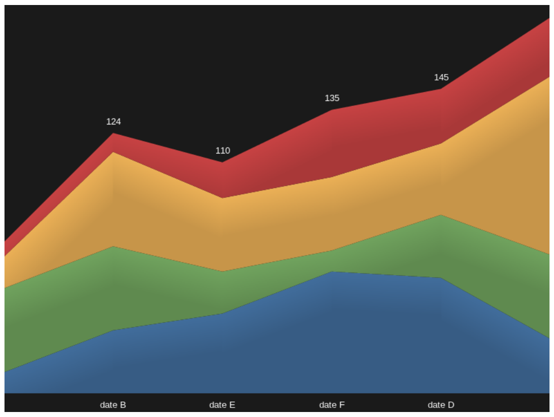

Snapshots XML files
GAV Flash Application
GAV Flash
ActionScript & MXML API (for Visualization)
Adobe Flex
ActionScript & MXML API (for User Interfaces)
Adobe Flash
ActionScript API (for Graphics)
Slider
Button
Polygons
Points

Figure 4.10: The general architecture of the components Adobe Flash, Adobe Flex, GAV Flash, and a GAV Flash application. This image is taken from NCVA [2011a].



(a) A squarified treemap displaying CDs from various interprets.



(b) A stacked area chart created using random data.

Figure 4.11: Examples of visualisations created with the JavaScript InfoVis Toolkit (JIT). The images are extracted from SenchaLabs [2011].

force-directed graphs, and many more. Examples of visualisations created with the JavaScript InfoVis Toolkit can be seen in Figure 4.11.

According to Nicolas Garcia Belmonte [2011], the JavaScript InfoVis Toolkit was acquired in 2010 by the Sencha Labs Foundation. However, the project remains open source and is still maintained by Nicolas Garcia Belmonte.

4.3.5 amCharts

amCharts [2011] offers a set of charts written in JavaScript and Flash. Currently, two different products are available: the amCharts bundle, and stock charts. The amCharts bundle consists of multiple visualisations such as bar charts, line charts, area charts, candlestick charts, pie charts, star plots, and many more. In addition, the amCharts bundle offers a number of visualisations written in Flash and Flex. However, amCharts has discontinued the development of its Flash-based charts. The second product, the stock charts, display time-based data such as financial data. An example of a stock chart can be seen in Figure 4.12.

The use of amCharts is free and all products can be downloaded from the corresponding web site. However, the products contain a small link to the amCharts' web site at the top left corner of each visualisation. In order to obtain a version without links, a commercial license has to be purchased.

4.3.6 Axiis

Axiis [2011] is a data visualisation framework written in Flex. It is built upon the Degrafa [2011] graphics framework and includes both pre-built visualisation components and classes which enable developers to create their own visualisations. Axiis is an open source project maintained by Tom Gonzalez, and Michael VanDaniker. The project started in 2009 with the aim to provide developers a framework for creating complex, interactive data visualisations.

Axiis is designed with its focus on enabling developers to write elegant code. Additionally, Axiis reduces the amount of code using inline Flex Binding. It is a granular framework which gives developers the possibility to combine building blocks in order to produce complex output.

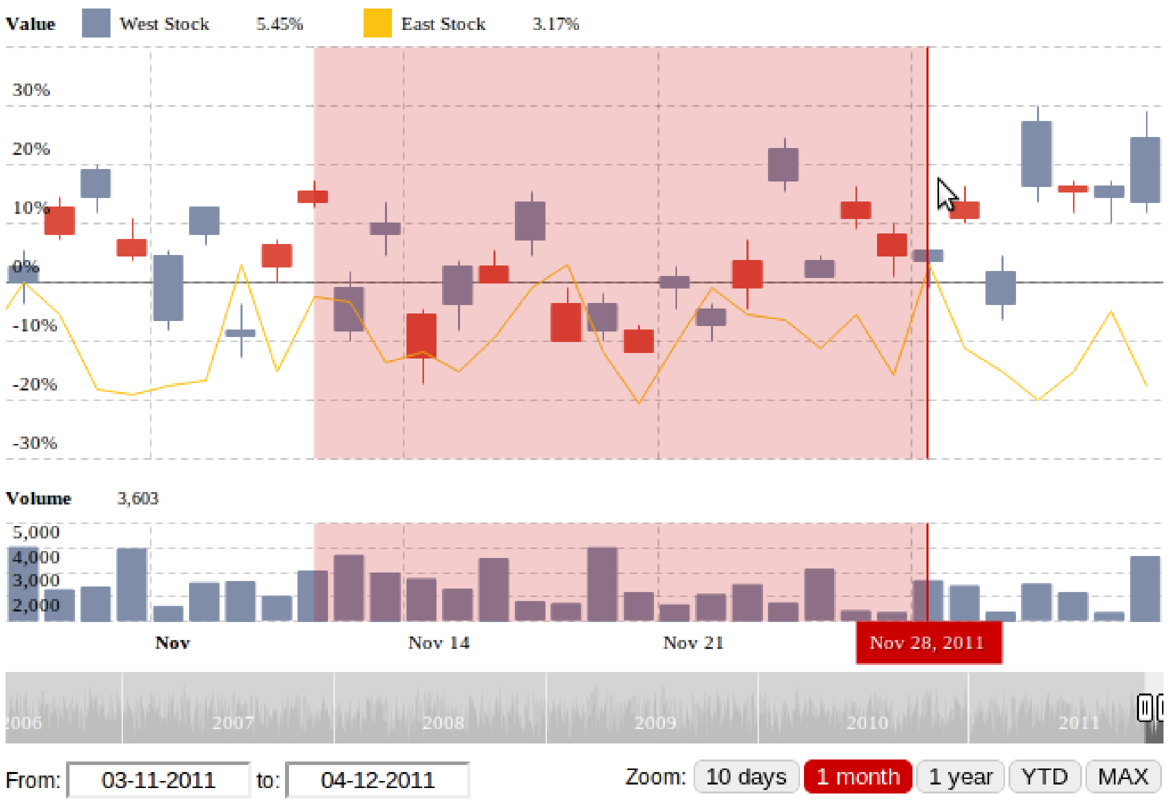
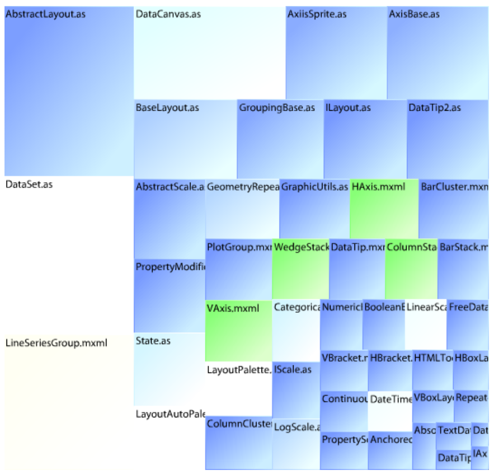
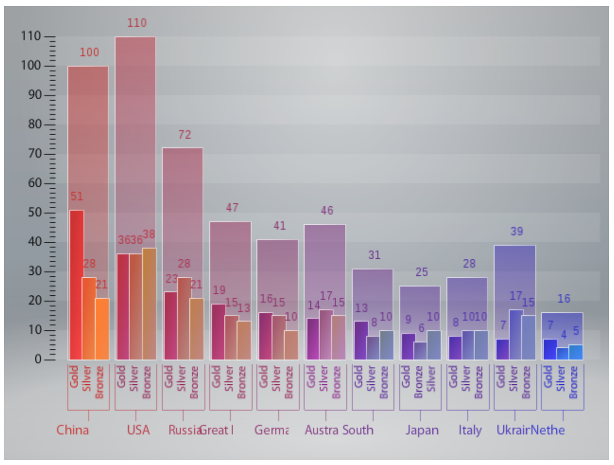


Figure 4.12: A stock chart created with amCharts. It displays time-based data and consists of several visualisation types such as a bar chart, a line chart, and a candlestick chart. The red area displays a selection.



(a) A treemap visualising the number of lines in each file within the Axiis framework.



(b) This figure shows a HCluster Column visualising the number of medals per nation in the 2008 Olympics. A HCluster Column is a bar chart which shows an additional histogram.

Figure 4.13: Examples created with Axiis. The images are taken from Axiis [2011].

Chapter 5

Liquid Diagrams Framework Version 2.0

Liquid Diagrams (LD) is a suite of information visualisation gadgets, developed using Flex [Andrews and Lessacher, 2010]. The Liquid Diagrams framework offers a set of functions for creating highly interactive visualisations. The functionality is carefully designed in order to not contain any visualisation logic. Instead, simple functionality such as functions for drawing lines, axes, legends, and data points is provided. The Liquid Diagrams framework was conceived by Keith Andrews and implemented by Martin Lessacher in a course at Graz University of Technology in 2009 [Lessacher, 2009]. Further development was performed in Lessacher's Master's thesis in 2010 [Lessacher, 2010].

Version 2.0 denotes the current state of the Liquid Diagrams framework. Version 1.0 denotes the state of the framework after the work of Lessacher [2010] was finished. Currently, there are three variants of the Liquid Diagrams framework:

1. **Gadget version:** The gadget version consists of gadgets which can be embedded into a spreadsheet on Google Docs [Google, 2011c]. It uses the Google Visualization API to retrieve the data from a spreadsheet. More details about the online version of the Liquid Diagrams framework can be seen in Section 5.2.
2. **Standalone version:** The standalone version allows the local execution of every gadget in the framework. Here, the visualisations are installed locally on the user's computer. No active internet connection is needed. The standalone version is covered in Section 5.3.
3. **Cookie version:** This version of the Liquid Diagrams framework works exactly like the gadget version. The difference between both versions is that the cookie version does not retrieve and save its settings using the Google gadget settings. Instead, the settings are saved using cookies (see Section 5.4).

5.1 Data Formats

Each visualisation in the framework requires its data to be specified according to certain criteria, in order to display it properly. Here, a distinction between five different data formats is made. The only data format which was modified in version 2.0 of the framework is the geo data format. The others remain as described by Lessacher [2010]. Table 5.1 summarises which data format is used by which visualisation.

5.1.1 LD Chart Data Format

The chart data format is used in the visualisations line chart, bar chart, and area chart. Here, the first column specifies the values shown on the x-axis. The first row contains the name of each data entity. An

Visualisation	Chart Data Format	Pie Data Format	Nightingale Data Format	Multidimensional Data Format	Hierarchical Data Format	Geo Data Format
Line Chart	√					
Bar Chart	√					
Area Chart	√					
Pie Chart		√				
Polar Area Diagram			√			
Bat's Wing Diagram			√			
Parallel Coordinates				√		
Star Plot				√		
Similarity Map				√		
Treemap					√	
Voronoi Treemap					√	
Heatmap						√

Table 5.1: A summary of the data formats used by the visualisations of the framework.

Year	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007
Lena	376	449	512	554	566	677	745	791	911	960	843
Leonie	0	0	0	158	242	453	661	851	966	942	822
Anna	973	1057	968	1042	1003	1016	923	916	883	871	821
Sarah	1063	925	958	1014	929	996	965	879	848	935	804
Julia	1491	1431	1322	1239	1020	934	809	835	815	760	748

Table 5.2: The LD chart data format. The first row specifies the x-axis ticks and the first column contains the names of the data entities.

example of this data format is shown in Table 5.2.

5.1.2 LD Pie Data Format

The pie chart visualisation uses the pie data format. An example of it can be seen in Table 5.3. Here, the name of the data entities are specified using the first column and the rest of the columns, starting with the second column contain data values visualised within the visualisation. The first row contains the names of the dimensions.

5.1.3 LD Nightingale Data Format

The data format used by the bat's wing diagram and the polar area diagram is similar to the pie data format. However, both diagrams are designed to display changes in the data over time. To adapt the pie data format to represent time, it is simply necessary to insert date values into the first column. An example of this data format can be seen in Table 5.4.

5.1.4 LD Multidimensional Data Format

The visualisations star plot, parallel coordinates, and similarity map use the multidimensional data format, shown in Table 5.5. The first column contains the names of the data entities. The names of the data dimensions are specified in the first row. In the star plot and parallel coordinates visualisations, the data dimensions are represented using axes. The values for attributes can be specified using numbers or strings.

World Regions	Population	Internet Users	Usage Growth
Africa	955206348	4514400	1031.2%
Asia	3776181949	114304000	406.1%
Europe	800401065	105096093	266.0%
Middle East	197090443	3284800	1176.8%
North America	337167248	108096800	129.6%
Latin America	576091673	18068919	669.3%
Oceania	33981562	7620480	165.1%

Table 5.3: The pie data format used by the pie chart visualisation.

Period	Other Causes	Wounds	Disease
1854-04-01	30	0	30
1854-05-01	20	0	30
1854-06-01	10	0	20
1854-07-01	20	0	100
1854-08-01	30	0	350
1854-09-01	60	30	330
1854-10-01	100	95	300
1854-11-01	90	180	350
1854-12-01	100	90	500
1855-01-01	200	80	800
1855-02-01	220	60	550
1855-03-01	120	40	300

Table 5.4: The Nightingale data format used by the bat's wing diagram and the polar area diagram.

Name	Manufacturer	Type	Calories	Protein (g)
100% Bran	N	cold	70	4
Almond Delight	R	cold	110	2
Apple Cinnamon Cheerios	G	cold	110	2
Apple Jacks	K	cold	110	2
Cap'n'Crunch	Q	cold	120	1
Cheerios	G	cold	110	6
Cream of Wheat (Quick)	N	hot	100	3

Table 5.5: Part of the well-known cereal dataset [StatLib, 2011] as an example of the multidimensional data format.

District	**Province**	**Area (ha)**	**Population**	**Density (per ha)**	**Foreigners**
Eisenstadt	Burgenland	4284.34	12744	3.0	1037
Klagenfurt	Carinthia	12003.16	93478	7.8	9411
Salzburg	Salzburg	6563.63	147732	22.5	30191
Graz	Styria	12748.16	253994	19.9	36145
Graz-Umgebung	Styria	110292.57	141226	1.3	6292
Wien	Wien	41464.84	1687271	40.7	339134

Table 5.6: An example of the hierarchical data format. The first column specifies the names of the data entities, the second column contains the name of the parent of each data item.

5.1.5 LD Hierarchical Data Format

The hierarchical data format shown in Table 5.6 is required by the visualisations treemap and Voronoi treemap. This data format specifies a data entity in each row. The first row contains the names of the data dimensions. The first column contains the names of the data entities. The second column specifies the name of the parent for each data entity. If a field in the second column is left blank, the gadget assumes that this entity is a top-level entity and therefore it is added to the root of the hierarchy. All further columns, starting with the third column are value columns, displayable within the visualisation.

5.1.6 LD Geo Data Format

The data for the heatmap gadget has to be specified, using the geo data format. Examples of this format can be seen in Table 5.8 and 5.7. It underwent some modifications in version 2.0 of the Liquid Diagrams framework. With this data format, a geographic area can be specified in two ways. First, in the one-column data format, the area is identified using the ISO-3166 ALPHA-2 code [ISO, 2011] of the parent area, followed by a dash ("-") and the ISO-3166 ALPHA-2 code of the child area. An example of this data format can be seen in Table 5.7.

The second data format is the two-column data format. An example can be seen in Table 5.8. Here, the areas are identified by the first column, specifying the ISO-3166 ALPHA-2 code of the parent, together with the second column, specifying the ISO-3166 ALPHA-2 code of the child. In order to be recognised as the two-column input format, the value in the first row of the second column must be named "ld-Child". Otherwise, the gadget assumes that the one-column input format is being used.

The second modification concerns the names of areas within the visualisation. The possibility was created to change the displayed name of an area within the visualisation. Normally, these names come from the loaded SVG shape files. They can be changed by inserting a data column, specifying the names. Here, the first value of the column must be "ld-Name" in order for it to be recognised properly. This modification is available in both data formats.

5.2 LD Gadget Version

A visualisation in the gadget version of Liquid Diagrams consists of a SWF file (containing the main body of the visualisation), an XML config file, some JavaScript code, and a data source. Currently, the only implemented data source is Google Spreadsheets. However, Liquid Diagrams is designed to be able to support various data sources. The only available limitation concerns the use of JavaScript, which needs to be supported by the data source.

Region Code	ld-Name	Car Registration Prefix	Population
AT-6-FB	Feldbach	FB	67,234.00
AT-6-G	Graz	G	257,328.00
AT-6-RA	Radkersburg	RA	23,044.00
AT-6-LB	Leibnitz	LB	77,135.00
AT-6-DL	Deutschlandsberg	DL	60,920.00
AT-6-VO	Voitsberg	VO	52,471.00
AT-6-GU	Graz Umgebung	GU	141,977.00
AT-6-WZ	Weiz	WZ	87,190.00

Table 5.7: The One-Column Geo Data Format. This data format is used by the heatmap visualisation. The areas are specified by only one column. A dash ("-") combines the ISO-3166 ALPHA-2 code [ISO, 2011] of the parent and the corresponding code of the child.

Region Code	ld-Child	ld-Name	Car Registration Prefix	Population
AT-6	FB	Feldbach	FB	67,234.00
AT-6	G	Graz	G	257,328.00
AT-6	RA	Radkersburg	RA	23,044.00
AT-6	LB	Leibnitz	LB	77,135.00
AT-6	DL	Deutschlandsberg	DL	60,920.00
AT-6	VO	Voitsberg	VO	52,471.00
AT-6	GU	Graz Umgebung	GU	141,977.00
AT-6	WZ	Weiz	WZ	87,190.00

Table 5.8: The Two-Column Geo Data Format. One of the two Geo Data Formats used by the heatmap gadget. Geographical areas are specified using the two-column format. In this case the first column specifies the parent and the second the child. The name of the second column must be "ld-Child". The third column with the name "ld-Name" specifies the names of the corresponding areas. This column is optional.

Google Spreadsheets is Goolge's answer to Microsoft Excel. It is an online spreadsheet application which allows the embedding of charts and gadgets to visualise one's own data. A gadget is a simple JavaScript and HTML application which can be created using the Google Gadgets API. The online version of Liquid Diagrams currently implements twelve different google gadgets, one for each visualisation. A gadget consists of an XML file, containing the gadget's preferences, its content and user preferences. Liquid Diagrams inserts a visualisation within the content section of the corresponding gadget's XML file using JavaScript. In addition, the data in the spreadsheet is retrieved using JavaScript, over the Google Visualization API. It is delivered to the visualisation after the visualisation is created. The visualisation is displayed inside a Google spreadsheet, visualising its data. The code to perform these actions was written by Martin Lessacher. A detailed description of the architecture can be found in Lessacher [2010].

Using Google Spreadsheets has the advantage of user collaboration. This works, because Google provides the functionality to share spreadsheets among users. This includes gadgets, previously submitted to and approved by Google. In this way, users can collaboratively explore the visualised data.

5.3 LD Standalone Version

An important extension to the Liquid Diagrams framework is the implementation of a standalone version of the framework. For the execution of this version, no internet connection is needed, because it runs as an independent application on the computer of the user. Section 7.1.2 explains how the standalone version was implemented by extending the debug version of the framework and how the architecture changed in comparison to the initial state.

The standalone version was implemented using Adobe AIR. This technology was already discussed in Section 3.4. It enhances the framework with advanced file import functionality and the possibility to specify and register file types in the operation system.

5.3.1 Applications

In contrast to the online version, the visualisations of the standalone version have to be installed in order be executed. Therefore, the decision was made to create several different applications, which can be launched from the user's desktop. The standalone version of the Liquid Diagrams framework consists the twelve individual visualisations currently implemented, as described in Chapter 6. They work exactly like in the online version, except for the additional data import functionality.

Unfortunately, Adobe AIR does not support multiple applications within one installer and therefore it was not possible install the whole framework at once. As a consequence of this circumstance, the decision was made to create a 13^{th} application named liquiddiagrams. It is a wrapper for all twelve individual visualisations. A screenshot of it can be seen in Figure 5.1. The main window of this application consists of twelve buttons enabling the user to launch each individual visualisation in the framework. By clicking on one of these buttons, a new window appears containing the corresponding visualisation. By default, the window's width is 800 pixels and its height is 600 pixels. The application allows the user to open as many visualisations as needed. Even two ore more visualisations of the same type are possible. This feature enables users to display the same data in different visualisations and with different options in each of them. The only limitation concerns the different data formats (see Table 5.1 in Section 5.1), which have to be followed in order to display the imported data properly.

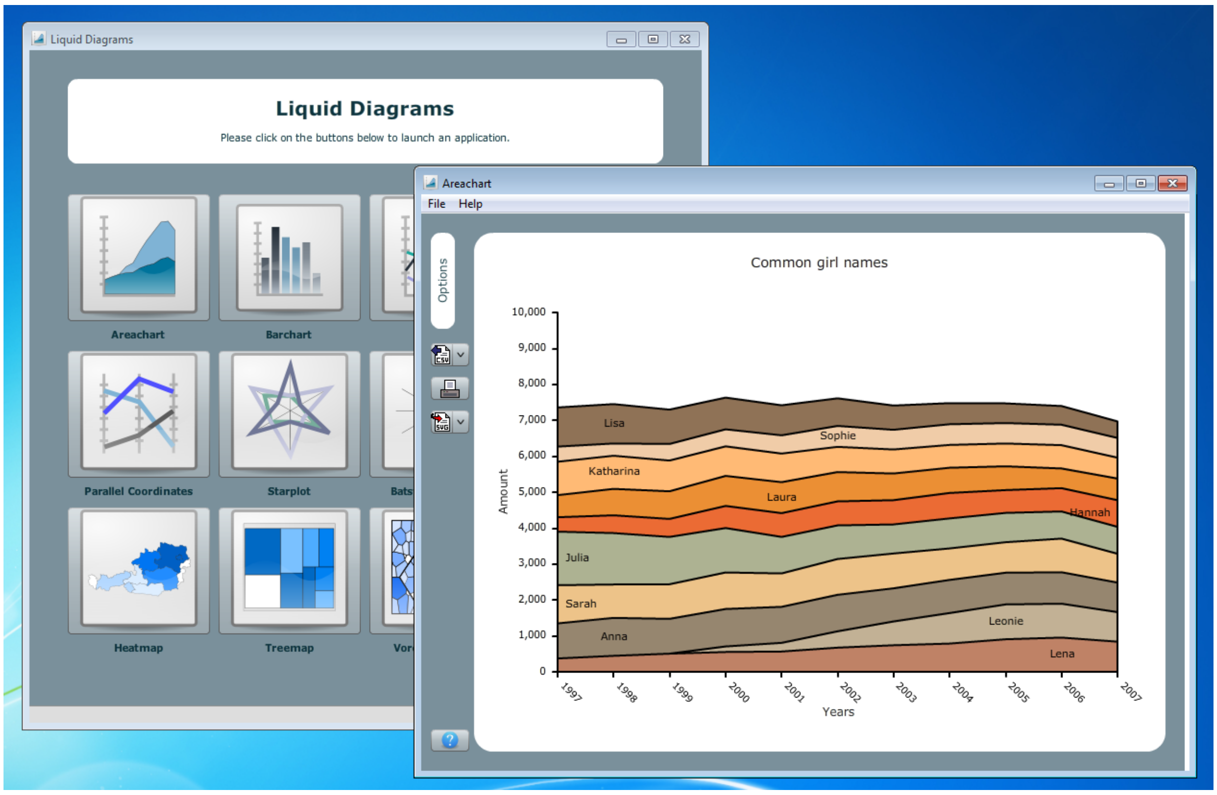


Figure 5.1: In the background the main window of the standalone version and the open area chart gadget in the front.

5.3.2 Visualisation Settings

In the gadget version of Liquid Diagrams, the settings of a visualisation come from the user preferences of the corresponding Google gadget. These are defined via the `UserPref` attributes in the XML file of the gadget. They can by reached and changed via the gadget menu, or if the gadget is moved to its own sheet via, the Edit Settings button on the upper right corner of the gadget.

For standalone version, a comfortable solution for managing the user settings was needed. An early solution was to include static default settings in the code of each visualisation. This was not a very satisfying approach, because there was no chance of changing the included settings without recompiling the visualisations.

To solve this problem, an XML file, containing the default settings of a gadget was written. This file is loaded at runtime during the initialisation process of the visualisation, at which point the corresponding settings are parsed and applied to it. In order to retrieve the settings properly, it was defined that the file which contains the default settings for a visualisation must have the name `settings.xml`. It has to be located in the corresponding resource folder of the visualisation. For example, the default settings for the bar chart visualisation can be found in `./res/barchart/settings.xml`. An example of a settings file can be seen in Listing 5.1. It shows the `settings.xml` file of the bar chart visualisation.

5.3.3 Data Import

In order to work properly, a standalone visualisation has to retrieve and subsequently visualise the retrieved data. Therefore, the fundamental question of an adequate data import functionality was raised. In the gadget version of the framework, data is retrieved from the spreadsheet via the Google Visualization

```
<data>
        <settings id="settings">
                <setting id="x_title" value="X Title"/>
                <setting id="title" value="Diagram Title"/>
                <setting id="number_display_format" value="1,000.00"/>
                <setting id="line_break_signs" value=":-/"/>
                <setting id="shorten_factor_x" value="1"/>
                <setting id="dynamic_y_axis" value="true"/>
                <setting id="legend" value="No Legend"/>
                <setting id="y_title" value="Y Title"/>
                <setting id="show_title" value="true"/>
                <setting id="show_x_title" value="true"/>
                <setting id="show_y_title" value="true"/>
                <setting id="use_ratio" value="false"/>
                <setting id="x_ratio" value="1"/>
                <setting id="y_ratio" value="1"/>
                <setting id="shorten_factor" value="1"/>
                <setting id="min_bar_width" value="10"/>
                <setting id="x_axis_label_style" value="45 Degrees"/>
                <setting id="color_scheme" value="Many"/>
                <setting id="date_display_format" value="2000-01-30"/>
        </settings>
</data>
```

Listing 5.1: The `settings.xml` file of the bar chart visualisation. It contains the visualisation's default settings loaded during its initialisation process. After the loading, the settings are applied to the visualisation.

API using JavaScript. The next step is the preprocessing of the data and finally the data is transferred to the gadget by a callback method. Lessacher [2010] explains this procedure in more detail.

In contrast to the gadget version, the data of the standalone version has to be imported using the import buttons at the left side of the visualisation, or when the liquiddiagrams application is launched using the file menu of an open window. A file chooser dialogue appears. When the user selects a data file, it is loaded and parsed by the framework. As a last step, the data import dialogue appears, where the user is able to modify parsing options before the final data is sent to the visualisation. More about the data import dialogue can be found in Section 5.5.8.

The import functionality currently supports the file formats `.csv`, `.xls` and `.ods`. By implementing this functionality, the focus was on extensibility with respect to the insertion of new file formats. In order to implement the import functionality for the file types `.xls` and `.ods`, two external libraries providing necessary functionality were used:

1. **as3xls:** The ActionScript library as3xls [2011] is used for parsing `.xls` files. Due to the fact that the original version of the library did not handle the parsing process correctly, a modified version by Wilson [2011] was used.
2. **FZip:** This library was used to extract the zipped content from `.ods` files in order to parse it. FZip was written by Wahlers and Herkender [2011].

In addition, there was a demand for sharing visualisations and settings among users like in the gadget version of the framework. To meet this demand, the import and export of project files was implemented.

5.3.4 Project File Import and Export

In the gadget version of the framework, visualisations can be shared among users by sharing a spreadsheet in Goolge Docs. Thereby, all the options are saved within the Google gadget settings. This mechanism enables any modified options to be restored when a gadget is loaded. When implementing the standalone version the decision was made that a similar functionality should be available in this version as well. That means that users should be able to save the current state of a visualisation with its corresponding settings and restore it later on. In this way, visualisations can be shared among users as well.

To meet this demand, the decision was made to implement the import and export of project files. A project file has the file extension `.ld` and is basically a `.zip` file, containing three files:

1. **context:** This file contains the context of the project file. Currently it just contains the name of the visualisation which is saved. In the future it could be extended to hold the version of the application or similar options.
2. **data.csv:** The data of the visualisation is saved within this file. It contains the filtered data currently displayed exactly when the project file was saved. For example, if some filtering options like hiding columns or rows were applied when the original data was loaded, only the filtered data values are saved in this file.
3. **settings.xml:** This file contains the settings which are active in the visualisation at the moment when the project file was exported. More about the visualisation settings can be found in Section 5.3.2.

There are two different ways to open a project file. The first is to select the option "Import Project File" in the file menu of the liquiddiagrams application. The second way is to simply open a previously exported project file from the file system. This is possible because the file type "Liquid Diagrams Project File" with its corresponding file extension (`.ld`) is registered with the operating system when the the standalone version of the Liquid Diagrams framework is installed. Adobe AIR makes this possible with an entry in the application's XML settings file. Listing 5.2 shows the XML item which has to be specified in the file, in order to register a file type for an application. A detailed discussion of the implementation of this functionality can be found in Section 8.2.

5.3.5 Release

Adobe AIR applications can be installed on the desktop. To create an AIR application installer, a production build of the AIR project has to be performed. It produces a digitally signed AIR installer, enabling the application to be installed. The production build can be performed using the internal release build functionality of Adobe Flash Builder, or by using Apache ANT. Liquid Diagrams uses Apache ANT. This has the advantage that the installers of all applications can be created by a simple double-click.

The functionality for creating an installer consists of two XML files, containing the ANT content and a Windows `.bat` file. The first XML file (shown in Listing 5.3) is responsible for all cleaning actions performed before the building process starts. It deletes the folder containing the installers built in a previous release, and recreates it again. The second XML file, seen in Listing 5.4, creates an installer. This is done using a self-signed certificate. Additionally, the file ensures that all necessary files needed by the application are included within the installer. To simplify the procedure of calling this files, a Windows `.bat` file was created. Instead of manually using the command console, this file offers the possibility of executing the files by performing a simple double-click. The `.bat` file can be seen in Listing 5.5. In order to create the installers for all applications of the framework at once, an additional `.bat` file (shown in Listing 5.6) is available. It executes the ANT files of all applications.

```xml
[...]

<fileTypes>
  <fileType>
    <name>ld.project.file</name>
    <extension>ld</extension>
    <!-- <description></description> -->
    <contentType>application/zip</contentType>
    <icon>
      <image16x16>areachart/icons/icon-areachart-16.png</image16x16>
      <image32x32>areachart/icons/icon-areachart-32.png</image32x32>
      <image48x48>areachart/icons/icon-areachart-48.png</image48x48>
      <image128x128>areachart/icons/icon-areachart-128.png</image128x128>
    </icon>

  </fileType>
</fileTypes>

[...]
```

Listing 5.2: This listing shows an entry in the XML settings file of an Adobe AIR application. It is responsible for registering a file type on the operation system.

```xml
<project>

    <!-- SDK properties -->
    <property name="SDK_HOME" value="C:/Program Files/Adobe/Adobe Flash Builder 4/
        sdks/4.1.0/"/>
    <property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
    <property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>

    <!-- Project properties -->
    <property name="PROJ_ROOT_DIR" value=".."/>
    <property name="release" location="${PROJ_ROOT_DIR}/release"/>

    <target name="clean" description="clean up">
        <delete dir="${release}"/>
    </target>

    <target name="init" depends="clean">
        <mkdir dir="${release}"/>
    </target>

</project>
```

Listing 5.3: This XML file is responsible for cleaning up the release folder. This is done using Apache ANT.

```
<project>
    <!-- SDK properties -->
    <property name="SDK_HOME" value="C:/Program Files/Adobe/Adobe Flash Builder 4/
        sdks/4.1.0/"/>
    <property name="ADL" value="${SDK_HOME}/bin/adl.exe"/>
    <property name="ADT.JAR" value="${SDK_HOME}/lib/adt.jar"/>

    <!-- Project properties -->
    <property name="PROJ_ROOT_DIR" value=".."/>
    <property name="APP_NAME" value="liquiddiagrams"/>
    <property name="APP_ROOT_DIR" value="${PROJ_ROOT_DIR}/src"/>
    <property name="APP_ROOT_FILE" value="${APP_NAME}.mxml"/>
    <property name="APP_DESCRIPTOR" value="${APP_ROOT_DIR}/${APP_NAME}-app.xml"/>
    <property name="AIR_NAME" value="${APP_NAME}.air"/>
    <property name="release" location="${PROJ_ROOT_DIR}/release"/>
    <property name="assets" location="${PROJ_ROOT_DIR}/res"/>
    <property name="STORETYPE" value="pkcs12"/>
    <property name="KEYSTORE" value="${PROJ_ROOT_DIR}/certificate.p12"/>
    <property name="STOREPASS" value="not-the-real-password"/>
    <property name="BUILD_DIR" location="${PROJ_ROOT_DIR}/bin-debug"/>
    <property name="APP_BUILD_FILE" value="${APP_NAME}.swf"/>

    <target name="test">
        <exec executable="${ADL}">
            <arg value="${APP_DESCRIPTOR}"/>
            <arg value="${APP_ROOT_DIR}"/>
        </exec>
    </target>

    <target name="package">
        <java jar="${ADT.JAR}" fork="true" failonerror="true">
            <arg value="-package"/>
            <arg value="-tsa"/>
            <arg value="none"/>
            <arg value="-storetype"/>
            <arg value="${STORETYPE}"/>
            <arg value="-keystore"/>
            <arg value="${KEYSTORE}"/>
            <arg value="-storepass"/>
            <arg value="${STOREPASS}"/>
            <arg value="${release}/${AIR_NAME}"/>
            <arg value="${APP_DESCRIPTOR}"/>
            <arg value="-C"/>
            <arg value="${BUILD_DIR}"/>
            <arg value="${APP_BUILD_FILE}"/>
            <arg value="-C"/>
            <arg value="${assets}"/>
            <arg value="/"/>
        </java>
    </target>
</project>
```

Listing 5.4: This file creates the installer for the liquiddiagrams application. It signs the installer with a certificate and includes all necessary files for the execution of the application within the installer.

```
CALL ant -f ant/common.xml init
CALL ant -f ant/build-liquiddiagrams.xml package
PAUSE
```

Listing 5.5: This Windows `.bat` file executes the files shown in Listings 5.3 and 5.4.

```
CALL ant -f ant/common.xml init
CALL ant -f ant/build-areachart.xml package
CALL ant -f ant/build-barchart.xml package
CALL ant -f ant/build-batswing.xml package
CALL ant -f ant/build-heatmap.xml package
CALL ant -f ant/build-linechart.xml package
CALL ant -f ant/build-parcoord.xml package
CALL ant -f ant/build-piechart.xml package
CALL ant -f ant/build-polararea.xml package
CALL ant -f ant/build-simmap.xml package
CALL ant -f ant/build-starplot.xml package
CALL ant -f ant/build-treemap.xml package
CALL ant -f ant/build-voronoi.xml package
CALL ant -f ant/build-liquiddiagrams.xml package
PAUSE
```

Listing 5.6: This Windows `.bat` file creates the installers of all applications of Liquid Diagrams. This is done by executing the corresponding Apache ANT files.

5.4 LD Cookie Version

According to Barth [2011] cookies are used for a web server to send state information to a user's browser and for the browser to return it to the originating server again. With the help of cookies, it is possible to preserve user-related data during navigation or across multiple visits of a website. This mechanism can be used to save data related to visualisations in the Liquid Diagrams framework.

As mentioned before in Section 5.3.2 the gadget version of the framework offers an instrument for saving visualisation-related data in the Google gadgets settings. However, there are some negative aspects of this approach:

- The only way to save the options of a visualisation is to open the Google gadget settings window to manually change the settings. This causes a reload of the gadget, whereupon the visualisation loads with the changed settings. It is not possible to directly modify the Google gadget settings from the visualisation. This causes a major disadvantage regarding the user experience while working with a visualisation. The user constantly has to switch to the gadget settings and wait for the gadget to load. A user might find this procedure annoying and frustrating.
- Some dynamic settings of a visualisation can not be saved by the Google gadget settings. These are mainly settings which result from the visualised data. An example of such a setting would be the colour of a specific data entity or different filtering options for diverse data entities. The reason why these settings cannot be saved by the Google gadget settings, is because it is impossible to pre-estimate how many data entities and data values are contained in the data. This would be necessary because the Google gadget settings consist of a predefined number of static XML elements, contained in the gadget's XML file.

To ameliorate this situation, the decision was made to implement a version of Liquid Diagrams which saves the settings of a visualisation as a cookie. By doing that, users do not have to constantly switch to the Google gadget settings any more and it is possible to save dynamic settings.

In the cookies version, the options are saved in a cookie within the user's browser with the help of JavaScript. The SWF file calls JavaScript methods via the ActionScript interface ExternalInterface and specifies callback methods which can be called in return. The JavaScript code consists of five main methods:

1. **storeCookieSettings(application, id, cookieSettings):** This function stores all the settings for an application. The type of the visualisation is defined by the parameter `application` and the `id` specifies the cookie identifier which is necessary in order to retrieve the correct settings. The existence of this identifier is a necessary evil, because otherwise it would not be possible to differentiate between two different visualisations of the same type. This identifier has to be set in the Google gadget settings.

2. **storeSingleCookieSetting(application, id, name, value):** Stores a single setting for a visualisation. If this setting already existed, its value is overridden with the new one.

3. **deleteSingleCookieSetting(application, id, name):** This function deletes a single setting from the cookie. This is done by setting the expiration date of the cookie to the past. It is automatically deleted by the user's browser.

4. **deleteCookieForApplication(application, id):** Deletes all the entries of an application. When this procedure is finished the call `getFlexApp(application).applicationCookiesDeleted()` is triggered.

5. **getCookieSettings(application, id):** This function returns all the settings for a specified application. If settings are found, it returns an array containing the settings via the call `getFlexApp(application).recieveCookieSettings(vis.settings)`. Otherwise, the gadget is notified by the JavaScript call `getFlexApp(application).noCookieFound()`.

In order to work properly, a gadget in the cookies version has to retrieve factory default settings on first startup. Therefore, the settings mechanism described in Section 5.3.2 was used. More about the implementation of the cookies version can be found in Section 8.2.

Unfortunately, this approach has some disadvantages as well. The first concerns the size of a cookie. According to the RFC standard, cookies have a limited size of at least 4096 bytes [Barth, 2011] which means that if the cookie option is used with multiple visualisations it could happen that the size of the cookie is exceeded. The reason for that is because all the gadgets of the online version of the framework run on the same host and therefore share the same cookie. If the size of the cookie is exceeded, the new entries will replace the oldest ones. Another disadvantage is that cookies are not shareable among users or browsers. This makes collaboration between users on a shared visualisation almost impossible and therefore one of the main features of the Liquid Diagrams framework is no longer accessible.

Both the cookies version of a gadget and the version where the options are saved in the Google gadget settings are available in parallel. This was done by creating two different Google gadget XML files. One is for the old version and the other one for the cookies version. If one of them is inserted in a spreadsheet on Google docs the same SWF file, containing the visualisation is loaded. The only difference of this procedure is that the initial options for the Flash file contain a cookie flag which is set in the cookies version. Using this flag in the ActionScript code, it can be easily differentiated between the two versions. The name of the XML file for the cookies version differs from the name of the XML

Visualisation	**General**	**Fonts**	**Labels**	**Axes**	**3D**	**Interaction**	**Map**	**Drawing**	**Algorithm**
Area Chart	√	√	√	√					
Bar Chart	√	√		√					
Bat's Wing Diagram	√	√	√			√		√	
Heatmap	√	√	√		√		√		
Line Chart	√	√		√					
Parallel Coordinates	√	√	√			√		√	
Pie Chart	√	√	√		√				
Polar Area Diagram	√	√	√					√	
Similarity Map	√	√	√					√	√
Star Plot	√	√	√			√		√	
Treemap	√	√	√					√	
Voronoi Treemap	√	√							√

Table 5.9: The tabs included in the Options Panel of each visualisation.

file of the old version by the additional suffix `-cookies`. For example, the name of the XML file for the voronoi treemap visualisation would be `voronoi-cookies.xml`. In this way, the users can decide which version to insert into their spreadsheet.

5.5 Components Shared Between Visualisations

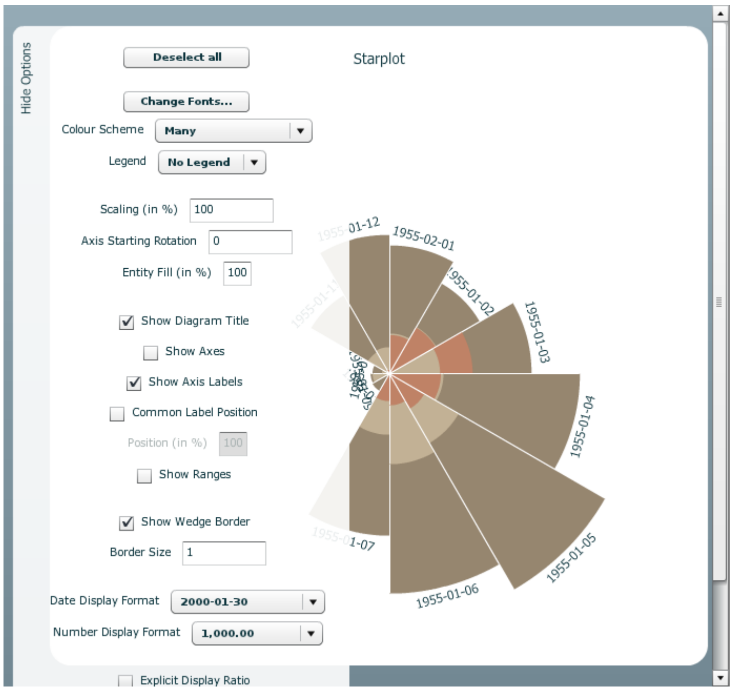
This section denotes both newly created components added to the framework, and components which already existed in version 1.0, which underwent major modifications. This includes components such as the improved version of the Options Panel, Import, Export and Print buttons, the Import Dialogue, and the About Panel.

5.5.1 Options Panel Menu Component

The new menu component is one of the most important changes in the Liquid Diagrams framework with regard to the user interface. In the previous version of the framework, Lessacher [2010] implemented the options in a way that they do not require much space when they are not currently needed. This was realised by adding a panel on the left side of each gadget. When this panel is clicked, it opens atop the visualisation, and options appear inside of the resized panel. An example of this implementation can be seen in Figure 5.2. It shows the resized Options Panel of the polar area diagram.

However, Figure 5.2 reveals a disadvantage of this approach. If too many different options are available for the visualisation, a scrollbar appears and consequently users have to scroll to reach the options at the bottom of the Options Panel. To overcome this issue, the decision was made to integrate vertical tabs into the Options Panel. These tabs group the option elements into categories which differ from gadget to gadget. In this way, it is ensured that the number of visible options does not overfill the available space. Figure 5.3 shows the Options Panel in the polar area diagram after the modifications were done.

Available tabs are General, Fonts, Labels, Axes, 3D, Interaction, Map, Drawing, and Algorithm. Table 5.9 shows which visualisation includes which tabs in its menu component. As can be seen, the tabs General and Fonts are included within every visualisation.



Hide Options

Deselect all

Change Fonts...

Colour Scheme: Many

Legend: No Legend

Scaling (in %): 100

Axis Starting Rotation: 0

Entity Fill (in %): 100

[✓] Show Diagram Title

[ ] Show Axes

[✓] Show Axis Labels

[ ] Common Label Position

Position (in %): 100

[ ] Show Ranges

[✓] Show Wedge Border

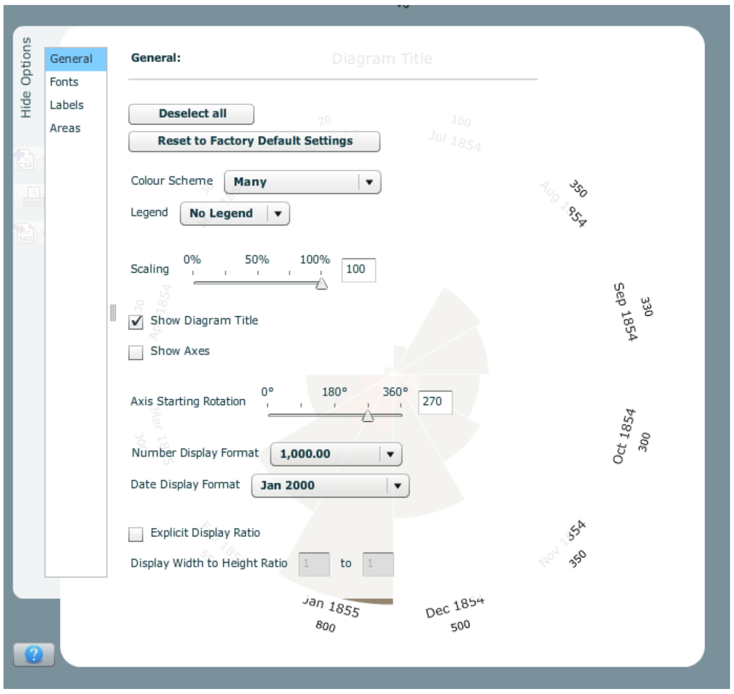
Border Size: 1

Date Display Format: 2000-01-30

Number Display Format: 1,000.00

[ ] Explicit Display Ratio

Figure 5.2: The old Options Panel displaying the options of the polar area diagram. Unfortunately, the window size is too small to display all the options, and a scrollbar is shown at the right side of the visualisation.



Hide Options

General
Fonts
Labels
Areas

General:

Deselect all

Reset to Factory Default Settings

Colour Scheme: Many

Legend: No Legend

Scaling: 0% 50% 100% 100

[✓] Show Diagram Title

[ ] Show Axes

Axis Starting Rotation: 0° 180° 360° 270

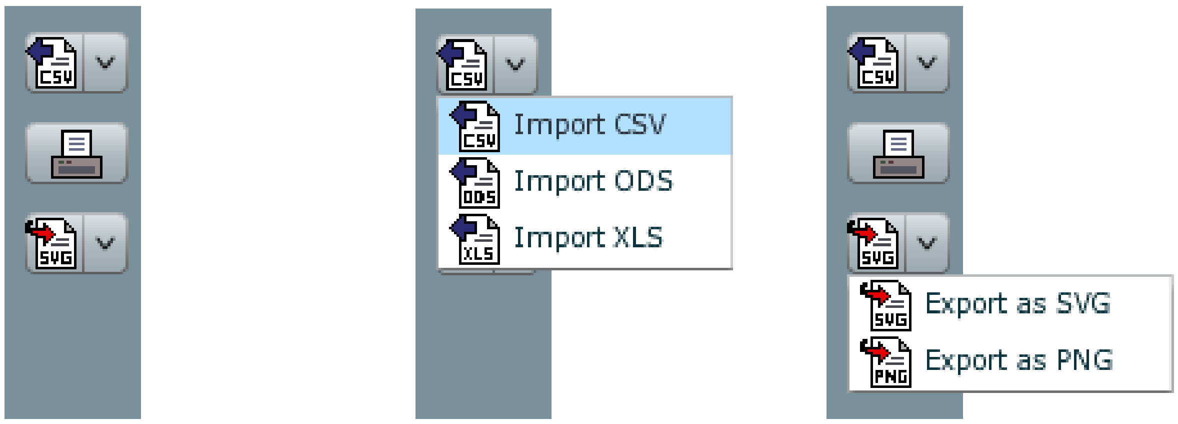
Number Display Format: 1,000.00

Date Display Format: Jan 2000

[ ] Explicit Display Ratio

Display Width to Height Ratio: 1 to 1

Figure 5.3: The improved Options Panel displaying the General tab of the polar area diagram. The use of tabs ensures that the user interface elements do not overfill the available space.



(a) The Import button on the top, the Print button in the middle and the Export button on the bottom.

(b) The Import button in its expanded state. Users can decide which file type to import.

(c) The expanded state of the Export button. The available export file types are SVG and PNG.

Figure 5.4: The new Import, Print, and Export buttons. Figure (a) shows the buttons in their normal state, (b) displays the data import formats and (c) shows the export formats.

5.5.2 Import, Export and Print Buttons

In version 1.0 of the Liquid Diagrams framework, the export and print functionality was accessible via the Options Panel. It was included at the bottom of the user interface elements. Due to the fact that the number of options in the Options Panel often exceeded the available space, users had to scroll to access the export and print functionality. As a consequence, users may not notice the buttons. This functionality was moved from the Options Panel and integrated into several independent buttons.

The Export and Print buttons are available in every gadget and can be found on the left side below the Options button. In addition, the standalone version includes an Import button in every visualisation as well. As the names indicate, the buttons trigger the import, export and print functionality respectively. Figure 5.4 shows the three user interface elements:

- **Import:**
  The Import button is only available in the standalone version. A click on the button opens a file dialogue where the user can select the desired file. The default file type which can be imported is `.csv`. To switch the file type, the user has to click on the right side of the button. By doing that, a drop down menu with the entries Import CSV, Import ODS and Import XLS appears. Figure 5.4b shows the button in its expanded state.

- **Print:**
  The visualisation can be printed via the Print button. In Figure 5.4a it is the button in the middle between the Import and Export button. In order to print the visualisation, a printer has to be installed on the computer.

- **Export:**
  The export button is responsible for exporting the visualisation. SVG and PNG are the available export file types. They can be seen in Figure 5.4c, which shows the expanded state of the button.

5.5.3 Axes

The Liquid Diagrams framework supports two different types of axes implemented by Lessacher [2010]:

1. **Normal Axes:** This is the normal axis type used in the visualisations line chart, bar chart, and area chart. It includes an x-axis and a y-axis. Their corresponding titles can by changed by clicking on the labels and changing the text within the appearing input field. The labels of the x-axis can be displayed using four different label styles:

   (a) **Centered:** The label is displayed horizontally centred.

   (b) **Line Break:** This option enables the use of characters to split the label. If the label text is for example "20.10.2011" and the specified line break sign is a point (".") the resulting label is split twice, resulting in a label consisting of three lines.

   (c) **Vertical:** Labels are rotated by 90°in order to provide more space for more labels.

   (d) **45 Degrees:** Labels are rotated by 45°.

2. **Slider Axes:** This axes type is used in the visualisations parallel coordinates, star plot, and bat's wing diagram. Its functionality differs from the normal axes type by including more interactive elements, such as drag and drop support and axis inversion. Additionally, it can be created using a star shape, used for the star plot and the bat's wing diagram or with parallel positioning of its axes used in the parallel coordinates visualisation. The additional functionality for this axes type includes:

   (a) **Drag and Drop:** The ordering of the axes can be changed by dragging an axis to a different location. This functionality is useful to detect correlations between data items whose axes are not positioned next to each other on start-up.

   (b) **Inversion:** This axes type supports the functionality to invert axes. It can be done by clicking the small arrow at the end of an axis. Upon that, the axis switches the positions of its highest and lowest values, resulting in an inverted axis.

   (c) **Filtering:** Data entities can be filtered using the filtering mechanism of Slider axes. Filtering can be accessed by checking the filtering option on the Axes tab of the Options Panel. The filter handles on each axis can be dragged to select a range.

5.5.4 Legend

Lessacher [2010] implemented two different types of legend:

1. **Normal Legend:** This legend type is used in visualisations where a specific colour is assigned to each data entity. Here, the legend displays a rectangle, filled with the corresponding colour for each entity. On the right side of the rectangles, the name of the data entities are written. Some visualisations such as line charts and parallel coordinates support displaying of data points. Using this option, a data point is displayed between the rectangle and the name of the entity.

2. **Colour Legend:** The colour legend is used by the visualisations heatmap, treemap, and Voronoi treemap. Here, no specific colours are assigned to data entities. Instead, the varying intensity of a main colour is used to express the values of data entities. The assignment of colours can be changed by selecting a different intensity distribution method:

   (a) **Uniform Distribution:** Here, the span between the highest and the lowest value is split into several uniform pieces, each representing a colour bin.

(b) **Continuous Distribution:** This distribution does not show any colour bins. Instead, a colour gradient is used.

(c) **Quantile Distribution:** An equal number of data entities are assigned to each colour bin. This means that the intensity of the colour bins is still uniformly distributed, but the underlying data ranges represented by the bins are distorted.

5.5.5 Text Input

The standard TextInput class of Flex was extended to support a time delay before applying the inserted values. This usability extension was made because of two problems with the standard TextInput class:

1. The standard TextInput component supports a number of events which can be triggered. These events are `change`, `dataChange`, `enter` and `textInput`. Using the `enter` event, the changed content of the input field is passed to a defined function when the user presses the enter button on the keyboard. However, observations have shown that users sometimes forget to press enter. In that case the inserted values will not be set and therefore not applied to the visualisation.

2. Using the events `change` or `textInput`, the inserted values will be submitted on every content change. That means if the user types for example “100”, the defined function will be triggered three times. The first the event will contain “1”, the second time “10”, and finally the third time “100”. This behaviour results in redrawing the visualisation three times as well.

To overcome these problems, a class named MenuTextInput was implemented using a timer to trigger a delayed call of the given function. The implementation of this class can be seen in Listing 5.7, an MXML declaration of it is shown in Listing 5.8 and the corresponding function which will be triggered after the delay can be seen in Listing 5.9.

The delayed call of a defined function addresses the first problem because the user does not have to press the enter button any more to submit the inserted values to the function. In addition, the second problem is solved as well because the timer of the MenuTextInput is reset every time the user continues to provide further input. This means if the user inserts new content or changes the original content of the input field before the defined timespan of 750 milliseconds is over, the timer starts again with its original timespan. As a result, the visualisation is redrawn only once.

5.5.6 Slider

The framework was extended by combining the standard Adobe Flex slider component with the Menu TextInput component described in Section 5.5.5. This was done for every text input field with fixed and predefined minimal and maximal input values. For example, in the pie chart visualisation, an input field specifies the radius of the labels in percent in relation to the radius of the pie pieces. The corresponding range is defined to extend from 0% to 200%. Using a slider component has the advantage that the user immediately sees the available range and does not have to insert several values into a single input field to test which value is still possible. In addition, when the slider component is combined with an input field, the user can insert a desired value directly by typing it. The slider for the previously mentioned example can be seen in Figure 5.5.

5.5.7 Colour Picker

As described in Lessacher [2010], the Liquid Diagrams framework does not use the standard colour picker form the Adobe Flex framework. Instead, a colour picker written by Nuzha [2010] is used. As

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TextInput xmlns:mx="http://www.adobe.com/2006/mxml"  xmlns:martin="martin.
    diagram.controls.*">
  <mx:Script>
    <![CDATA[

      private const DELAY:Number = 750;
      private var timer:Timer = new Timer(DELAY);

      public function timed(listener:Function) : void {

        if (!timer.running) {
          timer.addEventListener(TimerEvent.TIMER, listener);
          timer.start();
        }
        else {
          timer.reset();
          timer.start();
        }
      }

      public function getTimer() : Timer {
        return timer;
      }

    ]]>
  </mx:Script>
</mx:TextInput>
```

Listing 5.7: The implementation of the text field which adds a timer to the standard text input component. The delay before the passed function is triggered is set to 750 milliseconds. Listing 5.8 shows a declaration of this component.

```
<menu:MenuTextInput id="width_ratio" width="30" text="1" enabled="false"
  change="width_ratio.timed(widthRatioKeyPressed);" />
```

Listing 5.8: The MXML statement to insert a text field with a timed function. In this case, the name of the function is `widthRatioKeyPressed`. It can be seen in Listing 5.9.

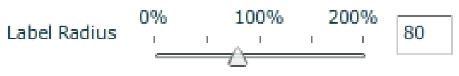


Figure 5.5: The slider component used for defining the label radius in the pie chart visualisation.

```
private function widthRatioKeyPressed(event:Event) : void {

  if ((event is KeyboardEvent && KeyboardEvent(event).charCode == 13) ||
    (event is TimerEvent)) {

    if (isNaN(Number(width_ratio.text)) || Number(width_ratio) < 0.01) {
      Alert.show("Please enter a valid number!");
      width_ratio.text = "1";
    }

    onRedraw();
    width_ratio.getTimer().stop();
  }
}
```

Listing 5.9: The implementation of a function which will be called after a short delay. If the user enters something in the input field with the id `width_ratio`, the timer will trigger a delayed call of this function. Executing this function will cause the timer to stop and a redraw of the visualisation will be triggered.

part of this work, it was extended in various ways. The original state of the colour picker can be seen in Figure 5.6a and the modified version is shown in Figure 5.6b.

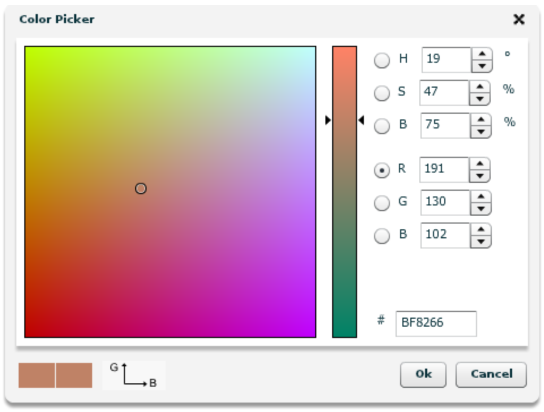
The main modification is the addition of several coloured squares at the bottom of the component. These make certain important colours directly available in a predefined palette. By pressing one of these, the colour of the square will be set within the picker. The available colours are red, green, blue, cyan, magenta, yellow, black, white, and three shades of grey.

Another enhancement is the improving of the usability of the component. This was done by adding the possibility of pressing the enter button when the input field on the right side of the component is used. When enter is pressed, the colour which is currently set is automatically applied and the colour picker window is closed. In addition, the possibility of entering three textual hexadecimal digits instead of six was implemented as well. If the user inserts for example the digits `FFF` into the input field and presses enter, the colour `#FFFFFF` (white) is applied.

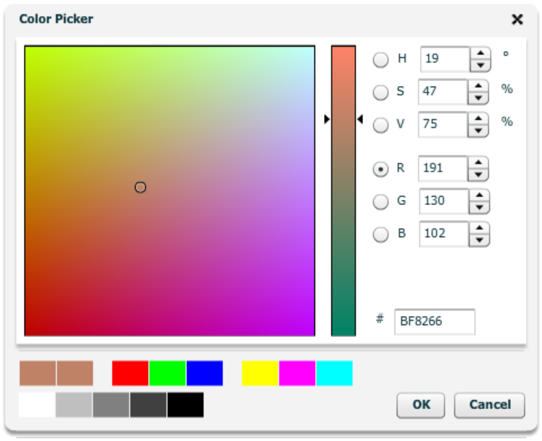
5.5.8 Import Dialogue

The import dialogue component appears when users import a data file in the standalone version of the framework. Its first purpose is to give users feedback about the data which is about to be loaded. The second purpose is to give the user the opportunity to modify the data before it is visualised.

To fulfil the first purpose, a scrollable table, containing the data is shown. In the header of each column, a drop-down box shows which data type the system identified for the data in the corresponding column. The available data types are `String`, `Number`, `Date` and `Percent`. In the case of a false identification of a data type, the user is able to set the right one by changing the data type manually. This can happen, for example, in the heatmap visualisation when a column which contains numbers should be interpreted as additional information for the areas and not as a variable mapped in the visualisation. When the user selects a data type which cannot be assigned to the column, the data type changes back to `String`. In addition, the drop-down box contains the option `Hide` which is in fact not really a data type. When the user selects it, the whole column will be excluded from the data when the import is performed. That means that the visualisation does not display the data contained in this column.



(a) The appearance of the old colour picker, implemented by Nuzha [2010].

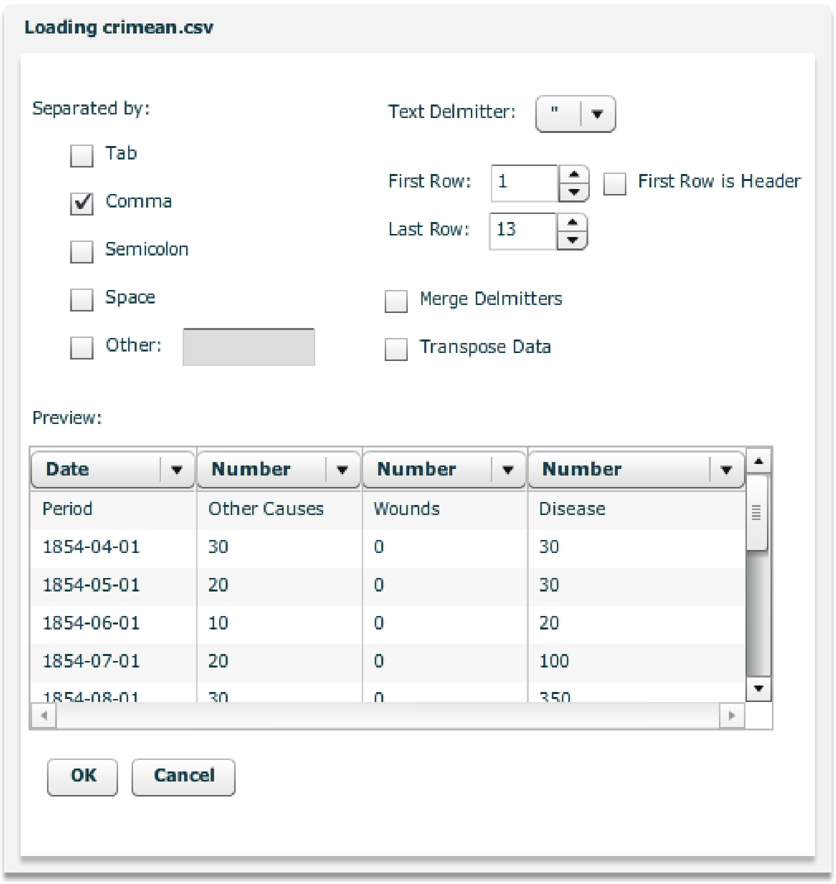


(b) The colour picker after the modifications were done. Several coloured rectangles at the bottom of the component were added. The user can directly select colours from this set of predefined primary colours and shades of grey.

Figure 5.6: The colour picker component. (a) the picker in its original state and (b) after improvements were implemented.

The import dialogue includes different possibilities to modify the imported data. Therefore, a distinction between a `.csv` file import and a `.ods` or `.xls` file import has to be made. A `.csv` file import, which can be seen in Figure 5.7, includes several check boxes defining which textual elements separate the data values from each other. The predefined textual elements are `Tab` (→), `Comma` (`,`), `Semicolon` (`;`), `Space`, and `Other`. By checking the option `Other` a text field is enabled, which can be used to define a custom separator. In addition, the user can check the check-box Merge Delimiter. This option enables the merging of the selected and defined separators. Merging means that if two or more separating textual elements are in succession without another element in between, the separators are handled as one separator. For example if the check boxes `Comma` and `Semicolon` are checked and the data input is `"A",;"B"` the result would be two data values "`A`" and "`B`". If this option were not selected, it would contain a third empty data value. Due to the fact that textual data values in a `.csv` file are contained within a text delimiter ("`'`" or "`"`"), a drop-down box containing these options was added to the import dialogue. In an attempt to make the handling of the dialogue for the user as easy as possible, the loaded data is analysed for the available separators before the dialogue is displayed to the user. By so doing, it is possible to check the check box of the most frequent separator in advance. This is done with the most frequent textual delimiter as well.

When loading an `.ods` or an `.xls` file, none of the above mentioned options are necessary, because these file types have a different structure and are not parsed by the mentioned delimiters. An example of an `.ods` import dialogue can be seen in Figure 5.8. However, there are some options which the two import dialogues have in common. One of these options is the functionality to transpose the loaded data. Transposing means switching the rows and columns of the data. This means, for example, that the third value in the second column becomes the second value in the third column. Another option is to exclude rows from the data file. This can be done using the user interface elements First Row and Last Row at the upper right side of the dialogue. The First Row input field specifies the first data row, at which the check-box indicates if the first row should be interpreted as header. This makes sense, because the first row normally contains the column headers and most of the gadgets need them for displaying the data. With the Last Row input field rows can be removed at the end of the data file. Using one of these



Loading crimean.csv

Separated by:

- [ ] Tab
- [✓] Comma
- [ ] Semicolon
- [ ] Space
- [ ] Other:

Text Delmitter: "

First Row: 1 — [ ] First Row is Header

Last Row: 13

- [ ] Merge Delmitters
- [ ] Transpose Data

Preview:

Date	Number	Number	Number
Period	Other Causes	Wounds	Disease
1854-04-01	30	0	30
1854-05-01	20	0	30
1854-06-01	10	0	20
1854-07-01	20	0	100
1854-08-01	30	0	350

Figure 5.7: The data import dialogue which appears when importing a `.csv` file into the standalone version of the framework. The user can choose the data separator and the text delimiter, exclude specific rows, transpose the data, and change the data type.

options, the table at the bottom of the dialogue changes immediately according to the performed actions and thus gives instant feedback to the user. Like hidden columns, removed lines will not be delivered to the visualisation and for this reason not visualised.

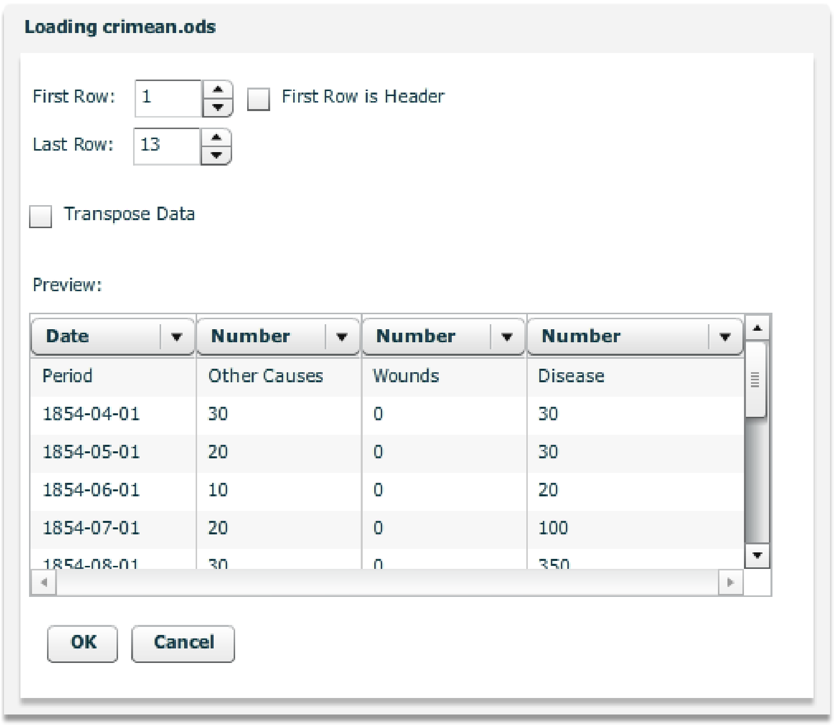
5.5.9 About Panel

The About Panel is a pop-up window which contains information about the Liquid Diagrams project. It is derived from the standard Flex Panel class and can be accessed via a button, displaying a question mark at the lower left corner of each gadget or in the standalone version by clicking on the help tab in the gadget specific menu. The graphic for the question mark on the button comes from the icon theme from the Elementary Project [2011]. A screenshot of the About Panel can be seen in Figure 5.9.

5.6 3D Functionality

The added 3D functionality is one of the largest modifications to the framework. The 3D functionality is currently used in the visualisations pie chart and heatmap, but could in principle be applied to other visualisations as well. There are two forms of 3D:

1. **3D View:** The first state simply displays a three-dimensional view of the visualisation. This is done


Loading crimean.ods

First Row: 1 ☐ First Row is Header
Last Row: 13

☐ Transpose Data

Preview:

Date	Number	Number	Number
Period	Other Causes	Wounds	Disease
1854-04-01	30	0	30
1854-05-01	20	0	30
1854-06-01	10	0	20
1854-07-01	20	0	100
1854-08-01	30	0	350

Figure 5.8: The data import dialogue when loading an `.ods` file. The option to choose the data separator and the text delimiter is disabled because the structure of the `.ods` file does not contain these options. The rest of the functionality can be performed like in the `.csv` data import dialogue.

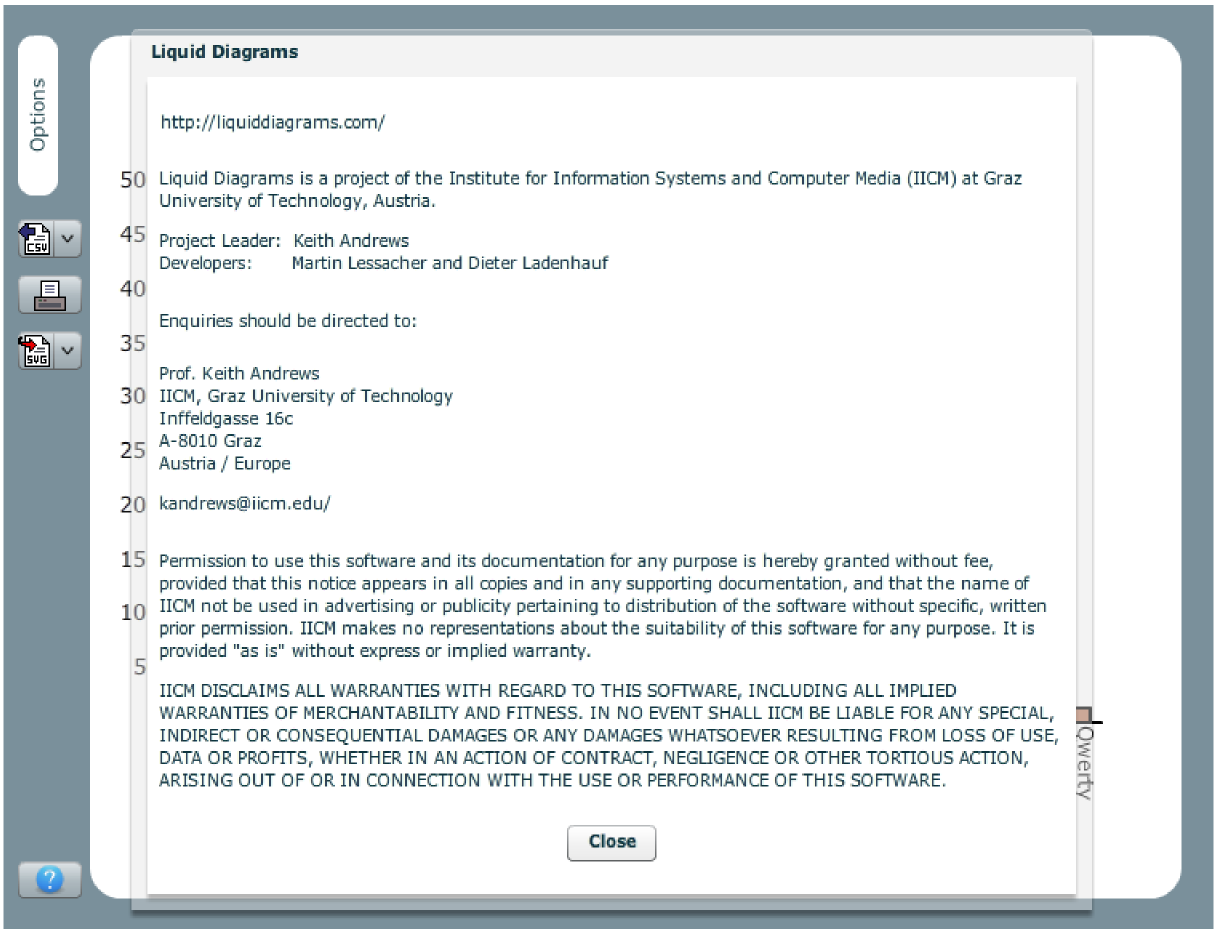
by adding a height to the items in addition to performing a rotation along the x-axis. Examples of 3D views can be seen in Figures 5.11 and 5.15.

2. **3D Extrusion:** The normal 3D view is used as a starting point and by using the height, an additional data dimension is added for the visualised data items. Examples of 3D extrusions can be seen in Figures 5.12 and 5.16.

The circumstance that each data entity can be split into several connected points is used by the 3D extension. Under the hood, the functionality saves the points of each data entity and transforms them according to the specified rotations. In the next step, a perspective projection is used to project the points from 3D space into 2D. The drawing sequence is determined with the help of a visible surface determination algorithm. Here, the painters algorithm is used. More about the implementation and the mathematical background of this functionality can be found in Section 8.3.

The 3D functionality can be modified via an additional tab in the Options Panel menu component. Figure 5.10 shows the 3D options tab. It includes the following user interface elements:

- **Enable 3D:** This user interface element is a check-box which enables the three-dimensional drawing. Its checking enables all user interface elements, except the “Extrusion Height”.
- **Enable Extrusion:** The extruded version of the visualisation is drawn if this check box is checked. In addition, it causes the slider component “Extrusion Height” to be enabled as well.
- **Show Edges:** This option determines whether the areas are separated with visible edges.



Liquid Diagrams

http://liquiddiagrams.com/

Liquid Diagrams is a project of the Institute for Information Systems and Computer Media (IICM) at Graz University of Technology, Austria.

Project Leader: Keith Andrews
Developers: Martin Lessacher and Dieter Ladenhauf

Enquiries should be directed to:

Prof. Keith Andrews
IICM, Graz University of Technology
Inffeldgasse 16c
A-8010 Graz
Austria / Europe

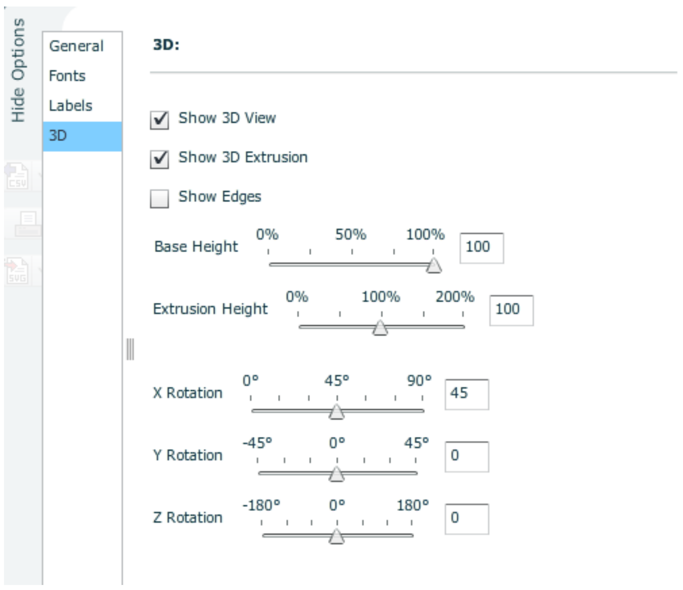
kandrews@iicm.edu/

Permission to use this software and its documentation for any purpose is hereby granted without fee, provided that this notice appears in all copies and in any supporting documentation, and that the name of IICM not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. IICM makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

IICM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL IICM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Close

Figure 5.9: The About Panel: a pop-up window that contains information about the Liquid Diagrams project.



3D:

- [✓] Show 3D View
- [✓] Show 3D Extrusion
- [ ] Show Edges

Base Height	0%	50%	100%	100
Extrusion Height	0%	100%	200%	100
X Rotation	0°	45°	90°	45
Y Rotation	-45°	0°	45°	0
Z Rotation	-180°	0°	180°	0

Figure 5.10: The 3D options tab of the menu component. The options determine the rotations, heights, and visibility of edges.

- **Base Height:** The base height determines the height of the three-dimensional objects. It can range from 0% to 100% and its default value is 100%. For this user interface element, a modified slider component (see Section 5.5.6) is used.
- **Extrusion Height:** This control sets the maximal additional height of data items when an extrusion is performed. This means that the extrusion height of the data item with the highest data value is set according to the percentage value defined by this control. The extrusion height of the other data items is set in relation to the value of this item. The final height of a data item is calculated by adding its extrusion height to its corresponding base height. The minimum value is 0%, its maximum is 200% and its factory default value is 100%.
- **X Rotation:** This option sets the rotation of the data items along the x-axis. The range which can be applied ranges from 0°to 90°. The default value is 45°.
- **Y Rotation:** The rotation in the y-direction. The possible values range from -45°to 45°and the factory default value is zero degree.
- **Z Rotation:** The rotation along the z-axis. By specifying a minimum value of -180°and a maximum of 180°the whole range of 360°is available for the rotation. This user interface element is only displayed if its usage makes sense with respect to the visualisation it is contained in. This is, for example, not the case in the heatmap visualisation, because a rotation along the z-axis would mean changing the geographic direction of the map.

The pie chart was the first visualisation where the 3D extension was added. By using it, it is possible to display the areas of the chart in 3D. An example can be seen in Figure 5.11, which displays the population of the Word's regions.

The pie chart visualisation implements two moving effects: firstly visualising the rotation from the two-dimensional to the three-dimensional view (see Figure 5.13) and secondly the transition from the normal three-dimensional view to its extruded pendant (see Figure 5.14). The effects take about two seconds and the visualisation is drawn as often as it can during the effect. This means that if the visualisation is more complex and it takes for this reason more time to draw it, it is automatically drawn less often than a simpler one. The classes PathDrawingEffect and PathDrawingEffectInstance in the package `extension3d` are responsible for performing the effects.

The second visualisation in which the three-dimensional extension is used is the heatmap. The drawing process of this gadget is more complicated and thus more time consuming than the drawing process of the pie chart. For this reason, no drawing effect was implemented. When a three-dimensional heatmap is drawn, the drawing-progress is shown by refreshing the visualisation every time 100 vertical areas are drawn. In this way, users immediately see progress as content appears on the screen.

The Liquid Diagrams framework allows the creation of prism maps using the Show Extrusion checkbox in the 3D tab of the Options Panel. Figure 5.16 shows an example of a prism map, displaying the population of the districts of Lower Austria in both dimensions.

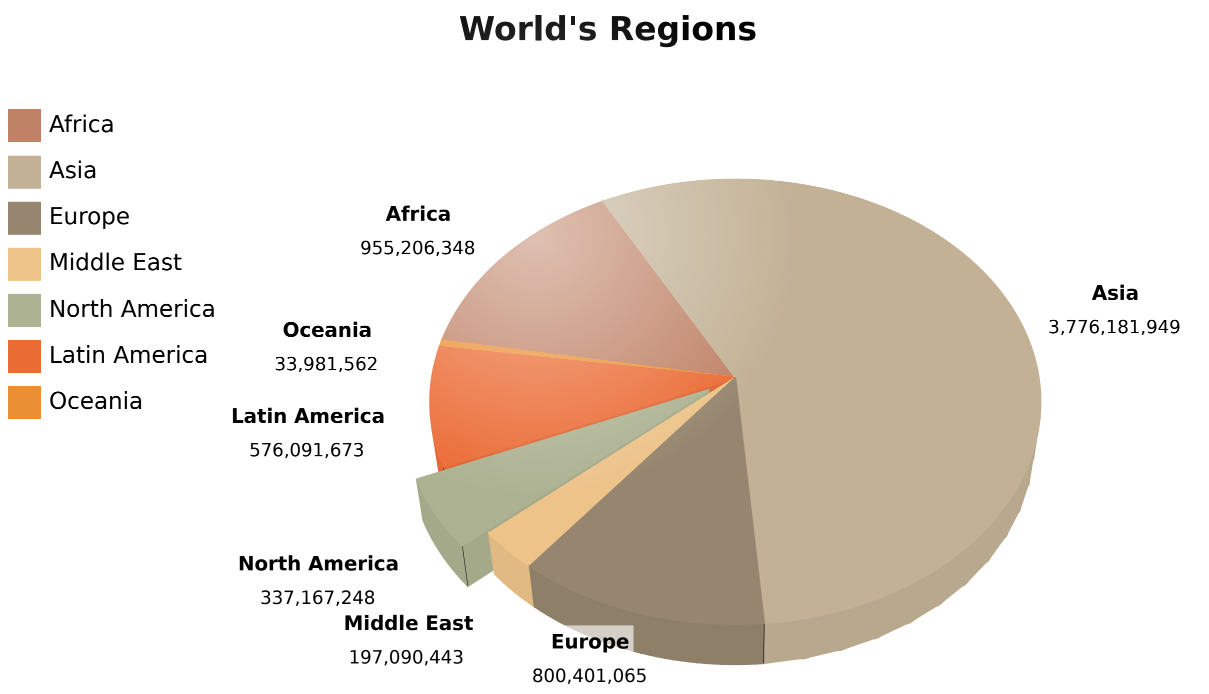


Figure 5.11: A three-dimensional pie chart visualising the population of the World's regions.

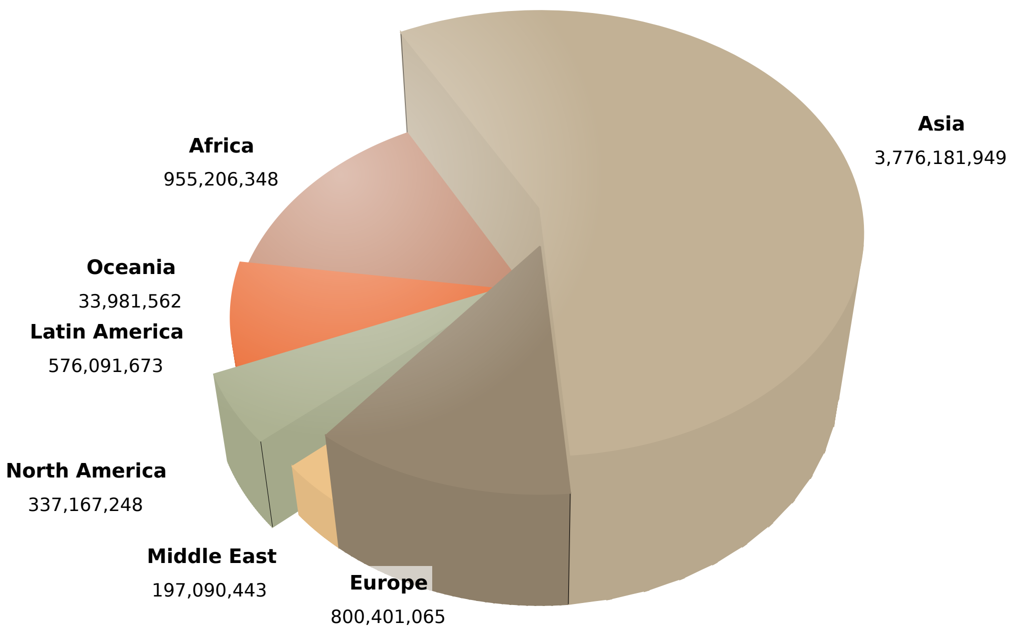
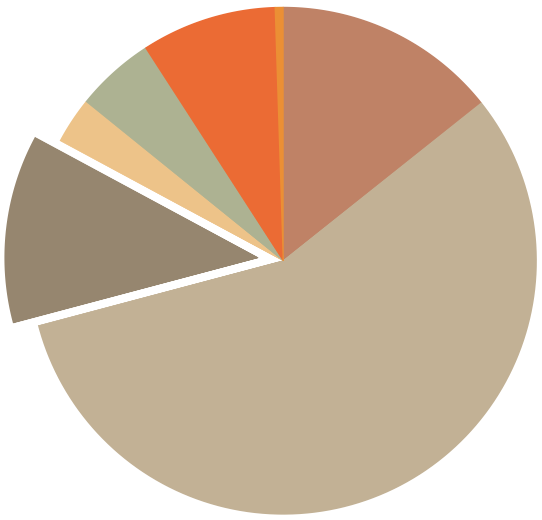
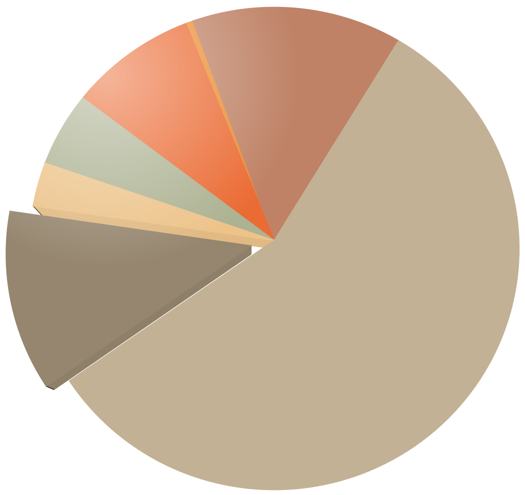


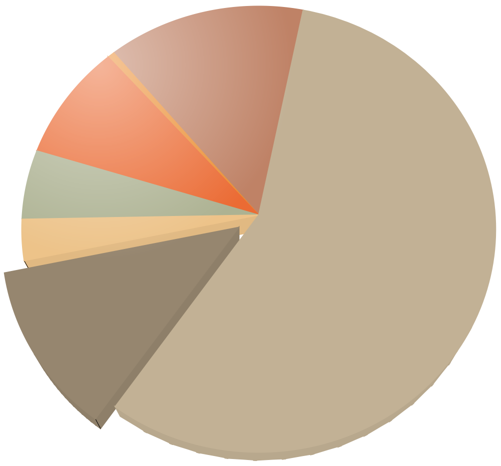
Figure 5.12: An extruded three-dimensional pie chart displaying the World's regions. Population data is expressed using the areas of the chart and its height encodes the number of internet users.



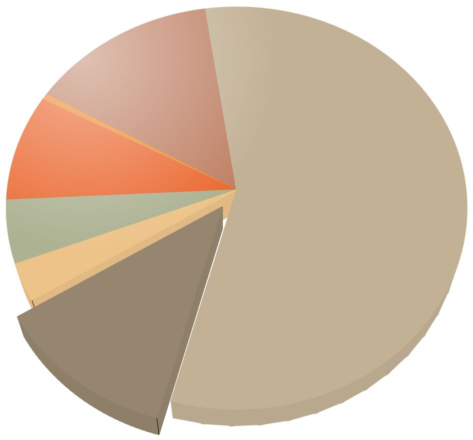
(a) Effect started (x-rotation: $0°$, y-rotation: $0°$).



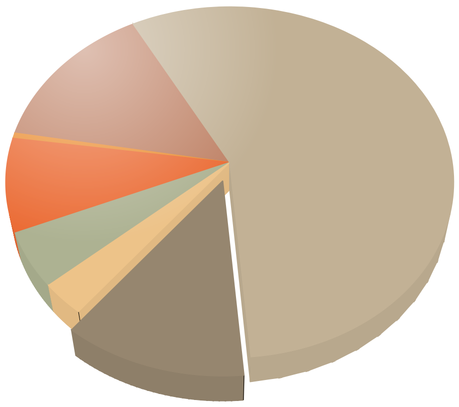
(b) Effect progress: 20% (x-rotation: $10°$, y-rotation: $-20°$).



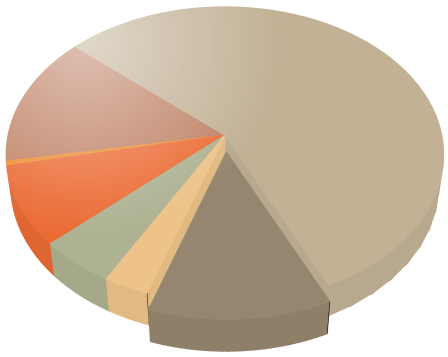
(c) Effect progress: 40% (x-rotation: $20°$, y-rotation: $-40°$).



(d) Effect progress: 60% (x-rotation: $30°$, y-rotation: $-60°$).

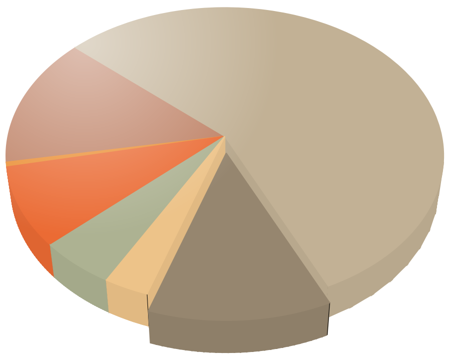


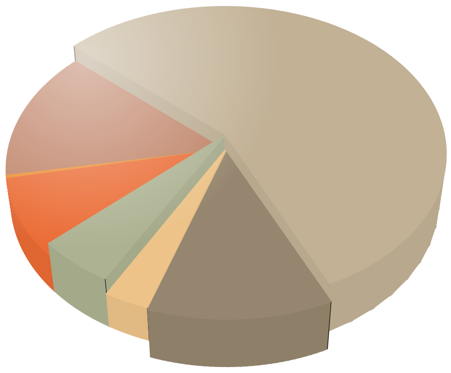
(e) Effect progress: 80% (x-rotation: $40°$, y-rotation: $-80°$).

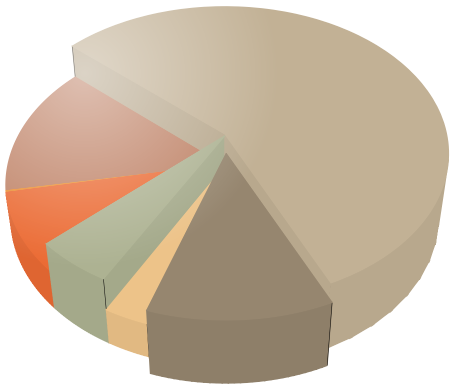


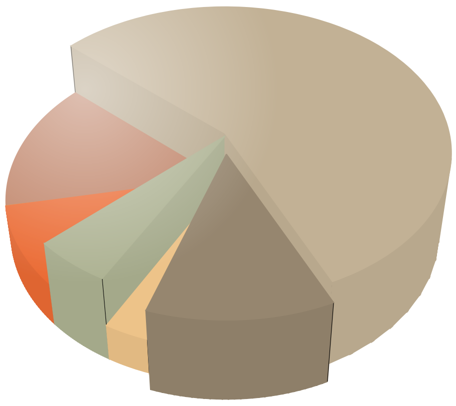
(f) Effect finished (x-rotation: $50°$, y-rotation: $-100°$).

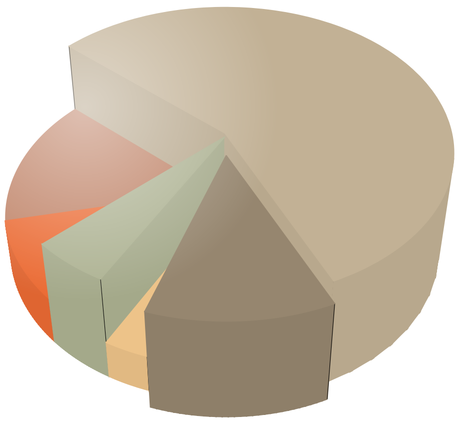
Figure 5.13: The animation sequence of the rotation effect visualising the rotation from the two-dimensional to the three-dimensional view. The same data as in Figure 5.11 is used.


(a) Effect started (extrusion height: 0%).


(b) Effect progress: 20% (extrusion height: 20%).


(c) Effect progress: 40% (extrusion height: 40%).


(d) Effect progress: 60% (extrusion height: 60%).


(e) Effect progress: 80% (extrusion height: 80%).

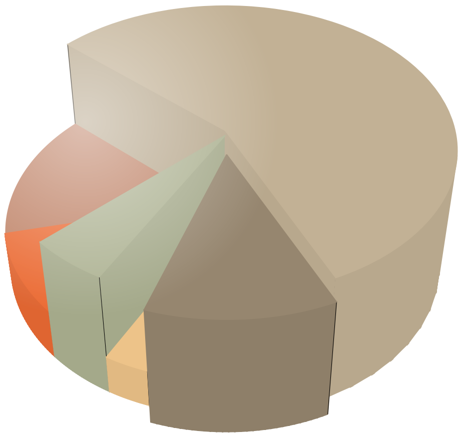

(f) Effect finished (extrusion height: 100%).

Figure 5.14: The animation sequence of the extrusion effect visualising the transition from the normal three-dimensional view to its extruded pendant. Population data is expressed using the areas of the chart and its height encodes the number of internet users in December 2000.

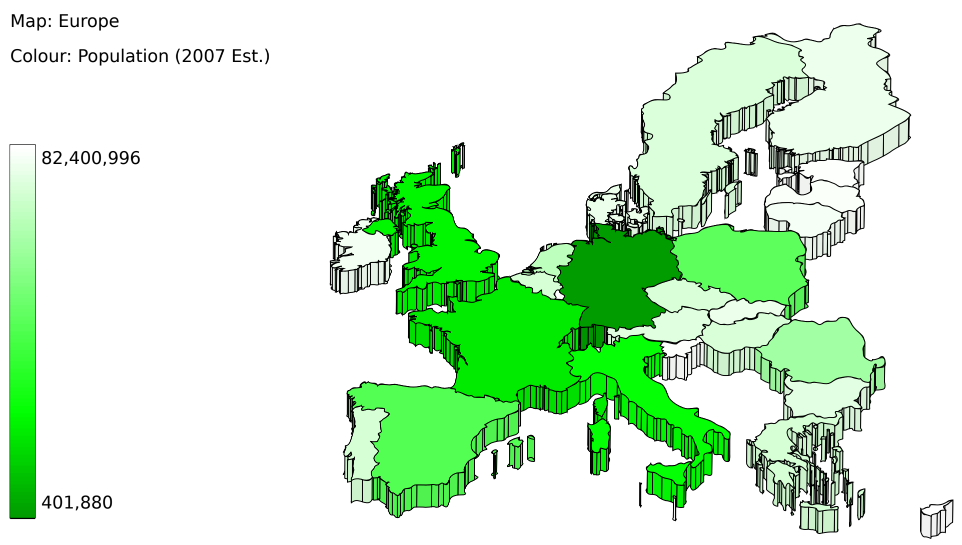


Figure 5.15: A three-dimensional heatmap displaying the number of internet users per country on the map of Europe. Only the EU27 are shown.

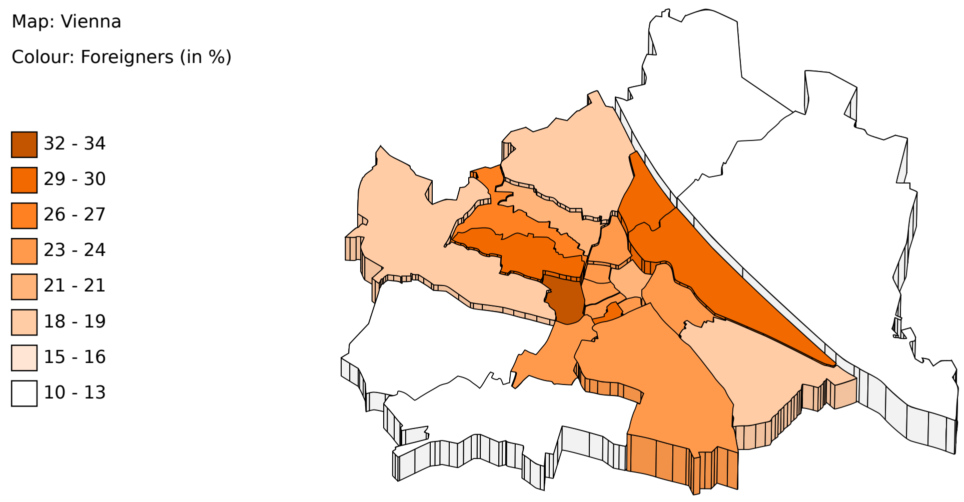


Figure 5.16: A prism map (heatmap with 3D extrusion), visualising the percentage of foreigners of the districs of Vienna with its shading. The height of the areas of the prism map is proportional to the number of inhabitants.

Chapter 6

Liquid Diagrams Visualisations

The Liquid Diagrams framework currently contains twelve different visualisations. Each of them is highly interactive and supports various methods of customisation. In general, one can differentiate between functionality only included in specific gadgets and features which are included in multiple gadgets. All gadgets support the following functionality implemented by Lessacher [2010]:

- **Colour Scheme:** Besides manually changing the colours of data entities, the possibility to use colour schemes is provided. Using a colour scheme results in a colour change to all data entities. If there are more data entities than colours available, the colours are repeated for the remaining data entities. The colour schemes were extracted from COLOURlovers [2011]. This functionality is not available in visualisations using the colour legend, as described in Section 5.5.4.

- **Aspect Ratio:** Every visualisation offers the functionality to manually set the aspect ratio (the width and the height) of the chart display area.

- **Number Display Format:** Three different display formats for numbers are available: “1.000,00”, “1,000.00”, and the option of no number formatting (“1000.00”).

In addition, visualisations which are able to display time-based data, offer the functionality to change the date display format. As described in Section 5.5.3 and 5.5.4, the Liquid Diagrams framework offers several axis and legend types. Table 6.1 shows which visualisations make use of which axis and legend types.

6.1 Line Chart

The LD line chart visualisation was implemented by Martin Lessacher [2010] and is used to display trends over time (see Section 2.3.1). An example of a line chart can be seen in Figure 6.1. The line chart offers the possibility to display data points for each data entity and uses the normal x-, and y-axis of the ChartPanel. Additionally, the standard legend described in Section 5.5.4 can be used.

The data displayed by the visualisations line chart (and bar chart, pie chart, and area chart in the following sections) is taken from StatCounter [2011] and illustrates the market share of mobile web browsers from December 2008 to October 2011.

Visualisation	Normal Axes	Slider Axes	Normal Legend	Colour Legend
Line Chart	√		√	
Bar Chart	√		√	
Area Chart	√		√	
Polar Area Diagram			√	
Bat's Wing Diagram		√	√	
Pie Chart			√	
Parallel Coordinates		√	√	
Star Plot		√	√	
Similarity Map			√	
Treemap				√
Voronoi Treemap				√
Heatmap				√

Table 6.1: The axis and legend types used by the different Liquid Diagrams visualisations.

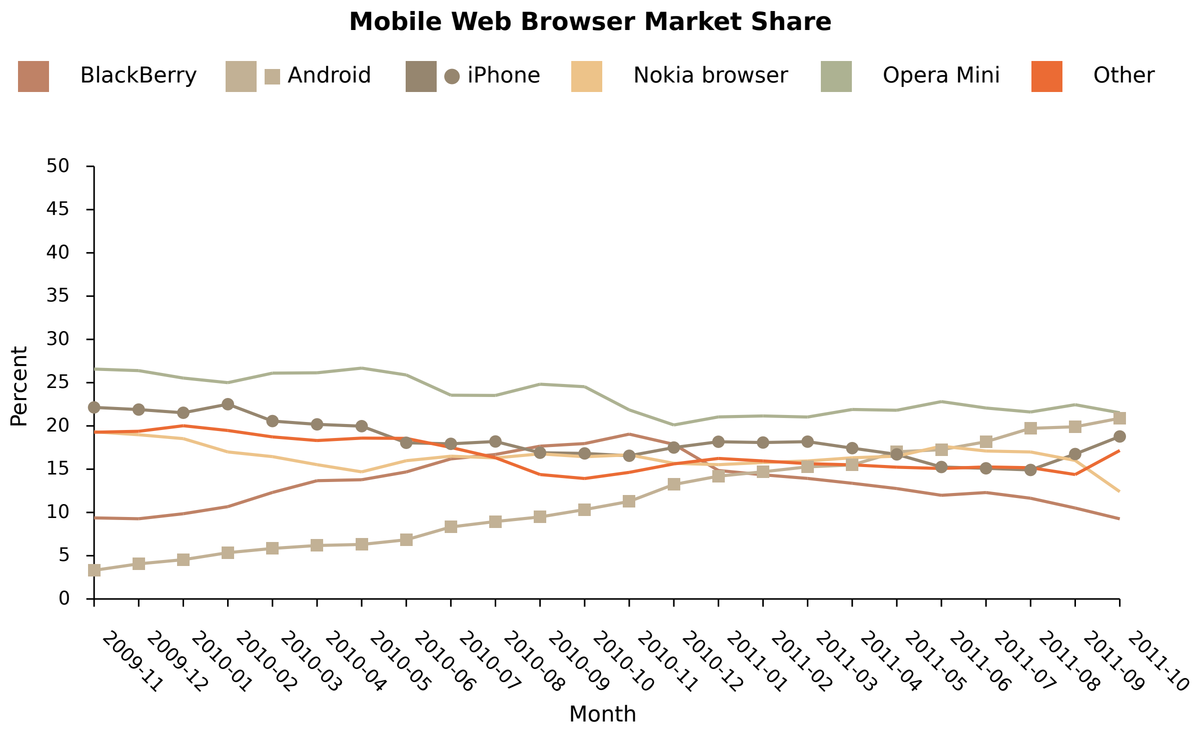


Figure 6.1: A Liquid Diagrams line chart visualising the market share of mobile browsers from December 2008 to October 2010. The data entities "Android" and "iPhone" make use of the functionality to display individual data points.

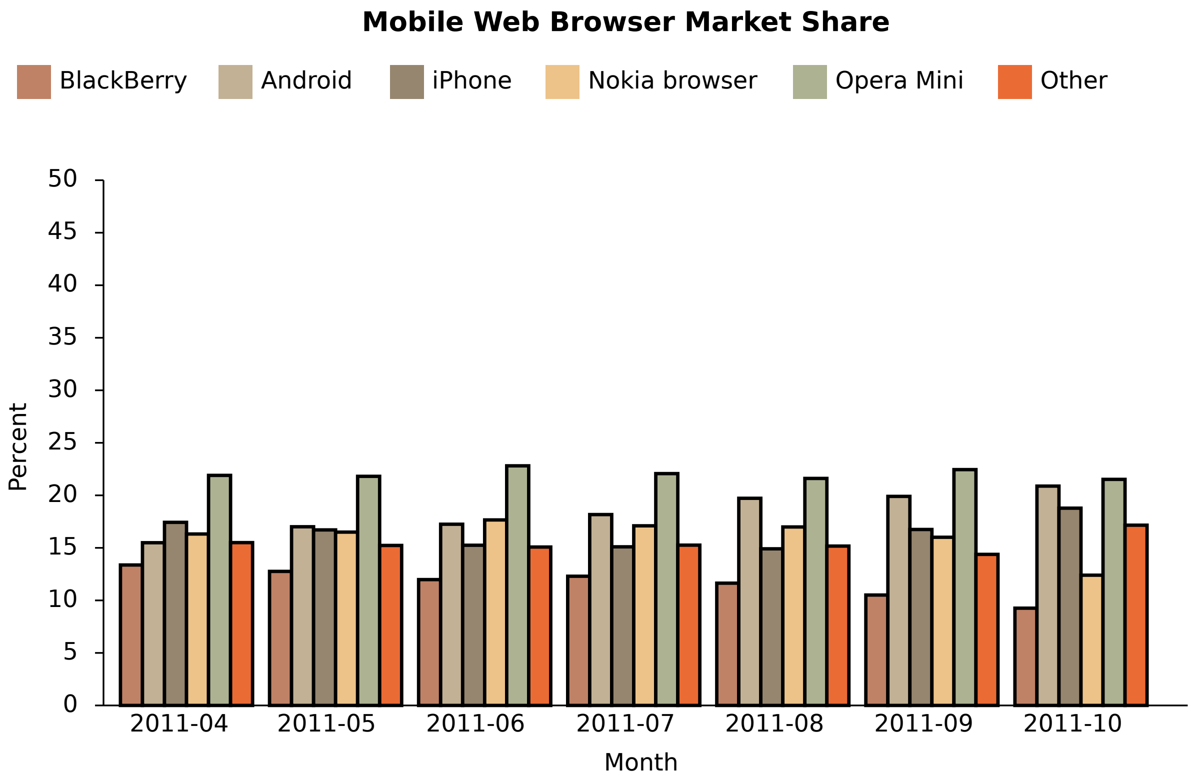


Figure 6.2: A bar chart created with the Liquid Diagrams framework displaying the market share of mobile browsers from April to October 2011. Using bar charts enables easy comparison of data entities.

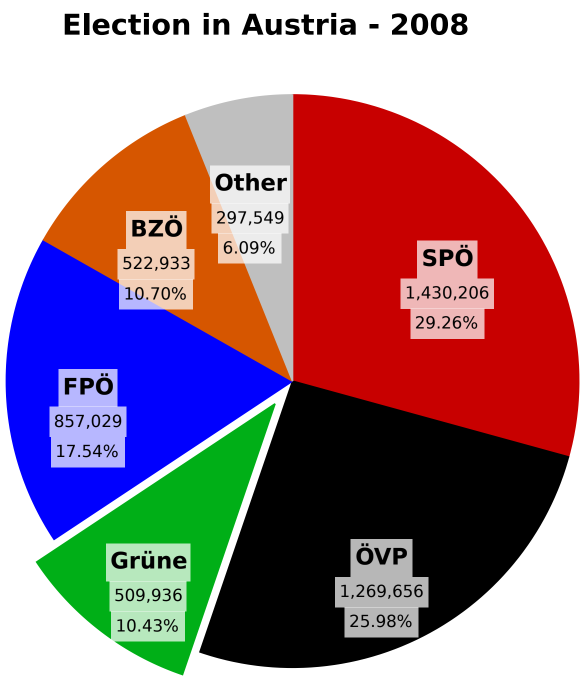
6.2 Bar Chart

As described in Section 2.3.2, bar charts simplify the procedure of comparing multiple data entities. The LD bar chart visualisation, which can be seen in Figure 6.2, was developed by Lessacher [2010]. It provides a mechanism for ensuring a minimum number of pixels for each single bar. Using this functionality, it could happen that all data entities cannot be displayed in the available space. In this case, the overlapping bars are cut off and a warning appears at the right side of the visualisation.

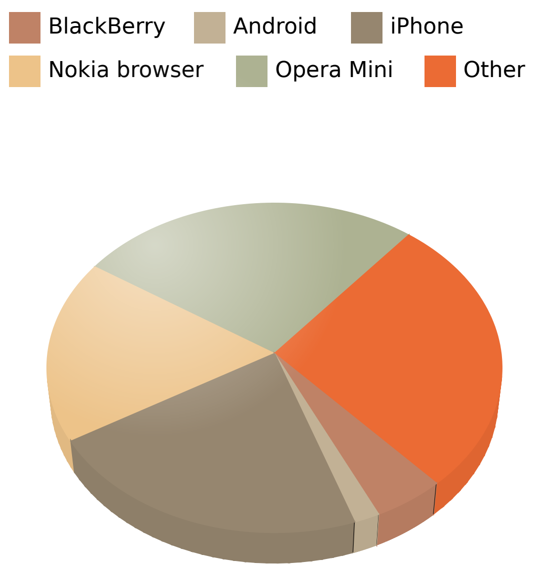
6.3 Pie Chart

The original Liquid Diagrams pie chart visualisation was implemented by Lessacher [2010]. The pie chart offers a scaling mechanism and three different labelling options which can be independently turned on and off. Additionally, it gives users the possibility to break out individual pie sectors in order to draw one's attention to a specific data entity.

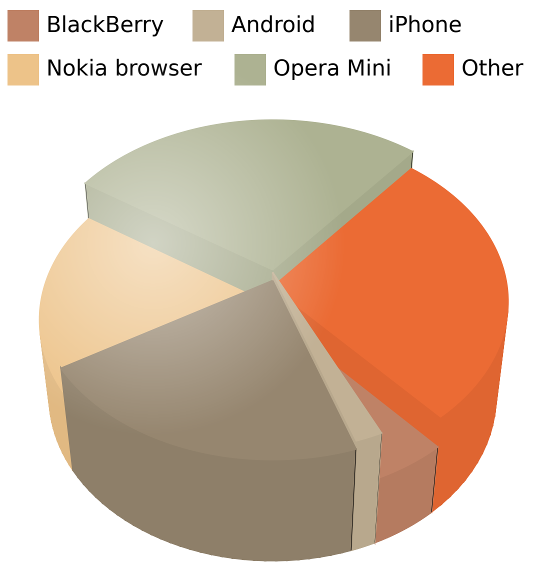
Based on Lessacher's work, a three-dimensional extension (see Section 5.6) was added to the pie chart. This extension enables the visualisation to be displayed as a three-dimensional pie chart. In addition, a mechanism for extruding areas (depth of the pie slices) was added, creating the possibility to display two different variables at once. The pie chart implements two effects, displaying the transitions between the different diagram states. The first effect shows the transition between the normal view and the three-dimensional view and the second effect displays the progress of the extruding functionality.



(a) The result of the 2008 Great Election in Austria. The advanced label options of the pie chart are enabled, displaying the data and the corresponding percentage of each sector. The green sector is broken out for emphasis.



(b) The market share of mobile web browsers in December 2008 visualised with a three-dimensional pie chart.



(c) An extruded version of the pie chart in (b). The size of the slices encodes the market share of mobile web browsers in December 2008 and their heights encode the market share of mobile web browsers in October 2011.

Figure 6.3: Three examples of pie charts. (a) standard pie chart with advanced label options, (b) three-dimensional pie chart and (c) extruded pie chart.

6.4 Area Chart

Like line charts, area charts are used to reveal trends over time (see Section 2.3.4). The Liquid Diagrams implementation of area charts, shown in Figure 6.4, was developed by Lessacher [2010]. It includes two versions of area charts:

- **Stacked Area Chart:** In this version, the areas are drawn on top of each other. An example of a stacked area chart can be seen in Figure 6.4a. An additional option, enabling the areas to be displayed according to their percentage, can be switched on using the Percent check-box on the General tab of the Options Panel. This option achieves better usage of the available space. An area chart using this functionality is shown in Figure 6.4b.
- **Overlay Area Chart:** Here, the areas of the chart overlap each other, resulting in an area chart such as that displayed in Figure 6.4c. In order that those data entities which are drawn first remain visible, transparency is applied to all areas.

6.5 Star Plot

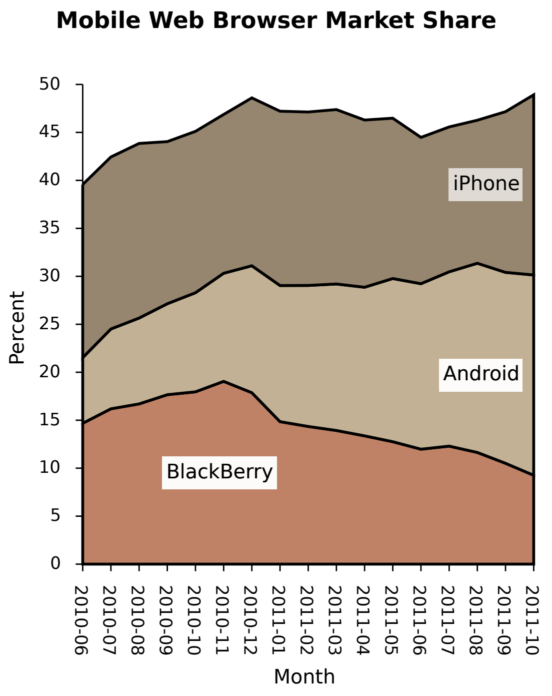
The star plot (Section 2.3.5) displays multidimensional data and dates back to Georg von Mayr [1877] in 1877. It consists of multiple axes, originating from the same center. The resulting shape looks like a star, giving the visualisation its name. Additionally, data points of the same entity on neighbouring axes are connected. An example of a star plot can be seen in Figure 6.5.

Liquid Diagram's star plot visualisation was implemented by Lessacher [2010]. It uses the slider axes (see Section 5.5.3). Therefore the functionality of rearranging axes by drag and drop, advanced filtering options, and the inversion of axes is available. Additional features include:

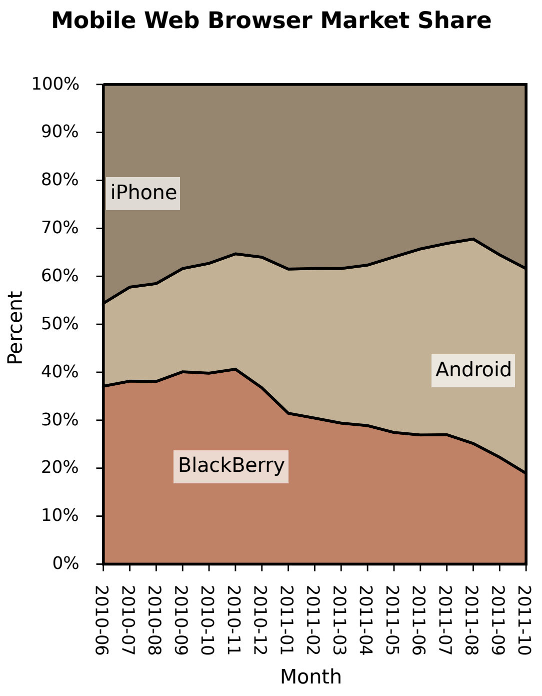
- **Inner Radius Offset:** Normally the axes are created originating from the same point in the center of the visualisation. This functionality enables users to define an offset for the starting point of the axes, enabling better visibility of values close to the center of the star plot.
- **Starting of Axes at Zero:** A check-box on the General tab of the Options Panel gives users the possibility to decide whether the axes start at zero, or at the lowest value mapped on it. This functionality concerns only axes which map numerical values.
- **Area Filling:** Areas formed by the lines of data entities can be filled using this functionality. It can only be enabled if no inner radius offset is specified.

6.6 Parallel Coordinates

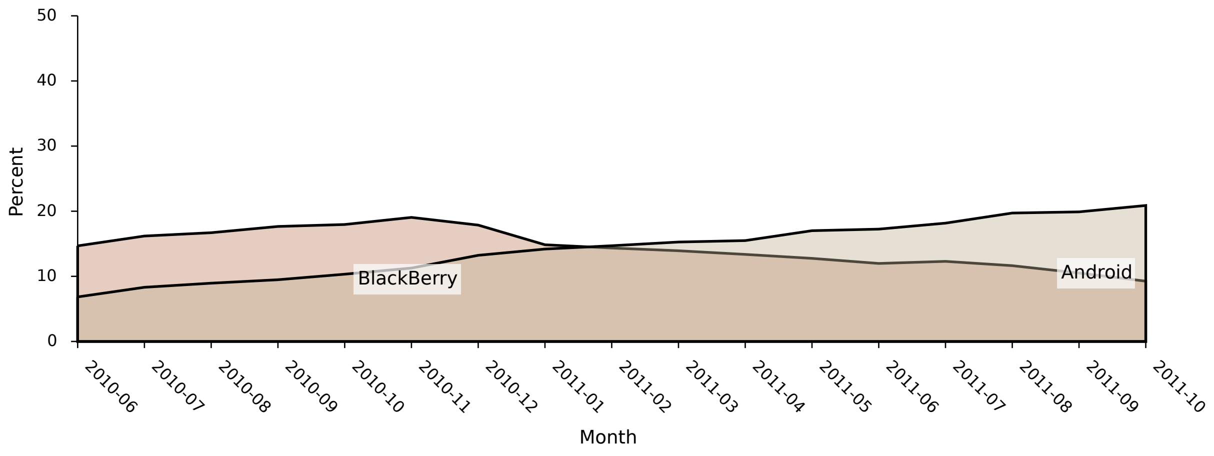
Like star plots, parallel coordinates are also used to visualise multidimensional data. The axes of this visualisation are arranged vertically, parallel to each other. Each of them represents a different data dimension. The data values are displayed by connecting lines between data points on neighbouring axes (see Section 2.3.6). The implementation included in the Liquid Diagrams framework was implemented by Martin Lessacher [2010] and based on Lessacher's work minor changes and bug fixes were performed. An example of the parallel coordinates visualisation can be seen in Figure 6.6. It displays the cereals dataset, filtered by the amount of sugar.



(a) A stacked area chart.



(b) A stacked area chart with the percentage option enabled.



(c) An overlay area chart displaying the market share of two mobile web browsers from June 2010 to October 2011.

Figure 6.4: Area charts: (a) stacked area chart, (b) stacked area chart with enabled percentage option, and (c) overlay area chart.

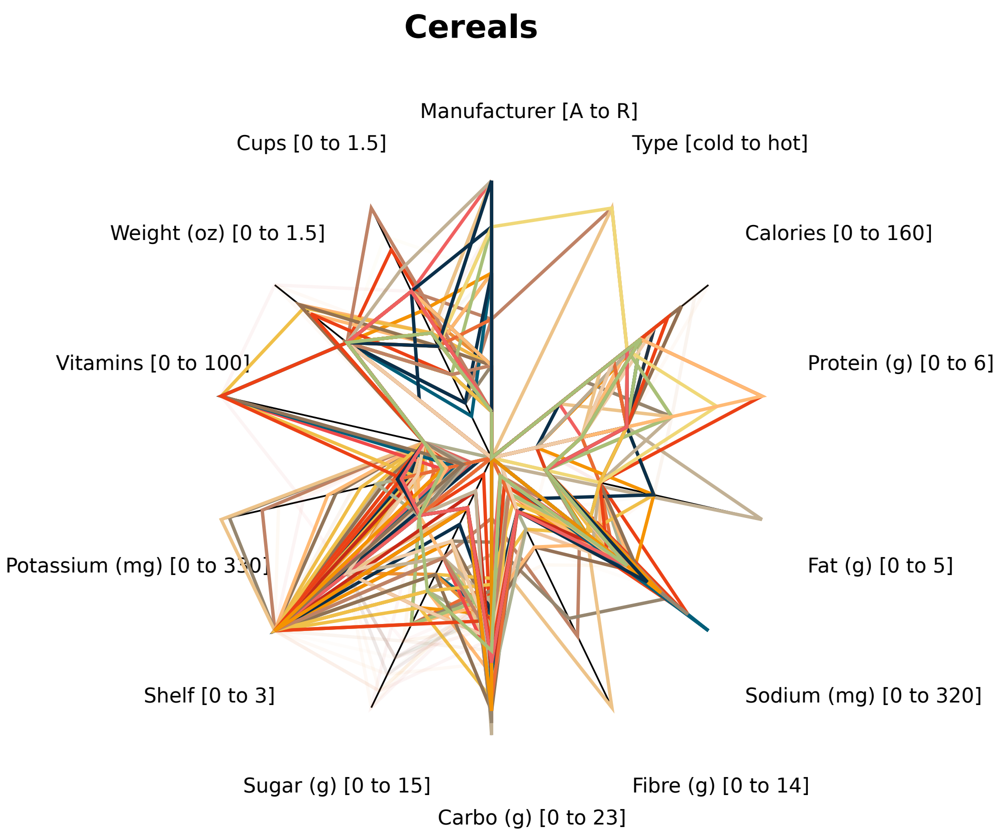


Figure 6.5: A star plot visualising the well-known cereals dataset. The axis containing the attribute sugar is filtered. This action results in a star plot excluding cereals with a high level of sugar.

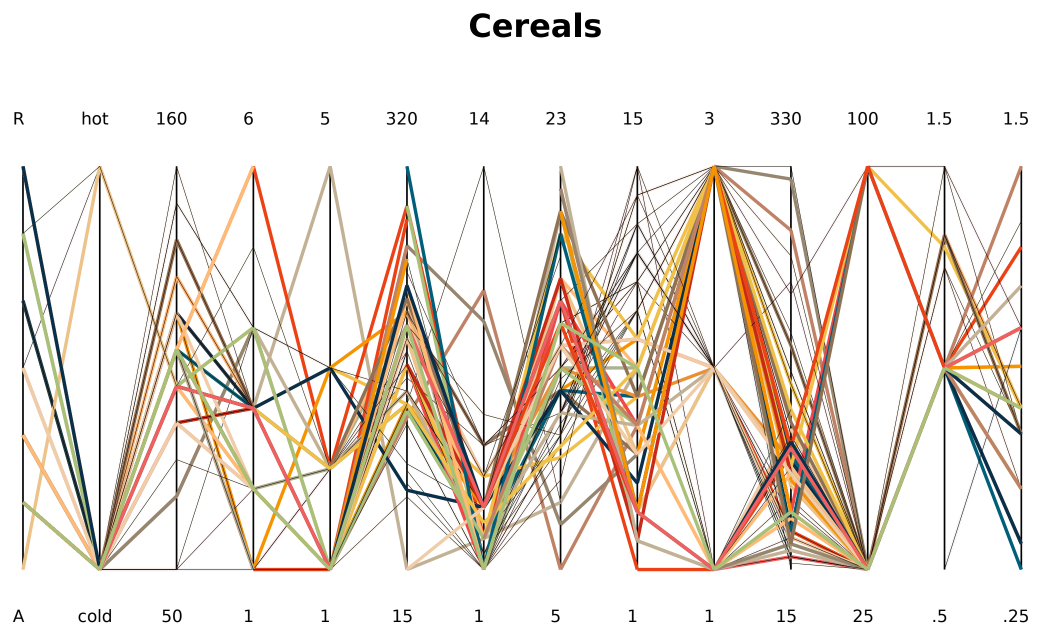


Figure 6.6: The cereals dataset visualised by the parallel coordinates gadget. Again, the axis representing the dimension "amount of sugar" was filtered in order to exclude cereals with a high sugar value. The filtered data entities are displayed using a high transparency.

The parallel coordinates visualisation uses the ChartPanel, a class containing the drawing functionality to create slider axes (see Section 5.5.3). Slider axes enable users to perform filtering actions and to rearrange and invert axes.

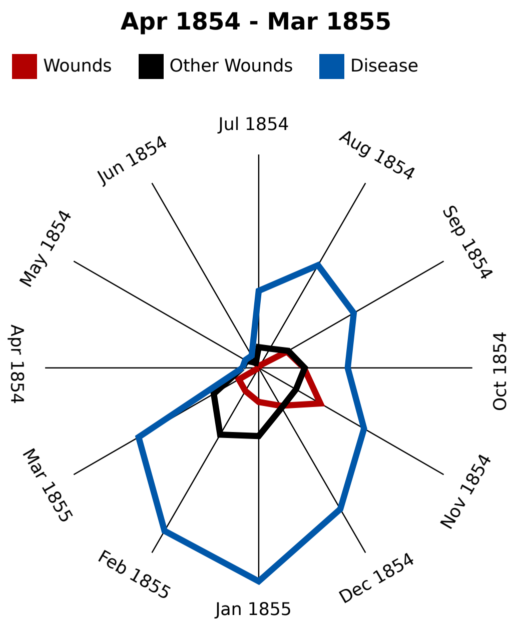
6.7 Bat's Wing Diagram

The bat's wing diagram was introduced by Florence Nightingale in 1858 (see Section 2.3.8). Its implementation in the Liquid Diagrams framework originates from Fernitz et al. [2010], a group of students at Graz University of Technology. It was created during a project in the course "Information Visualisation" in the summer term of 2010 [Andrews, 2011a]. Later, it was completely reimplemented and integrated into the Liquid Diagrams framework by Lessacher [2010].

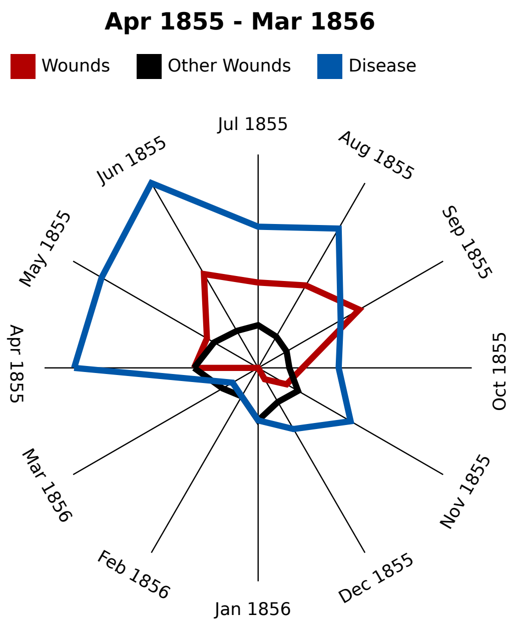
The implementation of the bat's wing diagram is similar to the implementation of the star plot. However, the main difference is that in the bat's wing diagram all axes display the same dimension and scale, reaching from zero to the highest data value. It is not possible to display multivariate data using the bat's wing diagram. Examples of bat's wing diagrams can be seen in Figure 6.7, in the form of redrawings of Florence Nightingale's original diagrams.

6.8 Polar Area Diagram

Like the bat's wing diagram, the polar area diagram (or wedge diagram) was invented by Florence Nightingale in 1859. It replaced the bat's wing diagram in Nightingale's later publications, because of the circumstance that the bat's wing diagram could be easily misinterpreted [Small, 1998]. The polar

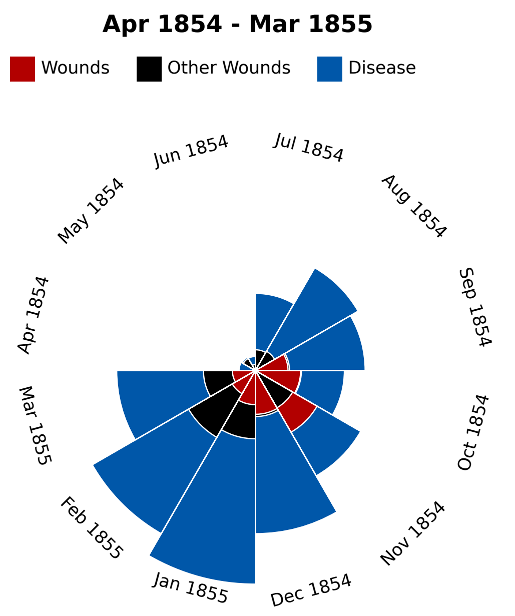


(a) The number of deaths from various causes from April 1854 to March 1855.

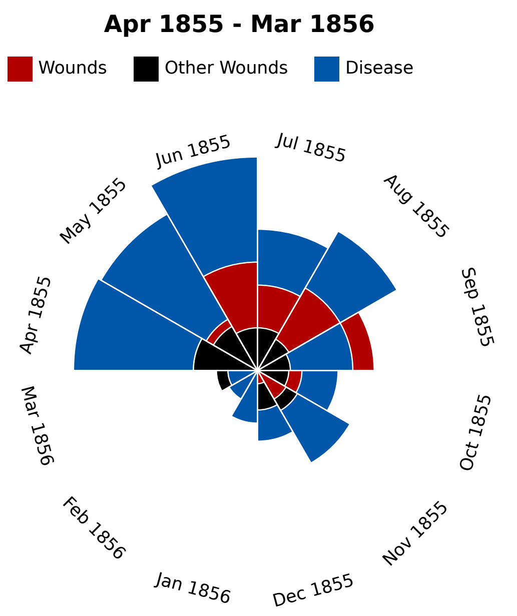


(b) The number of deaths from various causes from April 1855 to March 1856.

Figure 6.7: Modern versions of Florence Nightingale's original bat's wing diagrams. (a) shows the death rate from three different causes during the first year of the Crimean War (1853–1856) (b) shows the second year. A data value is represented by the length of a radial line, starting at the centre.



(a) The death rate from various causes from April 1854 to March 1855.



(b) The death rate from various causes from April 1855 to March 1856.

Figure 6.8: Florence Nightingale's polar area diagrams, redrawn using Liquid Diagrams. The data values in a polar area diagram are proportional to the areas of its wedges.

area diagram corrects this downside, because, the values are expressed as areas instead of radial lines (see Section 2.3.9). A redrawing of Nightingale's original polar area diagrams can be seen in Figure 6.8.

The implementation of the polar area diagram contained in Liquid Diagrams was originally developed by Fernitz et al. [2010] during the same project mentioned in Section 6.7. Like the bat's wing diagram it was also completely reimplemented and integrated into the Liquid Diagrams framework by Lessacher [2010].

6.9 Heatmap (Choropleth Map)

As described in Section 2.3.10, a geographic heatmap, also known as choropleth map, is a heatmap with underlying cartographic areas. Its areas are shaded in proportion to the data values assigned to it. A geographic heatmap with extruded areas is called a prism map. Geographic heatmaps date back to Pierre Charles Dupin, who used choropleth maps to show the distribution and intensity of illiteracy in France in 1826 [Dupin, 1826].

The heatmap gadget of the Liquid Diagrams framework was implemented by Lessacher [2010]. It uses SVG maps of cartographic areas to display a heatmap of the specified areas. Here, the areas are described using the geo data format (see Section 5.1.6). Building on Lessacher's work, the heatmap gadget underwent many changes and improvements. However, the main processing sequence of the heatmap visualisation remain unchanged. This sequence can be seen in Figure 6.9. The steps to create a heatmap visualisation are:

1. First, the heatmap gadget retrieves the specified data.

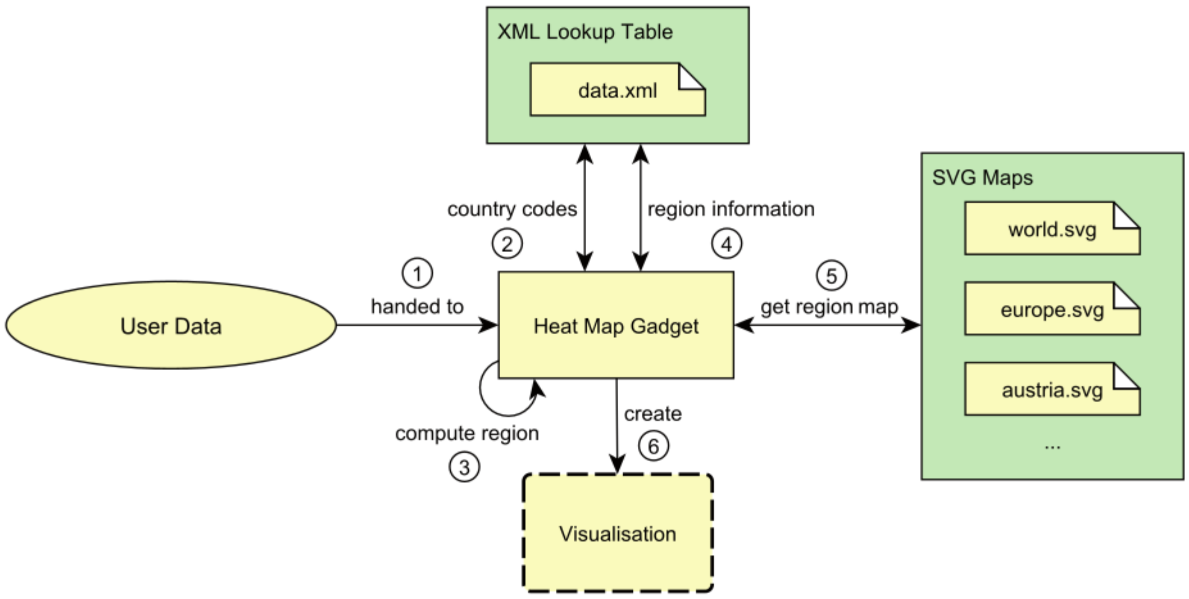


Figure 6.9: The steps to create a heatmap visualisation. This image is extracted from Lessacher [2010].

2. The ISO codes in the data are used to look up the countries and regions in the lookup file and to retrieve the corresponding region relationships.

3. A hierarchical data structure is created using this information.

4. The shape files for the corresponding regions are obtained from the lookup file.

5. The SVG map obtained in the previous step is loaded.

6. Finally, the visualisation is created.

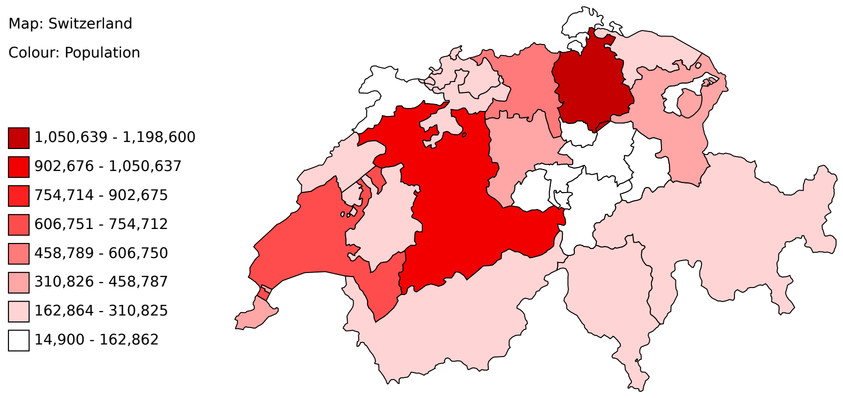
Examples of choropleth maps, created with the heatmap gadget can be seen in Figure 6.10. In version 2.0 of Liquid Diagrams, the heatmap gadget includes the 3D extension described in Section 5.6.

6.9.1 Categories

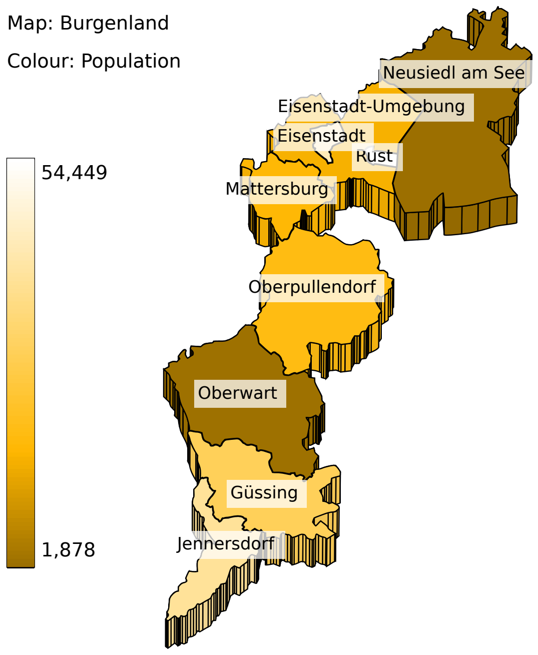
In version 1.0 of the Liquid Diagrams framework it was not possible to display non-numeric values in a heatmap. This circumstance was changed in version 2.0. In order to display the distribution of textual attributes of areas in a map, functionality to visualise categories was implemented. This functionality distinguishes between two kinds of textual attributes, which can occur in the data:

1. **Informations:** A data column is interpreted as additional information, if none of its values appears twice. The third column (Capital) in Table 6.2 is an example of a column containing information. The information strings are displayed in the complex tool-tip of the corresponding areas of the heatmap visualisation. They cannot be mapped onto areas of the map.

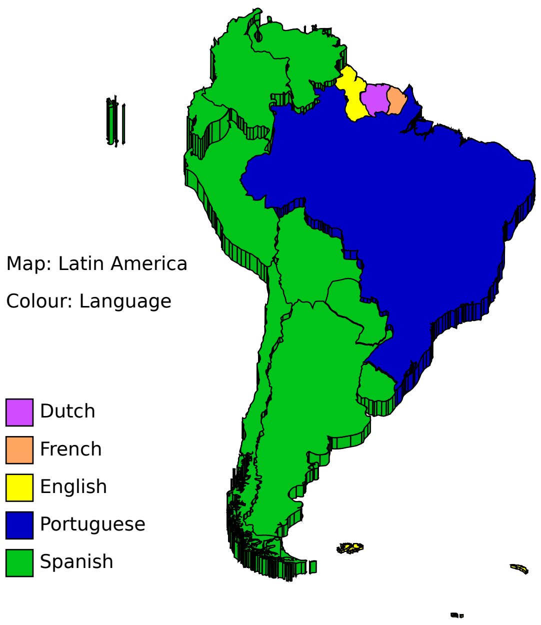
2. **Categories:** Categories represent textual values, which can be mapped onto areas of the map. A data column contains categories if it contains textual values and if one of the values is repeated more than once within the column. The fourth column (Governing Party) in Table 6.2 is an example of a data column containing categories. The column lists the governing parties of the specified areas. Due to the fact that the parties SPÖ and ÖVP occur more than once, the textual values of



(a) A choropleth map displaying the population of Switzerland. It uses uniform colour distribution.



(b) A three-dimensional heatmap visualising the districts of Burgenland, a region in Austria. Continuous colour distribution is used.



(c) A prism map of South America using the 3D extension of the framework. The colour expresses the official language of each country and the height is proportional to the population density per km^2.

Figure 6.10: The heatmap gadget allows the creation of choropleth maps, including three-dimensional choropleth maps.

Region	Population	Capital	Governing Party
AT-1	281190	Eisenstadt	SPÖ
AT-2	561094	Klagenfurt	FPK
AT-3	1597240	Sankt Pölten	ÖVP
AT-4	1408165	Linz	ÖVP
AT-5	530576	Salzburg	SPÖ
AT-6	1205909	Graz	SPÖ
AT-7	703512	Innsbruck	ÖVP
AT-8	366377	Bregenz	ÖVP
AT-9	1677867	Wien	SPÖ

Table 6.2: Visualising categories in the heatmap gadget. The third column (Capital) specifies additional information, because none of the textual data values occurs twice. The data values of the fourth column (Governing Party), on the other hand are interpreted as categories, because some of the strings occur more than once. The resulting heatmap can be seen in Figure 6.11.

the column are interpreted as categories. Consequently, the values of this column can be mapped onto the areas of the loaded map. The result of this mapping can be seen in Figure 6.11.

To visualise categories in a map, the desired data column has to be chosen from the drop-down box, listing the displayable columns. This drop-down box can be found in the Options Panel's General tab, or within the legend. By selecting a categories column, the colour distribution buttons in the legend are disabled, and two additional label options become available in the Label tab of the Options Panel.

However, there is a limitation of the categories functionality. By displaying categories, it is not possible to sum up the values of an area if a hierarchy exists. Normally, when numerical data is visualised on the map and multiple areas, for example countries are displayed, the value of a country is calculated by summing up the values of the regions of the country, if no explicit value for the country is specified. This procedure cannot be adapted to categories, because it is not possible to sum up two or more categories.

6.9.2 Tree Component

Hierarchically structured data can be visualised with the help of the heatmap gadget. For instance, it is possible to load the districts of multiple countries at once. In this case the map displaying the common parent of the districts is loaded. This would be the map containing all corresponding countries of the districts. The value for each of the countries is calculated by summing up the values of the corresponding districts. By clicking on one of the countries, a new map, containing the districts of that country is loaded and visualised by the heatmap gadget.

However, in order to improve the user experience, a tree component, showing the data hierarchy is also displayed. In this way, users immediately see the available hierarchy and are able to use it to navigate purposes. The tree component is only displayed, if a hierarchy is detected in the data. Otherwise the Show Tree Component check-box is disabled in the General tab of the Options Panel. It is the same tree component, available in the treemap and the Voronoi treemap gadget. Figure 6.12 shows this component inside the heatmap gadget.

If no “ld-Name”-column specifies the area names, the names inside the tree component remain empty on start-up. This is because the names of the areas normally come from the loaded SVG map. The heatmap gadget only loads maps if they are needed. This means that the heatmap gadget waits with the

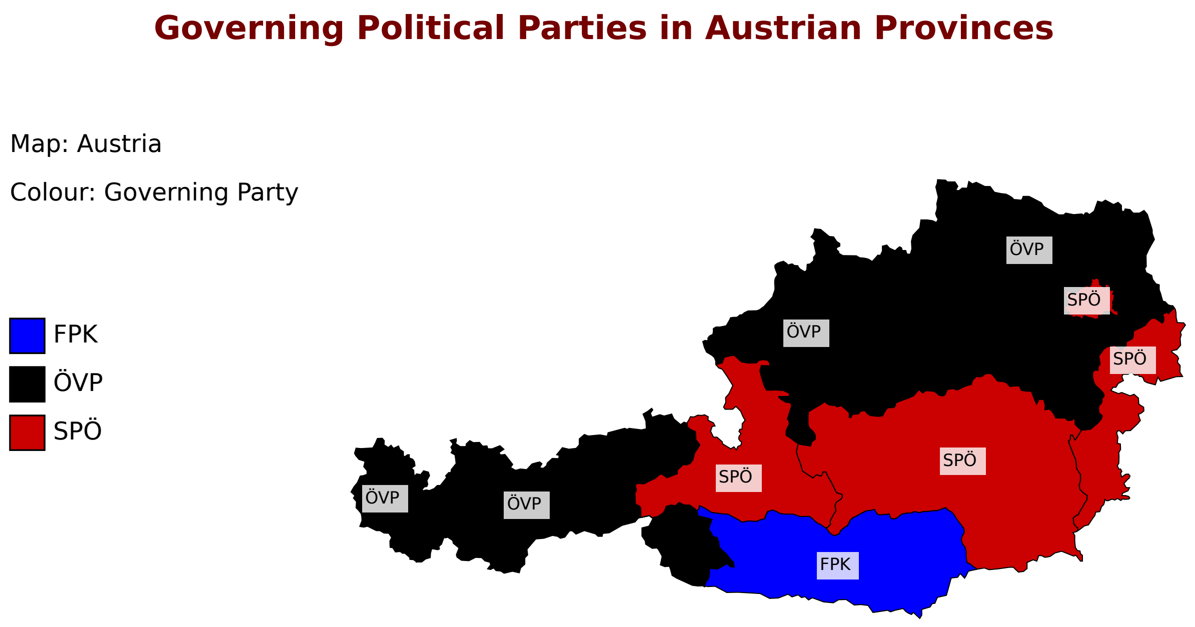


Figure 6.11: A heatmap with categories from the data in Table 6.2, showing governing political parties of the provinces of Austria.

loading of maps, until the user clicks on an area with an underlying hierarchy. The names of areas on lower level in the hierarchy only become known when the maps are loaded by the gadget.

6.9.3 SVG Maps

In version 2.0 of the Liquid Diagrams framework several modifications were made to the format of the Scalable Vector Graphics (SVG) maps used by the heatmap gadget. This includes the modification of the Document Type Definition (DTD) of the SVG files, the including of style information for each path, and the addition of several new attributes of the path element.

The DTD defines legal building blocks of a document [w3schools, 2011]. These specify the legal elements and attributes which define the document structure. The SVG maps used in the heatmap gadget underwent modifications regarding the addition of several new attributes of the `path` element. The DTD of the SVG files [W3C, 2011h] was extended. Otherwise the map files would have an invalid document structure and their validation would fail. Listing 6.1 shows an example of an SVG map using the new attributes. The new attributes of the path element are:

- **name:** The name of the region. This string is displayed inside of the tool-tip of the corresponding path. If the displayed data contains an “ld-Name” data column, the gadget does not use the name attribute. Instead, it visualises the names from the data column. The name attribute originates from Lessacher [2010].
- **center:** Lessacher [2010] introduced a “center” attribute for a path, indicating the position of a label. If no “center” attribute is given, the centre of the bounding box of the path is used for the label position.
- **viewbox:** This attribute specifies the bounding box of the path. It is used, amongst other things, to calculate a smaller view-box, if the option Draw Complete Map in the Map tab of the Options Panel is

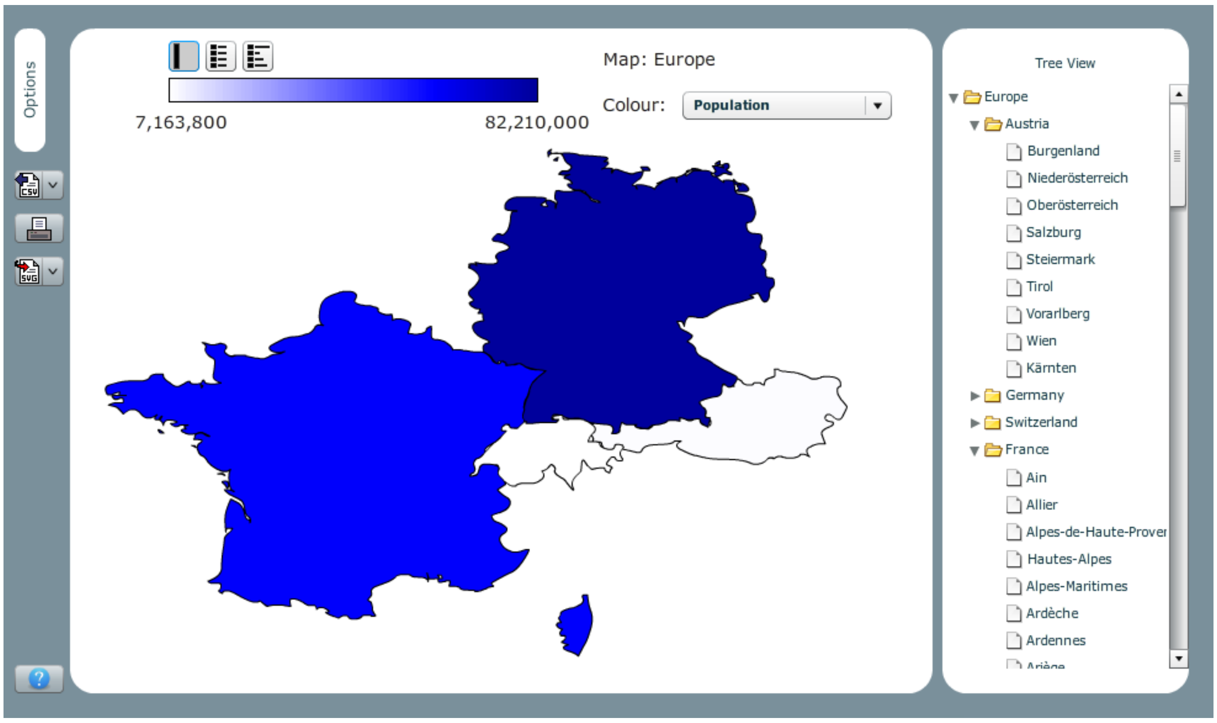


Figure 6.12: Austria, Germany, France and Switzerland visualised in the heatmap gadget. The hierarchy of the loaded data can be seen at the right side contained within the tree component.

disabled.

- **fips:** The paths of the SVG map of the United States contain this attribute. It specifies the Federal Information Processing Standard (FIPS) code of each path [NIST, 2011]. This code uniquely identifies each state of the United States.

In version 2.0 of the Liquid Diagrams framework, the number of available SVG maps increased significantly. Table 6.3 shows a summary of the SVG maps available within the heatmap gadget. The new maps added in version 2.0 of Liquid Diagrams are displayed in bold.

6.9.4 Drawing Process

Version 1.0 of the Liquid Diagrams framework introduced a component named ScalableContentPanel. As the name indicates this is a drawing container, which is fully scalable. It was used in the following way. First, the content of an SVG map was drawn within it. During the drawing process, the bounding box of every available region was collected and in the last step, the regions which had to be displayed were resized, using their combined bounding boxes as the new viewing area. However, this approach had one disadvantage: due to the fact, that the content was drawn first and resized later, elements such as outlines and text were resized and distorted as well, sometimes leading to disproportional large text or outlines. An example of this behaviour can be seen in Figure 6.13.

To overcome this problem, the ScalableContentPanel was replaced with a coordinate transformation mechanism. This mechanism takes the x and y coordinates of points in a SVG command as input and transforms them to screen coordinates. This is done using the combined bounding boxes of the paths to be displayed. For this reason the decision was made to include bounding boxes of SVG paths within each SVG file rather than having to calculate them on the fly. The procedure of this transformation is illustrated in Figure 6.14. Equations 6.1 and 6.2 are used to map a point P to screen coordinates.

```
 1 <?xml version="1.0" encoding="utf-8" standalone="no"?>
 2 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 3   "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" [
 4         <!ATTLIST path
 5                 name CDATA #IMPLIED
 6                 center CDATA #IMPLIED
 7                 viewbox CDATA #IMPLIED
 8                 fips CDATA #IMPLIED>
 9 ]>
10 <svg version='1.1' id='AT_6' xmlns='http://www.w3.org/2000/svg'
11     width='900' height='618' viewBox='0 0 900 618'>
12 
13     <path
14         id="AT_6_GU"
15         name="Graz-Umgebung"
16         style="fill:#FFFFFF;stroke:#000000;stroke-width:1;stroke-opacity:1"
17         viewbox="514.4063 245.8438 233.1563 235.3125"
18         center="599 311"
19         d="M 646.7500, 245.8438    ...     Z"
20     />
21 
22     <path
23         id="AT_6_G"
24         name="Graz"
25         style="fill:#FFFFFF;stroke:#000000;stroke-width:1;stroke-opacity:1"
26         viewbox="619.3243 354.0494 58.4617 58.8524"
27         d="M 656.5880, 411.0413    ...     Z"
28     />
29 
30     <path
31         id="AT_6_LB"
32         name="Leibnitz"
33         style="fill:#FFFFFF;stroke:#000000;stroke-width:1;stroke-opacity:1"
34         viewbox="609.3381 409.8089 128.8120 201.0005"
35         center="680 522"
36         d="M 664.7519, 610.3091    ...     Z"
37     />
38 
39    ...
40 
41 </svg>
```

Listing 6.1: An SVG map of the region of Styria in Austria. The DTD is extended to support the additional attributes “name”, “center”, “viewbox”, and “fips” (see lines 4-8). For simplicity, the SVG commands for the path element have been shortened and not all paths for the regions of Styria are printed.

World	Continents	Countries	Provinces
World	Africa	Austria	(AT) **Burgenland**
	Asia	Belgium	(AT) Carinthia
	Europe	France	(AT) **Lower Austria**
	Latin America	Germany	(AT) **Upper Austria**
	North America	Swizerland	(AT) **Salzburg**
	Oceania	United States	(AT) **Styria**
			(AT) **Tyrol**
			(AT) **Vorarlberg**
			(AT) **Vienna**

Table 6.3: The SVG maps available in the heatmap gadget. The new maps added in version 2.0 of Liquid Diagrams are given in bold.

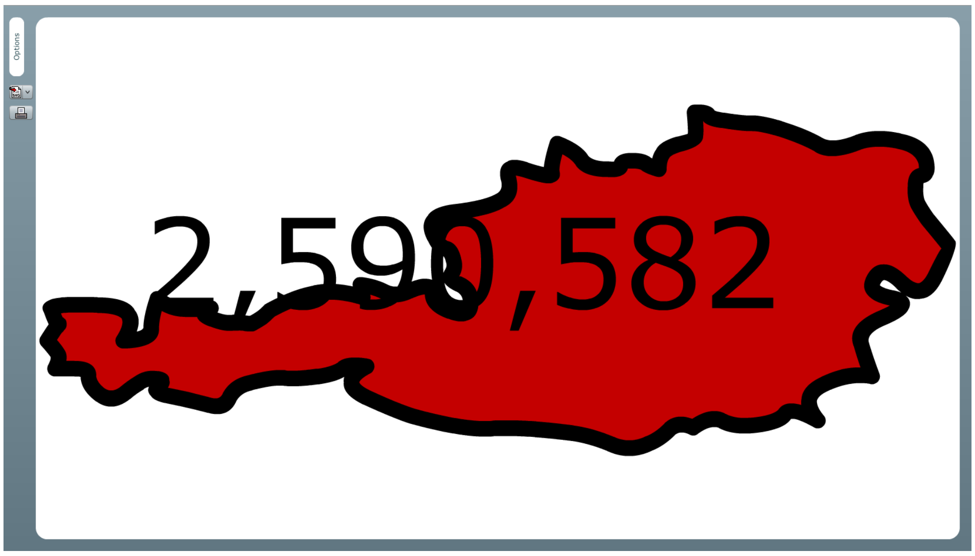
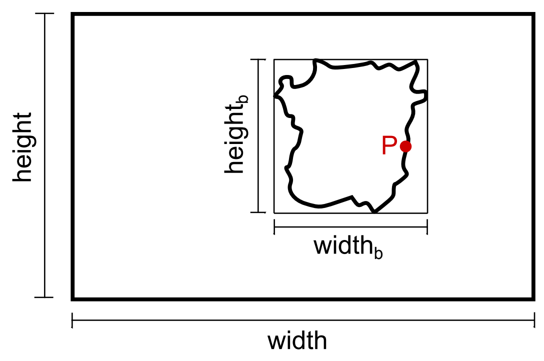
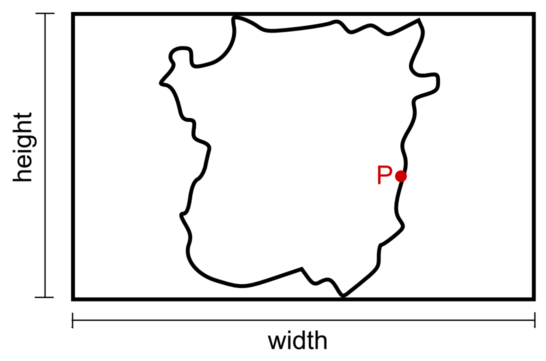


Figure 6.13: The disadvantage of the ScalableContentPanel. By scaling the available content to the screen dimensions, it could happen that borders and text become disproportionally large.



(a) The original state of an SVG path. The bounding box of the path is contained within its corresponding SVG file.



(b) The path is transformed to screen coordinates, using its bounding box.

Figure 6.14: The transformation of a path. (a) shows the original path and (b) shows the path after transformation.

P_x	The x coordinate of the point which is about to be transformed.
P_y	The y coordinate of the point.
x_0	The horizontal starting point (left) of the bounding box.
y_0	The vertical starting point (top) of the bounding box.
$width_b$	The width of the bounding box.
$height_b$	The height of the bounding box.
$width$	The width of the diagram area.
$height$	The height of the area reserved for the diagram.

$$x = \frac{(P_x - x_0)}{width_b} * width \tag{6.1}$$

$$y = \frac{(P_y - y_0)}{height_b} * height \tag{6.2}$$

The architecture of the coordinate transformation mechanism was carefully designed to be easily extendible. It could be extended to simultaneously display areas of different hierarchies. An example of this behaviour would be a map, displaying countries together with districts. This functionality was mostly already implemented within the heatmap gadget. However, the downside of this approach is that the underlying cartographic material has to be very accurate in order to support the visualisation of areas of multiple hierarchies at once. Sadly, the SVG maps of the heatmap gadget come from different sources and in most cases the borders of countries contained within different SVG maps do not match exactly, even if the countries share the same border in reality. Tofix this, the cartographic SVG material would need to be cleaned and matched (see Chapter 9).

6.10 Treemap

As described in Section 2.3.11, the treemap visualisation was invented by Ben Shneiderman in 1991 [Johnson and Shneiderman, 1991]. A treemap iteratively divides the available space into nested rectangles. Since the invention of this visualisation, various algorithms for creating treemaps were developed. Shneiderman [2009] lists various algorithms and discusses their strengths and weaknesses.

The treemap gadget of the Liquid Diagrams framework was developed by Lessacher [2010]. It implements the following algorithms:

- **Slice-and-Dice:** This is the original algorithm published by Johnson and Shneiderman [1991]. It alternates the splitting direction, using parallel horizontal and vertical lines. In this way the ordering of the data is preserved. However, the resulting aspect ratios can be quite high, resulting in very narrow horizontal or vertical strips.
- **Squarified:** This algorithm was introduced by Bruls et al. [2000] in 1999. Here, the ordering of items is relaxed in order to create a layout where the rectangles approximate squares. This procedure results in aspect ratios near 1:1.

In the LD treemap visualisation, the algorithm can be exchanged using the drop-down box located in the General tab of the Options Panel. Examples of the treemap visualisation are shown in Figure 6.15.

6.11 Voronoi Treemap

The Voronoi treemap uses Voronoi tesselation, a mathematical concept for dividing the available space according to the nearest-neighbour rule (see Section 2.3.12). The Voronoi treemap contained in the Liquid Diagrams framework was implemented by Martin Lessacher. A detailed discussion about the mathematical concept and its implementation can be found in Lessacher [2010].

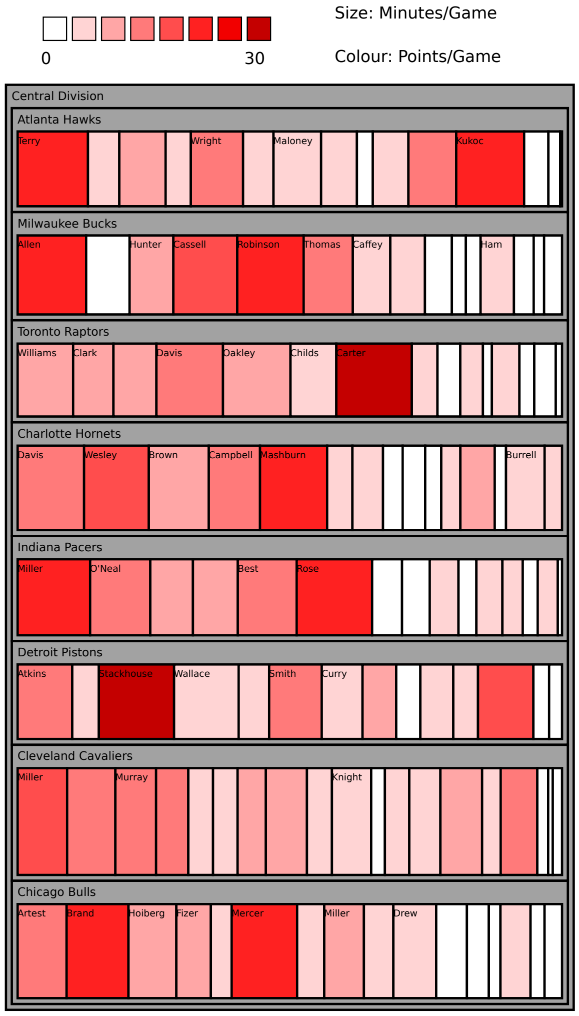
Liquid Diagram's Voronoi treemap can be seen in Figure 6.16. It contains the following features:

- **Shapes:** Several outer shapes containing the polygons created by the algorithm can be chosen. This includes rectangles, triangles, and regular polygons (where the number of sides can be chosen).
- **Algorithm Options:** Many algorithmic options can be changed in order to produce different output. The options are: the maximum number of iterations for the parent and the child, the area error tolerance, the number of retries, and the minimum area size for each child.

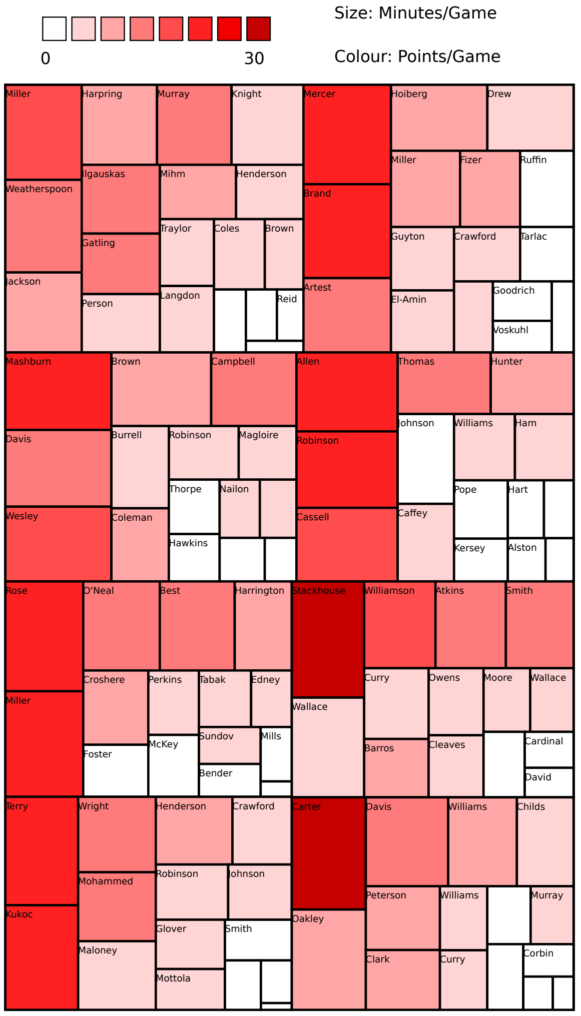
6.12 Similarity Map

In version 2.0 of the Liquid Diagrams framework a new visualisation, the similarity map, was added. It was developed by Bajramovic et al. [2011] in the course "Information Visualisation" [Andrews, 2011a] during the summer term 2011 at Graz University of Technology. The similarity map calculates the similarity between multidimensional data items and places items according to a force-directed placement algorithm. An example similarity map can be seen in Figure 6.17.

A force-directed placement algorithm determines the positions of nodes in a graph in an aesthetically pleasing way. This is done by assigning forces to each node in order to iteratively determine the position

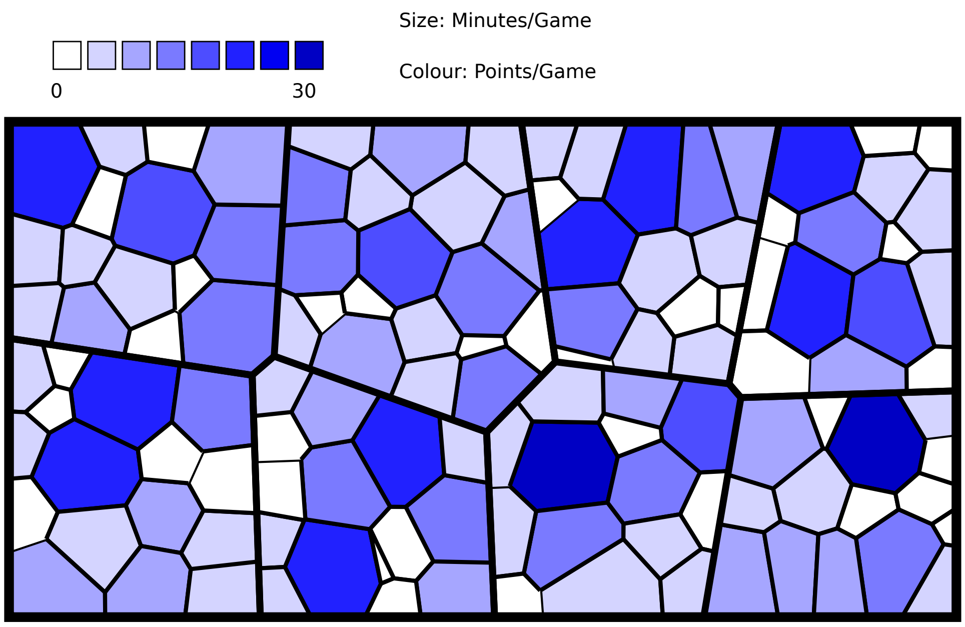


(a) Slice-and-Dice treemap visualising players in teams of the Central Division of the NBA.

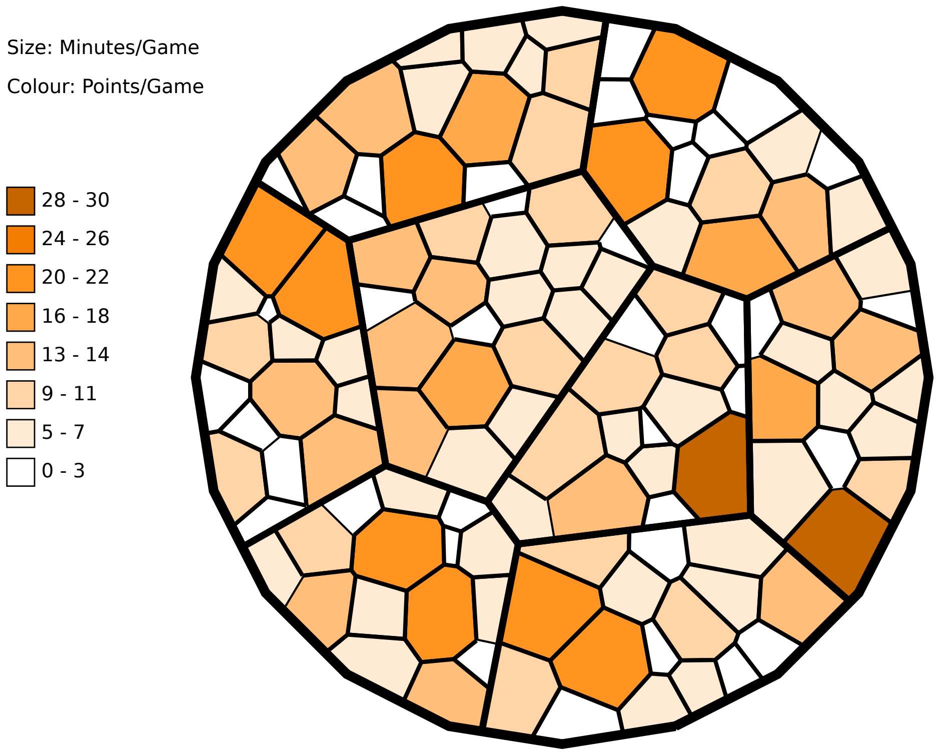


(b) Squarified treemap of the same data.

Figure 6.15: Treemaps created using two different algorithms. The area of each rectangle represents the minutes per game a player played and its shading expresses the points per game a player achieved. The statistics are taken from the example data with the treemap software from HCIL [2011].



(a) A Voronoi treemap visualising players in teams of the Central Division of the NBA.



(b) A Voronoi treemap shaped as a regular 20-sided polygon displaying the same data.

Figure 6.16: Voronoi treemaps displaying NBA statistics, created with Liquid Diagrams. The area of each polygon represents the minutes per game a player played and its shading expresses the points per game a player achieved. The statistics are extracted from the sample data provided with the treemap software from HCIL [2011].

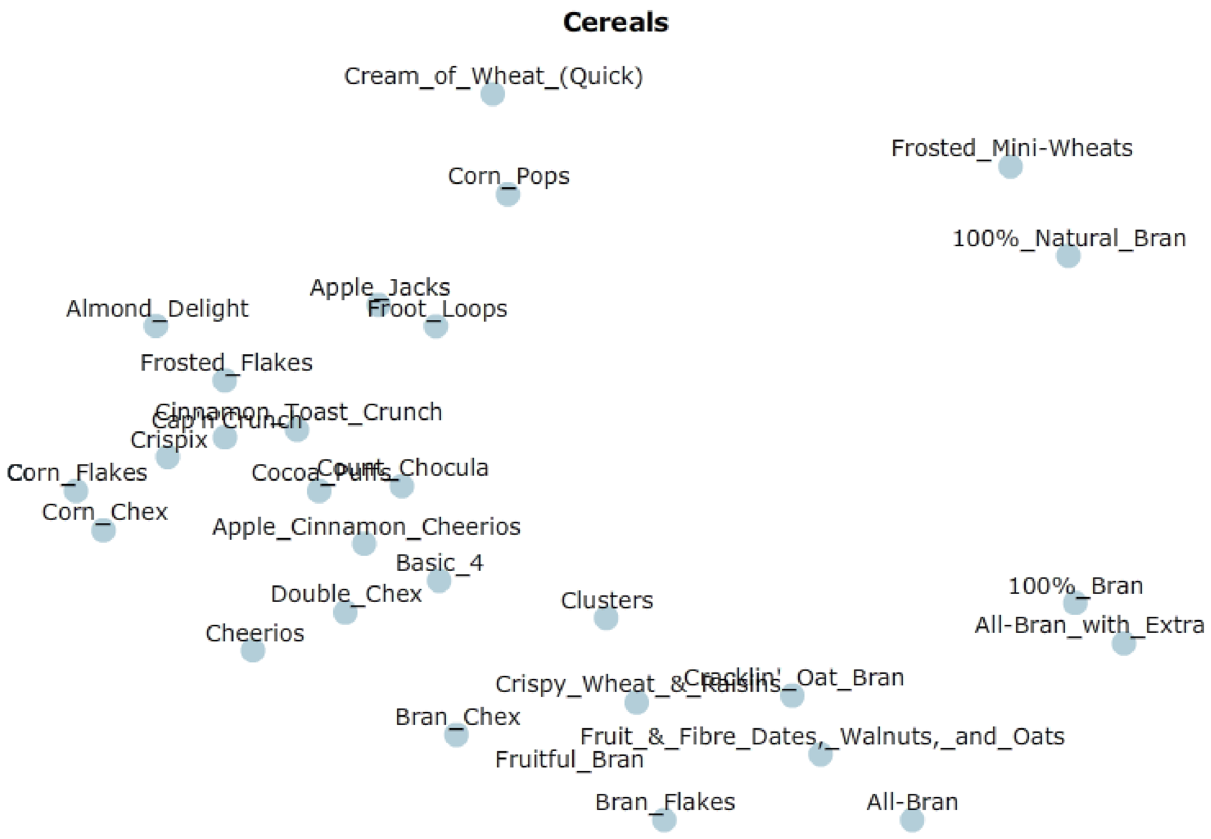


Figure 6.17: A similarity map, displaying 30 cereals from the well-known cereals dataset [StatLib, 2011]. Cereals with similar attributes will be placed closer to one another.

of a node. The algorithm takes a similarity matrix as input and the iterations are performed as long as the positions of the nodes change. Force-directed placement was invented by Eades [1984] in 1984. Further improvements to this field were made amongst others by Kamada and Kawai [1989] in 1989 and by Fruchterman and Reingold [1991] in 1991. Fruchterman and Reingold [1991] also coined the term "force-directed placement".

Bajramovic et al. [2011] implemented two different force-directed placement algorithms:

1. **Brute-Force Algorithm:** This is the original algorithm from Eades [1984]. It calculates all forces for all nodes. This results in a complexity of $\mathcal{O}(n^3)$.
2. **Chalmers '96 Algorithm:** This algorithm is an implementation of Chalmer's force-diected placement algorithm, published in 1996 [Chalmers, 1996]. It reduces the complexity of the iteration time to $\mathcal{O}(n)$ and results in an overall complexity of $\mathcal{O}(n^2)$. Due to this fact, this algorithm should be preferred for larger datasets.

To determine the similarity of the nodes, a similarity metric has to be used. Three different metrics, Euclidean distance, normalised Euclidean distance, and the cosine metric were implemented. The algorithm and the similarity metric can be chosen by selecting it from two drop-down boxes in the Algorithm tab of the Options Panel. The default algorithm is the Chalmers '96 algorithm and the default similarity metric is normalised Euclidean distance. The source code of this visualisation can be found in the package `simmap`.

Chapter 7

Changes to the Framework

When implementing version 2.0 of the Liquid Diagrams framework, it was necessary to make changes to pre-existing parts of the framework. This chapter is about these changes. Since most of the mentioned changes are deeply integrated within the framework, they were carefully designed and encapsulated in order to make further modifications as easy as possible.

First, the structure of the framework and its changes in comparison to its original structure will be discussed. Here, UML component and class diagrams will be used. Secondly, new features regarding the fonts of the Liquid Diagrams framework are shown and discussed. One of the major improvements is the loading of fonts at runtime.

7.1 Structure

7.1.1 Initial Structure

The structure of Liquid Diagrams framework version 1.0 can be seen in Figure 7.1. It displays the main Application files of the project LDFramework in a UML component diagram. For simplicity, only the files of the visualisations area chart, bar chart and bat's wing diagram are drawn. There are two MXML files for each visualisation:

1. The first file contains the code for the gadget version of the visualisation. In the initialisation process, the data of the spreadsheet and the settings for the visualisation are passed to the gadget via a callback method by using JavaScript. The detailed process of this procedure can be found in Lessacher [2010]. After this step is performed, the data is drawn by the visualisation.

2. The second file displays the same visualisation as the first file. However, the initialisation process is slightly different. The data and the options are no longer passed to the gadget via a JavaScript call. Instead, they are included statically within the code of the file. As a result, the gadget can be executed offline using a web browser and used for offline debugging purposes. This enables the developer to instantly test changes to the code.

   The reason Lessacher [2010] created this debugging mechanism was the enormous amount of work which had to be invested to test new or changed code in the gadget version of a visualisation. The procedure of testing the gadget online includes the insertion of `trace()` statements in the code and the manual upload of the changed SWF file to a web space. When this is done, the gadget can be tested with the help of Google Docs and a Flash debug and message tracing mechanism like Flashbug [Mariani and Efstathiou, 2011] for the Mozilla Firefox extension FireBug [Mozilla, 2011].

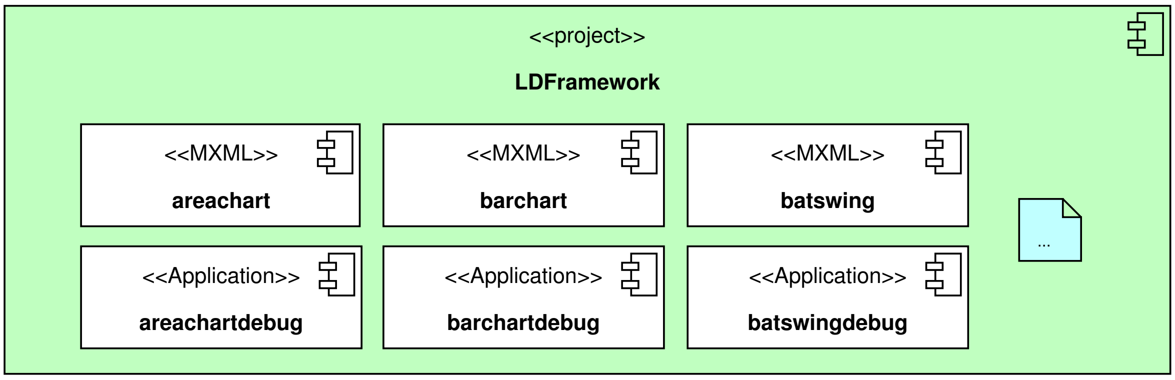


Figure 7.1: The project structure of the Liquid Diagrams framework version 1.0. The project LDFramework contains two MXML files for each visualisation. Both are of type Application and can be executed. The file with the suffix "debug" in its name contains static data and is intended for debugging purposes. The other file represents the gadget version of the visualisation. For simplicity, the files of only three different visualisations are drawn.

Apart from the different initialisation process of the debug file, the remaining code is the same in both the gadget and debug versions of each visualisation.

The existence of two files, which only differ in their initialisation process, leads to a major problem. As described above, the rest of the code in the two files is identical in order to provide and execute the same visualisation in the online version and in the offline version. This means that if the code in one file is changed, it has to be copied into the other file as well, in order to maintain the integrity of the visualisation. This procedure is extremely time-consuming and error-prone.

7.1.2 Current Structure

An early refactoring approach was the elimination of the debug file for each visualisation. This was done by encapsulating the initialisation part into separate classes. Here, the strategy pattern [Gamma et al., 1995, pages 315–324] was used. The strategy pattern is a design pattern defining a family of encapsulated algorithms which are interchangeable at runtime.

In addition, a debug flag in the class ChartPanel was defined. This enables the loading of a different initialisation strategy at startup. The ChartPanel is the perfect choice for locating the debug flag, because it represents the drawing area included in all visualisations. Figure 7.2 shows a class diagram of this procedure. For simplicity, only the classes of two visualisations (area chart and bar chart) are shown. The visualisations in Figure 7.2 have a member variable (`initializationStrategy`) which is a concrete instance of an implementation of the interface IInitializationStrategy. At startup, the debug flag is used to determine which IInitializationStrategy has to be instantiated. For example, if the area chart visualisation is started, an instance of AreaChartDebugInitStrategy is created if the call `chartPanel.isStandaloneCompiling()` returns true, and an instance of DefaultInitStrategy is instantiated if it returns false. This mechanism allows separate compilation of the debug and the gadget versions of a visualisation and solves the previously mentioned problem.

The next step in changing the framework structure was the integration of a separate standalone version of Liquid Diagrams. Here, the debug version of the framework was extended in various ways. Since a standalone version needs a different data source to the gadget version, a mechanism for loading a data

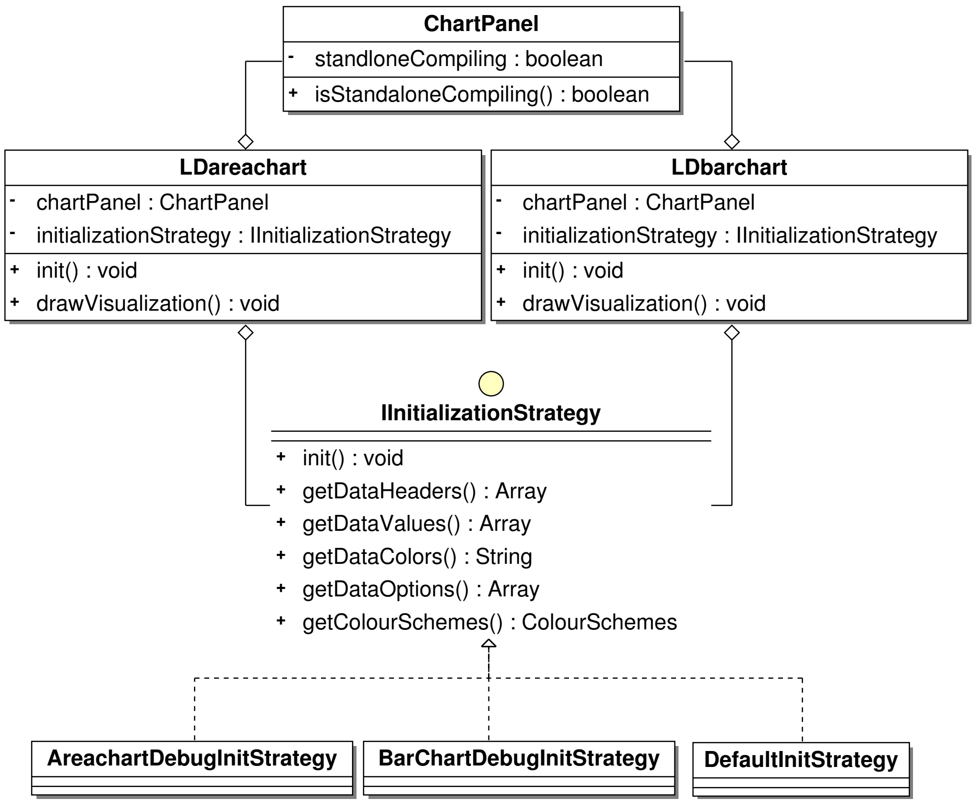

ChartPanel
- standloneCompiling : boolean
+ isStandaloneCompiling() : boolean
LDareachart
- chartPanel : ChartPanel
- initializationStrategy : IInitializationStrategy
+ init() : void
+ drawVisualization() : void
LDbarchart
- chartPanel : ChartPanel
- initializationStrategy : IInitializationStrategy
+ init() : void
+ drawVisualization() : void
IInitializationStrategy
+ init() : void
+ getDataHeaders() : Array
+ getDataValues() : Array
+ getDataColors() : String
+ getDataOptions() : Array
+ getColourSchemes() : ColourSchemes
AreachartDebugInitStrategy
BarChartDebugInitStrategy
DefaultInitStrategy

Figure 7.2: The first modification to the structure of Liquid Diagrams framework. Each visualisation has a concrete instance of a implementation of the interface IInitializationStrategy. A debug flag in the class ChartPanel decides which implementation of IInitializationStrategy is instantiated on startup. This is an implementation of the strategy pattern [Gamma et al., 1995, pages 315–324]. For simplicity, only the visualisations area chart and bar chart are drawn.

file was implemented. The original version of this mechanism was implemented by Haintz et al. [2011] in the course "Information Visualization" [Andrews, 2011a] during the summer term 2011 at Graz University of Technology. This extension resulted in many redundancies regarding the previously mentioned debug initialisation strategies, including static data and static options for each visualisation which were no longer needed.

The integration of Adobe AIR into the standalone version caused extensive changes to the structure of the framework as well, because in order to create a clean implementation, the project LDFramework (see in Figure 7.1) had to be divided into three different Adobe Flash Builder projects:

1. **LDFramework:** This project contains most of the code from the old project and works as a common codebase for the other two projects. In addition, it defines the interface IInitializationStrat egy which is implemented in the other two projects.
2. **LDFrameworkGadget:** The gadget version of the Liquid Diagrams framework. It includes a link to the code of the project LDFramework and implements the interface IInitializationStrategy. The main files of this project are of type Application and work as a wrapper for the main visualisation file, located in LDFramework. When this project is compiled, it outputs an SWF file for each visualisation which can be used as gadgets with Google Docs.
3. **LDFrameworkAIR:** This project contains the framework-specific code for the standalone version of Liquid Diagrams. This includes data import and data storage functionality. Like the project LD

FrameworkGadget it contains a link to the common codebase (LDFrameworkGadget) and implements the interface IInitializationStrategy. It defines a WindowedApplication, which is in fact a wrapper for the main file of every visualisation. Additionally, the WindowedApplication liquiddiagrams contains all visualisations at once. When this project is compiled, installers for standalone applications (see Section 5.3.5) can be created.

The original idea for this approach was taken from an article in the Adobe Developer Center [Prekaski, 2011]. All three projects and their interactions are illustrated in Figure 7.3. Details about the implementation can be found in Section 8.1.

7.2 Fonts

The font handling mechanism in the Liquid Diagrams framework underwent major improvement. Besides the restructuring and improving of the user interface elements, fonts are now loaded at runtime. This section describes which changes were made and why they were necessary. In addition, it is explained how to add new fonts to the framework.

The initial situation before refactoring the framework included a Font button in the Options Panel. A font chooser pop-up window appeared, where the user was able to select a font type and a font size for a specific font category. Six font categories (diagram title, axis title, axis label, legend, tool tip, and chart text) were available. By changing one of the settings a preview of the selected font in the chosen size appeared at the bottom of the pop-up window. A click on the OK-button closed the window and the font settings were adopted by the visualisation. Figure 7.4a shows the appearance of the initial font chooser in version 1.0 of Liquid Diagrams.

While modifying the Options Panel (as described in Section 5.5.1) a new Fonts tab was created, containing the font options. Besides the original functionality, four buttons were added to the font chooser component. The first button allows changing the font colour of the specified font category. By clicking it, a colour chooser pop-up window appears. After pressing the OK-button of the window, the selected colour is applied to the font category in the visualisation. As feedback, the button changes colour as well. The last three buttons give the user the possibility to change the font weight, font style and the text decoration of the corresponding font category. The font weight changes the font to bold, the font style to italic, and the text decoration to underline. With the bold and italic options the whole font family of the chosen font type can be selected. A screenshot of the new font dialogue can be seen in Figure 7.4b. Figure 7.5 shows the font options from Figure 7.4b applied to the pie chart visualisation.

Unfortunately, the new font options involved new problems which had to be solved. In Adobe Flex, fonts have to be embedded in order to be displayed properly. By adding the options to change the font weight and the font style (bold and italic) of a font, Flex was not able to display the fonts correctly in some situations. The problem occurred when the bold, italic or bold and italic text was rotated. This was the case in visualisations which included a horizontal axis (x-axis). The x-axis label styles included in the Liquid Diagrams framework contain an option whereby the label of the x-axis ticks can be rotated by 45°. When this option was selected, the label disappeared. The reason for this behaviour was that just one font of the whole font family was embedded into the code. Embedding the rest of the fonts would have meant increasing the size of the compiled flash file dramatically. For this reason, the decision was made to dynamically load selected fonts at runtime. The loading of fonts is done by the class FontLoader in package `common.fonts`.

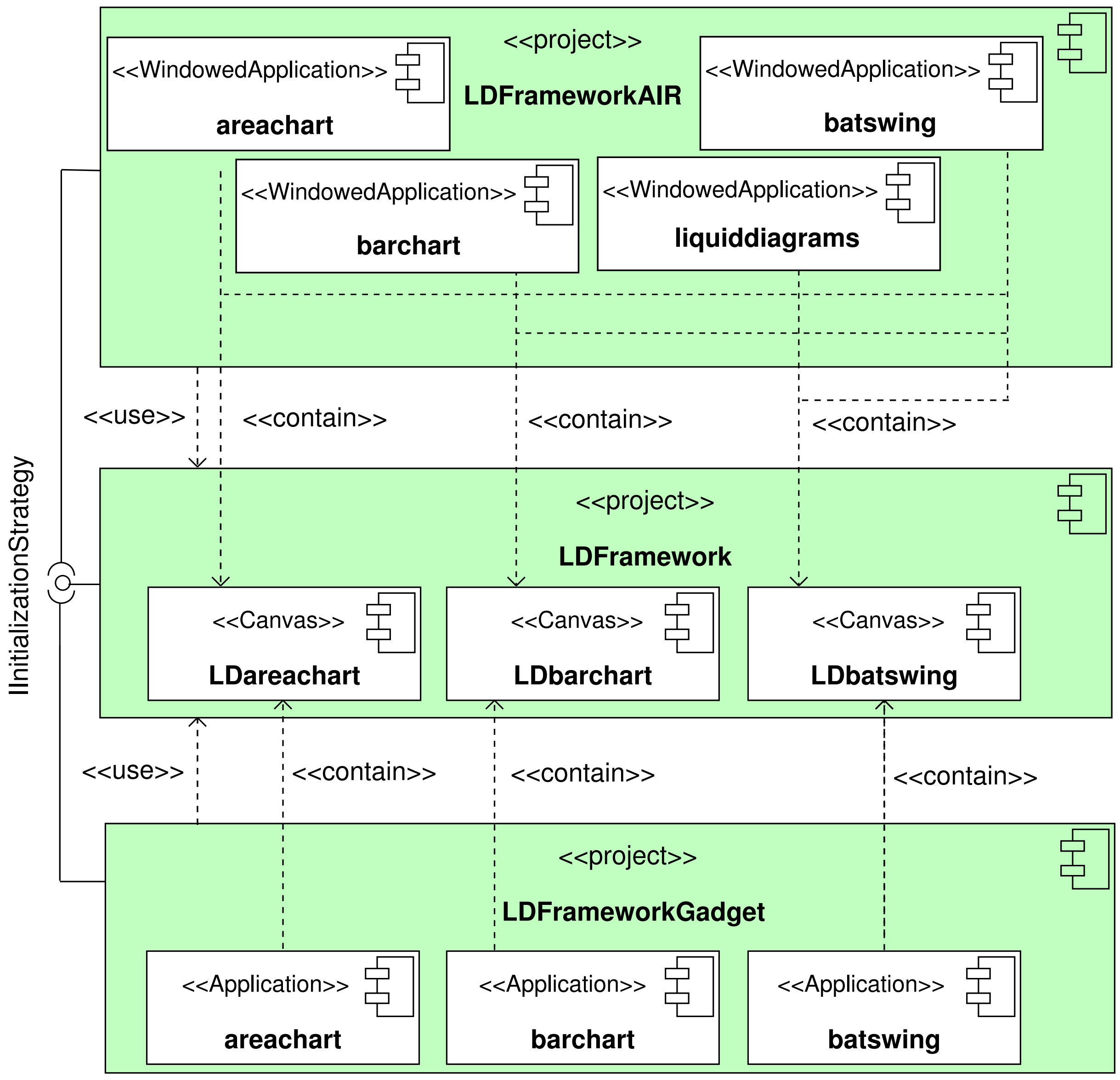
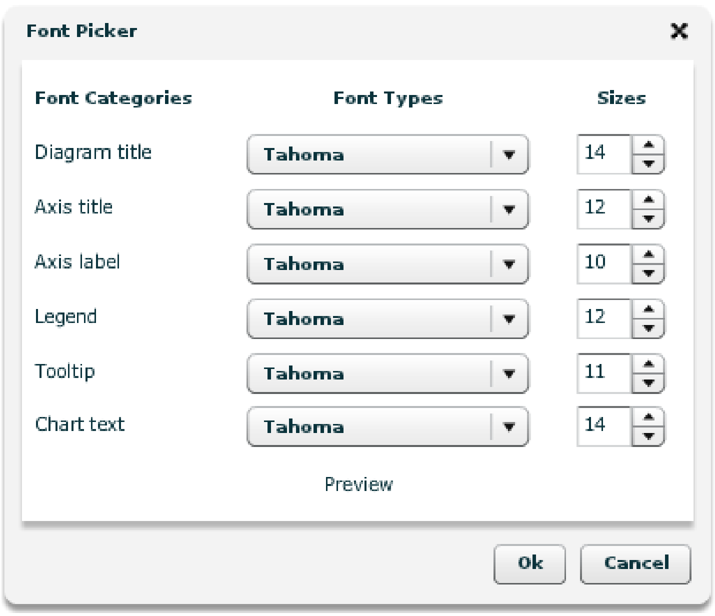


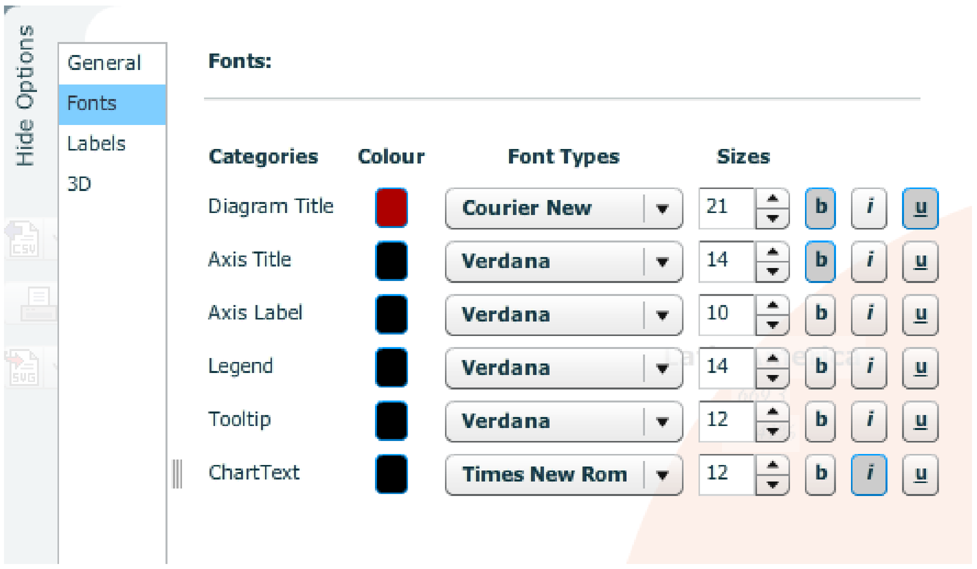
Figure 7.3: The general structure of the Liquid Diagrams framework version 2.0 in a UML component diagram. The framework is split into three different projects. The project in the middle (LDFramework) contains the resources which are referenced by the other two projects (LDFrameworkAIR and LDFrameworkGadget). It defines an interface for the initialisation process. The concrete implementations of this interface are included in the other projects. LDFrameworkAIR compiles the files for the standalone version of the framework and LDFrameworkGadget the ones for the gadget version. For simplicity, the diagram does not contain all application components.



Font Picker

Font Categories	Font Types	Sizes
Diagram title	Tahoma	14
Axis title	Tahoma	12
Axis label	Tahoma	10
Legend	Tahoma	12
Tooltip	Tahoma	11
Chart text	Tahoma	14

(a) The original font chooser pop-up window version 1.0.



Fonts:

Categories	Colour	Font Types	Sizes			
Diagram Title	🟥	Courier New	21	[✓ b]	i	[✓ u]
Axis Title	⬛	Verdana	14	[✓ b]	i	u
Axis Label	⬛	Verdana	10	b	i	u
Legend	⬛	Verdana	14	b	i	u
Tooltip	⬛	Verdana	12	b	i	u
ChartText	⬛	Times New Rom	12	b	[✓ i]	u

(b) The redesigned font chooser menu component in the opened Options Panel. It adds buttons for font colour, font style (italic), font weight (bold) and text decoration (underline).

Figure 7.4: The font component in the Liquid Diagrams Framework. (a) shows the initial state at version 1.0 (b) shows the font component after modification in version 2.0.

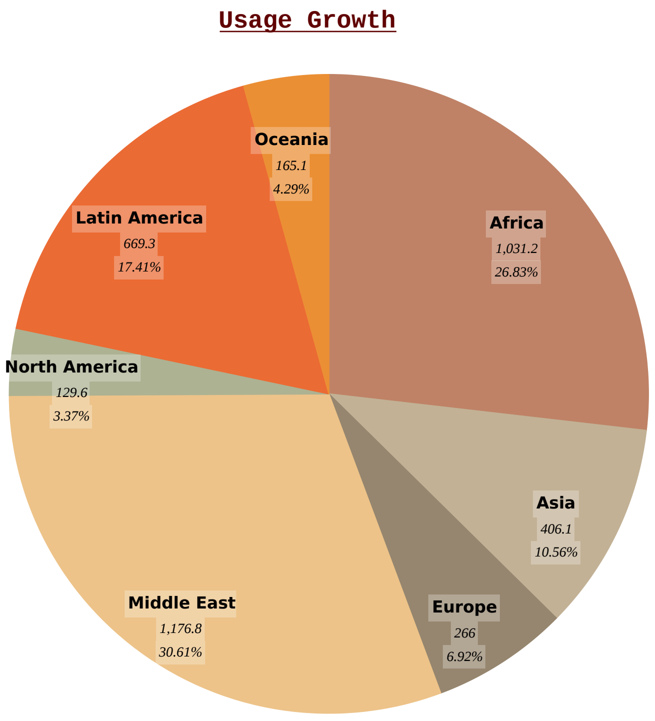


Figure 7.5: The Liquid Diagrams pie chart visualisation showing the font options from Figure 7.4b applied to the title (Diagram Title) and the slice labels (Axis Title and Chart Text).

Additionally, the text displayed in the visualisations was modified to support a coloured label background. This background can be enabled in various visualisations using the user interface controls in the Labels tab of the Options Panel. The background can be set to any transparency from 0% to 100%.

All these features are implemented in the newly created class ChartText. This class is derived from the standard Flex class Text and supports the automatic centering and the setting of the font style and background settings of the corresponding label. In addition, it automatically generates the SVG string used when exporting the visualisation as an SVG file.

Before refactoring the font component, the selected default font for text was Tahoma. Tahoma was designed as a two font typeface by Matthew Carter and Tom Rickner, as a system font for the operating system Windows95. [Will-Harris, 2003] The problem with Tahoma as the default font is that by including the option to set the font style of a font this option could not be applied to Tahoma because as a two font typeface it does not support italic or bold-italic. For this reason, the decision was made to set the default typeface of the visualisations to Verdana. The Verdana typeface is embedded into each visualisation in its normal form and the rest of Verdana's font family and the rest of the available font families are loaded at runtime if they are selected by the user. By default, the following fonts are currently available: Arial, Courier New, Georgia, Tahoma, Times New Roman and Verdana.

In order to import fonts at runtime into the main application, the font has to be pre-compiled into a Flash file. This task can be achieved using the code in Listing 7.1 inside an ActionScript project in Adobe Flash Builder. The Fonts are then available as SWF files.

The loading of fonts at runtime has the advantage that the file size of the main SWF file decreases, because most fonts are no longer embedded into the code. In numbers, the file size decreased from 2.1 megabytes (MB) to approximately 1.4 megabytes, an improvement of about 33%.

```
package {

  import flash.display.Sprite;

  public class Arial extends Sprite {

    [Embed(source='font/arial.ttf',
      embedAsCFF='false',
      fontFamily='Arial',
      fontName='Arial',
      fontWeight='normal',
      fontStyle='normal',
      mimeType='application/x-font',
      advancedAntiAliasing='true'
    )]
    public static var Arial:Class;
  }
}
```

Listing 7.1: The code necessary to embed a font in ActionScript. This code can be compiled into a SWF file, which can be loaded on runtime in order to access the embedded font in the main application.

Chapter 8

Selected Details of the Implementation

“ Words have been interchanged enough,
Let me at last see action too. ”

[Johann Wolfgang von Goethe, Faust: the first part of the tragedy, 1808.]

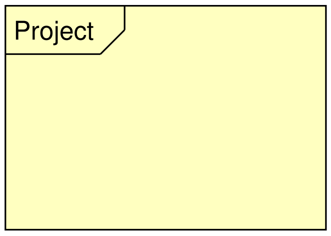
This chapter describes some of the finicky little details, which take a great deal of time to get right and which deserve to be explained in their own right. This includes the framework’s general structure and initialisation process, the import and export of project files, and its 3D functionality. In the following, UML class diagrams are used to explain the structure of the described details. A legend explaining elements used in these class diagrams can be seen in Figure 8.1.

8.1 General Structure and Initialisation Process

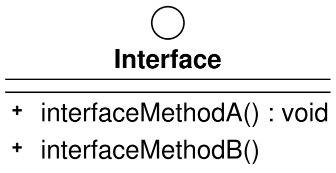
In Section 7.1, it was described how the structure of the Liquid Diagrams framework changed compared to its initial state. Three different projects enable the creation of the gadget version of the framework and its standalone pendant. The idea for this approach was taken from an article in the Adobe Developer Center written by Prekaski [2011]. The article describes the building of Adobe Flex and Adobe AIR applications from the same code base. The focus of this section is on how the general idea of this article was applied to the Liquid Diagrams framework.

As written in Section 7.1, the workspace of the Liquid Diagrams framework consists of three different projects: LDFramework, LDFrameworkGadget, and LDFrameworkAIR. The project LDFramework works as a common codebase for the other two projects and it contains all the shared code for all visualisations. The projects LDFrameworkAIR and LDFrameworkGadget reference the common codebase and output the final SWF files when they are compiled. There exists no link between LDFrameworkAIR and LDFrameworkGadget in order to strictly separate the different implementations.

In the following, the detailed initialisation process of a visualisation is described. The architecture involved can be seen in Figure 8.2. The project LDFramework contains the main files of the visualisations. These are MXML files which extend the Flex class Canvas and whose file names start with the prefix “LD”. In Figure 8.2, only two of them (LDareachart and LDbarchart) are displayed. Each of the files is contained within a corresponding MXML file from the project LDFrameworkAIR. One of these files can be seen in Listing 8.1. The project LDFrameworkGadget is not shown, because it includes mainly the same classes and relations as the project LDFrameworkAIR. The only difference is that the class StandaloneDefaultInitStrategy would be replaced by DefaultInitStrategy as the concrete implementation of



(a) UML Fragments are used to illustrate Adobe Flash Builder projects.



(b) An example of an interface.

Figure 8.1: Legend of UML class diagram elements. (a) shows an example of an Adobe Flash Builder project (b) illustrates the appearance of an interface.

the interface IInitializationStrategy.

At the beginning of the initialisation process of every application, the function `init()` from the main file of the included visualisation is called. This function can be seen in Listing 8.2. The context of the current application is determined by calling the function `chartPanel.isStandaloneCompiling()`. The implementation of this function can be seen in Listing 8.3. If it returns true, the application's context is an Adobe AIR project (LDFrameworkAIR) otherwise it is a gadget project (LDFrameworkGadget). The member variable `initializationStrategy` is retrieved by the class InitStrategyFactory. This is an implementation of the factory method pattern [Gamma et al., 1995, pages 107–116] and can be seen in Figure 8.4. Its purpose is the instantiation of a concrete type of the interface IInitializationStrategy without knowing the type. This is done using the Flex class ApplicationDomain, together with the textual representation of the concrete class.

This procedure is the main feature of the initialisation process. It enables the two projects LDFrameworkGadget and LDFrameworkAIR to coexist without knowing of their mutual co-existence. The shared codebase LDFramework does not need to know about the two projects which use it.

Since Adobe Flash Builder tries optimise the output when a project is compiled, it does not include code which is not referenced into the final SWF output. For that reason, the main file of an application has to include the concrete implementation of the interface IInitializationStrategy. This can be seen in Listing 8.1 line 16 where the member variable `onlyForCompilationInit` is declared, but only instantiated as a null reference.

8.2 Cookies and Project Files

In Section 5.4 it was described how cookies are used to store the options of a visualisation in the newly created cookies version of the Liquid Diagrams framework. The architecture invoked in this process was carefully designed in order to be extensible for various other storing methods. The ability to store project files, described in Section 5.3.4, is one of these methods. In the following, this architecture and its corresponding code are described. Figure 8.3 shows a diagram of the involved classes. As can be seen, the architecture makes use of the technique described in Section 8.1 of instantiating classes without knowing the concrete type.

The starting point of the export of settings is the main file of a visualisation. In Figure 8.3, this is the class LDvoronoi. It has a member variable `settingsHandler` of type SettingsHandler. In every other programming language, this class would be abstract but due to the fact that ActionScript does not

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <mx:WindowedApplication
 3   xmlns:mx="http://www.adobe.com/2006/mxml"
 4   layout="vertical" verticalAlign="top" horizontalAlign="left"
 5   xmlns:local="*" applicationComplete="init()"
 6   xmlns:s="library://ns.adobe.com/flex/spark" width="800" height="600"
 7   paddingLeft="0" paddingRight="0" paddingBottom="0" paddingTop="0"
 8   backgroundColor="0x7a909b" resize="onResize()"
 9   title="Liquid Diagrams - Polar Area Diagram">
10
11   <mx:Script>
12     <![CDATA[
13       import diagram.init.StandaloneDefaultInitStrategy;
14       import diagram.settings.StandaloneSettingsStorageProvider;
15
16       private static const onlyForCompilationInit:StandaloneDefaultInitStrategy =
             null;
17       private static const
             onlyForCompilationSettings:StandaloneSettingsStorageProvider = null;
18
19       private var application:LDpolararea;
20
21       private function init() : void {
22
23         this.application = new LDpolararea();
24         this.application.width = this.width - 5;
25         this.application.height = this.height - 20;
26
27         addChild(this.application);
28       }
29
30       private function onResize() : void {
31
32         if (application != null) {
33           application.width = this.width - 5;
34           application.height = this.height - 20;
35           application.onResize();
36         }
37       }
38
39     ]]>
40   </mx:Script>
41
42 </mx:WindowedApplication>
```

Listing 8.1: The main code of the polar area diagram in the project LDFrameworkAIR. Compilation outputs the SWF file for the standalone version of the Liquid Diagrams framework and instantiates and includes an instance of the class LDpolararea. In line 16, the member variable `onlyForCompilationInit` is included in this MXML file, because otherwise the Flash Builder would not include the code for the class StandaloneDefaultInitStrategy in the compiled output. This is because the class InitStrategyFactory (shown in Listing 8.4) does not know the type of the class it is instantiating.

```
 1  /**
 2   * init() is called after the initialisation process is finished
 3   *   (= first function to be run)
 4   */
 5  public function init():void {
 6
 7          if (initializationStrategy == null) {
 8                  initializeFonts();
 9
10                  var initFactory:InitStrategyFactory = new InitStrategyFactory(
                        chartPanel.isStandaloneCompiling());
11
12                  initializationStrategy = initFactory.getInitInstance();
13                  initializationStrategy.setApplication(this);
14                  initializationStrategy.setStandaloneDefaultOptions(
                        STANDALONE_DEFAULT_DATA_OPTIONS);
15                  initializationStrategy.init();
16          }
17  }
```

Listing 8.2: The function `init()` is called at the beginning of the initialisation process of every visualisation. The concrete instance of the interface IInitializationStrategy is retrieved in line 10 by calling the function `getInitInstance()` of an InitStrategyFactory. The visualisation itself does not know in which context it is running and therefore does not know which instance of IInitializationStrategy is instantiated. The ChartPanel's function `isStandaloneCompiling()` can be seen in Listing 8.3 and the InitStrategyFactory is shown in Listing 8.4.

```
public function isStandaloneCompiling() : Boolean {
  return Security.sandboxType.toString() == "application" ? true : false;
}
```

Listing 8.3: The function `isStandaloneCompiling()` in the class ChartPanel is responsible for determining in which context the current application runs. If this function returns true, the context is an Adobe AIR application, otherwise it is a gadget application.

```
package martin.diagram.init
{
        import flash.system.ApplicationDomain;
        import flash.utils.getDefinitionByName;

        public class InitStrategyFactory
        {
                private var isAir:Boolean;

                public function InitStrategyFactory(isAir:Boolean) {
                        this.isAir = isAir;
                }

                private function getClassToCreate(className:String) : Object {

                        var someClass:Object = null;
                        someClass = ApplicationDomain.currentDomain.getDefinition(
                            className);
                        return someClass;
                }

                public function getInitInstance() : IInitialisationStrategy {

                        var strategy:IInitialisationStrategy;
                        var cls:String = (isAir ? "diagram.init.
                            StandaloneDefaultInitStrategy" :
                                "diagram.init.DefaultInitStrategy");
                        var clsToCreate:Object = getClassToCreate(cls);
                        strategy = new clsToCreate();
                        return strategy;
                }
        }
}
```

Listing 8.4: The class InitStrategyFactory is an implementation of the factory method pattern [Gamma et al., 1995]. It instantiates objects of type IInitializationStrategy without knowing the concrete type. This is done using the Flex class ApplicationDomain, together with the name of the concrete class. If this code runs in an Adobe AIR application, the instantiated class is of type StandaloneDefaultInitStrategy, otherwise it is DefaultInitStrategy.

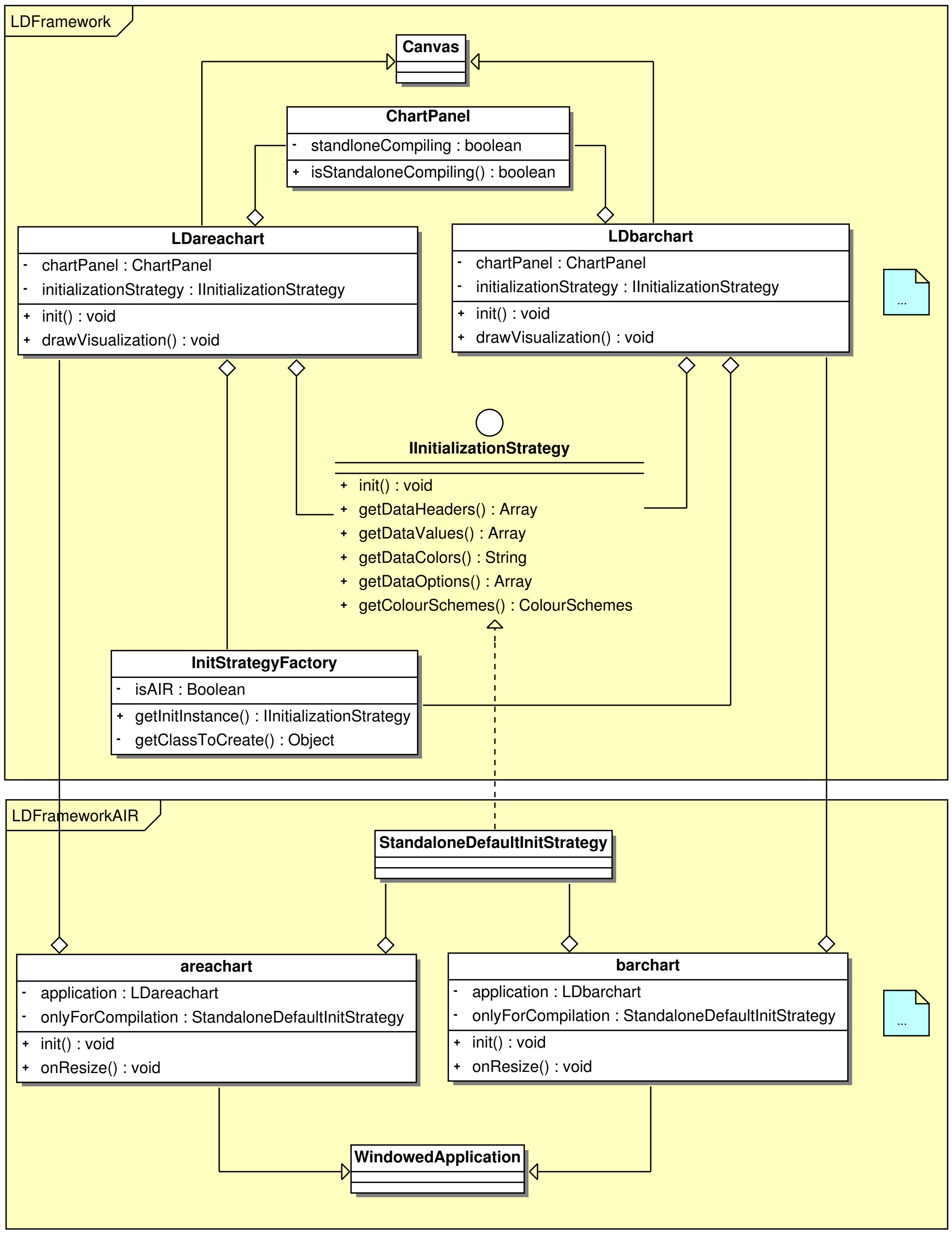

LDFramework
Canvas
ChartPanel
- standloneCompiling : boolean
+ isStandaloneCompiling() : boolean
LDareachart
- chartPanel : ChartPanel
- initializationStrategy : IInitializationStrategy
+ init() : void
+ drawVisualization() : void
LDbarchart
- chartPanel : ChartPanel
- initializationStrategy : IInitializationStrategy
+ init() : void
+ drawVisualization() : void
...
IInitializationStrategy
+ init() : void
+ getDataHeaders() : Array
+ getDataValues() : Array
+ getDataColors() : String
+ getDataOptions() : Array
+ getColourSchemes() : ColourSchemes
InitStrategyFactory
- isAIR : Boolean
+ getInitInstance() : IInitializationStrategy
- getClassToCreate() : Object
LDFrameworkAIR
StandaloneDefaultInitStrategy
areachart
- application : LDareachart
- onlyForCompilation : StandaloneDefaultInitStrategy
+ init() : void
+ onResize() : void
barchart
- application : LDbarchart
- onlyForCompilation : StandaloneDefaultInitStrategy
+ init() : void
+ onResize() : void
...
WindowedApplication

Figure 8.2: The classes involved in the initialisation process of two applications of the Liquid Diagrams framework. For simplicity, the files of only two applications (area chart and bar chart) are shown. The implementation of the main parts of the code can be seen in Listings 8.1, 8.2, 8.3, and 8.4. The project LDFrameworkGadget is not shown in this class diagram, because its classes and relations are almost the same as in the project LDFrameworkAIR.

allow setting methods as abstract, it is a normal class.The SettingsHandler has a reference to itself which makes it possible to create a chain of SettingsHandler objects. This is an implementation of the chain of responsibility pattern [Gamma et al., 1995, pages 223–232]. It allows the creation of an object structure which is able to sequentially process an event. In this case, the event is about storing and deleting settings from the storing mechanism. The SettingsHandler has a number of classes derived from it. The classes OptionsSettingsHandler, FontSettingsHandler and DataItemSettingsHandler are shown in Figure 8.3. In addition, the class LegendSettingsHandler is also available. Each of these classes has the following purpose:

- **OptionsSettingsHandler:** This class handles the settings for the standard options in each visualisation. The standard options mapping consists of an array `options`, several integer constants, and an array containing the textual representation of the settings in each visualisation. The integer constant specifies the position (index) in the array at which a certain setting and its textual name can be found.
- **FontSettingsHandler:** This SettingsHandler is responsible for the font options of each gadget. The font options can be found in the Options Panel by clicking on the Fonts tab. The FontSettingsHandler stores all the initial font settings and checks if one of the current settings differ from the default settings. Since the Liquid Diagrams framework supports runtime font loading, this class contains a reference to the FontLoader. More about fonts can be found in Section 7.2.
- **DataItemSettingsHandler:** In many of the visualisations in the Liquid Diagrams framework, users have the option to change the colour of a certain data entity. Options like this can be saved with the DataItemSettingsHandler. Currently, only the colour of a data entity is handled. However, the saving of options like selection and visibility could be integrated within this class as well.
- **LegendSettingsHandler:** This class saves the settings of the colour legend. This legend type is only available in the heatmap, treemap, and Voronoi treemap visualisations and computes the colour of each entity according to its specific data value and the data distribution type. The Legend SettingsHandler saves the distribution type, the data range, and the main colour of the legend.

The reason why the chain of responsibility pattern was used to implement of this functionality is because the visualisations in the framework have different requirements regarding the availability of settings. For example, the Voronoi treemap forms a SettingsHandler chain consisting of the classes OptionsSettingsHandler, FontSettingsHandler, and LegendSettingsHandler, on the contrary the area chart's chain consists of the classes OptionsSettingsHandler, DataItemSettingsHandler, and FontSettingsHandler.

In the next step of the storage procedure, the member variable `factory` comes into play. This variable is of the interface type ISettingsStorageFactory and its concrete implementation DefaultSettingsStorageFa ctory is responsible for creating a concrete ISettingsStorageProvider. This implementation of the factory method pattern [Gamma et al., 1995, pages 107–116] makes use of the already mentioned technique of creating class instances without knowing the concrete type. The implementations of the interface ISettingsStorageProvider (StandaloneSettingsStorageProvider and CookieSettingsStorageProvider) can be found in the projects LDFrameworkAIR and LDFrameworkGadget. The purpose of these two classes are:

- **StandaloneSettingsStorageProvider:** This ISettingsStorageProvider stores the settings of a visualisation in its member variables. This is done because in the standalone version of the framework, an instant storage mechanism like in the cookies version is not needed. The settings in the StandaloneSettingsStorageProvider are only used if a project is exported. Since the application liquiddiagrams in the standalone version of the framework allows the launch of multiple visualisations at the same time, a single instance of StandaloneSettingsStorageProvider is not enough.

That is why the class SettingsStorageProviderPool was created. It handles the administration of ISettingsStorageProvider instances by returning the correct instance for a particular visualisation.

- **CookieSettingsStorageProvider:** This class is responsible for saving settings to and from a cookie. The Flex class ExternalInterface is used to call the JavaScript functions described in Section 5.4.

8.3 3D Functionality

In order to present the class architecture of the 3D functionality in a structured way, a distinction is made between the starting procedure and its drawing process. Figure 8.4 displays the class diagram of all components engaged in the 3D starting procedure. The architecture of the final drawing process is shown in Figure 8.5.

If a user checks the Enable 3D check-box in the 3D tab of the Options Panel, the visualisation triggers a process involving the classes in Figure 8.4. Here, a distinction between a visualisation implementing effects (pie chart) and a visualisation without effects (heatmap) has to be made (see Section 5.6). An effect in Flex is basically a timer, which updates a target multiple times within a defined duration.

The pie chart uses the code in Listing 8.5 to set up its member variable `rotation3DEffect` and call the function `play()`. Setting up this attribute means setting a duration for the effect and defining its target, modified in each effect update. This target is the member variable `pathContainer`. The `rotation3DEffect` is of type PathDrawingEffect, a subclass of the Flex class TweenEffect. It is responsible for performing an effect, in which an implementation of the interface IEffectInstance is automatically updated, calling its function `onTweenUpdate()`. The concrete implementation of the interface, the class PathDrawingEffectInstance is responsible for updating the attribute `percentageFinished` of its target. In addition, it notifies an instance of the interface DrawingStrategy. The DrawingStrategy has two concrete implementations: Rotation3DDrawingStrategy and Extrusion3DDrawingStrategy, representing the two 3D states. Again, this is an implementation of the strategy pattern [Gamma et al., 1995, pages 315–324]. The concrete implementation of DrawingStrategy uses the attribute `percentageFinished` of the target of the effect, an object of type PathContainer, to apply transformations to concrete implementations of the interface IPath. An IPath represents a transformable data entity. The PathContainer stores an array of IPath objects and provides access to it, so the DrawingStrategy can perform its transformations.

The heatmap does not include any effects due to the fact that an IPath in this visualisation may consist of hundreds or thousands of points and transforming them would take too long. This is why the class LDHeatmap directly accesses the concrete implementation of the interface DrawingStrategy in order to start the 3D procedure.

The architecture involved in the transformation of data entities, represented by concrete instances of the interface IPath and the perspective projection, can be seen in Figure 8.5. For simplicity, multiple attributes and operations are excluded from the diagram. The interface IPath is implemented by the classes Path and PathDecorator. Path provides a default implementation of the methods defined in IPath. The class PathDecorator is the core element of the implemented decorator pattern [Gamma et al., 1995, pages 175–184]. This design pattern enables the dynamic addition of behaviour to classes by wrapping them. The PathDecorator wraps an implementation of IPath and its subclasses HeatMapPathDecorator and PieChartPathDecorator use this attribute to invoke the default implementation of the class Path in addition to their own implementation of certain functions.

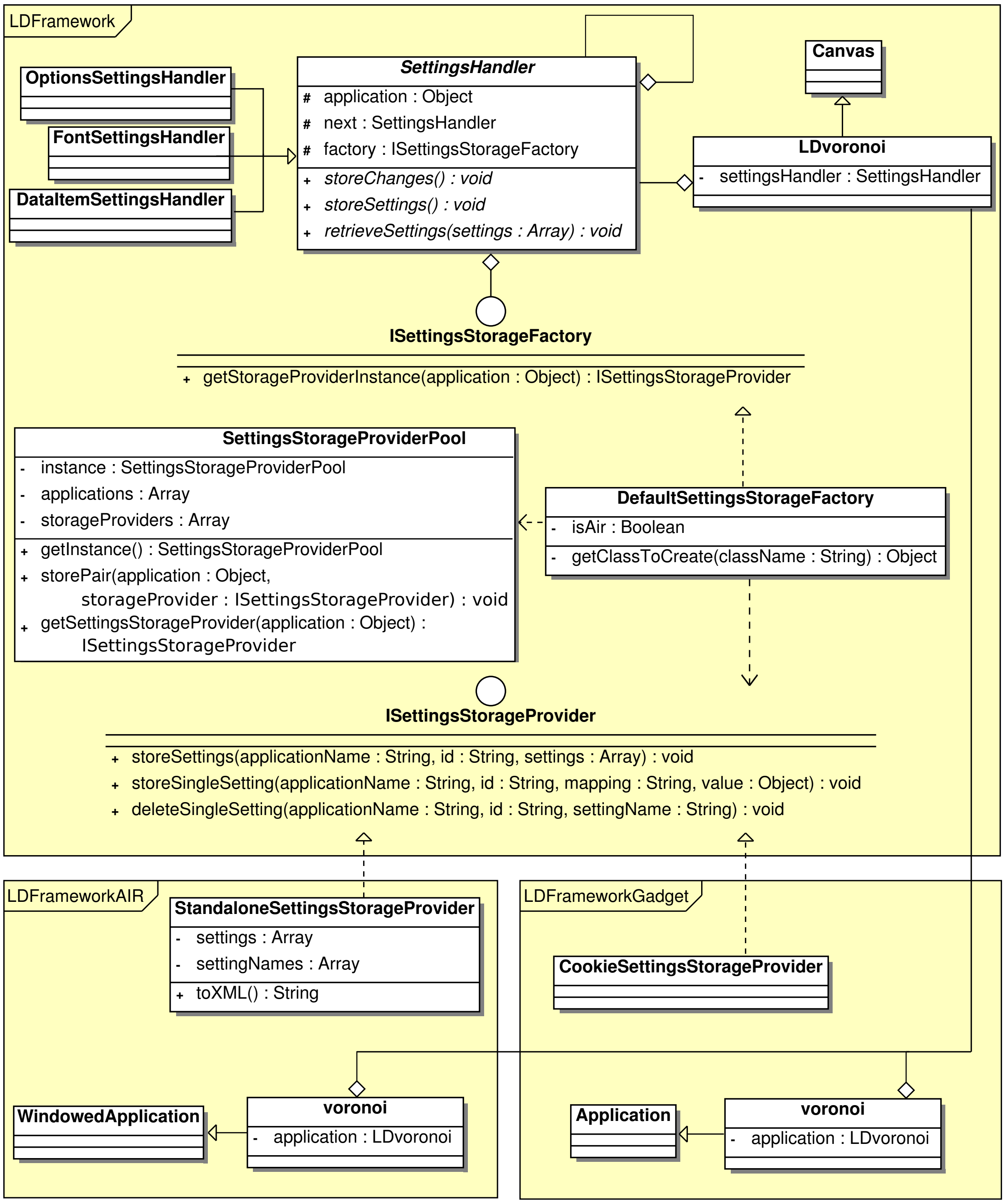

LDFramework
OptionsSettingsHandler
FontSettingsHandler
DataItemSettingsHandler
SettingsHandler
# application : Object
# next : SettingsHandler
# factory : ISettingsStorageFactory
+ storeChanges() : void
+ storeSettings() : void
+ retrieveSettings(settings : Array) : void
Canvas
LDvoronoi
- settingsHandler : SettingsHandler
ISettingsStorageFactory
+ getStorageProviderInstance(application : Object) : ISettingsStorageProvider
SettingsStorageProviderPool
- instance : SettingsStorageProviderPool
- applications : Array
- storageProviders : Array
+ getInstance() : SettingsStorageProviderPool
+ storePair(application : Object, storageProvider : ISettingsStorageProvider) : void
+ getSettingsStorageProvider(application : Object) : ISettingsStorageProvider
DefaultSettingsStorageFactory
- isAir : Boolean
- getClassToCreate(className : String) : Object
ISettingsStorageProvider
+ storeSettings(applicationName : String, id : String, settings : Array) : void
+ storeSingleSetting(applicationName : String, id : String, mapping : String, value : Object) : void
+ deleteSingleSetting(applicationName : String, id : String, settingName : String) : void
LDFrameworkAIR
StandaloneSettingsStorageProvider
- settings : Array
- settingNames : Array
+ toXML() : String
WindowedApplication
voronoi
- application : LDvoronoi
LDFrameworkGadget
CookieSettingsStorageProvider
Application
voronoi
- application : LDvoronoi

Figure 8.3: The classes involved in the storage procedure for settings in cookies and project files. Cookies are stored in the cookies version and project files can be created in the standalone version of the Liquid Diagrams framework.

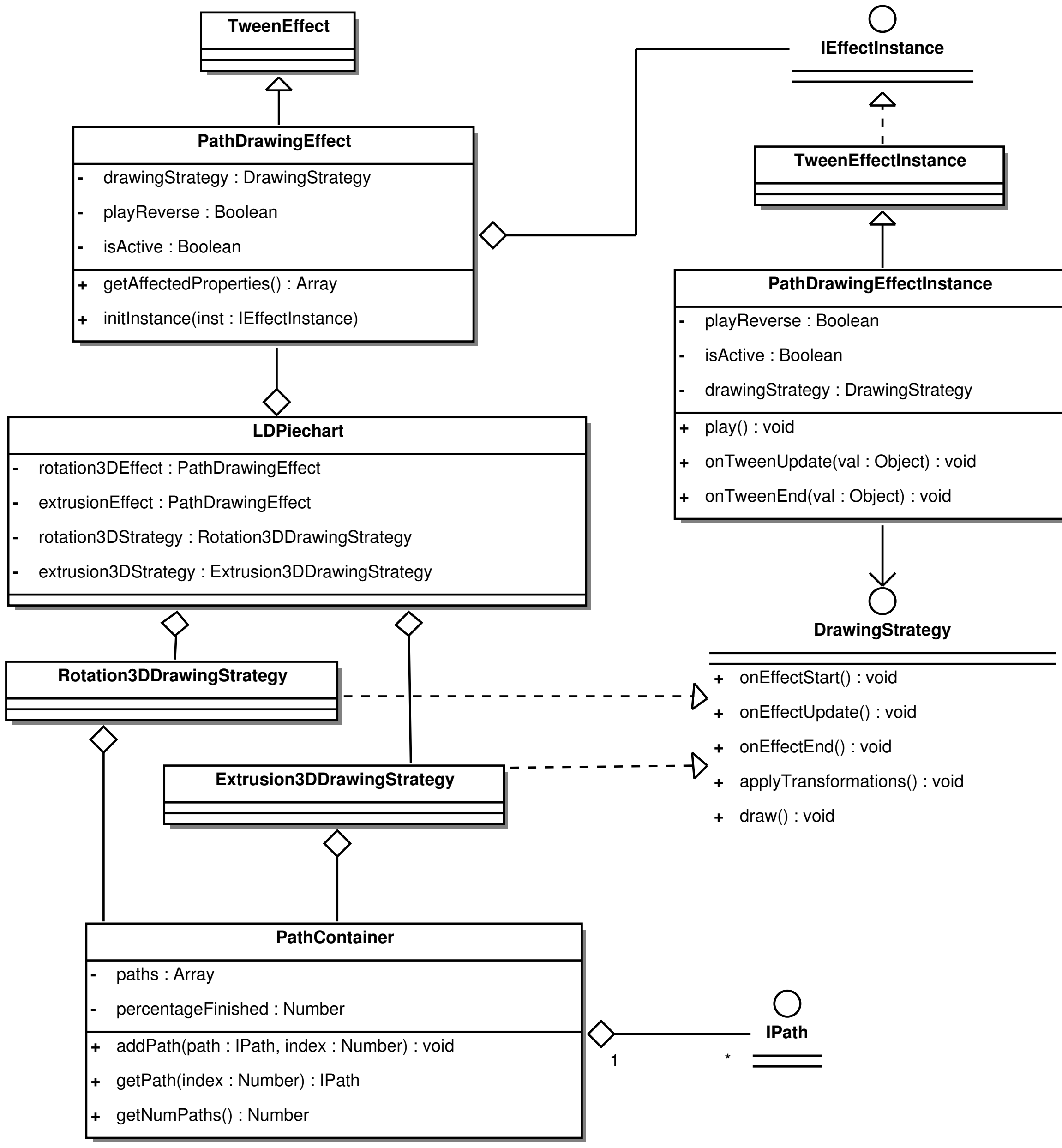

TweenEffect
PathDrawingEffect
- drawingStrategy : DrawingStrategy
- playReverse : Boolean
- isActive : Boolean
+ getAffectedProperties() : Array
+ initInstance(inst : IEffectInstance)
IEffectInstance
TweenEffectInstance
PathDrawingEffectInstance
- playReverse : Boolean
- isActive : Boolean
- drawingStrategy : DrawingStrategy
+ play() : void
+ onTweenUpdate(val : Object) : void
+ onTweenEnd(val : Object) : void
LDPiechart
- rotation3DEffect : PathDrawingEffect
- extrusionEffect : PathDrawingEffect
- rotation3DStrategy : Rotation3DDrawingStrategy
- extrusion3DStrategy : Extrusion3DDrawingStrategy
DrawingStrategy
+ onEffectStart() : void
+ onEffectUpdate() : void
+ onEffectEnd() : void
+ applyTransformations() : void
+ draw() : void
Rotation3DDrawingStrategy
Extrusion3DDrawingStrategy
PathContainer
- paths : Array
- percentageFinished : Number
+ addPath(path : IPath, index : Number) : void
+ getPath(index : Number) : IPath
+ getNumPaths() : Number
IPath
1
*

Figure 8.4: Classes involved in the 3D starting procedure. The diagram includes all classes engaged in performing the drawing effects in the pie chart visualisation.

```
private function play3DRotationEffect(reverse:Boolean = false) : void {

     setUpRotationStrategy();

     rotation3DEffect.drawingStrategy = rotation3DStrategy;
     rotation3DEffect.duration = 1500;
     rotation3DEffect.playReverse = reverse;
     rotation3DEffect.target = pathContainer;
     rotation3DEffect.play();
}

private function playExtrusionEffect(reverse:Boolean = false) : void {

     setUpExtrusionStrategy();

     extrusion3DStrategy.setMatrix(rotation3DStrategy.getMatrix());

     extrusionEffect.drawingStrategy = extrusion3DStrategy;
     extrusionEffect.duration = 2000;
     extrusionEffect.startDelay = 500;
     extrusionEffect.playReverse = reverse;
     extrusionEffect.target = pathContainer;
     extrusionEffect.play();
}
```

Listing 8.5: This listing shows the functions triggering the start of the rotation and extrusion effects in the pie chart visualisation. In each function, the implementation of the interface DrawingStrategy (`rotation3Deffect` or `extrusionEffect`) is initialised and the effect is started.

An example of the use of the decorator pattern is the implementation of the function `addPoint()` in class PieChartPathDecorator. The PieChartPathDecorator is responsible for all instances of IPath containing sectors in the pie chart gadget. Since sectors of a pie chart are parts of a circle, each outer line is curved. The curvature is defined by the start point, the end point, and a control point. However, the default implementation of a Path does not contain any functionality regarding the creation and saving of control points. For this reason, the `addPoint()` function within the PieChartPathDecorator creates and stores its own control points. In addition, it delegates the saving procedure of the start point and the end point (which is in fact the start point of the next pie sector) to the default implementation.

8.3.1 Rotation

The code implementing the rotation of IPaths can be seen in Listings 8.6, 8.7, and 8.8. The function `prepareTransformationMatrix()` within the class Rotation3DDrawingStrategy (shown in Listing 8.6) prepares the rotation matrices in addition to the translation parameters. these variables are then passed to the function `performTransformation()` (seen in Listing 8.7) in the class Path. It calls the function `Utils3D.transformVector()` for each point within the Path. Finally, the points are transformed using the class Utils3D. This functionality can be seen in Listing 8.8.

The mathematical calculation of a rotation is encapsulated in Flex in order to make its use more comfortable. Following Foley et al. [1990, page 215], the mathematical calculation of a rotation using matrices is shown in Equation 8.1. Here, one of the rotation matrices seen in the Equations 8.2, 8.3, and 8.4 have to be inserted. An important issue with rotations is that the object has to be translated to the

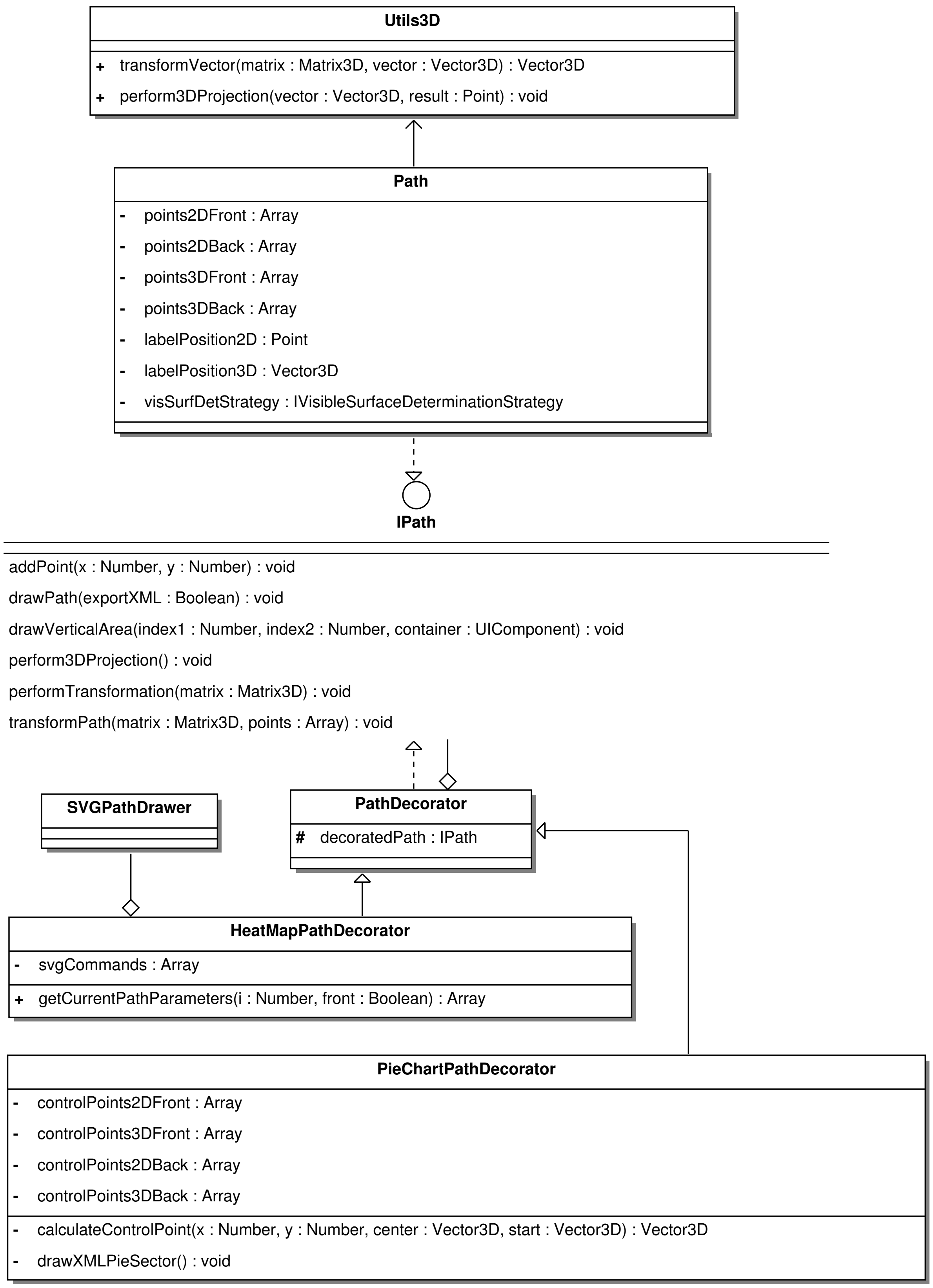

Utils3D
+ transformVector(matrix : Matrix3D, vector : Vector3D) : Vector3D
+ perform3DProjection(vector : Vector3D, result : Point) : void
Path
- points2DFront : Array
- points2DBack : Array
- points3DFront : Array
- points3DBack : Array
- labelPosition2D : Point
- labelPosition3D : Vector3D
- visSurfDetStrategy : IVisibleSurfaceDeterminationStrategy
IPath
addPoint(x : Number, y : Number) : void
drawPath(exportXML : Boolean) : void
drawVerticalArea(index1 : Number, index2 : Number, container : UIComponent) : void
perform3DProjection() : void
performTransformation(matrix : Matrix3D) : void
transformPath(matrix : Matrix3D, points : Array) : void
SVGPathDrawer
PathDecorator
# decoratedPath : IPath
HeatMapPathDecorator
- svgCommands : Array
+ getCurrentPathParameters(i : Number, front : Boolean) : Array
PieChartPathDecorator
- controlPoints2DFront : Array
- controlPoints3DFront : Array
- controlPoints2DBack : Array
- controlPoints3DBack : Array
- calculateControlPoint(x : Number, y : Number, center : Vector3D, start : Vector3D) : Vector3D
- drawXMLPieSector() : void

Figure 8.5: Classes involved in the final 3D drawing process. This involves the transformation and perspective projection of data entities.

```
private function prepareTransformationMatrix() : void {

      matrix = new Array();

      var matrix_z:Matrix3D = new Matrix3D();
      matrix_z.prependRotation(pathContainer.percentageFinished / 100 * rotationZ,
          Vector3D.Z_AXIS);

      matrix[0] = new Array();
      matrix[0].matrix = matrix_z;
      matrix[0].translationX = chartPanel.getDiagramOriginX();
      matrix[0].translationY = chartPanel.getDiagramOriginY();
      matrix[0].translationZ = 0;

      var matrix_y:Matrix3D = new Matrix3D();
      matrix_y.prependRotation(pathContainer.percentageFinished / 100 * rotationY,
          Vector3D.Y_AXIS);

      matrix[1] = new Array();
      matrix[1].matrix = matrix_y;
      matrix[1].translationX = chartPanel.getDiagramOriginX();
      matrix[1].translationY = chartPanel.getDiagramOriginY();
      matrix[1].translationZ = 0;

      var matrix_x:Matrix3D = new Matrix3D();
      matrix_x.prependRotation(pathContainer.percentageFinished / 100 * rotationX,
          Vector3D.X_AXIS);

      matrix[2] = new Array();
      matrix[2].matrix = matrix_x;
      matrix[2].translationX = pathContainer.getBiggestXCoord();
      matrix[2].translationY = pathContainer.getBiggestYCoord();
      matrix[2].translationZ = pathContainer.getPath(0).getZCoordinateBack();
}
```

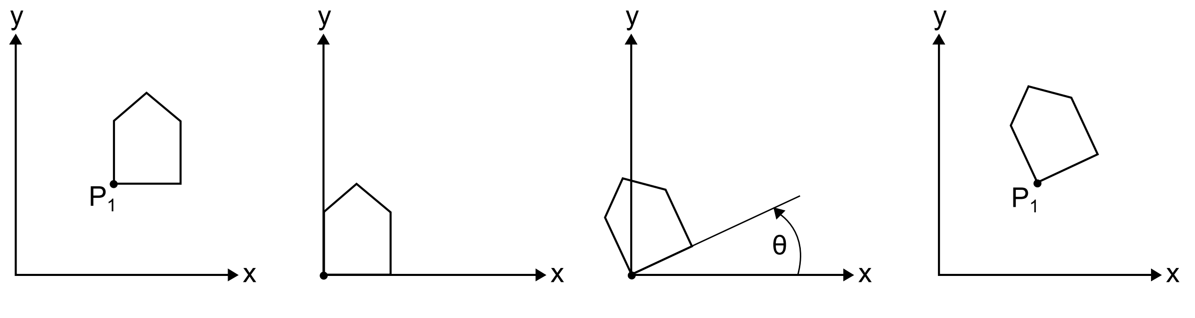
Listing 8.6: The function `prepareTransformationMatrix()` is located in the class Rotation3DDrawingStrategy and prepares the parameters necessary for a rotation and transition. The Flex framework encapsulates the mathematical background of this task. The direction of a rotation is specified using the constants `Vector3D.X_AXIS`, `Vector3D.Y_AXIS`, and `Vector3D.Z_AXIS`.

```
public function performTransformation(matrix:Matrix3D, translationX:Number = 0,
       translationY:Number = 0, translationZ:Number = 0) : void {

     points3DFront = transformPath(matrix, points3DFront, translationX,
          translationY, translationZ);
     points3DBack = transformPath(matrix, points3DBack, translationX,
          translationY, translationZ);

     // Label handling:
     labelPosition3D = Utils3D.transformVector(matrix, labelPosition3D,
          translationX, translationY, translationZ);
}

public function transformPath(matrix:Matrix3D, points:Array,
       translationX:Number = 0, translationY:Number = 0,
       translationZ:Number = 0) : Array {

     for (var i:Number = 0; i < points.length; i++) {
          points[i] = Utils3D.transformVector(matrix, points[i], translationX,
               translationY, translationZ);
     }

     return points;
}
```

Listing 8.7: This two functions represent the default implementation of transformations, located in the class Path. Here, the function `Utils3D.transformVector()` is called. It can be seen in Listing 8.8.

```
public static function transformVector(matrix:Matrix3D, vector:Vector3D,
       translationX:Number = 0, translationY:Number = 0,
       translationZ:Number = 0) : Vector3D {

     vector.x -= translationX;
     vector.y -= translationY;
     vector.z -= translationZ;

     vector = matrix.transformVector(vector);

     vector.x += translationX;
     vector.y += translationY;
     vector.z += translationZ;

     return vector;
}
```

Listing 8.8: A vector can be transformed using this function.



(a) The original house. **(b)** The point P_1 is translated to the origin. **(c)** The rotation is performed using the angle θ. **(d)** P_1 is translated back to its original position.

Figure 8.6: The rotation of an object. An object is first translated to the origin, rotated, then translated back to its original position. This figure is adapted from Foley et al. [1990, page 207].

origin, before the rotation is performed. This procedure can be seen in Figure 8.6.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R(\theta) * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{8.1}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.2}$$

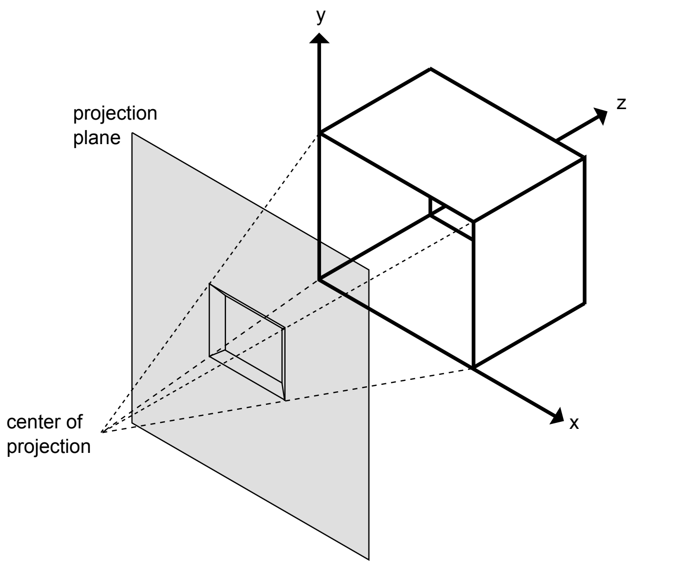
$$R_y(\theta) = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.3}$$

$$R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.4}$$

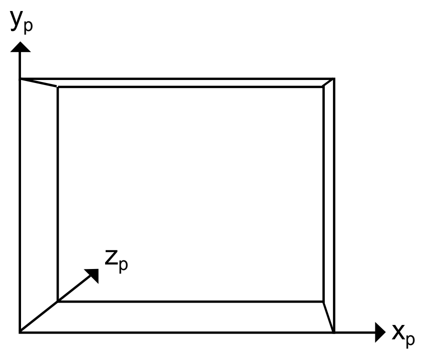
8.3.2 Perspective Projection

After the rotation of each point within an IPath, the points are still defined using three dimensions. In order to map the points to the two-dimensional display space, a perspective projection has to be performed. Liquid Diagrams uses a one-point perspective projection, as shown in Figure 8.7. The mathematical formula of a perspective projection can be seen in Equation 8.8. The terms $R(\theta_x)$, $R(\theta_y)$, and $R(\theta_z)$, defined using the Equations 8.5, 8.6, and 8.7 specify the rotation of the camera. The assignment of the variables used in the equations is:

θ: The rotation of the camera.



(a) Each point of a cube is mapped onto a plane cutting the z-axis.



(b) The resulting projection of the cube.

Figure 8.7: A one-point perspective projection. This image is adapted from Carlbom and Paciorek [1979, page 486].

a: The point in 3D space.

c: The position of the camera.

e: The position of the viewer.

b: The point in 2D space.

$$R(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix} \tag{8.5}$$

$$R(\theta_y) = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix} \tag{8.6}$$

$$R(\theta_z) = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.7}$$

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = R(\theta_x)R(\theta_y)R(\theta_z)\left(\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}\right) \tag{8.8}$$

Since no camera rotation is performed in the 3D extension of the Liquid Diagrams framework, the rotation matrices $R(\theta_x)$, $R(\theta_y)$, and $R(\theta_z)$ are equal to the identity matrix (see Equation 8.9) and therefore a perspective projection can be performed using Equation 8.10.

```
public static function perform3DProjection(vector:Vector3D, result:Point) : void {

     // Point in 3D space
     var ax:Number = vector.x;
     var ay:Number = vector.y;
     var az:Number = vector.z;

     var dx:Number;
     var dy:Number;
     var dz:Number;

     var tmp1:Number;
     var tmp2:Number;

     with (Math) {

     //     In case the camera rotation is not 0 some day in the future:
     //     tmp1 = (cos(angleY) * (az - cz) + sin(angleY) *
     //          (sin(angleZ) * (ay - cy) + cos(angleZ) * (ax - cx)));
     //     tmp2 = (cos(angleZ) * (ay - cy) - sin(angleZ) * (ax - cx));
     //
     //     dx = cos(angleY) * (sin(angleZ) * (ay - cy) + cos(angleZ) *
     //          (ax - cx)) - sin(angleY) * (az - cz);
     //     dy = sin(angleX) * tmp1 + cos(angleX) * tmp2;
     //     dz = cos(angleX) * tmp1 - sin(angleX) * tmp2;

          dx = ax - cx;
          dy = ay - cy;
          dz = az - cz;
     }

     result.x = ((dx - ex) * (ez / dz)) + cx;
     result.y = ((dy - ey) * (ez / dz)) + cy;
}
```

Listing 8.9: The function `perform3DProjection()`, located in class `Utils3D`, implements perspective projection.

$$R(\theta_x) = R(\theta_y) = R(\theta_z) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.9}$$

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \left(\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} - \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} \right) \tag{8.10}$$

In the last step, the transformed points have to be mapped onto a two-dimensional plane. Here, Equations 8.11 and 8.12 are used. The implementation of this functionality can be seen in Listing 8.9.

$$b_x = (d_x - e_x)(\frac{e_z}{d_z}) \tag{8.11}$$

$$b_y = (d_y - e_y)(\frac{e_z}{d_z}) \tag{8.12}$$

8.3.3 Visible Surface Determination

In order to draw the polygons of the IPaths in a correct order, a strategy concerning the determination of visible surfaces had to be implemented. Here, the Liquid Diagrams framework uses the depth-sort algorithm, introduced by Newell et al. [1972] in 1972. The depth-sort algorithm sorts the polygons according to their distance from the camera and draws them in order of decreasing distance. This means that polygons which are further away from the camera are drawn before polygons which are closer to the camera. This algorithm is also known as the painter's algorithm. [Foley et al., 1990, pages 673–675]

Chapter 9

Outlook And Future Work

Version 2.0 of the Liquid Diagrams framework makes a variety of improvements to the overall user experience and the availability of new functionality over version 1.0. However, there are still many aspects and features which could be improved or added in the future:

- **Support for mobile devices:** Since the structure of the framework was refactored by implementing the standalone version of the Liquid Diagrams framework, it could be easily extended to output a separate version targeted for use with mobile devices, such as mobile phones or tablets. This is possible because Adobe Flex add support for these devices in version 4.5.
- **Recreation of the cartographic material:** Section 6.9.4 described that it would be possible to display areas of different hierarchies within a single map in the heatmap visualisation. This functionality is mostly already implemented, but the downside of this approach is that the underlying SVG maps have to be very accurate in order to support this functionality. Unfortunately, this is not the case with the current SVG maps. For this reason, the SVG maps might be edited and made consistent.
- **An application for creating SVG maps:** Currently, the insertion of new SVG maps into the Liquid Diagrams framework is non-trivial. A graphical application which automates most of the tasks during the creation process of SVG maps could be created. The user could load SVG files into the application, make modifications (such as setting the name for a region), immediately see the resulting map, and save them back to the file system. This would be an enormous improvement regarding the map creation process.

Additionally, the following suggestions concerning future work were proposed by Lessacher [2010] and are still relevant:

- **Web Site:** A web site with spreadsheet functionality would mean an online version of Liquid Diagrams which could be independent from Google Docs. It could be similar to Many Eyes (described in Section 4.1.1), in enabling users to share visualisations among each other and to collaboratively explore and discuss data.
- **Scatterplot:** A scatterplot visualisation would be a useful extension to the framework because Liquid Diagrams currently does not support this visualisation type.
- **Time element:** A slider component could be implemented, enabling the functionality to look at changes in the data over time, similar to that in Gapminder (Motion Chart), discussed in Section 4.1.3.

Chapter 10

Concluding Remarks

This thesis began with an overview of the field of information visualisation, already existing information visualisation systems, and technologies which can be used to develop such systems. The main part of this thesis discussed and outlined the current version 2.0 of the Liquid Diagrams framework, a collection of gadgets enabling users to create highly interactive visualisations. Each visualisation can be exported as both raster graphics (PNG) and vector graphics (SVG).

It was argued that the amount of data produced every year reaches ever-higher dimensions. To process this amount of data and to extract information is highly difficult. However, the field of information visualisation takes the human visual perception system into account in order to present data in a well-structured and understandable manner. Interest in the field of information visualisation has been continuously expanding over recent years. It can be assumed that this trend will continue in the foreseeable future. Nowadays, people want to create visualisations using their own data and want to share them with others. This allows users to collaboratively explore data and gain new insights.

The work described in this thesis implements many new features regarding the user interface of Liquid Diagrams in order to improve the general user experience. In addition, advanced functionality such as the 3D extension was added. The Liquid Diagrams framework has now reached sufficient maturity that it can be released to a wider audience.

Bibliography

Adobe [2011a]. *Adobe AIR*. Adobe Systems. http://www.adobe.com/products/air.html. (Cited on page 25.)

Adobe [2011b]. *Adobe AIR - Deliver Rich Internet Applications on the Desktop*. Adobe Systems. wwwimages.adobe.com/www.adobe.com/products/air/pdfs/air_flex_datasheet.pdf. (Cited on page 25.)

Adobe [2011c]. *Adobe Flash Platform*. Adobe Systems. http://www.adobe.com/flashplatform/. (Cited on page 23.)

Adobe [2011d]. *Adobe Flex*. Adobe Systems. http://www.adobe.com/products/flex.html. (Cited on page 23.)

Adobe [2011e]. *Adobe Illustrator CS5*. Adobe Systems. http://www.adobe.com/products/illustrator.html. (Cited on page 28.)

Aichholzer, Oswin and Franz Aurenhammer [2002]. *Voronoi Diagrams - Computational Geometry's Favorite*. In *Special Issue on Foundations of Information Processing of TELEMATIK*, volume 1, pages 7–11. Institute for Theoretical Computer Science, Graz University of Technology. http://www.igi.tugraz.at/auren/psfiles/aa-vdcgf-02.ps.gz. (Cited on page 15.)

amCharts [2011]. *amCharts*. http://www.amcharts.com/. (Cited on page 43.)

Andrews, Keith [2011a]. *Information Visualisation*. Institute for Information Systems and Computer Media at Graz University of Technology. http://courses.iicm.tugraz.at/ivis/. Course Web Site. (Cited on pages 82, 93 and 99.)

Andrews, Keith [2011b]. *Information Visualisation, Course Notes*. http://courses.iicm.tugraz.at/ivis/ivis.pdf. (Cited on page 6.)

Andrews, Keith [2011c]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. http://ftp.iicm.edu/pub/keith/thesis/. (Cited on page xi.)

Andrews, Keith, Wolfgang Kienreich, Vedran Sabol, Jutta Becker, Georg Droschl, Frank Kappe, Michael Granitzer, Peter Auer, and Klaus Tochtermann [2002]. *The InfoSky Visual Explorer: Exploiting Hierarchical Structure and Document Similarities*. *Information Visualization*, 1, pages 166–181. ISSN 1473-8716. doi:10.1057/palgrave.ivs.9500023. (Cited on pages 15 and 19.)

Andrews, Keith and Martin Lessacher [2010]. *Liquid Diagrams: Information Visualisation Gadgets*. In *Proc. 14th International Conference Information Visualisation (IV'10)*, pages 104–109. IEEE Computer Society, Washington, DC, USA. ISBN 0769541658. doi:10.1109/IV.2010.100. (Cited on pages 1 and 45.)

Anscombe, Francis John [1973]. *Graphs in Statistical Analysis. The American Statistician*, 27(1), pages 17–21. `http://www.jstor.org/pss/2682899`. (Cited on pages 1 and 2.)

Apache [2011]. *OpenOffice Calc*. Apache Software Foundation (ASF). `http://www.openoffice.org/product/calc.html`. (Cited on page 38.)

as3xls [2011]. *as3xls - Read and Write Excel Files in Flex*. Google. `http://code.google.com/p/as3xls/`. (Cited on page 52.)

Axiis [2011]. *Axiis*. `http://www.axiis.org/index.html`. (Cited on pages 43 and 44.)

Bajramovic, Faruk, Arne Tauber, Ralph Woezelka, and Ferdinand Wörister [2011]. *Project Report - G05 - Force Directed Placement*. Graz University of Technology. Project Report. (Cited on pages 93 and 96.)

Balbi, Adriano and André-Michel Guerry [1829]. *Statistique comparée de l'état de l'instruction et du nombre des crimes dans les divers arrondissements des Académies et des Cours Royales de France*. Jules Renouard, Paris. BL:Tab.597.b.(38); BNF: Ge C 9014. (Cited on pages 15 and 17.)

Balzer, Michael and Oliver Deussen [2005]. *Voronoi Treemaps. Proc. IEEE Symposium on Information Visualization (InfoVis 2005)*, pages 49–56. ISSN 1522-404x. doi:10.1109/INFOVIS.2005.40. `http://kops.ub.uni-konstanz.de/xmlui/bitstream/handle/urn:nbn:de:bsz:352-opus-27261/Voronoi_Treemaps.pdf`. (Cited on page 19.)

Balzer, Michael, Oliver Deussen, and Claus Lewerentz [2005]. *Voronoi Treemaps for the Visualization of Software Metrics*. In *Proc. ACM Symposium on Software Visualization (SoftVis'05)*, pages 165–172. ACM, New York, NY, USA. ISBN 1595930736. doi:10.1145/1056018.1056041. `http://kops.ub.uni-konstanz.de/xmlui/bitstream/handle/urn:nbn:de:bsz:352-opus-24173/Voronoi_Treemaps_for_the_Visualization_of_Software_Metrics_2005.pdf`. (Cited on page 19.)

Barth, Adam [2011]. *HTTP State Management Mechanism*. RFC 6265 (Proposed Standard). `http://www.ietf.org/rfc/rfc6265.txt`. (Cited on pages 56 and 57.)

Bertin, Jacques [1967]. *Sémiologie graphique: Les diagrammes - Les réseaux - Les cartes*. Editions de l'Ecole des Hautes Etudes en Sciences. ISBN 2713212774. (Cited on page 3.)

Bertin, Jacques [1983]. *Semiology of Graphics*. University of Wisconsin Press. ISBN 0299090604. (Cited on page 3.)

Bloomberg [2011]. *Tableau Software, Inc.* Bloomberg Businessweek. `http://investing.businessweek.com/businessweek/research/stocks/private/snapshot.asp?privcapId=11421199`. (Cited on page 37.)

Bostock, Michael [2011]. *d3.js*. `http://mbostock.github.com/d3/`. (Cited on pages 40 and 42.)

Bostock, Michael and Jeffrey Heer [2009]. *Protovis: A Graphical Toolkit for Visualization. IEEE Transactions on Visualization and Computer Graphics*, 15, pages 1121 – 1128. ISSN 1077-2626. doi:10.1109/TVCG.2009.174. `http://vis.stanford.edu/files/2009-Protovis-InfoVis.pdf`. (Cited on page 40.)

Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer [2011]. *D^3 Data-Driven Documents. IEEE Transactions on Visualization and Computer Graphics*, 17, pages 2301–2309. ISSN 1077-2626. doi:10.1109/TVCG.2011.185. `http://vis.stanford.edu/files/2011-D3-InfoVis.pdf`. (Cited on pages 40 and 41.)

Brody, Howard, Michael R. Rip, Peter Vinten-Johansen, Nigel Paneth, and Stephen Rachman [2000]. *Map-Making and Myth-Making in Broad Street: The London Cholera Epidemic, 1854*. *The Lancet*, 356(9223), pages 64–68. doi:10.1016/S0140-6736(00)02442-9. `http://www.casa.ucl.ac.uk/martin/msc_gis/map_making_myth_making.pdf`. (Cited on page 3.)

Bruls, Mark, Kees Huizing, and Jarke van Wijk [2000]. *Squarified Treemaps*. In de Leeuw, W. and R. van Liere (Editors), *Proc. 2000 Joint Eurographics and IEEE TCVG Symposium on Visualization (VisSym'00)*, pages 33–42. Eurographics Association, Amsterdam, The Netherlands. ISBN 3211835156. ISSN 1727-5296. `http://www.win.tue.nl/~vanwijk/stm.pdf`. (Cited on page 93.)

Carlbom, Ingrid and Joseph Paciorek [1979]. *Geometric Projection and Viewing Transformations*. *ACM Computing Surveys*, 11, pages 280–. ISSN 0360-0300. doi:10.1145/356778.356788. `http://cs.uns.edu.ar/cg/clasespdf/p465carlbom.pdf`. (Cited on pages xi and 120.)

Chalmers, Matthew [1996]. *A Linear Iteration Time Layout Algorithm for Visualising High-Dimensional Data*. In *Proc. 7th Conference on Visualization '96 (VIS'96)*, pages 127–131. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0897918649. `www.dcs.gla.ac.uk/~matthew/papers/vis96.pdf`. (Cited on page 96.)

Chambers, John M., William S. Cleveland, Paul A. Tukey, and Beat Kleiner [1983]. *Graphical Methods for Data Analysis*. Duxbury Press. ISBN 053498052X. (Cited on pages 10 and 11.)

COLOURlovers [2011]. *COLOURlovers*. `http://www.colourlovers.com/`. (Cited on page 75.)

Corel [2011]. *CorelDRAW Graphics Suite X5*. Corel Corporation. `http://www.corel.com/servlet/Satellite/us/en/Product/1191272117978`. (Cited on page 28.)

Degrafa [2011]. *Degrafa*. `http://www.degrafa.org/`. (Cited on page 43.)

Descartes, René [1644]. *Le Monde de Mr Descartes, ou Le Trait de la Lumière*. (Cited on page 15.)

d'Ocagne, Maurice [1885]. *Coordonnées parallèles & axiales: méthode de transformation géométrique et procédé nouveau de calcul graphique déduits de la considération des coordonnées parallèles*. Cornell University Library. ISBN 1429700971. (Cited on page 10.)

DST [2011]. *Imports and Exports*. Statistics Denmark. `http://www.dst.dk/HomeUK/Statistics/Key_indicators/ExtTrade/ImpExp.aspx`. (Cited on page 8.)

Dupin, Charles [1826]. *Carte figurative de l'instruction populaire de la France*. Jobard. (Cited on pages 15, 17 and 84.)

Eades, Peter [1984]. *A Heuristic for Graph Drawing*. *Congressus Numerantium*, 42, pages 149 – 160. `www.cs.usyd.edu.au/~peter/old_spring_paper.pdf`. (Cited on page 96.)

EC [2011]. *Bilateral Relations - Statistics*. European Commission. `http://ec.europa.eu/trade/creating-opportunities/bilateral-relations/statistics/`. (Cited on page 8.)

Elementary Project [2011]. *elementary Icons 2.4*. `http://elementaryos.org/`. (Cited on page 66.)

Fernitz, Gernot, Dieter Ladenhauf, Christian Partl, and Patrick Plaschzug [2010]. *Implementation of the Bat's Wing Diagram and the Polar Area Diagram as Google Docs Gadget*. Graz University of Technology. Project Report. (Cited on pages 82 and 84.)

Fisher, Ronald Aylmer [1936]. *The Use of Multiple Measurements in Taxonomic Problems*. *Annals of Eugenics*, 7(7), pages 179–188. ISSN 1469-1809. doi:10.1111/j.1469-1809.1936.tb02137.x. `http://digital.library.adelaide.edu.au/dspace/bitstream/2440/15227/1/138.pdf`. (Cited on page 42.)

Foley, James D., Andries van Dam, Steven K. Feiner, and John F. Hughes [1990]. *Computer Graphics: Principles and Practice, Second Edition*. Addison-Wesley Professional. ISBN 0201848406. (Cited on pages 115, 119 and 122.)

Friendly, Michael [2007]. *A.-M. Guerry's Moral Statistics of France: Challenges for Multivariable Spatial Analysis*. *Statistical Science*, 22(3), pages 368–399. doi:10.1214/07-STS241. www.datavis.ca/papers/guerry-STS241.pdf. (Cited on page 15.)

Friendly, Michael and Daniel Denis [2005]. *The Early Origins and Development of the Scatterplot*. *Journal of the History of the Behavioral Sciences*, 41(2), pages 103–130. ISSN 1520-6696. doi:10.1002/jhbs.20078. http://www.datavis.ca/papers/friendly-scat.pdf. (Cited on page 13.)

Friendly, Michael and Daniel J. Denis [2011]. *Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization*. http://www.datavis.ca/milestones/. (Cited on pages 15 and 17.)

Fruchterman, Thomas M. J. and Edward M. Reingold [1991]. *Graph Drawing by Force-Directed Placement*. *Software: Practice and Experience*, 21, pages 1129–1164. ISSN 0038-0644. doi:10.1002/spe.4380211102. http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.13.8444&rep=rep1&type=pdf. (Cited on page 96.)

Gamma, Erich, Richard Helm, Ralph E. Johnson, and John Vlissides [1995]. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA. ISBN 0201633612. (Cited on pages 98, 99, 106, 109, 111 and 112.)

Gapminder [2011]. *About Gapminder*. http://www.gapminder.org/about-gapminder/. (Cited on page 33.)

Garcia Belmonte, Nicolas [2011]. *Sencha Acquires the JavaScript InfoVis Toolkit*. http://blog.thejit.org/2010/11/06/sencha-acquires-the-javascript-infovis-toolkit/. (Cited on page 43.)

Gay, Jonathan [2011]. *The History of Flash*. Adobe Systems. http://www.adobe.com/macromedia/events/john_gay/index.html. (Cited on page 23.)

GCS [2011]. *Scottish Export Statistics - Global Connections Survey*. Scottish Global Connections Survey. http://www.scotland.gov.uk/Topics/Statistics/Browse/Economy/Exports/GCSData. (Cited on page 8.)

Google [2011a]. *About Google Docs*. http://docs.google.com/support/bin/answer.py?answer=49008. (Cited on page 36.)

Google [2011b]. *About Google Gadgets*. http://code.google.com/apis/gadgets/. (Cited on page 36.)

Google [2011c]. *Google Docs*. http://docs.google.com. (Cited on pages 28 and 45.)

Google [2011d]. *Google Image Charts*. http://code.google.com/apis/chart/image/. (Cited on page 34.)

Google [2011e]. *Motion Chart*. http://code.google.com/apis/chart/interactive/docs/gallery/motionchart.html. (Cited on page 33.)

Granitzer, Michael, Wolfgang Kienreich, Vedran Sabol, Keith Andrews, and Werner Klieber [2004]. *Evaluating a System for Interactive Exploration of Large, Hierarchically Structured Document Repositories*. In *Proc. 10th IEEE Symposium on Information Visualization (InfoVis 2004)*, pages 127–134. ISSN 1522-404X. doi:10.1109/INFVIS.2004.19. (Cited on pages 15 and 19.)

Haintz, Christian, Stephan Moser, Michael Musenbrock, and Karin Pichler [2011]. *Flash/Flex Standalone Gadget - Liquid Diagrams*. Graz University of Technology. Project Report. (Cited on page 99.)

HCIL [2011]. *Treemap*. Human-Computer Interaction Lab. `http://www.cs.umd.edu/hcil/treemap/index.shtml`. (Cited on pages 94 and 95.)

Henry, Nathalie [2008]. *Exploring Social Networks with Matrix-based Representations*. PhD Thesis, Universite Paris Sud. `http://research.microsoft.com/en-us/um/people/nath/docs/Henry_thesis_oct08.pdf`. (Cited on page 3.)

Hilbert, Martin and Priscila López [2011]. *The World's Technological Capacity to Store, Communicate, and Compute Information*. *Science*, 332(6025), pages 60–65. ISSN 1095-9203. doi:10.1126/science.1200970. `http://www.sciencemag.org/content/332/6025/60.full.pdf`. (Cited on page 1.)

IBM [2011a]. *Fernanda B. Viégas*. `http://www.research.ibm.com/visual/fernanda.html`. (Cited on page 31.)

IBM [2011b]. *Many Eyes*. `http://many-eyes.com/`. (Cited on page 31.)

Inkscape [2011]. *Inkscape*. `http://inkscape.org/`. (Cited on page 28.)

Inselberg, Alfred [1985]. *The Plane with Parallel Coordinates*. *The Visual Computer*, 1(2), pages 69–91. ISSN 0178-2789. doi:10.1007/BF01898350. (Cited on page 10.)

Inselberg, Alfred [2009]. *Parallel Coordinates: Visual Multidimensional Geometry and Its Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN 0387215077. (Cited on page 10.)

ISO [2011]. *Country Names and Code Elements*. International Organization for Standardization. `http://www.iso.org/iso/country_codes/iso_3166_code_lists/country_names_and_code_elements.htm`. (Cited on pages 48 and 49.)

Jern, Mikael [2009]. *Collaborative Web-Enabled Geoanalytics Applied to OECD Regional Data*. In *Proc. 6th International Conference on Cooperative Design, Visualization, and Engineering (CDVE'09)*, pages 32–43. Springer-Verlag, Berlin, Heidelberg. ISBN 3642042643. `http://books.google.com/books?id=Z35Z49MHUmYC&pg=PA32`. (Cited on page 32.)

Johnson, Brian and Ben Shneiderman [1991]. *Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures*. In *Proc. 2nd Conference on Visualization (VIS'91)*, pages 284–291. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0818622458. `http://drum.lib.umd.edu/bitstream/1903/370/2/CS-TR-2657.pdf`. (Cited on pages 15 and 93.)

Kamada, Tomihisa and Satoru Kawai [1989]. *An Algorithm for Drawing General Undirected Graphs*. *Information Processing Letters*, 31(1), pages 7 – 15. ISSN 0020-0190. doi:10.1016/0020-0190(89)90102-6. `http://cs.wellesley.edu/~cs315/Papers/Kamada-Graphpdf`. (Cited on page 96.)

Know-Center [2011]. *InfoSky Demo*. `http://knowminer.know-center.tugraz.at/infosky-demo`. (Cited on page 15.)

Lessacher, Martin [2009]. *Liquid Diagrams: Visual Information Gadgets in Flex*. Graz University of Technology. Project Report. (Cited on pages 1 and 45.)

Lessacher, Martin [2010]. *Liquid Diagrams – A Suite of Visual Information Gadgets*. Master's Thesis, Graz University of Technology, Austria. `http://www.iicm.tugraz.at/thesis/MA_Martin_Lessacher.pdf`. (Cited on pages xi, 1, 45, 50, 52, 58, 61, 62, 75, 77, 79, 82, 84, 85, 88, 93, 97 and 123.)

Mariani, Gabriel and James Efstathiou [2011]. *Flashbug - An Extension for Firebug*. `http://blog.coursevector.com/flashbug`. (Cited on page 97.)

Mayr, Georg von [1877]. *Die Gesetzmässigkeit im Gesellschaftsleben, Statistische Studien*. R. Oldenbourg. `http://ia700208.us.archive.org/18/items/diegesetzmssig00mayruoft/diegesetzmssig00mayruoft.pdf`. (Cited on pages 10, 11 and 79.)

Mazza, Riccardo [2009]. *Introduction to Information Visualization*. Springer. ISBN 1848002181. (Cited on page 5.)

Microsoft [2011a]. *Expression Design 4*. `http://www.microsoft.com/expression/products/Design_Overview.aspx`. (Cited on page 28.)

Microsoft [2011b]. *Microsoft Excel*. `http://office.microsoft.com/excel/`. (Cited on page 38.)

Microsoft [2011c]. *Microsoft Silverlight*. `http://www.microsoft.com/silverlight/`. (Cited on page 25.)

Millward Brown [2011]. *Statistics : PC Penetration*. Adobe Systems Incorporated. `http://www.adobe.com/products/flashplatformruntimes/statistics.html`. (Cited on pages 21, 22 and 23.)

Mozilla [2011]. *FireBug - Web Development Evolved*. `http://getfirebug.com/`. (Cited on page 97.)

NComVA [2011a]. *About Norrköping Communicative Visual Analytics*. `http://www.ncomva.com/?page_id=59`. (Cited on page 32.)

NComVA [2011b]. *GAV Flash Toolkit*. `http://www.ncomva.com/?page_id=1106`. (Cited on page 41.)

NComVA [2011c]. *Statistics eXplorer Desktop Versions*. `http://www.ncomva.se/desktop/`. (Cited on page 37.)

NCVA [2011a]. *GAV Flash Tools*. `http://ncva.itn.liu.se/tools`. (Cited on pages 41 and 42.)

NCVA [2011b]. *National Center for Visual Analytics*. `http://ncva.itn.liu.se/ncva?l=en`. (Cited on page 32.)

NCVA [2011c]. *Open Statistics eXplorer*. `http://ncva.itn.liu.se/explorer/openexp?l=en`. (Cited on page 32.)

NCVA [2011d]. *Statistics eXplorer for Advanced Interactive Statistical Visualization*. `http://ncva.itn.liu.se/explorer?l=en`. (Cited on page 32.)

Newell, M. E., R. G. Newell, and T. L. Sancha [1972]. *A Solution to the Hidden Surface Problem*. In *Proc. ACM Annual Conference - Volume 1 (ACM'72)*, pages 443–450. ACM, New York, NY, USA. doi:10.1145/800193.569954. (Cited on page 122.)

NIST [2011]. *FIPS PUB 6-4 - Country Names and Codes of the US*. National Institute of Standards and Technology. `http://www.itl.nist.gov/fipspubs/fip6-4.htm`. (Cited on page 89.)

Noble, Joshua, Todd Anderson, Garth Braithwaite, Marco Casario, and Rich Tretola [2010]. *Flex 4 Cookbook*. O'Reilly Media. ISBN 0596805616. (Cited on page 23.)

NodeXL [2011]. *NodeXL*. `http://nodexl.codeplex.com/`. (Cited on page 38.)

Novell [2011]. *Moonlight*. Novell, Inc. `http://www.go-mono.com/moonlight/`. (Cited on page 25.)

Nuzha, Vasiliy [2010]. *Korax ColorPicker Control*. `http://kss.korax.ru/flex/cp/index.html`. (Cited on pages 62 and 65.)

OECD [2011]. *OECD Regional Statistics*. Organisation for Economic Co-operation and Development. `http://stats.oecd.org/OECDregionalstatistics/`. (Cited on page 32.)

Oracle [2011a]. *The History of Java Technology*. Oracle Corporation. `http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html`. (Cited on page 26.)

Oracle [2011b]. *Java*. Oracle Corporation. `http://www.oracle.com/us/technologies/java/`. (Cited on page 26.)

Oracle [2011c]. *The Java HotSpot Performance Engine Architecture*. Oracle Corporation. `http://java.sun.com/products/hotspot/whitepaper.html`. (Cited on page 26.)

Oracle [2011d]. *JavaFX*. Oracle Corporation. `http://javafx.com/`. (Cited on page 26.)

Parkinson, Cyril Northcote [1955]. *Parkinson's Law*. *The Economist*. `http://www.economist.com/node/14116121?story_id=14116121`. (Cited on page 1.)

Playfair, William [1786]. *The Commercial and Political Atlas: Representing, by Means of Stained Copper-Plate Charts, the Progress of the Commerce, Revenues, Expenditure and Debts of England During the Whole of the Eighteenth Century*. T. Burton. ISBN 0521855543. `http://books.google.com/books?id=lOWzGauOzSYC`. (Cited on pages 3, 7 and 8.)

Playfair, William [1801]. *The Statistical Breviary; Shewing the Resources of Every State and Kingdom in Europe*. T. Bensley, London. ISBN 0521855543. `http://books.google.com/books?id=Y4wBAAAAQAAJ`. (Cited on pages 3, 9, 40 and 41.)

Prekaski, Todd [2011]. *Building Flex and Adobe AIR Applications from the Same Code Base*. Adobe Systems. `http://www.adobe.com/devnet/air/flex/articles/flex_air_codebase.html`. (Cited on pages 100 and 105.)

Protovis [2011]. *Protovis*. Stanford Visualization Group. `http://mbostock.github.com/protovis/`. (Cited on pages 40 and 41.)

Ramos, Ernesto and David Donoho [2011]. *Car Dataset*. `http://stat-computing.org/dataexpo/1983.html`. (Cited on page 12.)

RIAStats [2011]. *Rich Internet Application Statistics*. DreamingWell. `http://www.riastats.com/`. (Cited on pages 21, 22 and 23.)

SenchaLabs [2011]. *JavaScript InfoVis Toolkit*. `http://thejit.org/`. (Cited on pages 41 and 43.)

Shneiderman, Ben [1992]. *Tree Visualization with Tree-Maps: 2-D Space-Filling Approach*. *ACM Transactions on Graphics*, 11, pages 92–99. ISSN 0730-0301. doi:10.1145/102377.115768. `http://hcil.cs.umd.edu/trs/91-03/91-03.html`. (Cited on pages xi, 15 and 18.)

Shneiderman, Ben [1996]. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. In *Proc. 1996 IEEE Symposium on Visual Languages (VL'96)*, pages 336–. IEEE Computer Society, Washington, DC, USA. ISBN 0-8186-7508-X. `http://www.cs.ubc.ca/~tmm/courses/cs533c-02/readings/shneiderman96eyes.pdf`. (Cited on page 6.)

Shneiderman, Ben [2009]. *Treemaps for Space-Constrained Visualization of Hierarchies*. `http://www.cs.umd.edu/hcil/treemap-history/index.shtml`. (Cited on pages 15 and 93.)

Small, Hugh [1998]. *Florence Nightingale's Statistical Diagrams*. In *Stats and Lamps Research Conference*. Florence Nightingale Museum. `http://www.york.ac.uk/depts/maths/histstat/small.htm`. (Cited on pages 13, 15 and 82.)

Spence, Ian [2005]. *No Humble Pie: The Origins and Usage of a Statistical Chart*. *Journal of Educational and Behavioral Statistics*, 30(4), pages 353–368. doi:10.3102/10769986030004353. `http://www.psych.utoronto.ca/users/spence/Spence%202005.pdf`. (Cited on pages 6 and 9.)

Spence, Robert [2007]. *Information Visualization: Design for Interaction*. 2nd Edition. Prentice-Hall, Upper Saddle River, NJ, USA. ISBN 0132065509. (Cited on pages 5 and 6.)

StatCounter [2011]. *Top 9 Mobile Browsers*. `http://gs.statcounter.com/#mobile_browser-ww-monthly-200812-201112`. (Cited on page 75.)

StatLib [2011]. *Cereal Dataset*. `http://lib.stat.cmu.edu/datasets/1993.expo/`. (Cited on pages 47 and 96.)

STATOWL [2011]. *Web Browser Plugin Market Share*. `http://www.statowl.com/plugin_overview.php`. (Cited on pages 21, 22 and 23.)

Steele, Julie and Noah Iliinsky [2011]. *Designing Data Visualizations: Intentional Communication from Data to Display*. O'Reilly Media. ISBN 1449312284. (Cited on page 7.)

Tableau [2011a]. *Tableau Public*. Tableau Software. `http://www.tableausoftware.com/products/public`. (Cited on page 37.)

Tableau [2011b]. *Who We Are*. Tableau Software. `http://www.tableausoftware.com/about/who-we-are`. (Cited on page 37.)

Tufte, Edward R. [1997]. *Visual Explanations: Images and Quantities, Evidence and Narrative*. Graphics Press. ISBN 0961392126. (Cited on page 3.)

Tufte, Edward R. [2001]. *The Visual Display of Quantitative Information*. 2nd Edition. Graphics Press, Cheshire, CT, USA. ISBN 0961392142. (Cited on pages 1, 3 and 7.)

Viégas, Fernanda B., Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon [2007]. *ManyEyes: A Site for Visualization at Internet Scale*. *IEEE Transactions on Visualization and Computer Graphics*, 13, pages 1121–1128. ISSN 1077-2626. doi:10.1109/TVCG.2007.70577. `http://www.research.ibm.com/visual/papers/viegasinfovis07.pdf`. (Cited on page 31.)

Viewpath [2011]. *Viewpath*. `http://www.viewpath.com/`. (Cited on page 36.)

VisuLab [2011]. *VisuLab - Interactive Data Visualisation in Microsoft Excel*. Swiss Federal Institute of Technology Zurich. `http://www.inf.ethz.ch/personal/hinterbe/Visulab/`. (Cited on pages 38 and 39.)

W3C [2011a]. *About SVG*. World Wide Web Consortium. `http://www.w3.org/Graphics/SVG/About.html`. (Cited on page 27.)

W3C [2011b]. *Cascading Style Sheets*. World Wide Web Consortium. `http://www.w3.org/Style/CSS/`. (Cited on page 21.)

W3C [2011c]. *Document Object Model (DOM)*. World Wide Web Consortium. `http://www.w3.org/DOM/`. (Cited on page 21.)

W3C [2011d]. *HTML*. World Wide Web Consortium. `http://www.w3.org/wiki/HTML`. (Cited on page 21.)

W3C [2011e]. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. World Wide Web Consortium. `http://www.w3.org/TR/SVG11/`. (Cited on page 27.)

W3C [2011f]. *Scripting and AJAX*. World Wide Web Consortium. `http://www.w3.org/standards/webdesign/script.html`. (Cited on page 21.)

W3C [2011g]. *The Secret Origin of SVG*. World Wide Web Consortium. `http://www.w3.org/Graphics/SVG/WG/wiki/Secret_Origin_of_SVG`. (Cited on page 27.)

W3C [2011h]. *SVG 1.1 (Second Edition): Document Type Definition*. World Wide Web Consortium. `http://www.w3.org/TR/SVG/svgdtd.html`. (Cited on page 88.)

w3schools [2011]. *DTD Tutorial*. Refsnes Data. `http://www.w3schools.com/DTD/default.asp`. (Cited on page 88.)

Wahlers, Claus and Max Herkender [2011]. *FZip*. `http://codeazur.com.br/lab/fzip/`. (Cited on page 52.)

Ward, Matthew O., Georges Grinstein, and Daniel Keim [2010]. *Interactive Data Visualization: Foundations, Techniques, and Applications*. A K Peters. ISBN 1568814739. (Cited on page 5.)

Ware, Colin [2004]. *Information Visualization: Perception for Design*. 2nd Edition. Morgan Kaufmann, San Francisco, CA, USA. ISBN 1558608192. (Cited on pages 5 and 6.)

Wikipedia [2011a]. *Diagram of the Causes of Mortality in the Army in the East*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Nightingale-mortality.jpg`. (Cited on page 16.)

Wikipedia [2011b]. *John Snow's Cholera Map*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Snow-cholera-map-1.jpg`. (Cited on page 4.)

Wikipedia [2011c]. *Minard's Map*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Minard.png`. (Cited on page 4.)

Wikipedia [2011d]. *One of William Playfair's First Pie Charts*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Playfair-piechart.jpg`. (Cited on page 9.)

Wikipedia [2011e]. *Plot of Anscombe's Quartet*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Anscombe%27s_quartet_3.svg`. (Cited on page 2.)

Wikipedia [2011f]. *Time Series of Exports and Imports of Denmark and Norway*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Playfair_TimeSeries-2.png`. (Cited on page 8.)

Wikipedia [2011g]. *William Playfair's First Bar Chart*. Wikimedia Commons. `http://en.wikipedia.org/wiki/File:Playfair_Barchart.gif`. (Cited on page 8.)

Will-Harris, Daniel [2003]. *Georgia & Verdana – Typefaces Designed for the Screen (Finally)*. `http://www.will-harris.com/verdana-georgia.htm`. (Cited on page 103.)

Wilson, Dan [2011]. *Fork of as3xls With Bugfixes*. GitHub Inc. `https://github.com/djw/as3xls/`. (Cited on page 52.)

Wright, John Kirtland [1938]. *Problems in Population Mapping. Notes on Statistical Mapping with Special Reference to the Mapping of Population Phenomena*. American Geographical Society, New York. (Cited on page 15.)